FPManger Unit Test Plan

Jim Hofman

4/21/2015

1. Verify the tables
   1. Load and initialize the tables
      1. Fault table
      2. Action table
      3. Response table
   2. Verify that the component values are equal to the values loaded
   3. This test simply shows the tables were loaded into FPManager correctly
2. Verify the commands
   1. Load and initialize the tables
   2. For each command sent, the command response is tested
   3. FPM\_FAULTID\_ON\_OFF
      1. Disable monitor 2
      2. Verify the monitor was disabled
      3. Enable monitor 2
      4. Verify the monitor is enabled
      5. Send a command with a bad opcode (not ENABLE/DISABLE)
      6. Verify COMMAND\_VALIDATION\_ERROR
      7. Verify *Unexpected Command Argument* EVR
   4. FPM\_ALL\_FAULTIDS\_ON\_OFF
      1. Disable all monitors
      2. Verify monitors are disabled
      3. Enable all monitors
      4. Verify monitors are enabled
      5. Send a command with a bad opcode
      6. Verify COMMAND\_VALIDATION\_ERROR
      7. Verify *Unexpected Command Argument* EVR
   5. FPM\_RESPONSE\_ON\_OFF
      1. Response 1 in enabled
      2. Response 2 is disabled
      3. Response 3 is locked and enabled
      4. Enable response 2
      5. Verify response 2 is enabled
      6. Disable response 1
      7. Verify the response is disabled
      8. Try to disable the locked response
      9. Verify the command is ignored
      10. Try to enable the locked response
      11. Verify the command is ignored
      12. Send a command with a bad opcode (not ENABLE/DISABLE)
      13. Verify COMMAND\_VALIDATION\_ERROR
      14. Verify *Unexpected Command Argument* EVR
   6. FPM\_ALL\_RESPONSES\_ON\_OFF
      1. Response 1 in disabled
      2. Response 2 is enabled
      3. Response 3 is locked and enabled
      4. Disable all responses
      5. Verify response 1 is disabled
      6. Verify response 2 is disabled
      7. Verify response 3 remains locked (enabled)
      8. Enable all responses
      9. Verify response 1 is enabled
      10. Verify response 2 is enabled
      11. Verify response 3 remains locked (enabled)
      12. Send a command with a bad opcode (not ENABLE/DISABLE)
      13. Verify COMMAND\_VALIDATION\_ERROR
      14. Verify *Unexpected Command Argument* EVR
   7. FPM\_RESET\_EVR\_THROTTLING
      1. Send command to Reset Evr Throttling
      2. Send a FPM\_RESET\_EVR\_THROTTLING command with a bad throttle group id
      3. Verify COMMAND\_VALIDATION\_ERROR
      4. Verify *Invalid Throttle Group* ID EVR
      5. Send command to Reset Evr Throttling
      6. Verify throttling is disabled
      7. Test throttling of EVRs in the Action Handler
         1. The *evr\_action\_threshold* is set to 10
         2. Invoke to Action – action id = 7
         3. Verify *Fault Response Completion Status* EVR is “RUNNING”
         4. Invoke the same Action 9 more times
         5. Verify *Same Action Received* EVRs
         6. Verify DUPLICATE\_ACTION\_EVENT is sent
         7. Invoke the same Action 1 more time
         8. Verify no EVR goes out
         9. Verify DUPLICATE\_ACTION\_EVENT is not sent
         10. Send *Reset Evr Throttling* command
         11. Send the same Action
         12. Verify *Fault Response Completion Status* EVR is “RUNNING”
         13. Verify *Same Action Received* EVRs
      8. Test throttling of EVRs in the FaultIn Handler
         1. Send *Reset Evr Throttling* command
         2. The *evr\_fault\_request\_threshold* is set to 9
         3. Invoke the same fault 9 times
         4. Verify 9 *Fault Response Request* EVRs
         5. Verify Last Fault and Fault Count tlm
         6. Invoke one more time
         7. Verify EVRs are not being sent out
         8. Verify tlm continues
         9. Verify EVENTS continue to be sent
         10. Send *Reset Evr Throttling* command
         11. Invoke the fault again
         12. Verify *Fault Response Request* EVR
         13. Verify Last Fault and Fault Count tlm
         14. Verify the EVENT was sent out
      9. Send *Reset Evr Throttling* command
      10. Verify *Fault Response Request* EVR
3. Verify the FaultIn Handler
   1. Load and initialize the tables
   2. Run standalone test to verify ASSERT
      1. Send a bad fault ID to the handler
      2. Verify the ASSERT
   3. Invoke a fault for each member of the table
   4. Verify *Fault Response Request* EVR
   5. Verify FaultCount telemetry
   6. Verify LastFault telemetry
   7. Verify that the proper events are sent
   8. Disable a fault monitor then trigger a fault
      1. Turn off Fault monitor 3
      2. Verify that no event is sent
      3. Verify that TLM continues
      4. Verify *Fault Response Request* EVR
4. Verify the Action Handler
   1. Standalone ASSERT Tests
      1. ASSERT on Action requested that is not the Action Table
         1. Invoke to Action with an invalid Action ID
         2. Verify ASSERT
      2. ASSERT when responses are in in both the Action Table and the Response Table
         1. Change a response number in the Action Table so that it is not in the Response Table
         2. Invoke an Action and verify the ASSERT
         3. Repeat for the Response Table
      3. ASSERT on an unknown status in the Response Table
         1. Change the Status value on an entry in the Response Table
         2. Invoke an Action and verify the ASSERT
   2. Invoke an Action for each entry in the Action Table
   3. Look in the Response Table to check status (ENABLED/DISABLED/LOCKED)
   4. If DISABLED send *Fault Response Status* EVR with RESP\_DISABLED status
   5. If ENABLED or LOCKED send *Fault Response Completion Status* EVR with RUNNING status
   6. Send the same action that is currently RUNNING
   7. Verify *Same Action Received* EVR
   8. Verify DUPLICATE\_ACTION\_EVENT go out Event port
   9. Verify EVR throttling
      1. Invoke the same action to the threshold limit of 9
      2. Verify *Same Action Received* EVRs
      3. Verify DUPLICATE\_ACTION\_EVENTs go out Event port
      4. Invoke the same action 1 more time
      5. Verify no new EVRs
      6. Verify no new EVENTS go out Event port
5. Verify the comResponseIn Handler
   1. Load and initialize the tables
   2. For each entry in the ACTION Table
      1. Write a unique positive context value into the Action Table to simulate an Action request.
      2. Invoke *cmd\_ResponseIn* with an opcode, the new context, and COMMAND\_OK response
      3. Verify the command count is incremented each time
      4. Verify the context remains the same throughout
      5. When complete verify that cmdCount == numCmds
      6. Verify the EVENT out: RESPONSE\_COMPLETE\_EVENT
      7. Verify *Fault Response Completion Status* EVR with a status of COMPLETE
   3. Reset the cmdCounts in the ACTION Table
   4. For each entry in the ACTION Table
      1. Write a unique positive context value into the Action Table to simulate an Action request
      2. Invoke *cmd\_ResponseIn* with an opcode, the new context, and COMMAND\_VALIDATION\_ERROR response
      3. Verify the context remains the same throughout
      4. Verify cmdError == true for each entry
      5. When complete verify that cmdCount == numCmds
      6. Verify the EVENT out: RESPONSE\_ERROR\_EVENT
      7. Verify *Fault Response Completion Status* EVR with a status of COMPLETE
6. Verify serialized data
   1. Load and initialize the tables
   2. For all the Actions test the opCodes and arguments
   3. Invoke the action
   4. Get the serialized data that was sent out from the Action handler
   5. Deserialize the data (OpCode, arg1, arg1)
   6. Save the opCode
   7. Get the Arguments
   8. Deserialize Arg1
   9. Deserialize Arg2
   10. Verify the match what was sent out