Code 582
Flight Software Branch

CORE FLIGHT SYSTEM HOUSEKEEPING APPLICATION BUILD 2.4.1.0

FLIGHT SOFTWARE BUILD VERIFICATON TEST REPORT

Flight Software Branch – Code 582

Version 1.0

SIGNATURES

Submitted by:

X WALTER MOLESKI

Digitally signed by WALTER MOLESKI
DN: c=US, o=U.S. Government, ou=NASA, ou=People, cn=WALTER MOLESKI, 0.9.2342.19200300.100.1.1=wmoleski
Date: 2016.11.09 15:12:25 -05'00'

Walt Moleski/582
cFS Flight Software Tester

Approved by:

X SUSANNE STREGE

Digitally signed by SUSANNE STREGE
DN: c=US, o=U.S. Government, ou=NASA, ou=People, cn=SUSANNE STREGE, 0.9.2342.19200300.100.1.1=sstrege
Date: 2016.11.09 14:20:50 -05'00'

Susanne Strege/582
cFS Flight Software Product Development Lead

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) Housekeeping (HK) Flight Software (FSW) Test Team build 2.4.1.0 verification testing. It is used to verify that the HK FSW has been tested in a manner that validates that it satisfies the functional and performance requirements defined within the cFS HK Requirements Document. This Test Report summarizes the FSW test history, the build verification process, the build test configuration, and the test execution and results.

1.2 APPLICABLE DOCUMENTS

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

Parent Documents (Mission and FSW)

- 582-2007-034 cFS Housekeeping Requirements Document, Version 1.1
- 582-2008-012 cFS Deployment Guide

Reference Documents

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

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

1.3 DOCUMENT ORGANIZATION

Section 1 of this document presents some introductory material.

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

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

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

Section 5 describes the test execution and results by subsystem.

Appendix A - provides the Requirements Traceability Matrix

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

1.4 DEFINITIONS

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

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

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

- Requirements Tested Passed: Requirement was fully tested in a build test procedure and passed all tests.
- Requirements Tested Failed: Requirement was fully tested in a build test procedure and failed one or more aspect of the testing.
- Requirements Tested Partially: Requirement was tested partially in a build test procedure. To be fully tested, the partially tested requirement 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 have been validated, needs to have a CCB to close the DCR
- Closed: The DCR is closed and have been resolved and tested to satisfaction
- Closed with Defect: The DCR is closed and the defect is most likely assigned a differed DCR number associated with another subsystem.

2 OVERVIEW

2.1 FLIGHT DATA SYSTEM CONTEXT

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

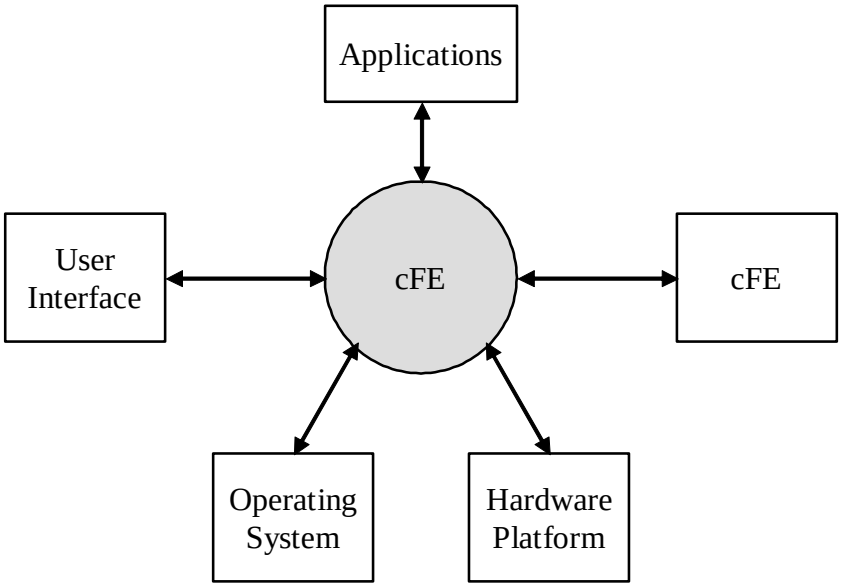

Figure 2-1 cFS System Context

The Housekeeping (HK) component of the Core Flight System (cFS) is responsible for building and sending combined telemetry messages from individual system applications. Combining messages is performed in order to minimize downlink telemetry bandwidth. Combining certain data from multiple messages into one message eliminates the message headers that would be required if each message was sent individually. HK provides the capability to generate multiple combined packets so that data can be organized and output at different rates (e.g. a fast, medium and slow packet).

Figure 2-2 shows the context diagram for the cFS Housekeeping (HK) Application. During initialization, HK subscribes to housekeeping messages from other applications. The Scheduler Application (SCH) sends periodic commands to HK. Ground commands come from the Command Ingest task (CI). Combined output messages, and events messages are routed to the appropriate task(s) by the cFE SB Application. The copy table defines the output message formats. HK learns of ground updates to the copy table through the cFE Table Services application.

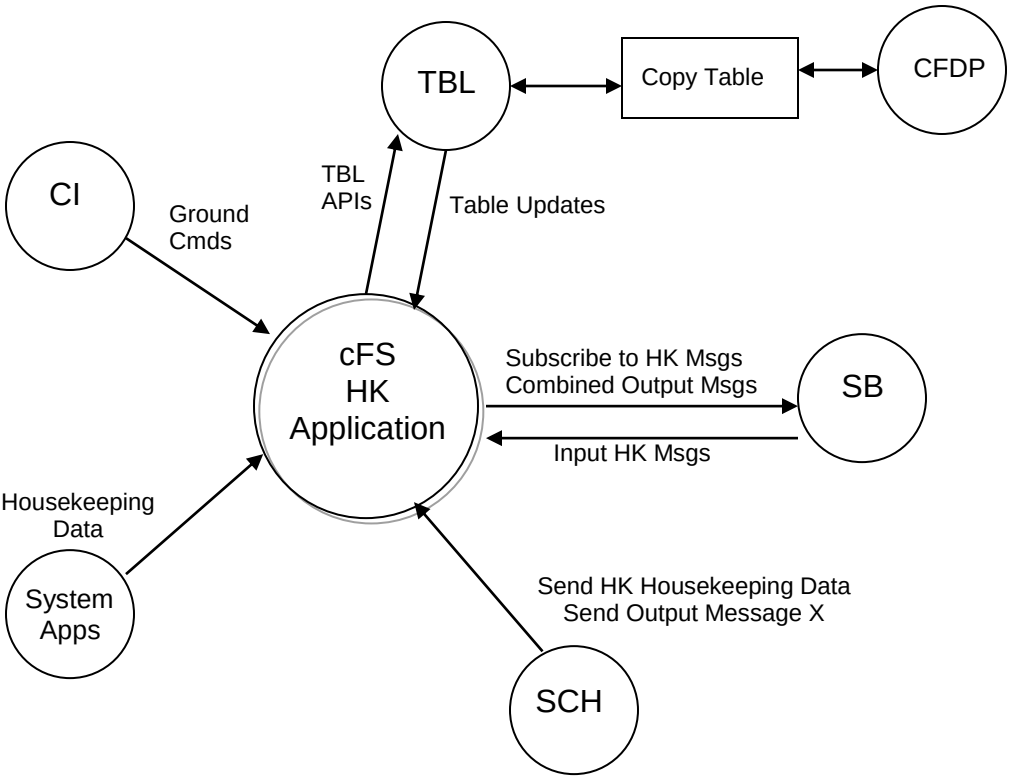

Figure 2-2 cFS HK Context

2.2 TEST HISTORY

HK 1.0.0.0 – Build Verification Testing completed by Walt Moleski 09/19/2008
HK 2.0.0.0 – Build Verification Testing completed by Walt Moleski 08/25/2009
HK 2.1.0.0 – Build Verification Testing completed by Walt Moleski 07/07/2010
HK 2.2.0.0 – Build Verification Testing completed by Walt Moleski 07/07/2011
HK 2.3.0.0 – Build Verification Testing completed by Walt Moleski 04/03/2012
HK 2.3.0.0a – Testing completed by Walt Moleski on 04/16/2012
HK 2.4.0.0 – Build Verification Testing completed by Walt Moleski on 08/30/2012
HK 2.4.0.0 – Build Verification Testing completed by Walt Moleski on 11/09/2016

2.3 TESTING OVERVIEW

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

- 1 test application: tst_hk
- 6 main test procedures: hk_gencmds, hk_basichousekeeping hk_missingdata, hk_stresshousekeeping, hk_stressmissingdata, hk_stresstableload
- 6 test procedures that are called by the main procedures: hk_sendoutmsg, hk_copytable1, hk_copytable2, hk_copytable3, hk_copytable4, hk_copytable5
- 1 header file used by the test procedures: cfs_hk_requirements. This header defines the requirements array that is used by all of the main test procedures except for hk_gencmds. It also defines constants that are used by these procedures.
- Tests require the ASIST Ground System

The tst_hk test application is used to send schedule requests for the output of HK's housekeeping data to the HK 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 also provides the ability send input messages to HK and to request that HK sends output messages to the ground. TST_HK has 2 ground commands that are used by the HK test procedures:

- SendInMsg: This command is used to simulate sending cFS application housekeeping packets to HK. The input to this command includes the MsgId (uint16), DataSize (uinit16 specifying 3, 4, 8, 16, or 32 bytes), and DataPattern(uint32). The output of this command is a packet sent on the software bus with a data portion that is the size specified in DataSize with data that is a repeating of the DataPattern to fill out the data size.
- SendOutMsg: This command is used to send a schedule request to the HK application to send a combined output packet down to the ground. The input to this command in a MsgId (uint 16) and the output is a message sent to the HK requested that the combined output packet with MsgID
-

These 6 main HK test procedures do the following:

Procedure	**Description**
hk_gencmds	The purpose of this test is to verify that Housekeeping (HK) general commands function properly. The HK_NoOp and HK_Reset commands will be tested as well as invalid commands and an application reset to see if the HK application behaves appropriately. It should be noted that this procedure uses the RAW command with hard-coded MsgIds to send invalid commands to the HK Application.
hk_basichousekeeping	The purpose of this test is to verify that Housekeeping (HK) can collect housekeeping data from an average number of input message streams (20) and combine the input message data into an average number of output messages (3). It also tests HK sending its housekeeping data and updating the copy table
hk_missingdata	The purpose of this test is to verify that Housekeeping (HK) correctly handles missing housekeeping packets. It also tests the collection of housekeeping data from an average number of input message streams (20). It will also test that HK can combine input message data into an average number of output messages (3). This test is executed with two configurations of the HK software. One with the Discard Combo Packets set to NO and one with it set to YES.
hk_stresshousekeeping	The purpose of this test is to stress the Housekeeping subsystems by setting up a copy table with a large number of input messages and the maximum number of output messages. It also tests sending invalid message ids in the Output Message x requests and receiving input packets whose lengths are smaller than what was defined in the copy table
hk_stressmissingdata	The purpose of this test is to stress the Housekeeping (HK) application by sending it data with a large number of input messages missing. This test is executed with two configurations of the HK software. One with the Discard Combo Packets set to NO and one with it set to YES.
hk_stresstableload	The purpose of this test is to stress Housekeeping (HK) by loading a new copy table 4 times in a row.

The 6 test procedures described in the table below are called by the 6 main test procedures. The purpose of the copytable procedures is to generate the files for the copytables used during BVT. Note that the message ids used are borrowed from the other cFS applications (MM, FM, MD, and SCH). The definition of the 5 copytables is included in Appendix C.

Procedure	**Description**
hk_sendoutmsg	The procedure is used by all tests that need to receive and validate HK Combined Output Packets. It sends a request to TST_HK to send a specific output packet and then verifies that the data is as expected.

Procedure	Description
hk_copytable1	This procedure defines a copytable with 20 input messages, 3 output messages, pieces of each input packet go to each output packet. Table will have 60 entries. Used for: GenCmds, BasicHousekeeping, MissingData, Stress Table Load
hk_copytable2	This procedure defines a copytable with 20 input messages, 3 output messages, pieces of each input packet go to each output packet. Similar to Table 1, but change 1 input packet id, 1 output packet id, and size of 1 output packet, data from 1 input packet should be output in only output packet 2. Used for: BasicHousekeeping, MissingData, StressTableLoad
hk_copytable3	This procedure defines a copytable with 2 input messages, 6 output messages (Table will have 128 entries). Used for StressHousekeeping, StressMissingData, StressTableLoad
hk_copytable4	This procedure defines a copytable with 11 input messages, 2 output messages, odd sized input and output packets. Used for StressHousekeeping, StressTableLoad
hk_copytable5	This procedure defines a copytable with 19 input messages, 4 output messages, data that has gaps (i.e. – copying 1...4 from input packet to 1...4 in output packet and then put 5...8 from a different input packet into 5...8 in output packet), odd sized output packets, odd byte copies. Used for StressHousekeeping, StressTableLoad

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.
CFE_shutdown	Directive combines the "close_data_center" and "exit" ASIST shutdown commands.
create_tbl_file_from_cvt	Procedure that creates a load file from the specified arguments and cvt
ftp_file	To ftp a file to/from the FSW/GSW.
load_start_app	Procedure to load and start a user application from the /s/opr/accounts/cfebx/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_hk (version 2.4.1.0)	Test application with 2 primary commands that the HK test procedures use to send messages to the HK application. They are: • sendinmsg: sends a simulated message to HK for process as defined in the copy table • sendoutmsg: sends a message to HK to send out the specified output message
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.

Name	Description
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
HK Requirements	1.1
HK Application	2.4.1.0
TST_HK Application	2.4.1.0
CFE	6.5.0.0
ASIST	20.2
VxWorks	6.9

3 BUILD VERIFICATION TEST PREPARATION

3.1 SCENERIO DEVELOPMENT

There were no new scenarios developed for build verification test 2.4.1.0. All scenarios are stored on the MKS server, in cFS-Repository HK test-and-ground directory within the test-review-packages subdirectory in the Scenarios folder. It should be noted that as HK 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 the 6 test procedures. All test procedures were written using the STOL scripting language. The naming convention for files created by the test procedures was: scx_cpu<#>_<procedure name>_GMT.<ext>.

3.3 TEST PRODUCTS

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

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

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

4 BUILD VERIFICATION TEST EXECUTION

4.1 TESTBED OVERVIEW

HK FSW testing took place in the cFS FSW Development and Test Facility. A high level view of the cFS FSW Test Bed is shown in Figure 4-1. This facility is located in GSFC Building 23, Room N410. This facility consists of two ASIST workstations running ASIST version 9.7k and three MPC750 CPU boards running VxWorks 6.4. CPU1 is primarily used for development testing while CPU2 and CPU3 are used for build verification testing.

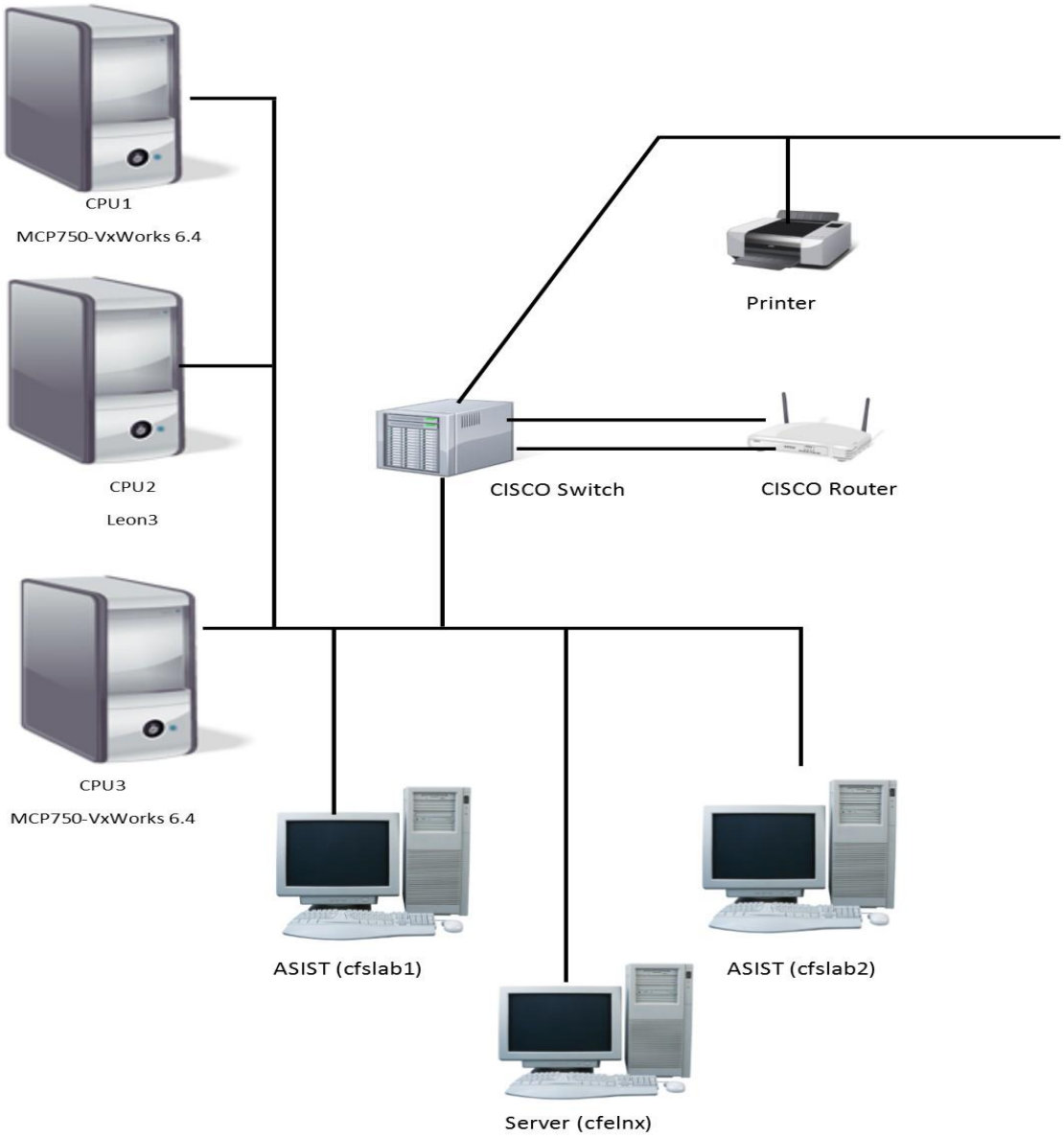

Figure 4-1 cFS FSW Development and Testing Facility

4.2 REQUIREMENTS VERIFICATION MATRIX

	Housekeeping (HK)
Requirements Tested Passed	15
Requirements Tested Failed	0
Requirements Tested Partially	0
Total Tested	15
Deferred	0
Total	15

4.3 REQUIREMENTS PARTIALLY TESTED

No requirements were partially tested.

4.4 REQUIREMENTS/FUNCTIONALITY DEFERRED

No requirements were deferred for later build testing

4.5 REQUIREMENTS/FUNCTIONALITY DEFERRED FOR MISSION TESTING

No requirements were deferred for mission testing.

5 BUILD VERFICIATON TEST RESULTS

5.1 OVERALL ASSESSMENT

During this build test of the HK Application, the software preformed as expected. Below is a summary of the results:

- 15 requirements passed demonstration
- 0 requirements were validated by analysis.
- 0 requirements were deferred for testing later
- 3 existing DCRs were validated

5.2 PROCEDURE DESCRIPTION

Procedure	Description	Requirements tested
HK_GenCmds	The purpose of this test is to verify that Housekeeping (HK) general commands function properly. The HK_NoOp and HK_Reset commands will be tested as well as invalid commands and an application reset to see if the HK application behaves appropriately.	HK1000, HK1001, HK1002, HK1003, HK1004, HK3000, HK4000
HK_BasicHousekeeping	The purpose of this test is to verify that Housekeeping (HK) can collect housekeeping data from an average number of input message streams (20) and combine the input message data into an average number of output messages (3). It also tests HK sending its housekeeping data and updating the copy table.	HK2000, HK2001, HK2001.1, HK3000, HK4000
HK_MissingData	The purpose of this test is to verify that Housekeeping (HK) correctly handles missing housekeeping packets. It also tests the collection of housekeeping data from an average number input message streams (20). It will also test that HK can combine input message data into an average number of output messages (3).	HK2000, HK2001, HK2001.2, HK2001.3, HK2001.5, HK2001.6, HK2001.7, HK3000, HK4000
HK_StressHousekeeping	The purpose of this test is to stress the Housekeeping subsystems by setting up a copy table with a large number of input and the maximum number of output messages. It also tests sending invalid message ids in the Output Message x requests and receiving input packets whose lengths are smaller than what was defined in the copy table	HK2000, HK2001, HK2001.1, HK2001.3, HK2001.5, HK3000, HK4000
HK_StressMissingData	The purpose of this test is to stress the Housekeeping (HK) application by sending it data with a large number of input messages missing.	HK2000, HK2001, HK2001.2, HK2001.3, HK2001.6, HK3000, HK4000

Procedure	Description	Requirements tested
HK_StressTableLoad	The purpose of this test is to stress Housekeeping (HK) by loading a new copy table 4 times in a row	HK2000, HK2001, HK2001.1, HK3000, HK4000

5.3 ANALYSIS REQUIREMENTS VERIFICATION

No requirements were verified using analysis.

5.4 DCRS

No new DCRs were generated during HK 2.4.1.0 testing.

5.4.1 DCRs Verified

The following DCRs were verified during testing.

DCR	Description	Test Method	Test Approach
4070	HK – Uninitialized variable causes compiler warning	Demonstration	Make process did not generate any compiler warnings
145911	HK – CFE_EVS_SendEvent format warnings	Demonstration	Make process did not generate any compiler warnings
145936	HK – Integrate and implement Babelfish ticket fixes: - Ticket #27 – Fix compiler errors/warnings with strict build settings - Ticket #39 – Allow C99 code in apps	Demonstration	Make process did not generate any compiler warnings

5.4.2 Outstanding DCRs

DCR	Description	State
4115	HK - Add Trick Simulation Support (JSC Request)	Submitted

5.5 NOTES

The HK application was tested in two configurations for 2.4.1.0. The normal configuration had the Discard Combo Packets configuration parameter set to NO. The other configuration had this parameter set to YES in order to test the new requirements added to HK.

There were no significant findings and/or anomalies reported during testing but due to the serial nature of the build testing integration testing is the ultimate verification of the HK applications performance in a system-like scenario.

APPENDIX A - RTTM

The HK Build 2.4.1.0 RTTM can be found on the MKS server, in cFS-Repository HK test-and-ground/results folder.

APPENDIX B - COMMAND, TELEMETRY, AND EVENTS VERIFICATION MATRIX

Command	Test Procedure(s)	Notes/Comments
HK_NOOP	HK_GenCmds	
HK_RESETCTRS	HK_GenCmds	

Telemetry	Test Procedure(s)	Notes/Comments
HK_CMDPC	HK_GenCmds	
HK_CMDEC	HK_GenCmds	
HK_CMBPKTSENT	HK_GenCmds HK_BasicHousekeeping HK_MissingData HK_StressHousekeeping HK_StressMissingData HK_StressTableLoad	Initialization/ Reset only Processing data Processing data Processing data Processing data Processing data
HK_MISSDATACTR	HK_GenCmds HK_MissingData HK_StressHousekeeping HK_StressMissingData	Initialization/ Reset only Processing data Processing data Processing data
HK_MEMPOOLHNDL		Tested manually
HK_COMBINED_PKT1	HK_BasicHousekeeping HK_MissingData HK_StressHousekeeping HK_StressMissingData HK_StressTableLoad	
HK_COMBINED_PKT2	HK_BasicHousekeeping HK_MissingData HK_StressHousekeeping HK_StressMissingData HK_StressTableLoad	
HK_COMBINED_PKT3	HK_BasicHousekeeping HK_MissingData HK_StressHousekeeping HK_StressMissingData HK_StressTableLoad	
HK_COMBINED_PKT4	HK_BasicHousekeeping HK_MissingData HK_StressHousekeeping HK_StressMissingData HK_StressTableLoad	
HK_COMBINED_PKT5	HK_StressHousekeeping HK_StressMissingData	
HK_COMBINED_PKT6	HK_StressHousekeeping HK_StressMissingData	

File and Table Telemetry	Test Procedure(s)	Notes/Comments
HK_COPY_TBL	HK_GenCmds HK_BasicHousekeeping HK_MissingData HK_StressHousekeeping HK_StressMissingData	

	HK_StressTableLoad	

Event Message Ids	**Test Procedure(s)**	**Notes/Comments**
HK_INIT_EID 1	HK_GenCmds HK_BasicHousekeeping HK_MissingData HK_StressHousekeeping HK_StressMissingData HK_StressTableLoad	
HK_CC_ERR_EID 2	HK_GenCmds	
HK_CMD_LEN_ERR_EID 3	HK_GenCmds	
HK_NOOP_CMD_EID 4	HK_GenCmds	
HK_RESET_CNTRS_CMD_EID 5	HK_GenCmds	
HK_ACCESSING_PAST_PACKET_END_EID 6	HK_MissingData; HK_StressHousekeeping	
HK_MEM_POOL_MALLOC_FAILED_EID 7		Tested during unit testing
HK_CANT_SUBSCRIBE_TO_SB_PKT_EID 8		Tested during unit testing
HK_MEM_POOL_FREE_FAILED_EID 9		Tested during unit testing
HK_UNEXPECTED_GETSTAT_RET_EID 10		Tested during unit testing
HK_UNKNOWN_COMBINED_PACKET_EID 11	HK_StressHousekeeping	
HK_OUTPKT_MISSING_DATA_EID 12	HK_GenCmds HK_MissingData HK_StressHousekeeping HK_StressMissingData	
HK_EVS_REG_ERR_EID 13		Tested during unit testing
HK_CR_PIPE_ERR_EID 14		Tested during unit testing
HK_SUB_CMB_ERR_EID 15		Tested during unit testing
HK_SUB_REQ_ERR_EID 16		Tested during unit testing
HK_SUB_CMD_ERR_EID 17		Tested during unit testing
HK_CR_POOL_ERR_EID 18		Tested during unit testing
HK_CPTBL_REG_ERR_EID 19		Tested during unit testing
HK_RTTBL_REG_ERR_EID 20		Tested during unit testing
HK_CPTBL_LD_ERR_EID 21		Tested during unit testing
HK_CPTBL_MNG_ERR_EID 22		Tested during unit testing
HK_RTTBL_MNG_ERR_EID 23		Tested during unit testing
HK_CPTBL_GADR_ERR_EID 24		Tested during unit testing
HK_RTTBL_GADR_ERR_EID 25		Tested during unit testing
HK_RCV_MSG_ERR_EID 26		Tested during unit testing
HK_UNEXPECTED_GETSTAT2_RET_EID 27		Tested during unit testing
HK_MSG_LEN_ERR_EID 28	HK_GenCmds	

APPENDIX C - COPYTABLE DEFINITIONS

Table 1:
20 input messages, 3 output messages, pieces of each input packet go to each output packet. Table will have 60 entries.
Used for: GenCmds, BasicHousekeeping, MissingData, Stress Table Load

Table 2:
20 input messages, 3 output messages, pieces of each input packet go to each output packet. Similar to Table 1, but change 1 input packet id, 1 output packet of 1 output packet, data from 1 input packet should be output in only output packet 2
Used for: BasicHousekeeping, MissingData, StressTableLoad

Table 3:
2 input messages (128 copy table entries), 6 output messages
Used for StressHousekeeping, StressMissingData, StressTableLoad

Table 4:
11 input messages, 2 output messages, odd sized input and output packets
Used for StressHousekeeping, StressTableLoad

Table 5:
19 input messages, 4 output messages, data that has gaps (i.e. – copying 1..16 from input packet to 1….16 in output packet and then put 17….32 from a differ packet into 17…..32 in output packet)
Used for StressHousekeeping, StressTableLoad

Table 1

20 input messages, 3 output messages, pieces of each input packet go to each output packet. Table will have 60 entries.

				data to output 1 CPU Msgids (0x) 89c, 99c, a9c			data to output 2 CPU Msgids (0x) 89d, 99d, a9d			data to output 3 CPU Msgids (0x) 89e, 99e, a9e			
Pkt	CPU Msgids (0x) cpu1,cpu2,cpu3	Data length (bytes)	Data Pattern (lw)	InOffset (bytes)	# bytes	OutOffset (bytes)	InOffset (bytes)	# bytes	OutOffset (bytes)	InOffset (bytes)	# bytes	OutOffset (bytes)	Data Pattern (2nd run)
1	887, 987, a87	4	x0123 4567	12	4	12	13	2	50	15	1	30	xa987 6543
2	888, 988, a88	8	x1234 5678	12	4	16	17	2	48	19	1	28	x9876 5432
3	889, 989, a89	16	x2345 6789	12	4	20	21	2	46	27	1	26	x8765 4321
4	88a, 98a, a8a	32	x3456 789a	12	4	24	37	2	44	43	1	24	x7654 3210
5	88b 98b, a8b	32	x4567 89ab	12	4	28	38	2	42	43	1	22	xf012 3456
6	88c, 98c, a8c	16	x5678 9abc	12	4	32	22	2	40	27	1	20	xef01 2345
7	88d, 98d, a8d	8	x6789 abcd	12	4	36	18	2	38	19	1	18	xdef0 1234
8	88e, 98e, a8e	4	x789a bcde	12	4	40	14	2	36	15	1	16	xcdef 0123
9	88f, 98f, a8f	4	x89ab cdef	12	4	44	13	2	34	15	1	14	xbcde f012
10	890, 990, a90	8	x9abc def0	12	4	48	17	2	32	19	1	12	xabcd ef01
11	891, 991, a91	8	xabcd ef01	12	4	52	18	2	30	19	1	13	missing
12	892, 992, a92	4	xbcde f012	12	4	56	14	2	28	15	1	15	x89ab cdef
13	893, 993, a93	4	xcdef 0123	12	4	60	13	2	26	15	1	17	x789a bcde
14	894, 994, a94	8	xdef0 1234	12	4	64	17	2	24	19	1	19	x6789 abcd
15	895, 995, a95	16	xef01 2345	12	4	68	25	2	22	27	1	21	x5678 9abc
16	896, 996, a96	32	xf012 3456	12	4	72	41	2	20	43	1	23	x4567 89ab
17	897, 997, a97	32	x7654 3210	12	4	76	42	2	18	43	1	25	x3456 789a
18	898, 998, a98	16	x8765 4321	12	4	80	26	2	16	27	1	27	x2345 6789
19	899, 999, a99	8	x9876 5432	12	4	84	18	2	14	19	1	29	x1234 5678

20	89a, 99a, a9a	4	xa987 6543	12	4	88	14	2	12	15	1	31	x0123 4567
				80			40			20			

InHdr 12
OutHdr 12

Input Id	Data Packet 1
1	0123 4567
2	1234 5678 1234 5678
3	2345 6789 2345 6789 2345 6789 2345 6789
4	3456 789a 3456 789a 3456 789a 3456 789a 3456 789a 3456 789a 3456 789a 3456 789a
5	4567 89ab 4567 89ab 4567 89ab 4567 89ab 4567 89ab 4567 89ab 4567 89ab 4567 89ab
6	5678 9abc 5678 9abc 5678 9abc 5678 9abc
7	6789 abcd 6789 abcd
8	789a bcde
9	89ab cdef
10	9abc def0 9abc def0
11	abcd ef01 abcd ef01
12	bcde f012
13	cdef 0123
14	def0 1234 def0 1234
15	ef01 2345 ef01 2345 ef01 2345 ef01 2345
16	f012 3456 f012 3456 f012 3456 f012 3456 f012 3456 f012 3456 f012 3456 f012 3456
17	7654 3210 7654 3210 7654 3210 7654 3210 7654 3210 7654 3210 7654 3210 7654 3210
18	8765 4321 8765 4321 8765 4321 8765 4321
19	9876 5432 9876 5432
20	a987 6543

Output id	
	0123 4567 1234 5678 2345 6789 3456 789a 4567 89ab 5678 9abc 6789 abcd 789a bcde 89ab cdef 9abc def0 abcd ef01 bcde fo12 cdef 0123 def0 1234 ef01 2345 f012 3456 7654 3210 8765 4321
1	9876 5432 a987 6543
	6543 5432 4321 3210 1234 0123 f012 ef01 f012 ef01 bcde abcd bcde abcd 9abc 89ab 5678 4567
2	3456 2345
3	f001 ef12 de23 cd34 bc45 ab56 9a10 8921 7832 6743

Run 2 Input Packets	Data Packet 2
1	a987 6543
2	9876 5432 9876 5432

3	8765 4321 8765 4321 8765 4321 8765 4321
4	7654 3210 7654 3210 7654 3210 7654 3210 7654 3210 7654 3210 7654 3210 7654 3210
5	f012 3456 f012 3456 f012 3456 f012 3456 f012 3456 f012 3456 f012 3456 f012 3456
6	ef01 2345 ef01 2345 ef01 2345 ef01 2345
7	def0 1234 def0 1234
8	cdef 0123
9	bcde f012
10	abcd ef01 abcd ef01
11	
12	89ab cdef
13	789a bcde
14	6789 abcd 6789 abcd
15	5678 9abc 5678 9abc 5678 9abc 5678 9abc
16	4567 89ab 4567 89ab 4567 89ab 4567 89ab 4567 89ab 4567 89ab 4567 89ab 4567 89ab
17	3456 789a 3456 789a 3456 789a 3456 789a 3456 789a 3456 789a 3456 789a 3456 789a
18	2345 6789 2345 6789 2345 6789 2345 6789 2345 6789
19	1234 5678 1234 5678
20	0123 4567

Output id

	a987 6543 9876 5432 8765 4321 7654 3210 f012 3456 ef01 2345 def0 1234 cdef 0123 bcde f012 abcd ef01 **abcd ef01** 89ab cdef 789a bcde 6789 abcd 5678 9abc 4567 89ab 3456 789a 2345 6789
1	1234 5678 0123 4567
	4567 5678 6789 789a 6789 789a 89ab 9abc cdef **ef01** cdef def0 0123 1234 2345 3456 5432 6543
2	7654 8765
3	01**01** 12ef 23de 34cd 45bc 56ab 109a 2189 3278 4567

Run 3

1	1111 1111
2	1212 1212 1212 1212
3	1313 1313 1313 1313 1313 1313 1313 1313
4	1414 1414 1414 1414 1414 1414 1414 1414 1414 1414 1414 1414 1414 1414 1414 1414
5	1515 1515 1515 1515 1515 1515 1515 1515 1515 1515 1515 1515 1515 1515 1515 1515
6	1616 1616 1616 1616 1616 1616 1616 1616
7	1717 1717 1717 1717
8	1818 1818
9	1919 1919
10	1a1a 1a1a 1a1a 1a1a
11	1b1b 1b1b 1b1b 1b1b
12	1c1c 1c1c

13	1d1d 1d1d
14	1e1e 1e1e 1e1e 1e1e
15	1f1f 1f1f 1f1f 1f1f 1f1f 1f1f 1f1f 1f1f
16	2020 2020 2020 2020 2020 2020 2020 2020 2020 2020 2020 2020 2020 2020 2020 2020
17	2121 2121 2121 2121 2121 2121 2121 2121 2121 2121 2121 2121 2121 2121 2121 2121
18	2222 2222 2222 2222 2222 2222 2222 2222
19	2323 2323 2323 2323
20	2424 2424

Output id

	1111 1111 1212 1212 1313 1313 1414 1414 1515 1515 1616 1616 1717 1717 1818 1818 1919 1919
	1a1a 1a1a 1b1b 1b2b 1c1c 1c1c 1d1d 1d1d 1e1e 1e1e 1f1f 1f1f 2020 2020 2121 2121 2222 2222
1	2323 2323 2424 2424
	2424 2323 2222 2121 2020 1f1f 1e1e 1d1d 1c1c 1b1b 1a1a 1919 1818 1717 1616 1515 1414 1313
2	1212 1111
3	1a1b 191c 181d 171e 161f 1520 1421 1322 1223 1124

Table 2

20 input messages, 3 output messages, pieces of each input packet go to each output packet. Similar to Table 1, but change 1 input packet id, 1 output packet id, and size of 1 output packet, data from 1 input packet should be output in only output packet 2

				data to output 1 CPU Msgids (0x) 89c, 99c, a9c			data to output 2 CPU Msgids (0x) 89d, 99d, a9d			data to output 4 CPU Msgids (0x) 89f, 99f, a9f		
Pkt	CPU MsgIds (0x) cpu1,cpu2,cpu3	Data length (bytes)	Data Pattern (lw)	InOffset (bytes)	# bytes	OutOffset (bytes)	InOffset (bytes)	# bytes	OutOffset (bytes)	InOffset (bytes)	# bytes	OutOffset (bytes)
1	887, 987, a87	4	x1111 1111	12	4	12	13	2	50	15	1	35
2	888, 988, a88	8	x2222 2222	12	4	16	17	2	48	19	1	28
3	889, 989, a89	16	x3333 3333	12	4	20	21	2	46	27	1	26
4	88a, 98a, a8a	32	x4444 4444	12	4	24	37	2	44	43	1	24
5	88b 98b, a8b	32	x5555 5555	12	4	28	38	2	42	43	1	22
6	88c, 98c, a8c	16	x6666 6666	12	4	32	22	2	40	27	1	20
7	88d, 98d, a8d	8	x7777 7777	12	4	36	18	2	38	19	1	18
8	88e, 98e, a8e	4	x8888 8888	12	4	40	14	2	36	15	1	16
9	88f, 98f, a8f	4	x9999 9999	12	4	44	13	2	34	15	1	14
10	890, 990, a90	8	xAAAA AAAA	12	4	48	17	2	32	19	1	12
11	891, 991, a91	8	xBBBB BBBB	12	4	52	18	2	30	19	1	13
12	892, 992, a92	4	xCCCC CCCC	12	4	56	14	2	28	15	1	15
13	893, 993, a93	4	xDDDD DDDD	12	4	60	13	2	26	15	1	17
14	894, 994, a94	8	xEEEE EEEE	12	4	64	17	2	24	19	1	19
15	8a2, 9a2, aa2	16	XFFFF FFFF	12	4	68	25	2	22	27	1	21
16	896, 996, a96	32	x1616 1616	12	4	72	41	2	20	43	1	23
17	897, 997, a97	32	x1717 1717	12	4	76	42	2	18	43	1	25
18	898, 998, a98	16	x1818 1818	12	4	80	26	2	16	27	1	27
19	899, 999, a99	8	x1919 1919	12	4	84	18	2	14	19	2	29

20	89a, 99a, a9a	4	x2020 2020				14	2	12			
				76			40			20		

InHdr 12
OutHdr 12
changes

1 input packet id:	InPkt 15 changed to id
1 output packet id:	OutPkt 3 changed to OutPkt 4
Size of 1 output packet:	OutPkt 1 changed from 80 bytes to 76 bytes
1 input packet should put in OutPkt 2 only:	Packet id 20 no longer output on OutPkt 1 or OutPkt 4. In OutPkt 4 InPkt 19 has 2 bytes output to keep size same

Input Id	Data Packet (1st time)
1	1111 1111
2	2222 2222 2222 2222
3	3333 3333 3333 3333 3333 3333 3333 3333
4	4444 4444 4444 4444 4444 4444 4444 4444 4444 4444 4444 4444 4444 4444 4444 4444
5	5555 5555 5555 5555 5555 5555 5555 5555 5555 5555 5555 5555 5555 5555 5555 5555
6	6666 6666 6666 6666 6666 6666 6666 6666
7	7777 7777 7777 7777
8	8888 8888
9	9999 9999
10	AAAA AAAA AAAA AAAA
11	BBBB BBBB BBBB BBBB
12	CCCC CCCC
13	DDDD DDDD
14	EEEE EEEE EEEE EEEE
30	FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF
16	1616 1616 1616 1616 1616 1616 1616 1616 1616 1616 1616 1616 1616 1616 1616 1616
17	1717 1717 1717 1717 1717 1717 1717 1717 1717 1717 1717 1717 1717 1717 1717 1717
18	1818 1818 1818 1818 1818 1818 1818 1818
19	1919 1919 1919 1919
20	2020 2020

Output id

```
                1111 1111  2222 2222 3333 3333  4444 4444 5555 5555  6666 6666 7777 7777  8888 8888 9999
                9999   aaaa aaaa bbbb bbbb  cccc cccc dddd dddd eeee eeee ffff  ffff  1616 1616 1717 1717  1818
1               1818 1919 1919

                2020 1919 1818 1717 1616 ffff eeee dddd cccc bbbb aaaa 9999 8888 7777 6666 5555 4444 3333
2               2222 1111
4               aabb 99cc 88dd 77ee 66ff 5516 4417 3318 2219 1911
```

Table 3
2 input messages (128 entries in table), 6 output messages

Entries			Pkt 1	Pkt 2	
		CPU MsgIds (0x) cpu1,cpu2,cpu3 data length (bytes) Data Pattern (lw)	887, 987, a87 32 x0123 4567	89a, 99a, a9a 32 x89ab cdef	
1	data to output 1 CPU Msgids (0x) 89c, 99c, a9c	InOffset (bytes) # bytes OutOffset(bytes	12 1 12	43 1 13	65
2	data to output 2 CPU Msgids (0x) 89d, 99d, a9d	InOffset (bytes) # bytes OutOffset(bytes	13 1 12	42 1 13	66
3	data to output 3 CPU Msgids (0x) 89e, 99e, a9e	InOffset (bytes) # bytes OutOffset(bytes	14 1 12	41 1 13	67
4	data to output 4 CPU Msgids (0x) 89f, 99f, a9f	InOffset (bytes) # bytes OutOffset(bytes	15 1 12	40 1 13	68
5	data to output 5 CPU Msgids (0x) 8a0, 9a0, aa0	InOffset (bytes) # bytes OutOffset(bytes	16 1 12	39 1 13	69
6	data to output 6 CPU Msgids (0x) 8a1, 9a1, aa1	InOffset (bytes) # bytes OutOffset(bytes	17 1 12	38 1 13	70
7	data to output 1 CPU Msgids (0x) 89c, 99c, a9c	InOffset (bytes) # bytes OutOffset(bytes	18 1 14	37 1 15	71
8	data to output 2 CPU Msgids (0x) 89d, 99d, a9d	InOffset (bytes) # bytes OutOffset(bytes	19 1 14	36 1 15	72
9	data to output 3 CPU Msgids (0x) 89e, 99e, a9e	InOffset (bytes) # bytes OutOffset(bytes	20 1 14	35 1 15	73
10	data to output 4 CPU Msgids (0x) 89f, 99f, a9f	InOffset (bytes) # bytes OutOffset(bytes	21 1 14	34 1 15	74
11	data to output 5 CPU Msgids (0x) 8a0, 9a0, aa0	InOffset (bytes) # bytes OutOffset(bytes	22 1 14	33 1 15	75
12	data to output 6 CPU Msgids (0x) 8a1, 9a1, aa1	InOffset (bytes) # bytes OutOffset(bytes	23 1 14	32 1 15	76
13	data to output 1 CPU Msgids (0x) 89c, 99c, a9c	InOffset (bytes) # bytes OutOffset(bytes	24 1 16	31 1 17	77
14	data to output 2 CPU Msgids (0x)	InOffset (bytes) # bytes	25 1	30 1	78

	89d, 99d, a9d	OutOffset(bytes	16	17	
15	data to output 3 CPU Msgids (0x) 89e, 99e, a9e	InOffset (bytes) # bytes OutOffset(bytes	26 1 16	29 1 17	79
16	data to output 4 CPU Msgids (0x) 89f, 99f, a9f	InOffset (bytes) # bytes OutOffset(bytes	27 1 16	28 1 17	80
17	data to output 5 CPU Msgids (0x) 8a0, 9a0, aa0	InOffset (bytes) # bytes OutOffset(bytes	28 1 16	27 1 17	81
18	data to output 6 CPU Msgids (0x) 8a1, 9a1, aa1	InOffset (bytes) # bytes OutOffset(bytes	29 1 16	26 1 17	82
19	data to output 1 CPU Msgids (0x) 89c, 99c, a9c	InOffset (bytes) # bytes OutOffset(bytes	30 1 18	25 1 19	83
20	data to output 2 CPU Msgids (0x) 89d, 99d, a9d	InOffset (bytes) # bytes OutOffset(bytes	31 1 18	24 1 19	84
21	data to output 3 CPU Msgids (0x) 89e, 99e, a9e	InOffset (bytes) # bytes OutOffset(bytes	32 1 18	23 1 19	85
22	data to output 4 CPU Msgids (0x) 89f, 99f, a9f	InOffset (bytes) # bytes OutOffset(bytes	33 1 18	22 1 19	86
23	data to output 5 CPU Msgids (0x) 8a0, 9a0, aa0	InOffset (bytes) # bytes OutOffset(bytes	34 1 18	21 1 19	87
24	data to output 6 CPU Msgids (0x) 8a1, 9a1, aa1	InOffset (bytes) # bytes OutOffset(bytes	35 1 18	20 1 19	88
25	data to output 1 CPU Msgids (0x) 89c, 99c, a9c	InOffset (bytes) # bytes OutOffset(bytes	36 1 20	19 1 21	89
26	data to output 2 CPU Msgids (0x) 89d, 99d, a9d	InOffset (bytes) # bytes OutOffset(bytes	37 1 20	18 1 21	90
27	data to output 3 CPU Msgids (0x) 89e, 99e, a9e	InOffset (bytes) # bytes OutOffset(bytes	38 1 20	17 1 21	91
28	data to output 4 CPU Msgids (0x) 89f, 99f, a9f	InOffset (bytes) # bytes OutOffset(bytes	39 1 20	16 1 21	92
29	data to output 5 CPU Msgids (0x) 8a0, 9a0, aa0	InOffset (bytes) # bytes OutOffset(bytes	40 1 20	15 1 21	93
30	data to output 6 CPU Msgids (0x) 8a1, 9a1, aa1	InOffset (bytes) # bytes OutOffset(bytes	41 1 20	14 1 21	94
31	data to output 1 CPU Msgids (0x)	InOffset (bytes) # bytes	42 1	13 1	95

	89c, 99c, a9c	OutOffset(bytes	22	23	
32	data to output 2 CPU Msgids (0x) 89d, 99d, a9d	InOffset (bytes) # bytes OutOffset(bytes	43 1 22	12 1 23	96
33	data to output 3 CPU Msgids (0x) 89e, 99e, a9e	InOffset (bytes) # bytes OutOffset(bytes	12 1 22	43 1 23	97
34	data to output 4 CPU Msgids (0x) 89f, 99f, a9f	InOffset (bytes) # bytes OutOffset(bytes	13 1 22	42 1 23	98
35	data to output 5 CPU Msgids (0x) 8a0, 9a0, aa0	InOffset (bytes) # bytes OutOffset(bytes	14 1 22	41 1 23	99
36	data to output 6 CPU Msgids (0x) 8a1, 9a1, aa1	InOffset (bytes) # bytes OutOffset(bytes	15 1 22	40 1 23	100
37	data to output 1 CPU Msgids (0x) 89c, 99c, a9c	InOffset (bytes) # bytes OutOffset(bytes	16 1 24	39 1 25	101
38	data to output 2 CPU Msgids (0x) 89d, 99d, a9d	InOffset (bytes) # bytes OutOffset(bytes	17 1 24	38 1 25	102
39	data to output 3 CPU Msgids (0x) 89e, 99e, a9e	InOffset (bytes) # bytes OutOffset(bytes	18 1 24	37 1 25	103
40	data to output 4 CPU Msgids (0x) 89f, 99f, a9f	InOffset (bytes) # bytes OutOffset(bytes	19 1 24	36 1 25	104
41	data to output 5 CPU Msgids (0x) 8a0, 9a0, aa0	InOffset (bytes) # bytes OutOffset(bytes	20 1 24	35 1 25	105
42	data to output 6 CPU Msgids (0x) 8a1, 9a1, aa1	InOffset (bytes) # bytes OutOffset(bytes	21 1 24	34 1 25	106
43	data to output 1 CPU Msgids (0x) 89c, 99c, a9c	InOffset (bytes) # bytes OutOffset(bytes	22 1 26	33 1 27	107
44	data to output 2 CPU Msgids (0x) 89d, 99d, a9d	InOffset (bytes) # bytes OutOffset(bytes	23 1 26	32 1 27	108
45	data to output 3 CPU Msgids (0x) 89e, 99e, a9e	InOffset (bytes) # bytes OutOffset(bytes	24 1 26	31 1 27	109
46	data to output 4 CPU Msgids (0x) 89f, 99f, a9f	InOffset (bytes) # bytes OutOffset(bytes	25 1 26	30 1 27	110
47	data to output 5 CPU Msgids (0x) 8a0, 9a0, aa0	InOffset (bytes) # bytes OutOffset(bytes	26 1 26	29 1 27	111
48	data to output 6 CPU Msgids (0x)	InOffset (bytes) # bytes	27 1	28 1	112

	8a1, 9a1, aa1	OutOffset(bytes	26	27	
49	data to output 1 CPU Msgids (0x) 89c, 99c, a9c	InOffset (bytes) # bytes OutOffset(bytes	28 1 28	27 1 29	113
50	data to output 2 CPU Msgids (0x) 89d, 99d, a9d	InOffset (bytes) # bytes OutOffset(bytes	29 1 28	26 1 29	114
51	data to output 3 CPU Msgids (0x) 89e, 99e, a9e	InOffset (bytes) # bytes OutOffset(bytes	30 1 28	25 1 29	115
52	data to output 4 CPU Msgids (0x) 89f, 99f, a9f	InOffset (bytes) # bytes OutOffset(bytes	31 1 28	24 1 29	116
53	data to output 5 CPU Msgids (0x) 8a0, 9a0, aa0	InOffset (bytes) # bytes OutOffset(bytes	32 1 28	23 1 29	117
54	data to output 6 CPU Msgids (0x) 8a1, 9a1, aa1	InOffset (bytes) # bytes OutOffset(bytes	33 1 28	22 1 29	118
55	data to output 1 CPU Msgids (0x) 89c, 99c, a9c	InOffset (bytes) # bytes OutOffset(bytes	34 1 30	21 1 31	119
56	data to output 2 CPU Msgids (0x) 89d, 99d, a9d	InOffset (bytes) # bytes OutOffset(bytes	35 1 30	20 1 31	120
57	data to output 3 CPU Msgids (0x) 89e, 99e, a9e	InOffset (bytes) # bytes OutOffset(bytes	36 1 30	19 1 31	121
58	data to output 4 CPU Msgids (0x) 89f, 99f, a9f	InOffset (bytes) # bytes OutOffset(bytes	37 1 30	18 1 31	122
59	data to output 5 CPU Msgids (0x) 8a0, 9a0, aa0	InOffset (bytes) # bytes OutOffset(bytes	38 1 30	17 1 31	123
60	data to output 6 CPU Msgids (0x) 8a1, 9a1, aa1	InOffset (bytes) # bytes OutOffset(bytes	39 1 30	16 1 31	124
61	data to output 1 CPU Msgids (0x) 89c, 99c, a9c	InOffset (bytes) # bytes OutOffset(bytes	40 1 32	15 1 33	125
62	data to output 2 CPU Msgids (0x) 89d, 99d, a9d	InOffset (bytes) # bytes OutOffset(bytes	41 1 32	14 1 33	126
63	data to output 3 CPU Msgids (0x) 89e, 99e, a9e	InOffset (bytes) # bytes OutOffset(bytes	42 1 32	13 1 33	127
64	data to output 4 CPU Msgids (0x) 89f, 99f, a9f	InOffset (bytes) # bytes OutOffset(bytes	43 1 32	12 1 33	128

Run 1 Input Pkt

	0123 4567 0123 4567 0123 4567 0123 4567 0123 4567 0123 4567 0123 4567
1	0123 4567
	89ab cdef 89ab cdef 89ab cdef 89ab cdef 89ab cdef 89ab cdef 89ab cdef 89ab
2	cdef

Output Pkt

1	01ef 45ab 01ef 45ab 01ef 45ab 01ef 45ab 01ef 45ab 01ef
2	23cd 6789 23cd 6789 23cd 6789 23cd 6789 23cd 6789 23cd
3	45ab 01ef 45ab 01ef 45ab 01ef 45ab 01ef 45ab 01ef 45ab
4	6789 23cd 6789 23cd 6789 23cd 6789 23cd 6789 23cd 6789
5	01ef 45ab 01ef 45ab 01ef 45ab 01ef 45ab 01ef 45ab
6	23cd 6789 23cd 6789 23cd 6789 23cd 6789 23cd 6789

Run 2 Input Pkt

	1234 5678 1234 5678 1234 5678 1234 5678 1234 5678 1234 5678 1234 5678
1	1234 5678
2	Don't send

Output

1	12ef 56ab 12ef 56ab 12ef 56ab 12ef 56ab 12ef 56ab 12ef
2	34cd 7889 34cd 7889 34cd 7889 34cd 7889 34cd 7889 34cd
3	56ab 12ef 56ab 12ef 56ab 12ef 56ab 12ef 56ab 12ef 56ab
4	7889 34cd 7889 34cd 7889 34cd 7889 34cd 7889 34cd 7889
5	12ef 56ab 12ef 56ab 12ef 56ab 12ef 56ab 12ef 56ab
6	34cd 7889 34cd 7889 34cd 7889 34cd 7889 34cd 7889

Run 3 Input Pkt

	1111 2222 1111 2222 1111 2222 1111 2222 1111 2222 1111 2222 1111 2222
1	1111 2222
	aaaa bbbb aaaa bbbb aaaa bbbb aaaa bbbb aaaa bbbb aaaa bbbb aaaa bbbb
2	aaaa bbbb

Output

1	11bb 22aa 11bb 22aa 11bb 22aa 11bb 22aa 11bb 22aa 11bb
2	11bb 22aa 11bb 22aa 11bb 22aa 11bb 22aa 11bb 22aa 11bb
3	22aa 11bb 22aa 11bb 22aa 11bb 22aa 11bb 22aa 11bb 22aa
4	22aa 11bb 22aa 11bb 22aa 11bb 22aa 11bb 22aa 11bb 22aa
5	11bb 22aa 11bb 22aa 11bb 22aa 11bb 22aa 11bb 22aa
6	11bb 22aa 11bb 22aa 11bb 22aa 11bb 22aa 11bb 22aa

Table 4

11 input messages, 2 output messages, odd sized input and output packets

Pkt	CPU MsgIds (0x) cpu1,cpu2,cpu3	Data length (bytes)	Data Pattern (lw)	data to output 1 CPU Msgids (0x) 89c, 99c, a9c			data to output 2 CPU Msgids (0x) 89d, 99d, a9d			
				InOffset (bytes)	# bytes	OutOffset (bytes)	InOffset (bytes)	# bytes	OutOffset (bytes)	
1	887, 987, a87	4	x0123 4567	12	2	12	12	1	12	
2	888, 988, a88	8	x1234 5678	13	2	14	12	1	13	
3	889, 989, a89	4	x2345 6789	14	2	16	12	1	14	
4	88a, 98a, a8a	8	x3456 789a	15	2	18	12	1	15	
5	88b 98b, a8b	4	x4567 89ab	12	2	20	12	1	16	
6	88c, 98c, a8c	3	x5678 9a	13	2	22	12	1	17	odd sized input packet
7	88d, 98d, a8d	8	x6789 abcd	14	2	24	12	1	18	
8	88e, 98e, a8e	8	x789a bcde	15	2	26	12	1	19	
9	88f, 98f, a8f	4	x89ab cdef	12	2	28	12	1	20	
10	890, 990, a90	8	x9abc def0	13	2	30	12	1	21	
11	891, 991, a91	4	xabcd ef01	14	2	32	12	1	22	
hdt		12			22			11		

pkt 2 is odd sized output packet

Input	Data
1	0123 4567
2	1234 5678 1234 5678
3	2345 5678
4	3456 789a 3456 789a
5	4567 89ab
6	5678 9a
7	6789 abcd 6789 abcd
8	789a bcde 789a bcde
9	89ab cdef
10	9abc def0
11	abcd ef01

Output

1 0123 3456 6789 9a34 4567 789a abcd de78 89ab bcde ef01
2 0112 2334 4556 6778 899a ab

Table 5

19 input messages, 4 output messages, data that has gaps (i.e. – copying 1..4 from input packet to 1….4 in output packet and then put 5….8 from a different input packet into 5…..8 in output packet) , odd sized output packets, odd byte copies.

				data to output 1 CPU Msgids (0x) 89c, 99c, a9c			data to output 2 CPU Msgids (0x) 89d, 99d, a9d			data to output 3 CPU Msgids (0x) 89e, 99e, a9e			data to outpu CPU Msgids (0x) 89	
Pkt	CPU MsgIds (0x) cpu1,cpu2,cpu3	Data length (bytes)	Data Pattern (lw)	InOffset (bytes)	# bytes	OutOffset (bytes)	InOffset (bytes)	# bytes	OutOffset (bytes)	InOffset (bytes)	# bytes	OutOffset (bytes)	InOffset (bytes)	# bytes
1	887, 987, a87	4	x0123 4567	12	1	12	12	2	48	13	1	48	15	1
2	888, 988, a88	8	x1234 5678	12	2	13	14	2	46	13	3	45	19	1
3	889, 989, a89	16	x2345 6789	12	3	15	15	2	44	13	1	44	27	1
4	88a, 98a, a8a	32	x3456 789a	12	4	18	16	2	42	13	3	41	43	1
5	88b 98b, a8b	32	x4567 89ab	12	4	22	17	2	40	13	1	40	43	1
6	88c, 98c, a8c	16	x5678 9abc	12	3	26	18	2	38	13	3	37	27	1
7	88d, 98d, a8d	8	x6789 abcd	12	2	29	17	2	36	13	1	36	19	1
8	88e, 98e, a8e	4	x789a bcde	12	1	31	13	2	34	13	3	33	15	1
9	88f, 98f, a8f	4	x89ab cdef	12	1	32	14	2	32	13	1	32	15	1
10	890, 990, a90	8	x9abc def0	12	2	33	15	2	30	13	3	29	19	1
11	891, 991, a91	8	xabcd ef01	12	3	35	16	2	28	13	1	28	19	1
12	892, 992, a92	4	xbcde f012	12	4	38	14	2	26	13	3	25	15	1
13	893, 993, a93	4	xcdef 0123	12	4	42	13	2	24	13	1	24	15	1
14	894, 994, a94	8	xdef0 1234	12	3	46	16	2	22	13	3	21	19	1
15	8a2, 9a2, aa2	16	xef01 2345	12	2	49	17	2	20	13	1	20	27	1
16	896, 996, a96	32	xf012 3456	12	1	51	18	2	18	13	3	17	43	1
17	897, 997, a97	32	x7654 3210	12	1	52	19	2	16	13	1	16	43	1
18	898, 998, a98	16	x8765 4321	12	2	53	18	2	14	13	3	13	19	1
19	899, 999, a99	8	x9876 5432	12	3	55	16	2	12	13	1	12	19	1
hdr	12				46			38			37			1

Input Id	Data Packet
1	0123 4567
2	1234 5678 1234 5678
3	2345 6789 2345 6789 2345 6789 2345 6789
4	3456 789a 3456 789a 3456 789a 3456 789a 3456 789a 3456 789a 3456 789a 3456 789a
5	4567 89ab 4567 89ab 4567 89ab 4567 89ab 4567 89ab 4567 89ab 4567 89ab 4567 89ab
6	5678 9abc 5678 9abc 5678 9abc 5678 9abc
7	6789 abcd 6789 abcd
8	789a bcde
9	89ab cdef
10	9abc def0 9abc def0
11	abcd ef01 abcd ef01
12	bcde f012
13	cdef 0123
14	def0 1234 def0 1234
15	ef01 2345 ef01 2345 ef01 2345 ef01 2345
16	f012 3456 f012 3456 f012 3456 f012 3456 f012 3456 f012 3456 f012 3456 f012 3456
17	7654 3210 7654 3210 7654 3210 7654 3210 7654 3210 7654 3210 7654 3210 7654 3210
18	8765 4321 8765 4321 8765 4321 8765 4321
18	9876 5432 9876 5432

Output	
	0112 3423 4567 3456 789a 4567 89ab 5678 9a67 8978 899a bcab cdef bcde f012 cdef 0123 def0 12ef 01f0 7687 6598
1	7654
2	9876 4321 1076 3456 0123 def0 ef01 f012 abcd f09a cdef 9abc 89ab 9abc 6789 3456 8923 5678 0123
3	7665 4321 5412 3456 01f0 1234 efde f012 cdbc def0 ab9a bcde 8978 9abc 6756 789a 4534 5678 23
4	6778 899a abbc cdde eff0 0112 2334 4556 1021 32