**CDH Software**

**FDIR Test Document**

|  |  |
| --- | --- |
| **Prepared by** | Keenan Burnett |
| **Revision** | 1.0 |
| **Date** | 2016.01.11 |
| **Reference** | FDIR Test Document |
| **Signed By** | Keenan Burnett |

# Document Change Log

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Description of Changes** | | | **Modified Section** | **Revision** | **Date** | |
| Initial Issue | | |  | 1.0 | 2016.01.11 | |
|  | | |  |  |  | |
|  | | |  |  |  | |
|  |  | | | |

# Procedures

**(To be used in conjunction with the PUS Service Interfaces document, and the Command Line Interface document)**

**The goal here is to reproduce each of the possible failures that the FDIR software is supposed to handle and sure that the problem is resolved by the software automatically.**

1. **SPIMEM\_READ FAILS**
   1. Modify the code so that it appears that one of the SPI memory chips is dead (start with chip # 1). Verify that the code can recover from this by setting chip #1 as being dead and using only chip 2 and 3.Set all chips as dead artificially within the code and verify that -1 is returned.
   2. Modify the code in one the tasks that uses this function slightly so that it will continuously call it improperly. FDIR should eventually delete and recreate this tasks (find a way of verifying that this happens). If the issue persists, FDIR should force the system into SAFE\_MODE. Verify that this happens as well.
   3. Write a bit of code so that one of the tasks acquires the Spi0\_Mutex and then runs into an infinite loop (never releasing the mutex). Verify that the FDIR task deletes and recreates this task, and when the issue continues to persist, it drops into SAFE\_MODE.
   4. Add a bit of code within check\_if\_ready( ), so that if the chip being queried is chip #1, then it always returns a response indicating this it is not ready.Verify that this chip gets set as being dead and then the error handling returns after it starts to use the other SPI chips.Lastly, make it so that all the chips fail in this way and verify that the system enters into INTERNAL\_MEMORY\_FALLBACK mode.
2. **SPIMEM\_WRITE FAILS**
   1. Repeat 1.(a-d) for spimem\_write.
3. **A SPIMEM chips stops working properly.**
   1. Disable the hardware of one of the chips and repeat the testing which occurred in 1.d
4. **Executing a scheduled command failed**
   1. Cause a scheduled command to repeatedly fail.
   2. Verify that the command is tried again by the fdir software.
   3. Verify that if the error persists, that the system drops into SAFE\_MODE.
5. **READ/WRITE from a fifo failed.**
   1. Initialize a fifo as being empty (such as the ones that connect tasks to the OBC\_PACKET\_ROUTER).
   2. (LOW COUNTER): Verify that the fifo is recreated and the issue is resolved.
   3. (HIGH COUNTER): Verify that the task which is supposed to be using this faulty fifo is “reset”.
   4. Verify that if the error persists, that the system drops into SAFE\_MODE.
   5. Repeat this for each of the tasks that utilizes a fifo and observe the results / make corrections where necessary.
6. **//**
7. **Variable not updated during HK collect**
   1. Pick a variable within the OBSW and cause it’s collection to fail.Verify that this causes the system to drop into SAFE\_MODE.
   2. Pick a parameter contained in the SSMs and modify the sensor retrieval software such that if a certain parameter is asked for, then it simply does not respond to the OBC, ex: SPI\_TEMP.
   3. Verify that the FDIR task attempts and succeeds in resetting the faulty SSM.
   4. Verify that as the issue persists that the OBC attempts to reprogram the SSM which malfunctioned (if the correct program is contained the OBC’s SPI memory, verify that this resolves the issue.
   5. Verify that if the error persists, that the system drops into SAFE\_MODE.
8. READ WITH SPIMEM FAILED: **//**
9. //
10. CHIP “BUSY” during initialization
    1. Cause the check\_if\_ready( ) functions to report that all chips are busy at the initialization part of the code.
    2. Verify that the system drops into SAFE\_MODE.
11. ERASING CHIP TAKES TOO LONG:
    1. Decrease the timeout for a chip erase
    2. Verify that this error is triggered.
    3. Verify that the timeout is increased by FDIR.
    4. Verify that if the error persists, that the system drops into SAFE\_MODE.
12. **LOAD SECTOR INTO BUFFER RETURNED INCORRECT # OF BYTES**
    1. Modify this function so that it fails for all memory chips.
    2. Verify that the system enters into INTERNAL\_MEMORY\_FALLBACK mode.
13. **UPDATE\_SPIBUFFER\_WITH\_NEW\_PAGE RETURNED INCORRECT # OF BYTES**
    1. Repeat the testing in 12. But for this function.
14. **ERASE\_SECTOR\_ON\_CHIP FAILED**
    1. Manually decrease the timeout for this function so that it will fail.
    2. Verify that FDIR increases the timeout variable being used in this function.
    3. Keep the timeout low, thereby forcing this function to always fail. Verify that the system drops into INTERNAL\_MEMORY\_FALLBACK mode as a result.
15. **WRITE\_SECTOR\_BACK\_TO\_SPIMEM FAILED**
    1. Repeat the testing in 12. But for this function.
16. **WRITE-IN-PROGRESS WHEN TRYING TO READ**
    1. Repeat the testing in 12. But for this function.
17. **ALL SPIMEM CHIPS ARE MALFUNCTIONING**
    1. Verify that the system enters INTERNAL\_MEMORY\_FALLBACK mode.
18. **SPIMEM\_READ FAILED DURING RTC\_INIT**
    1. Cause this function to return a failure response within rtc\_init.
    2. Verify that the system enters into SAFE\_MODE.
19. **//**
20. **//**
21. **//**
22. **//**
23. **GETTING SENSOR DATA FROM SSM TOOK TOO LONG**
    1. In the SSM code, when the SSM is supposed to retrieve sensor values and then report them to the OBC, insert a 50ms delay.
    2. Verify that the FDIR software increases the timeout allowed for an SSM response.
    3. Verify that the issue is eventually resolved in this manner.
    4. Verify that if the issue persists (make a longer timeout … 250 ms), then the SSM being communicated with is reset.
    5. Verify that if the issue persists (make a longer timeout … 250 ms), then the SSM being communicated with is reprogrammed. If the correct program is contained in the OBC’s SPI memory, then you should observe that this issue is resolved.
    6. Verify that if the error persists, that the system drops into SAFE\_MODE.
    7. Repeat the above tests for “setting variables” in an SSM.
    8. Repeat for each task-SSM combination. (coms.c + COMS SSM, …)
24. //
25. **A TC/TM TRANSACTION BETWEEN OBC & COMS SSM FAILS**
    1. Repeat the tests performed in 23, for sending / receiving PUS packets with the SSM.
26. //
27. //

**Test Ground Station Operability (For each of the following, verify that the action can be executed using the GSSW terminal)**

1. **Enter Low Power Mode**
2. **Exit Low Power Mode**
3. **Enter Safe Mode**
4. **Exit Safe Mode**
5. **Enter Coms Takeover Mode**
6. **Exit Coms takeover Mode**
7. **Pause SSM Operations**
8. **Resume SSM Operations**
9. **Reprogram SSM**
10. **Reset SSM**
11. **Reset Task**
12. **Delete Task**

# Test Results

Report the results of each major test here and the actions that were taken to correct any issues (or simply insert a link to the issue that was opened)