

**UC45 Flash data integrity self-test at FFP**

Contents

[1. Introduction 3](#_Toc398904359)

[2. Business Story 4](#_Toc398904360)

[3. Requirements 4](#_Toc398904361)

[3.1 Use cases 4](#_Toc398904362)

[Use Case 1: Valve action 4](#_Toc398904363)

[Use Case 2: Status check 4](#_Toc398904364)

[Use Case 3: Mode change 4](#_Toc398904365)

[Use Case 4: Local display 4](#_Toc398904366)

[Use Case 5: Reboot device and recover 4](#_Toc398904367)

[Use Case 6: Self-test status monitor 4](#_Toc398904368)

[3.2 Steps to use it 5](#_Toc398904369)

[4. Implementations 6](#_Toc398904370)

[4.1 Algorithm change 6](#_Toc398904371)

[4.2 Memory or code cost 6](#_Toc398904372)

[4.3 Special storage 6](#_Toc398904373)

[4.4 Impact on other implementation 6](#_Toc398904374)

[5. How to test it 7](#_Toc398904375)

[5.1 Test cases 7](#_Toc398904376)

[5.2 Extremes 7](#_Toc398904377)

[6. Impact on the rest of the code 7](#_Toc398904378)

[7. Revision History 8](#_Toc398904379)

# Introduction

The purpose of this document is to present the design of [flash data integrity self-test at FFP] for FF SVI project.

The design follows the flash test design in APP and is based on the following assumptions:

* ROM (flash) failure detection by the code running from that ROM is the matter of luck
  1. Therefore, if a failure is detected, the action to drive the valve to the failsafe position must be immediate
* There is 1 block of flash memory to do self-test at runtime
* The begin address is the address of the vector table
* The end address is address of variable “mychecksum”
* APP and FFP share the same algorithm for verifying the CRC [refer to APP design document to find more detail]

Note that the FF CPU will verify the integrity of the active code running in the device. There may be a second image that is downloaded in the other partition, but the software will NOT verify the integrity of that code until that location is activated.

# Business Story

Detects the error early and causes the positioner to de-energize the valve.

# Requirements

NOTE: The cases below cannot be always guaranteed because they are implemented by the code running from corrupted image, and the code itself may be compromised. However, the requirements spell out the best outcome possible in the event of flash failure.

## Use cases

## Valve action

The device detects memory failure and it is switching to de-energized position.

This is met by the following design:

1. IPC communication to APP is disabled on two levels
   1. IPC UART is reconfigured to a wrong speed, so that APP doesn’t receive any valid messages
   2. IPC HART Master is disabled so that FFP doesn’t send messages
2. Actions per #1 are performed
   1. Immediately on failure detection
   2. Periodically to improve the chances of success

In a short period of time (default 2 seconds), APP will detect loss of setpoint and de-energize the valve and set SP\_TIMEOUT fault. This fault shall be recoverable: if APP resumes receiving the setpoint, it will continue controlling the valve.

After some time, for example some minutes, APP generates IPC\_LOST fault [new design non-recoverable fault] and APP change mode to FAILSAFE, and the valve is de-energized.

## Status check

User wants to know if the device has flash memory integrity issue

Open Resource block – Read the Block Error, check if exist BLK\_ERR\_MEMORY\_FAILURE.

## Mode change

When self-test fails, the APP changes to FAILSAFE mode and user can view IPC\_LOST and SP\_TMOUT failure occurs after certain time. RB actual modes change to OOS mode. RB target mode stays the same as before.

## Local display

When self-test fails, the local display don’t refresh all information from FFP, and display “FF PARAMS” in cycle menu with status “BAD” and value “NULL”.

## Reboot device and recover

The user reboots the device and the device’s behavior depends on the new round of check result, since it’s based on flash memory and it is expected to be unrecoverable.

## Self-test status monitor

When the algorithm is abnormal, for example not running for some minutes, open resource block and read block error, check if exists BLK\_ERR\_DEV\_NEEDS\_MAINT\_NOW.

It only happens when CPU is fully occupied, it rarely happens, this case could be tested by code review.

By investigation that one round of flash self-test cost less than 2 seconds, the timer for above use cases to take actions are:

Valve action: about 3 minutes

Block error check: about 2 seconds

Block mode change: about 2 seconds

Local display: about 20 seconds

Self-test status monitor: 120 minutes

## Steps to use it

(flow diagram if necessary)

# Implementations

## Algorithm change

Invoke self-test initiation at beginning of idle loop which is before while (TRUE), to initialize the parameters. For FFP these specific parameters are needed:

* FLASH\_START which is address of vector table
* Timer\_Ticker which indicate the start time of self-test.

Enable OSTaskIdleHook, and invoke the existing CRC algorithm, which is seflttest\_FlashCrc() located in MN\_\_\diagnostics. For FFP these specific parameters are needed:

* Address of mychecksum
* Timer\_Ticker which to refresh time stamp of self-test.

Fault handler when CRC fails:

* Set BLK\_ERR\_MEMORY\_FAILURE at p\_resource->block\_err
* Set actual mode of RES block to OOS
* Stop Hart master to stop IPC communication
* LUI display FF PARAMS in cycle menu to be bad and NULL
* APP stays in FAILSAFE mode, and generate faults SP\_TMOUT and, later, IPC\_LOST when IPC stops for certain minutes.
* Flash-self keep running in idle task.

Invoke selftest\_FlashCrcMon() by Appl\_background\_RESB(), to monitor the execution of self-test.

This specific parameter is needed:

* Timer\_Ticker which to adjudge the status of self-test.

Fault Handler:

When self-test abnormal, it is rare possibility but it happens in case of CPU too busy. This is a catastrophic failure that may indicate that flash failure detection code itself is corrupted. In this case, the FFP intended behavior and APP behavior remain the same as if flash fault was detected, except that RB block error is set to DEV\_NEEDS\_MAINT\_NOW.

## Memory or code cost

Negligible

## Special storage

Negligible.

## Impact on other implementation

If power consumption rises unacceptably, we need to slow down the running of check.

# How to test it

## Test cases

### Pre-test

Ensure that “HART over FF” command 255.2 can disable IPC traffic and SP\_TMOUT and later IPC\_LOST faults are set correctly and the local UI behaves as described above

Re-enable IPC traffic and bring the device to normal working state

### Simulate flash fault

Using “HART over FF” command 255.4, corrupt RAM variable flashtest whose can be found in the map file accompanying the build. The type of flashtest is  
struct flashtest\_t {  
 u32 flashpoint;  
 u32 flashticks;  
 u16 crc;  
 u16 CheckWord;  
}  
Note that the corrupted value must still have a valid CheckWord. E.g. increment crc and CheckWord by the same amount.

FFP flash failure shall be detected in a few seconds; soon after APP will de-energize the valve (SP\_TMOUT) and within a couple of minutes go failsafe with the IPC\_LOST fault.

Re-enable IPC traffic using “HART over FF” command 255.2 periodically/continuously. Test that the SP\_TMOUT fault is present or re-detected if cleared.

### Simulate flash test timeout

Write to RAM variable flashticks a value like TMOUT\_FLASH\_CRC\_TEST=2\*3600\*1000/200; this should trigger the failure.

## Extremes

# Impact on the rest of the code

# Revision History

The table below describes the revision history of this document.

|  |  |  |
| --- | --- | --- |
| Rev. | Changed figure, table, chapter | Title or brief description |
| A | 08/27/2014 | Initial version by Eric Jiang. |
| B | 09/19/2014 | Update use case 3. |
| C | 12/08/2014 | Update use case 3. |
| D | 07/12/2015 | Clarified assumptions and requirements; proposed test cases |
| E | 12/12/2020 | Corrected test procedure for R3. |
|  |  |  |
|  |  |  |