Robust Inter-processor communications in SVI FF

# Use cases

## Loss of communications

This may be random (a glitch) or intentional (e.g. detected flash failure in FFP).

When IPC communications are lost, FF communications shall continue without interruption (in case of flash failure, to the best of the device’s ability).

To correct an IPC problem, RB.RESTART=Processor may be written.

## Fatal exception in FFP

In this case, FFP goes through a reset and initialization, and the time it takes will surely exceed the runtime timeout in APP. APP will register “IPC LOST” fault and go failsafe, thus requiring operator’s actions. Since it is known that Softing FF stack may generate fatal exceptions (TFS\*), it is best to reset APP at the time of exception to allow a much more forgiving timeout. The in-place reset feature of APP will hold the valve in the current position until FFP is ready to control.

# Requirements

## BREAK synchronization

When FFP catches a fatal exception, it shall emit a BREAK over IPC UART. APP shall be able to detect a BREAK and reset via MN\_RT\_ASSERT.

Explanation: IPC UART is the only working connection between FFP and APP, so it is the only choice. At the time of exception, normal communications over UART are not possible, so a BREAK is the only choice. On the APP side, if a BREAK is detected two times in a row, APP will enter failsafe mode, which is a reasonable standard behavior.

## Communications recovery

FFP shall attempt to recover from IPC errors without blocking FF communications

## Start-up

On start-up, FFP shall not hang on “connecting” to APP thus blocking FF link. Instead, it shall communicate with last known good device address. In case of failure, a connection is (periodically) attempted, and in the event of “device id” mismatch, new address is remembered and the device is reset. This is primarily interesting to manufacturing.

## Default tags etc.

This, too, is primarily interesting to manufacturing.

If the device enters FF communications and then “device id” is changed, full reinitialization of Softing stack will create new defaults. This shall be accomplished by writing RB.RESTART=169.

# Derived implementation requirements

## FFP crash

Any FFP crash/fatal exception that causes a reset shall cause APP reset as well. This shall be done by routing all exceptions via mn\_assert which in turn shall send a BREAK signal via IPC UART. APP shall reset upon receiving a BREAK and shall not preserve any non-persistent faults

## FFP reset requested

Normal FFP reset on RB.RESTART=Processor request shall attempt to reset APP using command 42. If that fails, FFP shall invoke crash mechanism described above.

### FFP reset – incidental

When FFP wants to reset itself a part of requested operation, except as described in the next section, and is unable to reset APP using command 42, FFP shall not reset, and the operation shall fail.

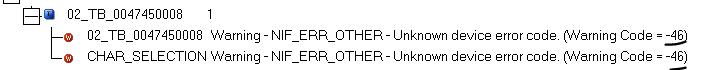
### Eliminating window of vulnerability

In a few cases operation in FFP and APP absolutely must be synchronized. An example is activation of newly downloaded firmware, where the final step is simultaneous reset. These cases shall be handled as in [subsection 2](#_FFP_reset_requested).

# Known snags

## Error reporting

When IPC is stopped, the reported error is -46 (unknown):

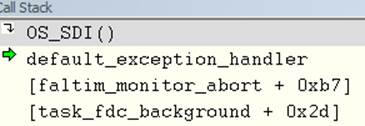


Our code has nothing like that, so it is buried probably in the bowels of Softing stack or in NI error interpretation.

## Problem with timing

If IPC is stopped before NI tools are synchronized with the device, Softing stack generates fatal exception after 50 s of failure to establish a VCR.

Call stack:



See also a screenshot below. Softing account:

<SOFTING>

The exception you listed is caused by a problem with VCR’s in a completely different code section:

Function faltim monitor\_abort() is called on a timer event in task\_fdc\_background() to check active connections.

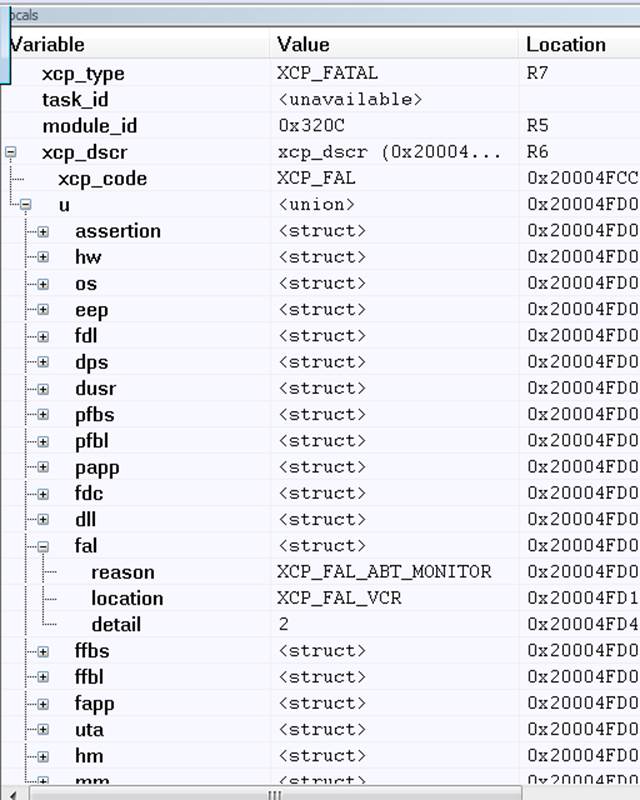
Detail = 1 means a problem with a management VCR.

Detail = 2 means a problem with Function Block Application VCR, index 0.

…

This indicates that there was no communication on this VCR for about 50 seconds(!) and that the stack did not properly generate or process an abort. Or the timer is running way too fast, or the global data structure conn\_description got corrupted.

</SOFTING>



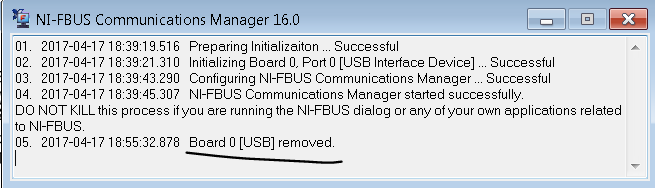
There is nothing we can do about this. The good news is that after a reset, communications are restored.

## Problems resetting when IPC stopped

### Symptoms

When IPC is stopped, and then RB.RESTART=Processor is issued, NI tools are not very stable and may need to be restarted. In most cases, the problem is with NI Communications Manager (the driver).

In some cases, the driver crashes (you can check in systray if it is running). Sometimes, the driver loses the interface, as shown in a screenshot below.



In yet other cases, the problem is with NI Configurator: it fails to synchronize with the device.

### Countermeasures

It is handy to have Smarts Assistant connected over FF at all times. The reason is that connection goes through IPC and will fail if IPC is topped. However, when connected, SA can still run commands intercepted in FFP (currently, subcommands of 255).

If communications fail with NI tools, try SA with a harmless command, like 255.9 “Read active flash bank”.

If it fails, restart the NI Communications Manager. SA will lose connection, and will not be able to connect if IPC is still stopped. Take NI Configurator online and see if communications are restored, and whether IPC can be restored by RB.RESTART.

If a command with SA succeeds, the problem is likely with NI Configurator. Try Updating the view (on any level of hierarchy) or removing the device and letting it be found again. This helps often but not always; you may need to restart NI Configurator.

# Testing and debugging notes

## Remote testing/debugging

The testing tools are:

1. NI Configurator to monitor FF communications
2. Smarts Assistant connected via HART over FF (SA-FF)
3. Smarts Assistant occasionally connected via ISP (SA-ISP)
4. Local UI (for resetting APP while in Failsafe)

Connecting and disconnecting SA-ISP has the effect of a power cycle – both CPUs reset. This is the only use of SA-ISP in this context.

If FFP is reset while SA-ISP is connected, the FFP enters “bootloader mode” and our firmware is not running. Occasionally, it is true for APP, too. Connecting and disconnecting SA-ISP will revive the system.

## Stopping IPC

If IPC is stopped,

1. TB dynamic parameters shall have status BAD with substatus “Device Failure”
2. TB mode shall be OOS
3. APP shall be in failsafe within 2-3 seconds
4. FF communications shall be normal, except for errors in reading/writing TB parameters owned by APP. See, however, [Known snags](#_Known_snags)

### On APP side

With factory mode enabled, inject fault “IPC Disconnect” (68) using command 130.139.

### On FFP side

Use command 255.16 with parameter “Hard kill IPC communications”

## Restarting IPC

Writing RB.RESTART=Processor shall cause a reset with restored IPC.

If IPC was stopped on FFP side, command 255.16 “Hard restart IPC communications” shall restore IPC. APP will still be in Failsafe.

If IPC was stopped on APP side, reset from local UI (physical! Emulation is not available) shall restore IPC. APP will still be in Failsafe.

## Synchronized panic reset

With everything running fine, and with no traps in either CPU and faults in APP cleared, cause a trap in FFP (e.g. by corrupting a protected RAM object. `flashtest’ of the section below will do.

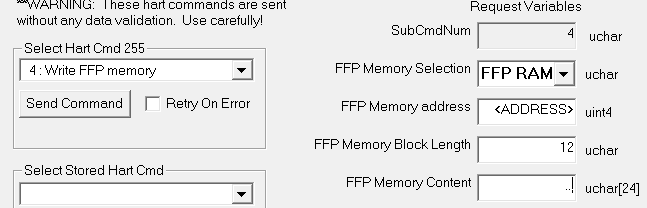
Expected result:

1. Both CPUs reset and have a trap
   1. Trap in FFP corresponds to what caused it
   2. Trap in FFP is an assert pointing to stm\_uart.c
2. APP is in normal mode
3. The valve didn’t move appreciably (in-place reset)

The same corruption repeated twice in a row (within 20 s):

1. Both CPUs have two traps; one as before, and the second “Repeat offense” -10.
2. APP has “MPU fault” and is in failsafe
3. The valve is de-energized

## Simulated flash failure in FFP

1. Since flash self-test structure is now checksum-protected, corrupting one of its elements will create trap -7 (RAM failure) instead of flash failure. To elicit flash failure, write the whole structure complete with correct checksum:
2. In the map file, find ` flashtest’ in the map file; its size should be 12 bytes.
3. Using SA with HART over FF, command 255.4, write e.g.   
   0x80 0x63 0x00 0x08 0x6b 0xe8 0x04 0x00 0x1e 0xb8 0x0c 0x0c  
     
   (Details: you can read flashtest object and increment (or decrement) by e.g. 1 byte 8 (current crc) and byte 11 (the checksum). Indices heare are 0-based)
4. NOTE that it may require to send the command per step 3 several times in a row to achieve the simulated flash failure
5. The expected result is
   1. All the effects of stopped IPC
   2. IPC cannot be restarted with command 255.16
   3. Writing RB.RESTART=Processor restores the IPC after reset.