Titel	FSD300
Products	Safety Simplifier
Requirements	61508
Purpose	Detailed tests
History	Jesper Ribbe2018-01-11 V1 initial version Mike Fu 2018-04-18 V2 Add CPU1 Module test 52~56, CPU2 29~31 Toby Li 2018-08-14 V3 Update HW module test and add check SW with diff HW. Mike Fu 2018-09-01 V4 Update comments Jesper Ribbe 2018-09-11 V5 Renamed from “SW_module_tests” to “SW_HW_module_integrartion_tests” Mike Fu 2018-10-09 V6 Add CPU2 Module test32~35 Add CPu2 module test 57 Mike Fu 2018-10-10 V7 Update CPu2 module test 35 Lobin Lin 2025-08-03 V8 Update for Simplifier 2.0

Table of Contents

Introduction

This document contains both test specification and test report. New version of the document will be issues if new SW.
As some tests covers several SW requirements, the tests are just numbered, and then there is a a list of requirements that are met by the test.
Many tests need a special config file. The file name and GIT commit hash is shown for each test.

SafetyTest

Software module tests for CPU1

Module Test #1 - Power-up stable test

Test Specification:

Number of units used	Set up with 16 Nodes works, and tested at node 5.
Config file	stressTest2.ssplog
Alternative used	1

Perform a check of the power-up stable, The purpose is to verify if enter fatal error at power-up.
Procedure:

1. Start the unit.
2. The unit should restart per 5 seconds all the time, and not enter any fatal error. Note that if enter fatal error, the unit still keep in fatal error mode, not restart gain.

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>

Check the power-up stable about 12 hours, the system restart per 5 seconds all the time stable, not enter fatal error.

Module Test #2 - Runtime CPU2CPU RX DMA stable test

Test Specification:

Number of units used	Set up with 16 Nodes works, and tested at node 5.
Config file	stressTest2.ssplog
Alternative used	1

Perform a check of the code that checks the CPU2CPU DMA RX stable with bad data. The purpose is to verify that this code will behave correctly if wrong data is detected runtime, and the system should still works normal.

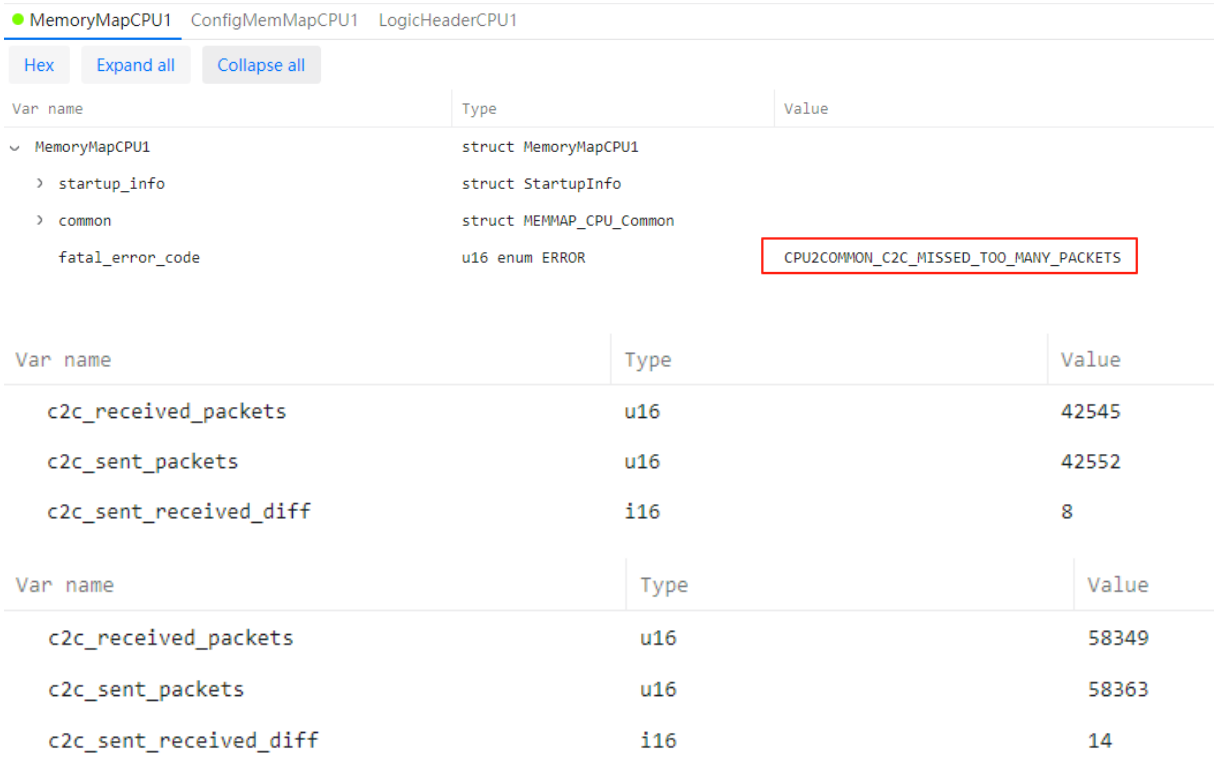
Procedure:

1. Start the unit. Run “j2.py inject -i 2 -t s 11264” via usb
2. Note that the SW will flush out one recieve byte if board button short pressed.
3. Quickly short pressed board button to flush out one receive buffer to destroy the data. It can filter out incorrect data and discard received data packets. If the number of data packets exceeds 20, an error will be triggered.

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>

Use board button pressed in board for test. If the number of c2c_sent_receive_diff increased when button short press quickly, and the system still works normal. If the number of data packets exceeds 20, an error will be triggered.



Var name	Type	Value
c2c_received_packets	u16	12287
c2c_sent_packets	u16	12307
c2c_sent_received_diff	i16	20

Module Test #3 - Runtime CPU2CPU TX DMA with bad CRC test

Test Specification:

Number of units used	Set up with 16 Nodes works, and tested at node 5.
Config file	stressTest2.ssplog
Alternative used	1

Perform a check of the code that checks the CPU2CPU DMA TX stable with bad CRC. The purpose is to verify that this code will behave correctly if some unexpected data with CRC error, and the unit should still works normal.

Procedure:

1. Start the unit. Run “j2.py inject -i 3 -t s 11264” via usb
2. Quickly short pressed board button to flush out one receive buffer to destroy the data. The system should still works normal, but CRC error should data should be detected at CPU2.

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>

if button pressed in board one time. The num_c2c_not_plus1_pkt_count will add one. Attached the value as below:

num_c2c_not_plus1_pkt_count	u8	4

Var name	Type	Value
num_c2c_not_plus1_pkt_count	u8	5

Columns Filter logs Hex Total logs: 4 Hidden logs: 2029 Autoscroll Clear list

CPU	Name	Timestamp	ftick_idx	Message
2	C2C_Error	55969.5450 ms	12	type=RX_BAD_CRC, arg1=44320, arg2=44511, arg3=255
1	C2C_Error	55971.6776 ms	13	type=RX_SAME_COUNTER, arg1=149, arg2=149, arg3=0
1	C2C_Error	55971.6785 ms	13	type=RX_BAD_COUNTER, arg1=150, arg2=149, arg3=0
1	C2C_Error	55972.6757 ms	13	type=RX_BAD_COUNTER, arg1=150, arg2=151, arg3=0

REMOVE Module Test #4 - Runtime CPU2CPU TX DMA with destroy Tx length

Test Specification:

Number of units used	Set up with 16 Nodes works, and tested at node 5.
Config file	stressTest2.ssplog
Alternative used	1

this test is not valid anymore due to using fixed size packets

Perform a check of the code that checks the CPU2CPU DMA TX stable with destroy Tx length. The purpose is to verify that this code will behave correctly if unexpected data is detected runtime.

Procedure:

1. In Sconstruct, enable the SAFETY_TEST_INJECT = 4 test. Recompile and flash CPU1 via USB. Note that the SW will send out two packets with unexpected length data per 256 packets, one with len+1, the other one with len-1.
2. Start the unit.
3. Run showDebug.py --driver pyserial --filter cpu2dbgData.dbg to watch rxCpu2cpuCRCerrors and rxSearchPreambleUnexpects should be detected.

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	

Use showDebug.py --driver pyserial --filter cpu2dbgData.dbg for watch the value of rxCpu2cpuCRCerrors and rxSearchPreambleUnexpects. The number is always change. attached the value as below:

Module Test #5 - Inject fatal error test

Test Specification:

Number of units used	Set up with 16 Nodes works, and tested at node 5.
Config file	stressTest2.ssplog
Alternative used	1

Perform a check of the code that fatal error works. The purpose is that verify fatal error able to works, and display the fatal error number at the extern LED board. The unit also able to restart if hold pressed board button or cap button when unit enter fatal error.

Procedure:

1. Start the unit. Run “j2.py inject -i 4 -t s 11264” via usb
2. The unit should enter fatal error after about five seconds. All SIOs, SIO power, relay power, plus 12V should be off.

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>

When power up the unit, The uint enter fatal error CPU1COMMON_NO_ERROR (11264) after five seconds. All SIOs, SIO power, relay power, plus 12V be turned off at fatal error status. Attached the pic as below:

Module Test #6 - Runtime CPU2CPU not send packets 30ms test

Test Specification:

Number of units used	Set up with 16 Nodes works, and tested at node 5.
Config file	stressTest2.ssplog
Alternative used	1

Perform a check of the code that checks the CPU2CPU TX not sending packets for 30ms. The purpose is to verify that this code will behave correctly into CPU1COMMON_C2C_MISSED_TOO_MANY_PACKETS when CPU2 not received packet from CPU1 more than 30ms.

Procedure:

1. Start the unit. Run “j2.py inject -i 1 -t s 11304” via usb
2. Note that CPU1 not send out datas to CPU2 about 30ms after delay five seconds.

3. The unit should enter fatal error CPU1COMMON_C2C_MISSED_TOO_MANY_PACKETS (11304) after about five seconds.

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>

Module Test #7 - Runtime CPU2CPU RX timeout test.

Test Specification:

Number of units used	Set up with 16 Nodes works, and tested at node 5.
Config file	stressTest2.ssplog
Alternative used	1

Perform a check of the code that checks the CPU2CPU RX timeout. The purpose is to verify how long to time out about not received packet for CPU2 to indicate fatal error. A real RX time out is difficult to trigger error at HW, so the code has to fake not reload the RX timing after five seconds.

Procedure:

1. Start the unit. Run “j2.py inject -i 6 -t s 11264” via usb
2. Note that for easy to measured the time lenght, the SW use TP4 high pulse lenght to indicate the time RX time out. TP4 should be set high after unit power-up 5 seconds, and then set to low if SW detect timeout.
3. The unit should enter fatal error after about five seconds.

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>

Module Test #8 - Runtime CPU2CPU TX pin a short time short-circuit to test

Test Specification:

Number of units used	Set up with 16 Nodes works, and tested at node 5.

Config file	stressTest2.ssplog
Alternative used	1

Perform a check of the code that checks the CPU2CPU TX pin short circuit to GND with a short time. The purpose is to verify that it should works normal if TX short circuit to GND a short time(less than 20ms). A real short time short circuit to GND is difficult to handle by HW, so the code use TP4 to output 3ms high pulse per 256ms, and TP4 driver a NPN transistor to pull low the CPU1 TX PIN.

Procedure:

1. Start the unit. Run “j2.py inject -i 7 -t s 11264” via usb
2. Mount an extern board to CPU1 TX pin.
3. Start the unit.
4. The unit should still works normal.

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>

Module Test #9 - Runtime CPU2CPU RX pin a short time short-circuit to GND test.

Test Specification:

Number of units used	Set up with 16 Nodes works, and tested at node 5.
Config file	stressTest2.ssplog
Alternative used	1

Perform a check of the code that checks the CPU2CPU RX pin short circuit to GND with a short time. The purpose is to verify that it should works normal if RX short circuit to GND a short time(less than 20ms). A real short time short circuit to GND is difficult to handle by HW, so the code use TP4 to output 3ms high pulse per 256ms, and TP4 driver a NPN transistor to pull low the CPU1 TX PIN.

Procedure:

1. Start the unit. Run “j2.py inject -i 7 -t s 11264” via usb
2. Mount an extern board to CPU1 RX pin.
3. Start the unit.
4. The unit should still works normal.

Test Execution:

Date	

Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>

Module Test #10 - Runtime CPU2CPU TX overrun test

Test Specification:

Number of units used	Set up with 16 Nodes works, and tested at node 5.
Config file	stressTest2.ssplog
Alternative used	1

Perform a check of the code that checks the CPU2CPU TX overrun. The purpose is to verify that this code will behave correctly check with CPU1 TX overrun. A real DMA TX overrun is difficult to trigger error, so the code has to fake to send out many data at the same time after five seconds.

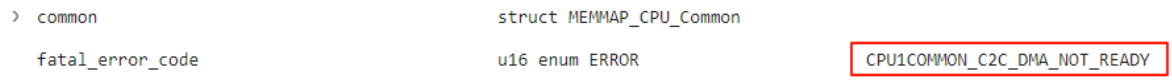
Procedure:

1. Start the unit. Run “j2.py inject -i 1 -t s 11307” via usb
2. The unit should enter fatal error CPU1COMMON_C2C_DMA_NOT_READY (11307) after about five seconds.

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>

The unit into fatal error and indicate CPU1COMMON_C2C_DMA_NOT_READY (11307) error. Use MemmapRead to see data as below attached pictiure.



Module Test #11 - Runtime CPU2CPU not allow to send CPU2CPU data to CPU2 test

Test Specification:

Number of units used	Set up with 16 Nodes works, and tested at node 5.
Config file	stressTest2.ssplog
Alternative used	1

Perform a check of the code that send C2C data to CPU2 when C2C not allow to send it yet.The purpose is to verify that this code will behave correctly into CPU1COMMON_C2C_DMA_NOT_READY (11307) when a data send but DMA_TX is disable.

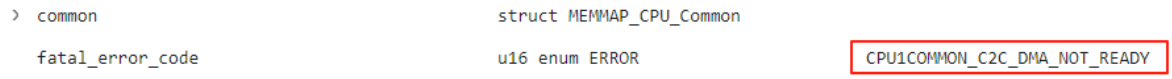
Procedure:

1. Start the unit. Run “j2.py inject -i 2 -t s 11307” via usb
2. The unit should enter fatal error CPU1COMMON_C2C_DMA_NOT_READY (11307)

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>

The unit into fatal error and indicate CPU1COMMON_C2C_DMA_NOT_READY (11307) error. Use MemmapRead to see data as below attached pictiure.



Module Test #12 - Production data crc invalid test

Test Specification:

Number of units used	Set up with 16 Nodes works, and tested at node 5.
Config file	stressTest2.ssplog
Alternative used	1

Perform a check of the code that checks the production data crc. The purpose is to verify that this code will behave correctly struck in small bootloader when the production data CRC error.

Procedure:

1. Modify the booty code to determine if it is stuck in the smallbootloader mode when the pd crc calculation is correct
2. Run J-flash to flash CPU1. Start the unit.
3. Run “j2.py probe” will see that the IN_SMALL_BOOTLOADER bit is 1
4. Run “j2.py pd_write test.mbd test.pd ” via usb
5. Restart the unit. Run the third step will yield the same result.
6. Flash the correct image for CPU1 by J-FLASH and execute step 4, and unit will work properly

Test Execution:

Date	<>
Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>

The unit will struck in small bootloader.

```
PS D:\dev\jtremote\ABJier\src\SRC900-002-Jab2> .\j2.py -pCOM29 probe
Got JabusAnswerProbe(header=JabusHeader(dst=0, length=16, cmd=0), main_cpu_nr=1, op_flags=32769, cdata=ProbeConstantData
(jabus_hw_type=0, product_table_ptr=0, product_table_size=0, protocol_product_specific_type=0))
IN_SMALL_BOOTLOADER  : 1
IN_BIG_BOOTLOADER    : 0
SETTINGS_WRITE_OK    : 0
ROUTING_OK           : 0
NORMAL_OP            : 0
FATAL_ERROR          : 0
HALF_DUPLEX          : 0
RESTART_NEEDS_FF     : 0
FLOW_CTRL            : 0
CANNOT_RUN_FW        : 0
BOOTY                : 32768
```

Module Test #13 - C2C communicate missed packets from CPU2 test

Test Specification:

Number of units used	Set up with 16 Nodes works, and tested at node 5.
Config file	stressTest2.ssplog
Alternative used	1

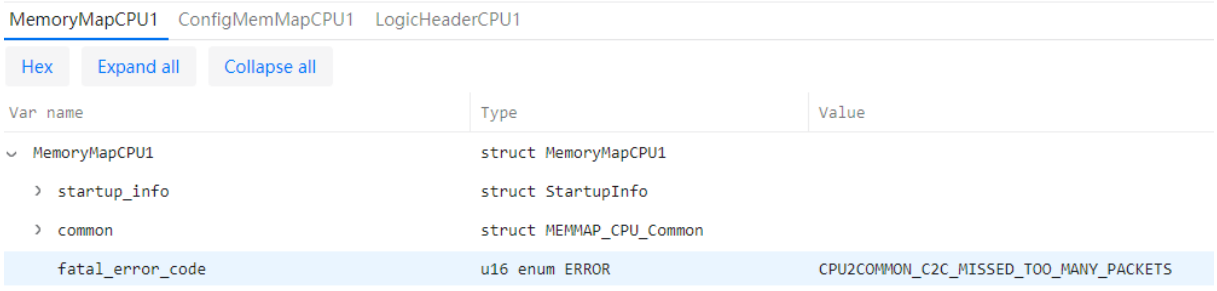
Perform a check of the code that checks the CPU1 missed packets from CPU2. The purpose is to verify that this code will behave correctly if CPU1 receive CPU2 with lost packets. The CPU1 allow to lost about 20 packets from CPU2. A real lost packets more than 20 packets is difficult to happen, so the CPU2 code has to fake to miss 20 packets one time to trigger CPU1 enter fatal error CPU2COMMON_C2C_MISSED_TOO_MANY_PACKETS (12328) .

Procedure:

1. Start the unit. Run “j2.py inject -i 2 -t s 12328 -r” via usb
2. The unit should enter fatal error CPU2COMMON_C2C_MISSED_TOO_MANY_PACKETS (12328) after about five seconds.

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>



Module Test #14 - Not increase the TX packet count test

Test Specification:

Number of units used	Set up with 16 Nodes works, and tested at node 5.
Config file	stressTest2.ssplog
Alternative used	1

Perform a check of the code that CPU1 not increase the TX packet count test. The purpose is to verify that this code will behave correctly check CPU1 should able to indicate error if packet count not changed.

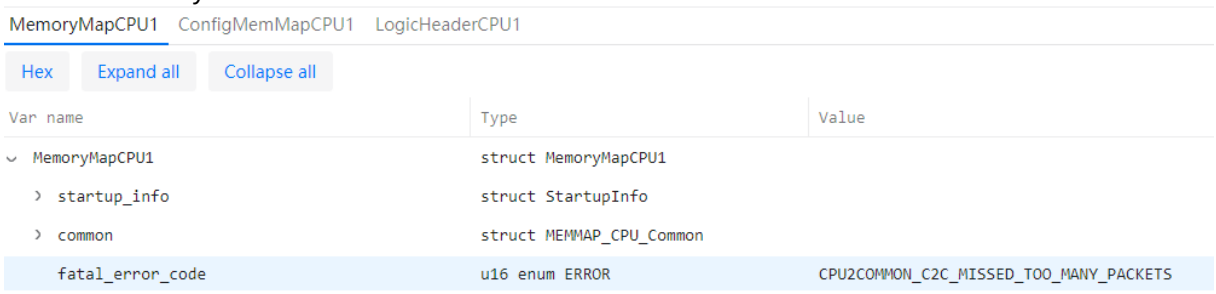
Procedure:

1. Start the unit. Run “j2.py inject -i 2 -t s 12328 -r” via usb
2. It should enter fatal error to indicate

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>

The unit into fatal error and indicate CPU2COMMON_C2C_MISSED_TOO_MANY_PACKETS (12328) error about delay 5 seconds.



Module Test #15 – CPU2 invalid SW version test.

Test Specification:

Number of units used	Set up with 16 Nodes works, and tested at node 5.
Config file	stressTest2.ssplog
Alternative used	1

Perform a check of the code that checks the CPU2 SW version. The purpose is to verify that this code will behave correctly if CPU2 SW version error. It shall enter fatal error CPU2 SW version if CPU2 flash a mismatch SW version.

Procedure:

1. Flash both CPUs SW to let unit work normal.
2. Start the unit. The unit shall works normal, not give out fatal error.
3. Flash CPU2 with incorrect SW version.
4. Restart the unit
5. The unit should enter fatal error INVALID_CPU2_VERSION

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>

Module Test #16 – CPU2 blank test

Test Specification:

Number of units used	Set up with 16 Nodes works, and tested at node 5.
Config file	stressTest2.ssplog
Alternative used	1

Perform a check of the code that checks the CPU2 empty, The purpose is to verify that able to detect with CPU2 without SW.

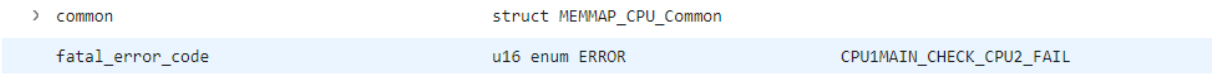
Procedure:

1. Run J-flash to erase CPU2 chip to empty via Jlink.
2. Start the unit.
3. It should enter fatal error to indicate CPU1MAIN_CHECK_CPU2_FAIL(1231)
4. Flash CPU2 with correct SW via Jlink, then the unit works normal again.

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>

Following the up procedure, use show debug to see data as below attached picture.



Module Test #17 - Fast IRQ follow control test.

Test Specification:

Number of units used	Set up with 16 Nodes works, and tested at node 5.
Config file	stressTest2.ssplog
Alternative used	1

Perform a check of the code that check the fast IRQ follow control. The purpose is to verify the following control at fast IRQ run correct. A real fast IRQ follow control error cannot be triggered, so the code has to be modified to “think” there is an error occurring.

Procedure:

1. Start the unit. Run “j2.py inject -i 1 -t s 11284 -r” via usb
2. The unit should enter fatal error CPU1COMMON_FL_CTRL_FTICK (11284) after about five seconds.

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>

The unit into fatal error and indicate CPU1COMMON_FL_CTRL_FTICK (11284) error about delay 5 seconds.

Module Test #18 – 1Khz IRQ follow control

Test Specification:

Number of units used	Set up with 16 Nodes works, and tested at node 5.
Config file	stressTest2.ssplog
Alternative used	1

Perform a check of the code that check the 1Khz IRQ follow control. The purpose is to verify the following control at 1Khz IRQ run correct. A real 1Khz IRQ follow control error cannot be triggered, so the code has to be modified to “think” there is an error occurring.

Procedure:

1. Start the unit. Run “j2.py inject -i 1 -t s 11286 -r” via usb
2. The unit should enter fatal error CPU1COMMON_FL_CTRL_1KHZ_IRQ (11286) after about five seconds.

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	

Compiler SW	
Config file GIT commit ID	
Result	<pass>

The unit into fatal error and indicate CPU1COMMON_FL_CTRL_1KHZ_IRQ (11286) error about delay 5 seconds.

Module Test #19 - Mainloop follow ctl test

Test Specification:

Number of units used	Set up with 16 Nodes works, and tested at node 5.
Config file	stressTest2.ssplog
Alternative used	1

Perform a check of the code that check the mainloop follow control. The purpose is to verify the following control at mainloop run correct. A real mainloop follow control error cannot be triggered, so the code has to be modified to “think” there is an error occurring.

Procedure:

1. Start the unit. Run “j2.py inject -i 1 -t s 11285 -r” via usb
2. The unit should enter fatal error CPU1COMMON_FL_CTRL_MAINLOOP (11285) after about five seconds.

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>

The unit into fatal error and indicate CPU1COMMON_FL_CTRL_MAINLOOP (11285) error about delay 5 seconds.

Module Test #20 – Board button press test

Test Specification:

Number of units used	Set up with 16 Nodes works, and tested at node 5.
Config file	stressTest2.ssplog
Alternative used	1

Perform a check of the code that check the mainloop follow control. The purpose is to verify the board button press correct. The code use TP4 to output 10ms high pulse when board button pressed.

Procedure:

1. Start the unit. Run “j2.py inject -i 9 -t s 11264 -r” via usb
2. The unit should still works normal.

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>

Module Test #21 – check that startup-check for memories work.

Test Specification:

Number of units used	Set up with 16 Nodes works, and tested at node 5.
Config file	stressTest2.ssplog
Alternative used	1

Perform a check of the code that checks the all ram memory works at start up. The purpose is to verify that all ram from 0x20000000 to 0x20018000 able to write and read out correct by word. A real RAM memory error cannot be triggered, so the code has to be modified to “think” there is an error occurring.
Procedure:

1. Start the unit. Run “j2.py inject -i 1 -t s 11324 -r” via usb
2. The unit shall enter fatal error CPU1COMMON_RAM_TEST_START_UP (11324), the extern LED board display the correspond error number, and the two board LEDs should quick blink orange.

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>

The unit can enter fata error CPU1COMMON_RAM_TEST_START_UP (1131), the extern LED board display the correspond error number, and the two board LEDs should quick blink orange

Module Test #22 - Continuous ram test

Test Specification:

Number of units used	Set up with 16 Nodes works, and tested at node 5.
Config file	stressTest2.ssplog
Alternative used	1

Perform a check of the code that checks the RAM memory. The purpose is to verify that this code will behave correctly if a RAM error is detected runtime. A real RAM memory error cannot be triggered, so the code has to be modified to “think” there is an error occurring.

Procedure:

1. Start the unit. Run “j2.py inject -i 1 -t s 11325 -r” via usb
2. The unit shall enter fatal error CPU1COMMON_RAM_TEST_RUNNING (11325), the extern LED board display the correspond error number, and the two board LEDs should quick blink orange.

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>

The unit can enter fata error CPU1COMMON_RAM_TEST_RUNNING (11326), the extern LED board display the correspond error number, and the two board LEDs should quick blink orange

Module Test #23 – Continue RAM check timeout test

Test Specification:

Number of units used	Set up with 16 Nodes works, and tested at node 5.
Config file	stressTest2.ssplog
Alternative used	1

Perform a check of the code that checks the RAM memory running check . The purpose is to verify that this code will behave correctly if a RAM running checking not run about 60 seconds. A real RAM memory error cannot be triggered, so the code has to be modified to “think” there is an error occurring.

Procedure:

1. Start the unit. Run “j2.py inject -i 1 -t s 11326 -r” via usb
2. The unit shall enter fatal error CPU1COMMON_RAM_TEST_TIMEOUT(11326), the extern LED board display the correspond error number, and the two board LEDs should quick blink orange.

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>

The unit can enter fata error CPU1COMMON_RAM_TEST_TIMEOUT(1133), the extern LED board display the correspond error number, and the two board LEDs should quick blink orange

Module Test #24 – Flash test during running.

Test Specification:

Number of units used	Set up with 16 Nodes works, and tested at node 5.
Config file	stressTest2.ssplog
Alternative used	1

Perform a check of the code that checks the flash memories integrity. The purpose is to verify that this code will behave correctly if the flash memory broken. A real flash memory broken is difficult to happend, so the code has to fake to miss one word of flash to calculate CRC to trigger fatal error.

Procedure:

1. Start the unit. Run “j2.py inject -i 1 -t s 11278 -r” via usb
2. The unit shall enter fatal error CPU1COMMON_FLASH_CRC_INVALID (11278), the extern LED board display the correspond error number, and the two board LEDs should quick blink orange.

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>

Following up procedure test, CPU1 able to enter fatal error CPU1COMMON_FLASH_CRC_INVALID (11278) after five seconds.

Module Test #25 - Fast irq between time limit test

Test Specification:

Number of units used	Set up with 16 Nodes works, and tested at node 5.
Config file	stressTest2.ssplog
Alternative used	1

Perform a check of the code that checks the max between two fast IRQ time limit . The purpose is to verify that this code will behave correctly if one of fast IRQ between time over than limit. A real fast IRQ between time cannot be over than the limit 75us, so the code has to be modified fast IRQ period to 77.5us to trigger the error occurring after five seconds.

Procedure:

1. Start the unit. Run “j2.py inject -i 1 -t s 11465 -r” via usb
2. The unit shall enter fatal error CPU1COMMON_FTICK_BETWEEN_TIME (11465) after five seconds, the extern LED board display the correspond error number, and the two board LEDs should quick blink orange.

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>

The unit into fatal error and indicate IRQ_FAST_BETWEEN_TIME (11465) error after five seconds. Attached picture as below:

```
› common                          struct MEMMAP_CPU_Common
  fatal_error_code                u16 enum ERROR                CPU1COMMON_FTICK_BETWEEN_TIME
```

Module Test #26 - Fast irq average time limit test

Test Specification:

Number of units used	Set up with 16 Nodes works, and tested at node 5.
Config file	stressTest2.ssplog
Alternative used	1

Perform a check of the code that checks the fast IRQ average time limit. The purpose is to verify that this code will behave correctly if fast IRQ average time over than limit. A real fast IRQ between time cannot be over than the limit 65us, so the code has to be modified fast IRQ period to 67.5us to trigger the error occurring after five seconds.

Procedure:

1. Start the unit. Run “j2.py inject -i 1 -t s 11466 -r” via usb
2. The unit shall enter fatal error CPU1COMMON_FTICK_AVG_TIME (11466) after five seconds, the extern LED board display the correspond error number, and the two board LEDs should quick blink orange.

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>

The unit into fatal error and indicate CPU1COMMON_FTICK_AVG_TIME (11466) error after five seconds. And the value irqTiming.avgTimeBetweenFastIrqs is 67.5us.

Module Test #27 - Fast irq run for time test

Test Specification:

Number of units used	Set up with 16 Nodes works, and tested at node 5.
Config file	stressTest2.ssplog
Alternative used	1

Perform a check of the code that checks the fast IRQ for time limit. The purpose is to verify that this code will behave correctly if fast IRQ for time over than limit. A real fast IRQ for time cannot be over than the limit 62us, so the code has to be inject a delay about 55us to trigger the error occurring after five seconds.

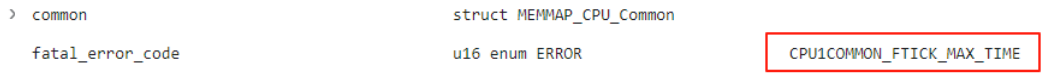
Procedure:

1. Start the unit. Run “j2.py inject -i 1 -t s 11464 -r” via usb
2. The unit shall enter fatal error CPU1COMMON_FTICK_MAX_TIME (11464) after five seconds, the extern LED board display the correspond error number, and the two board LEDs should quick blink orange.

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>

The unit into fatal error and indicate CPU1COMMON_FTICK_MAX_TIME (11464) error after five seconds.



Module Test #28 - 1KHZ irq between time limit test

Test Specification:

Number of units used	Set up with 16 Nodes works, and tested at node 5.
Config file	stressTest2.ssplog
Alternative used	1

Perform a check of the code that checks the max between two 1Khz IRQ time limit . The purpose is to verify that this code will behave correctly if one of 1Khz IRQ between time over than limit. A real 1Khz IRQ between time cannot be over than the limit 1340 us, so the code has to be modified 1Khz IRQ period to trigger the error occurring after five seconds.

Procedure:

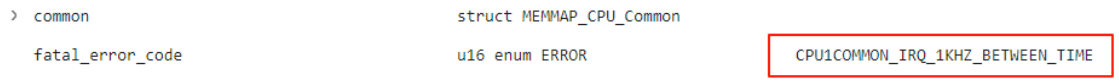
1. Start the unit. Run “j2.py inject -i 1 -t s 11475 -r” via usb

2. The unit shall enter fatal error CPU1COMMON_IRQ_1KHZ_BETWEEN_TIME (11475) after five seconds, the extern LED board display the correspond error number, and the two board LEDs should quick blink orange.

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>

The unit into fatal error and indicate CPU1COMMON_IRQ_1KHZ_BETWEEN_TIME (11475) error after five seconds. And read the value of irqTiming.maxTimeBetweenTwo1KhzIRQ is 1375 us bigger than 1340us.attached picture as below:



Module Test #29 - 1KHZ irq average time limit test

Test Specification:

Number of units used	Set up with 16 Nodes works, and tested at node 5.
Config file	stressTest2.ssplog
Alternative used	1

Perform a check of the code that checks the 1Khz IRQ average time limit. The purpose is to verify that this code will behave correctly if 1Khz IRQ average time over than limit. A real 1Khz IRQ between time cannot be over than the limit 1050us, so the code has to be modified 1050 IRQ period to 1062.5us to trigger the error occurring after five seconds.

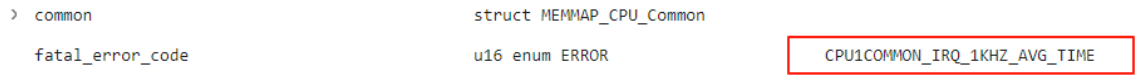
Procedure:

1. Start the unit. Run “j2.py inject -i 1 -t s 11476 -r” via usb
2. The unit shall enter fatal error CPU1COMMON_IRQ_1KHZ_AVG_TIME (11476) after five seconds, the extern LED board display the correspond error number, and the two board LEDs should quick blink orange.

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>

The unit into fatal error and indicate CPU1COMMON_IRQ_1KHZ_AVG_TIME (11476) error after five seconds. And use read the irqTiming.avgTimeBetween1KhzIRQ is 1062.5 bigger than 1050 or less than 996. Attached picture as below:



Module Test #30 - 1KHZ irq run for time limit test

Test Specification:

Number of units used	Set up with 16 Nodes works, and tested at node 5.
Config file	stressTest2.ssplog
Alternative used	1

Perform a check of the code that checks the fast 1Khz for time limit. The purpose is to verify that this code will behave correctly if 1Khz IRQ for time over than limit. A real 1Khz IRQ for time cannot be over than the limit 400us, so the code has to be inject a delay about 300us to trigger the error occurring after five seconds.

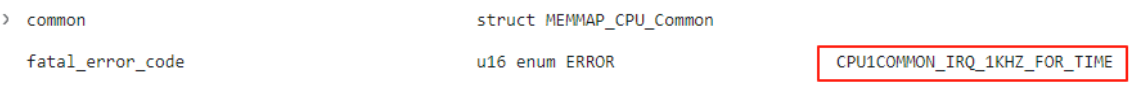
Procedure:

1. Start the unit. Run “j2.py inject -i 1 -t s 11474 -r” via usb
2. The unit shall enter fatal error CPU1COMMON_IRQ_1KHZ_FOR_TIME (11474) after five seconds, the extern LED board display the correspond error number, and the two board LEDs should quick blink orange.

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>

The unit into fatal error and indicate CPU1COMMON_IRQ_1KHZ_FOR_TIME (11474) error after five seconds. Attached picture as below:



Module Test #31 - Main loop between time limit test

Test Specification:

Number of units used	Set up with 16 Nodes works, and tested at node 5.
Config file	stressTest2.ssplog
Alternative used	1

Perform a check of the code that checks the max between time limit. The purpose is to verify that this code will behave correctly if one of main loop between time over than limit. A real mainloop between time cannot be over than the limit 30ms, so the code has to be insert a 30ms delay to trigger the error occurring after five seconds.

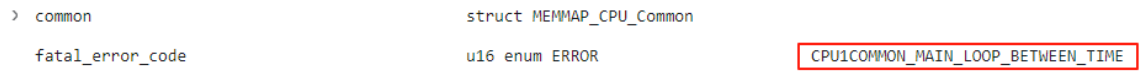
Procedure:

1. Start the unit. Run “j2.py inject -i 1 -t s 11484 -r” via usb
2. The unit shall enter fatal error CPU1COMMON_MAIN_LOOP_BETWEEN_TIME (11484) after five seconds, the extern LED board display the correspond error number, and the two board LEDs should quick blink orange.

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>

unit into fatal error and indicate CPU1COMMON_MAIN_LOOP_BETWEEN_TIME (11484) error after five seconds. Attached picture as below:



Module Test #32 – Main loop average time limit test

Test Specification:

Number of units used	Set up with 16 Nodes works, and tested at node 5.
Config file	stressTest2.ssplog
Alternative used	1

Perform a check of the code that checks the main loop average time limit. The purpose is to verify that this code will behave correctly if main loop average time over than limit. A real main loop average time cannot be over than the limit 1000us, so the code has to be inject a delay about 600us in the main loop to trigger the error occurring.

Procedure:

1. Start the unit. Run “j2.py inject -i 1 -t s 11486 -r” via usb
2. The unit shall enter fatal error CPU1COMMON_MAIN_LOOP_AVG_TIME (11486) about one second delay later, the extern LED board display the correspond error number, and the two board LEDs should quick blink orange.

Test Execution:

Date	
Target SW CPU1	

Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>

The unit into fatal error and indicate CPU1COMMON_MAIN_LOOP_AVG_TIME (11486).

Module Test #33 - Main loop run for time limit test

Test Specification:

Number of units used	Set up with 16 Nodes works, and tested at node 5.
Config file	stressTest2.ssplog
Alternative used	1

Perform a check of the code that checks the main loop for time limit. The purpose is to verify that this code will behave correctly if main loop for time over than limit. A real main loop for time cannot be over than the limit 5000us, so the code has to be inject a delay about 5000us to trigger the error occurring after five seconds.

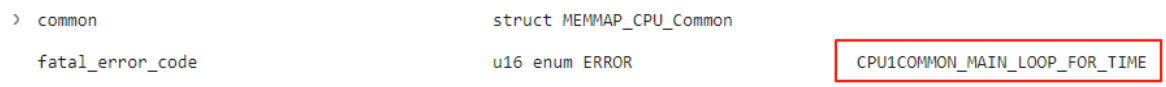
Procedure:

1. Start the unit. Run “j2.py inject -i 1 -t s 11485 -r” via usb
2. The unit shall enter fatal error CPU1COMMON_MAIN_LOOP_FOR_TIME (11485) after five seconds, the extern LED board display the correspond error number, and the two board LEDs should quick blink orange.

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>

The unit into fatal error and indicate CPU1COMMON_MAIN_LOOP_FOR_TIME (11485) error after five seconds. Attached picture as below:



Module Test #34 – SIOs analogue compare test

Test Specification:

Number of units used	1
Config file	stressTest2.ssplog
Alternative used	Alternative 9 for test output, and 10 for test input.

Perform a check of the code that checks SIO analogue compare status. The purpose is to verify that this code will behave correctly check with compare the SIO analogue value, and able to trigger fatal error if each SIO AD mismatch. The SIO analogue at both CPU compare should satisfy all of below condition, if one of then fail to meet the conditions, there is should be an error occurred. The Max difference diff time is 200ms.

• At output, CPU1 SIO voltage should be in [0.75*powerSuply : powerSupply], if not record as error type 1
• At output, CPU1 SIO voltage should be less than CPU2
For PCB018J: The min difference between CPU2 SIO voltage to CPU1 shall be in range[0.105780886-0.04,0.219250099+0.04]*SIOpower. If less than min, it shall record error as error type 2. If over than max, it shall record error as error typ 6.
For PCB018G: SIO voltage of diff= CPU2 -CPU1 shall be: If SIOpower over than 17V: [0.005,0.21698]; If SIOpower less than 17V: [0.004,0.13644]

• At output, the max difference of CPU2 SIO voltage between SIO power should be:
For PCB018J: less than 0.0891*CPU1 SIOpower
For PCB018G: less than 0.75V when SIO power voltage less than 14V, and less than 1/18 of SIOpower voltage when over than 14V. If not record as error type 3.
• At input, the max difference of SIO CPU1 and CPU2 should be:
For PCB018J: less than 0.0891*CPU1 SIOpower
For PCB18G: less than 0.75V when SIO power voltage less than 14V, and less than 1/18 of SIOpower voltage when over than 14V. If the max voltage not correct, record as error type 4, else if the min voltage not correct, record error type as 5.

Note that we just need check the code able to detect the up difference error type and able to give out fatal error if still fail more than 200ms here, don’t need to check all SIO HW. Please refer to the HW module test for the detail SIO test if need. So just test with SIO 2 here.

Procedure:

1. Test with 24V. And use the alternative 9 for test. Note that for output. the real ad detected voltage should be 24/(64.9+(3.24*16/(3.24+16)))*(3.24*16/(3.24+16))*(64.9+3.24)/3.24= 20.1195V.
2. Adjust R213 to 2.2K, when turn on SIO, then read out the analogureErrorLog data we should able to see it record the error type 1 and give out fatal error SIO_ANALOGURE_MISMATCH(8). If we adjust to 2.2K: 24/(64.9+(2.2*16/(2.2+16)))*(2.2*16/(2.2+16))*(64.9+3.24)/3.24 = 14.6063 less than 24V.
3. Adjust R213 to 4.7K, when turn on SIO, then read out the analogureErrorLog data we should able to see it record the error type 3 and give out fatal error SIO_ANALOGURE_MISMATCH(8). If we adjust to 4.7K: 24/(64.9+(4.7*16/(4.7+16)))*(4.7*16/(4.7+16))*(64.9+3.24)/3.24 = 26.7557 bigger than 24V.
4. Test with 24V, and use the alternative 9 for test.
5. Adjust R208 to 560R or 470R, and connect 24V input to SIO1, then read out analogureErrorLog data we should able to see it record the error type 4 or 5 and give out fatal error.

560R: | ((24/(16+0.56)*0.56)/0.51*(16+0.51)-24)/24 | = 0.0947 greater than 0.0891
470R: | ((24/(16+0.47)*0.47)/0.51*(16+0.51)-24)/24 | = 0.07619 less than 0.0891

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>

Module Test #35 - SIO current test

Test Specification:

Number of units used	1
Config file	stressTest2.ssplog
Alternative used	1

Perform a check of the code that checks SIO current detect status. The purpose is to verify that this code will behave correctly check with SIO current value, and able to trigger fatal error if the current over load. There are have tow ways to limit the current, one is the SW current sampling by AD, the general formula use as I**2*t for limit the energy, the other one the hardware limit the max current is 8A. Note that for able to take the capacitance load, when the over current be trigged by the HW, there's have a 1us delay to keep the SIO power, and also have 200 times IRQ allow for able to charge to the capacitance load.

Procedure:

1. Check the over current with AD should able to be turn off with difference load. Note that the formula of the current works as limit the energe as (mmDebug.sioStatus.cpu1SIOCurrentMa-settings.sioSettings→currentMax[2])**2 * t. Here we just test with current 2.4A,4A,6A for check the time length to turn off the SIO power.
2. Connect to a bigger load to able to trig the HW over current works for test. It should have 200 times over current IRQ be trigged, we can see it at SIO power control PIN status to know the times.

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>

Follow the up test. The time length of current during the SIO turn off is satisfy to I**2 * t, and also trigger fatal error SIO_OVER_CURRENT_ADC(22). And the HW over current able to trigger, and also

have 200 times delay to trigger fatal error over current SIO_OVER_CURRENT_IRQ(10). Attached process and pictures as below, if you need check more detail, please look the HW module test.

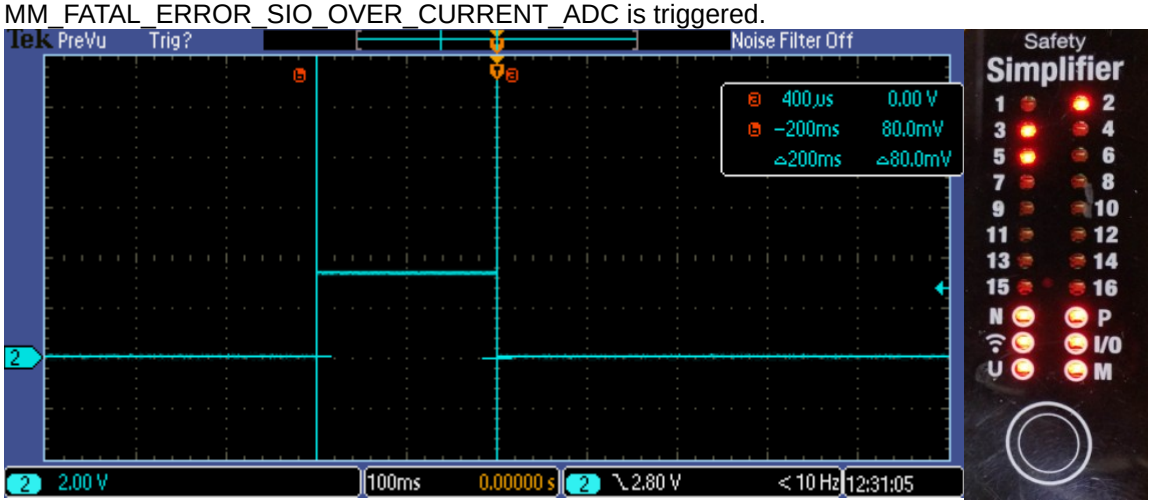
To record the time length of current duration when current exceed the preset value.
Here we set currentMax[2] as 2400mA and enableOverTimeMaxMs as 200ms.

a) When SIO1, SIO2, SIO4, SIO8 and SIO9 use 20ohm/10W resistor as load and all SIOs are on.
Adjust input voltage to let current to be near 2400mA.
The following picture shows current decreases down to 38mA which means protection is activated.

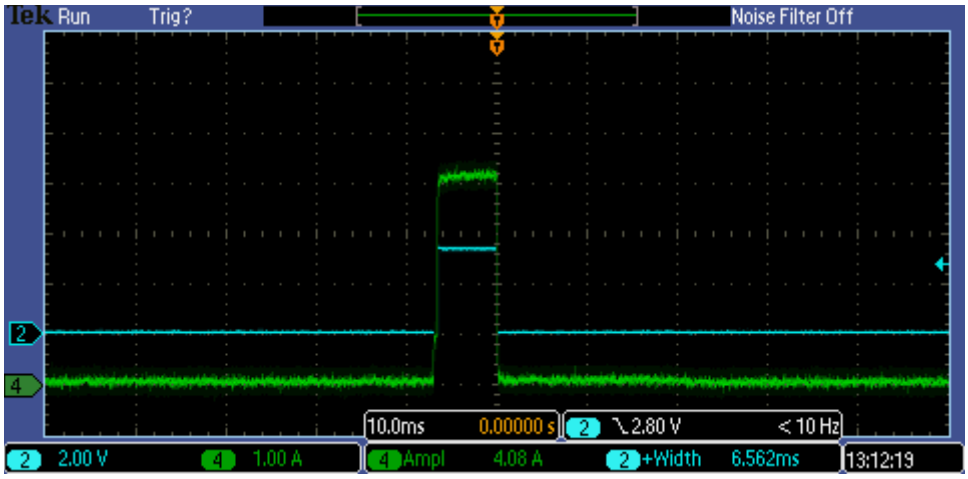
```
D:sioStatus.sioCurrentDbgBuffer[27]: 2400          ( 2400 0x0960)
D:sioStatus.sioCurrentDbgBuffer[28]: 2397          ( 2397 0x095D)
D:sioStatus.sioCurrentDbgBuffer[29]: 2397          ( 2397 0x095D)
D:sioStatus.sioCurrentDbgBuffer[30]: 2398          ( 2398 0x095E)
D:sioStatus.sioCurrentDbgBuffer[31]: 2400          ( 2400 0x0960)
D:sioStatus.sioCurrentDbgBuffer[32]: 2398          ( 2398 0x095E)
D:sioStatus.sioCurrentDbgBuffer[33]: 2398          ( 2398 0x095E)
D:sioStatus.sioCurrentDbgBuffer[34]: 2398          ( 2398 0x095E)
D:sioStatus.sioCurrentDbgBuffer[35]: 2398          ( 2398 0x095E)
```

```
D:sioStatus.sioCurrentDbgBuffer[56]: 2374          ( 2374 0x0946)
D:sioStatus.sioCurrentDbgBuffer[57]: 2374          ( 2374 0x0946)
D:sioStatus.sioCurrentDbgBuffer[58]: 2372          ( 2372 0x0944)
D:sioStatus.sioCurrentDbgBuffer[59]: 2374          ( 2374 0x0946)
D:sioStatus.sioCurrentDbgBuffer[60]: 38            (   38 0x0026)
D:sioStatus.sioCurrentDbgBuffer[61]: 37            (   37 0x0025)
```

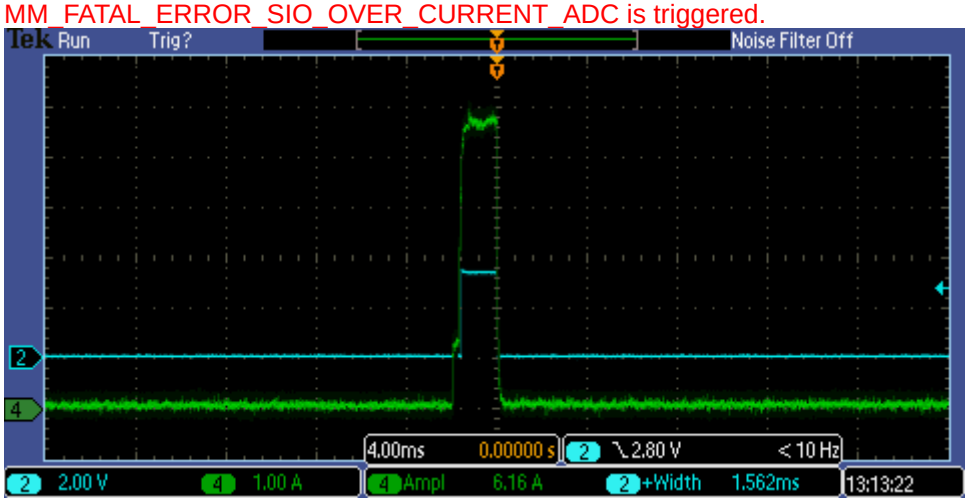
The time length when current is near preset over-current value is about 200ms.
MM_FATAL_ERROR_SIO_OVER_CURRENT_ADC is triggered.



b) Current at 4.08A. Protection is activated after duration of 4.08A beyond 6.562ms.
MM_FATAL_ERROR_SIO_OVER_CURRENT_ADC is triggered.



c) Current at 6.16A. Protection is activated after duration of 6.16A beyond 1.562ms. MM_FATAL_ERROR_SIO_OVER_CURRENT_ADC is triggered.



Module Test #36 – Production Data check

Test Specification:

Number of units used	1
Config file	stressTest2.ssplog
Alternative used	1

Perform a check of the code that checks the production data. The purpose is to verify that this code will behave correctly check with the production data if blank, or invalid id number. For difference HW should have difference HwVersion, PCB018H02=>13, PCB018H04=>14, PCB018K=>25 or later.

Procedure:

1. Production data blank check. Run J-flash to reflash CPU1 chip to correct SW via Jlink.
2. Start the unit. It should enter small bootloader.
3. Invalid id Number 0 check. Write the invalid serialNumber 0. Then restart the unit, the system should enter fatal error CPU1COMMON_MY_ID_MISMATCH (11298).
4. Follow point1,2 again.
5. Invalid id Number 0xffffffff check. Write the invalid serialNumber 0xffffffff. Then restart the unit, the system should enter fatal error CPU1COMMON_MY_ID_MISMATCH (11298).

6. Follow point1,2 again.
7. Write correct production data. Use Write the correct production data. Then the system works normal.
8. Invalid HW version check. Write with the unexpected hwVersion=2 Then restart the unit, the system should enter fatal error CPU1COMMON_BAD_HW_VERSION (11300).

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>

Module Test #37 – Logic valid check

Test Specification:

Number of units used	1
Config file	stressTest2.ssplog
Alternative used	None

Perform a check of the code that check without logic, invalid logic CRC and others kind parameters invalid should be detect. The purpose is to verify that this code will behave correctly if logic not exist, logic CRC invalid or other parameter not correct. The code check the logic header CRC, flash CRC, flash size, headerVersion, magicVal, ramDataAddress, slotTime, mainLoopFun valid, init valid, compilerGeneration and init() to init logic successful. A real logic invalid is difficult to trigger an logic error, so the SW has to inject a fake to trigger error.

Procedure:

1. **Header CRC check.** Start the unit. Run “j2.py sfe -e 21507 -s 1 -r” via usb, the unit should eneter fatal error CPU1LOGIC_HEADER_BAD_CRC (21507). Note that the code revert (~(LOGIC_HEADER.crc)) to campare with the code running calculated.
2. **Flash CRC check.** Start the unit. Run “j2.py sfe -e 21509 -s 1 -r” via usb, the unit should eneter fatal error CPU1LOGIC_BAD_CODE_CRC (21509).Note that the code revert ~LOGIC_HEADER.flashCRC to campare with the code running calculated.
3. **Flash size check.**Start the unit. Run “j2.py sfe -e 21510 -s 1 -r” via usb, the unit should eneter fatal error CPU1LOGIC_BAD_FLASH_SIZE (21510). Note that the code revert flashSize != ~0x4000 to campare with the header flash size.
4. **headerVersion check.** Start the unit. Run “j2.py sfe -e 21508 -s 1 -r” via usb, the unit should eneter fatal error CPU1LOGIC_HEADER_BAD_VERSION (21508).Note that the code revert ”(__LOGIC_HEADER__.common_header.header_version != ~MM_CONST_LOGIC_HEADER_VERSION)” to campare with the headerVersion.
5. **magicVal check.** Start the unit. Run “j2.py sfe -e 21506 -s 1 -r” via usb, the unit should eneter fatal error CPU1LOGIC_HEADER_BAD_MAGIC (21506). Note that the code revert

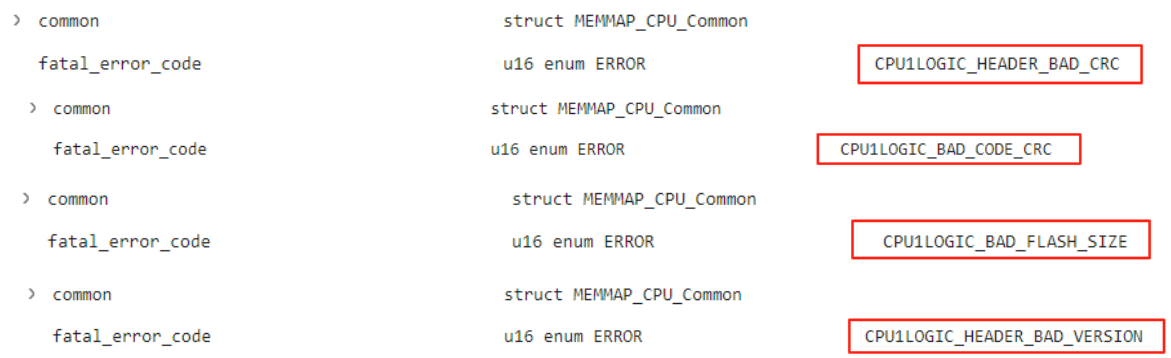
”(__LOGIC_HEADER__.common_header.magic_val != ~MM_CONST_MAGIC_MAIN_VAL)” to campare with the magicVal.

6. **ramDataAddress check.** Start the unit. Run “j2.py sfe -e 21515 -s 1 -r” via usb, the unit should eneter fatal error CPU1LOGIC_BAD_RAM_DATA_ADDRESS (21515). Note that the code revert ” __LOGIC_HEADER__.common_header.ram_data_address != ~__ram_logic_origin” to campare with the ramDataAddress.
7. **slotTime check.** Start the unit. Run “j2.py sfe -e 21516 -s 1 -r” via usb, the unit should eneter fatal error CPU1LOGIC_BAD_SLOT_TIME (21516). Note that the code use ”(__LOGIC_HEADER__.cfg_params.slottime_ms >= 1)” to campare with the slotTime.
8. **mainLoopFunc valid check.** Start the unit. Run “j2.py sfe -e 21518 -s 1 -r” via usb, the unit should eneter fatal error CPU1LOGIC_NO_MAINLOOP_FUNCTION (21518). Note that the code use ”(LOGIC_HEADER.common_header. init_main_loop != NULL)” to fake init_main_loop valid error.
9. **init valid check.** Start the unit. Run “j2.py sfe -e 21517 -s 1 -r” via usb, the unit should eneter fatal error CPU1LOGIC_NO_INIT_FUNCTION (21517). Note that the code use ”(LOGIC_HEADER.common_header.init !=NULL)” to fake init valid error.
10. **compilerGeneration check.** Start the unit. Run “j2.py sfe -e 21519 -s 1 -r” via usb, the unit should eneter fatal error CPU1LOGIC_BAD_COMPILER_GENERATION (21519). Note that the code use ”(LOGIC_HEADER.common_header.compiler_generation == 0)” to compare with the revert compilerGeneration value.
11. **Init() inited successful check.** Start the unit. Run “j2.py sfe -e 21511 -s 1 -r” via usb, the unit should eneter fatal error CPU1LOGIC_INIT_FUNCTION_FAIL (21511). Note that the code use ”(LOGIC_HEADER.common_header.init() == 0)” to fake fatal error if init the logic successful.

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>

Following the up test. The unit able to enter CONFIGURE_ERROR1~12 to indicate difference fatal error. Attached pictures as below.



Module Test #38 – Settings serial number test

Test Specification:

Number of units used	1
Config file	stressTest2.ssplog
Alternative used	1

Perform a check of the code that checks the settings serial number. The purpose is to verify that this code will behave correctly check with verify the settings serial number. It should be able to verify the network ID and itself serial number.

Note that we test with node 5 for itself serial number is 6049.

Procedure:

1. Start the unit, Use “j2.py sread -f test.set” to read out the settings data and save at test.set
2. Modify the node 0 serial Number to 0,modified as “serial_numbers = 0, 6205, 6206, 6207, 6049, 6208, 6209, 6210, 6211, 6212, 6213, 6214, 6215, 6061, 6050, 6064”
3. Use ” j2.py swrite test.set” to write the settings data. Then the unit should enter fatal error and indicate fatal error CPU1COMMON_NETWORK_ID_INVALID (11297).
4. Modify the node 5 serial Number to unexpected number, modified as “serial_numbers = 6204, 6205, 6206, 6207, 1234, 6208, 6209, 6210, 6211, 6212, 6213, 6214, 6215, 6061, 6050, 6064”
5. Follow point 3 again. Then the unit should enter fatal error CPU1COMMON_MY_ID_MISMATCH (11298).
6. Modify the node 5 serial Number to unexpected number, modified as “serial_numbers = 6204, 6205, 6206, 6207, 0, 6208, 6209, 6210, 6211, 6212, 6213, 6214, 6215, 6061, 6050, 6064”
7. Follow point 3 again. Then the unit should enter fatal error .CPU1COMMON_MY_ID_INVALID (11296).
8. Follow point 3 again. Then the unit should able to write the correct serial number to the settings data and restart the unit. Then the system works normal again. Follow point 1 to

readout the settings data, we can see the serialNumbers[4] have be write to the correct serial number 6049.

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>

Follow up the up test, the unit able to indicate fatal error at CPU1COMMON_NETWORK_ID_INVALID (11297) , CPU1COMMON_MY_ID_MISMATCH (11298) and CPU1COMMON_MY_ID_INVALID (11296), and also able to return normal if serial numbers set as 0. Attached pictures as below.

```
› common                    struct MEMMAP_CPU_Common
  fatal_error_code          u16 enum ERROR          CPU1COMMON_MY_ID_INVALID
› common                    struct MEMMAP_CPU_Common
  fatal_error_code          u16 enum ERROR          CPU1COMMON_MY_ID_MISMATCH
› common                    struct MEMMAP_CPU_Common
  fatal_error_code          u16 enum ERROR          CPU1COMMON_NETWORK_ID_INVALID
```

Module Test #39 – RADIO TX packet not send to CPU2 test

Test Specification:

Number of units used	node1
Config file	stressTest2.ssplog
Alternative used	8

Perform a check of the code that checks the radio memory. The purpose is to verify that this code will behave correctly check with verify radio memory safety check at both CPU.

Note that for stressTest2.ssplog alternative 8 node 1 at mike branch. Set the mem2 as safety and to control the SIO output with square off=3000ms,on=3000ms

Procedure:

1. Use “sspdl.py cfgdl -a8 tests\stressTest2.ssplog.zip -n1 -c1/2” to flash CPU1/2 CFGs SW.
2. In Sconstruct, enable the SAFETY_TEST_INJECT = 0x40 test. Recompile and flash CPU1 via USB. Note that CPU1 should not send radio TX packet to CPU2 after five seconds.
3. Start the unit.
4. The unit should works normal first, and change menu to IO, the led should indicate the SIO13 output normal at first five seconds.
5. The unit shall enter CPU2 fatal error MEMORY_MISMATCH (141) about six seconds delay when SIO output high few, the extern LED board display the correspond error number, and the two board LEDs should quick blink orange.

Test Execution:

Date	

Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>

Module Test #40 – RADIO RX packet not send to CPU2 test

Test Specification:

Number of units used	16
Config file	stressTest2.ssplog
Alternative used	1

Perform a check of the code that checks the radio memory. The purpose is to verify that this code will behave correctly check with verify radio memory safety check at both CPU.

Note that for stressTest2.ssplog alternative 1 node 5 at mike branch, it set the node[1].mem2 as safety and to control the SIO output with square off=2000ms,on=2000ms

Procedure:

1. Use “sspdl.py cfgdl -a1 tests\stressTest2.ssplog.zip -n5 -c1/2” to flash CPU1/2 CFGs SW.
2. In Sconstruct, enable the SAFETY_TEST_INJECT = 0x41 test. Recompile and flash CPU1 via USB. Note that CPU1 should not send radio RX packet to CPU2 after five seconds.
3. Start the unit.
4. The unit should works normal first, and change menu to IO, the led should indicate the SIO2~14 output normal at first five seconds.
5. The unit shall enter CPU2 fatal error OUTPUT_PRE_SIGNAL_DIFF (146) about six seconds delay when SIO output high few, the extern LED board display the correspond error number, and the two board LEDs should quick blink orange.

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>

Module Test #41 – RADIO both TX and RX packet not send to CPU2 test

Test Specification:

Number of units used	16 nodes and tested with node1 and node5.

Config file	stressTest2.ssplog
Alternative used	1

Perform a check of the code that checks the radio memory. The purpose is to verify that this code will behave correctly check with verify radio memory safety check at both CPU.

Procedure:

1. Use “sspdl.py cfgdl -a1 tests\stressTest2.ssplog.zip -n5/1 -c1/2” to flash CPU1/2 CFGs SW.
2. In Sconstruct, enable the SAFETY_TEST_INJECT = 0x42 test. Recompile and flash CPU1 via USB at both two nodes.
3. Start both units.
4. The units should works normal first, and change menu to IO, the led should indicate the SIO2~14 output normal at first five seconds.
5. The unit shall enter CPU2 fatal error OUTPUT_PRE_SIGNAL_DIFF (146) at node5 and MEMORY_MISMATCH (141) at node 1 about six seconds delay when SIO output high, the extern LED board display the correspond error number, and the two board LEDs should quick blink orange.

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>

Remove Module Test #42 – RADIO packets not send to CPU2 with short time test

Test Specification:

Number of units used	2
Config file	stressTest2.ssplog
Alternative used	2

Note: The current C2C length is determined by the structure, which carries RADIO data. If no radio data is sent, it is similar to the test above.

Perform a check of the code that checks the radio memory. The purpose is to verify that this code will behave correctly check with verify radio memory safety check at both CPU when few radio packets lost.

Note that for the alternative 2, node 1 SIO2~14 be controled by the node[2].mem2, and node2 SIO2`14 be controlled by node[1].mem2. And both mem2 square as on=200ms, off=200ms.

Procedure:

1. In Sconstruct, enable SAFETY_TEST_INJECT = 0x43 test. Recompile and flash CPU1 via USB.

2. Start the unit. Note that at the first five seconds, CPU1 not lost radio packet to CPU2, after it CPU1 should not send radio packets to CPU1 about 10ms with per 60ms. Run “showDebug.py --driver pyserial --view Fake” the ram safetyFake.test1 and safetyFake.test2 should keep 0 at the first five seconds, after that they’re increase slowly to indicate the numbers of missed send packets to CPU2. And we also can measured the hige pulse length of TP4 which indicate is CPU1 not send radio packet to CPU2.
3. The unit should still works normal.

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	

Follow up the up test, the unit still works normal. We can see that at the first five second, both ram safetyFake.test1 and test2 keep 0, and then they increase. Measured the TP4 pulse with oscilloscope, the high width=10ms,low=50ms.

Remove Module Test #43 – RADIO packets not set to CPU2 with long time test

Test Specification:

Number of units used	2
Config file	stressTest2.ssplog
Alternative used	2

Note: The current C2C length is determined by the structure, which carries RADIO data. If no radio data is sent, it is similar to the test above.

Perform a check of the code that checks the radio memory. The purpose is to verify that this code will behave correctly check with verify radio memory safety check at both CPU when many radio packets lost to send to CPU2.

Note that for the alternative 2, node 1 SIO2~14 be controled by the node[2].mem2, and node2 SIO2`14 be controlled by node[1].mem2. And both mem2 square as on=200ms, off=200ms.

Procedure:

1. In Sconstruct, enable SAFETY_TEST_INJECT = 0x44 test. Recompile and flash CPU1 via USB.
2. Start the unit. Note that at the first five seconds, CPU1 not lost radio packet to CPU2, after it CPU1 should not send radio packets to CPU1 about 100ms with per 150ms. Run “showDebug.py --driver pyserial --view Fake” the ram safetyFake.test1 and safetyFake.test2 should keep 0 at the first five seconds, after that they’re increase slowly to indicate the numbers of missed send packets to CPU2. And we also can measured the hige pulse length of TP4 which indicate is CPU1 not send radio packet to CPU2.
3. The unit should works normal at the first five seconds, and then enter fatal error MEMORY_MISMATCH (141).

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	

Follow up the up test, the unit still works normal at the first five seconds, and then enter CPU2 fatal error MEMORY_MISMATCH (141). We can see that at the first five second, both ram safetyFake.test1 and test2 keep 0, and then they increase quickly. Measured the TP4 pulse with oscilloscope, the high width=100ms, low=50ms.

Module Test #44 – RADIO packets send to CPU2 with wrong datas

Test Specification:

Number of units used	2
Config file	stressTest2.ssplog
Alternative used	2

Perform a check of the code that checks the radio memory. The purpose is to verify that this code will behave correctly check with verify radio data safety check at both CPU when CPU1 send with wrong datas to CPU2.

Note that for the alternative 2, node 1 SIO2~14 be controled by the node[2].mem2, and node2 SIO2`14 be controlled by node[1].mem2. And both mem2 square as on=200ms, off=200ms.

Procedure:

1. In Sconstruct, enable SAFETY_TEST_INJECT = 0x45 test. Recompile and flash CPU1 via USB.
2. Start the unit. CPU2 should enter fatal error.

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>

Follow up the up test, CPU2 enter fatal error OUTPUT_PRE_SIGNAL_DIFF (146).

Module Test #45 – RADIO packets CRC including check.

Test Specification:

Number of units used	2
Config file	stressTest2.ssplog
Alternative used	1

Perform a check of the code that checks radio packet CRC seed with radio channel, Sws version, static full data and radio data. The purpose is to verify that radio should not link up if send out radio crc without seed with one of up data..

Note that test with node1 and node5, and set both node to disable CAN.

Procedure:

1. **Seed CRC without Sws version test.** Start the unit. Run “j2.py sfe -e 11264 -s 20 -r” for node 1 CPU1 via USB. The radio should not link up between node 1 and node5.
2. **Seed CRC without radio channel.** Start the unit. Run “j2.py sfe -e 11264 -s 13 -r” for node 1 CPU1 via USB. The radio should not link up between node 1 and node 5.
3. **Seed CRC without simple data(radio raw packet).** Start the unit. Run “j2.py sfe -e 11264 -s 14 -r” for node 1 CPU1 via USB. The radio should not link up between node 1 and node 5.
4. **Seed CRC without full data(radio extra data).** Start the unit. Run “j2.py sfe -e 11264 -s 21 -r” for node 1 CPU1 via USB. The radio should not link up between node 1 and node 5.
5. Restart the unit. The radio should link up between node 1 and node 5.

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>

Module Test #46 – RADIO packets send to CPU2 with wrong datas

Test Specification:

Number of units used	2
Config file	stressTest2.ssplog
Alternative used	2

Perform a check of the code that checks the radio memory. The purpose is to verify that this code will behave correctly check with verify radio data safety check at both CPU when CPU1 send with wrong datas to CPU2.

Note that for the alternative 2, node 1 SIO2~14 be controled by the node[2].mem2, and node2 SIO2`14 be controlled by node[1].mem2. And both mem2 square as on=200ms, off=200ms.

Procedure:

1. In Sconstruct, enable SAFETY_TEST_INJECT = 0x4A test. Recompile and flash CPU1 via USB.

2. Start the unit. CPU2 should enter fatal error.

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>

Follow up the up test, CPU2 enter fatal error OUTPUT_PRE_SIGNAL_DIFF (146). Attached picture as below.

Module Test #47 – CPU1 REL clock lost detect test

Test Specification:

Number of units used	node5
Config file	stressTest2.ssplog
Alternative used	None

Perform a check of the code that checks the CPU1 REL clock. The purpose is to verify that this code will behave correctly check with verify rel clock lost should be detected.

Procedure:

1. Start the unit. Run “j2.py inject -i 1 -t s 11322 -r” via usb.
2. The unit shall enter fatal error CPU1COMMON_RELAY_STATUS_MISMATCH (11322).

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>

Module Test #48 – check that timeout from radio

Test Specification:

Number of units used	3
Config file	stressTest2.ssplog
Alternative used	7

Perform a check of the code that checks the timeout from radio and CAN. The purpose is to verify that this code will behave correctly check with six timeout of link_timeout_long, link_timeout_short, can_timeout_long, can_timeout_short, link_direct_timeout_long, link_direct_timeout_short.

Note that for the alternative 7 node3 configure as below:

```
  dummy1 = link_timeout_long(node = const 1)
dummy2 = link_timeout_short(node = const 1)
dummy3 = can_timeout_long(node = const 1)
dummy4 = can_timeout_short(node = const 1)
dummy5 = link_direct_timeout_long(node = const 1)
dummy6 = link_direct_timeout_short(node = const 1)

#out1 =  s1s
out2 =  dummy1
out4 =  dummy2
out6 =  dummy3
out8 =  dummy4
out10 = dummy5
out12 = dummy6

system:
  slot_time = 32          # 32*62.5 us = 2ms
  can_interval = 64       # 64*62.5 us = 4ms
  can_baudrate = 250      # 250 kbit/s
  ossd_low = 1            # 1*62.5 us
  radio_timeout_short = 10    #   10*num_nodes*(slot_time*62.5us). Also used for CAN timeouts.
  radio_timeout_long = 50    #   50*num_nodes*(slot_time*62.5us). Also used for CAN timeouts.
```

Procedure:

1. In Sconstruct, enable SAFETY_TEST_INJECT = 0x60 test. Recompile and flash node1 CPU1 via USB. Note that CPU1 should disable(3s)/enable(3s) send out radio packets after five seconds, but send out the radio data to CPU2. Keep node2 and node3 with normal works well SW. And TP4 output high when disable send radio packets.
2. Link direct timeout test.
    a) Settings node1 and node 2 with enable CAN and Radio, and set node3 with enable Radio but disable CAN.
    b) Start the 3 units. Use oscilloscope measured node1 TP4, node3 out10 and out12.
        - The link_timeout_long(out2) and link_timeout_short(out4)should be keep on, because repeater is enable, node3 should able to get node1 data from node 2.
        - The can_timeout_long(out6) and can_timeout_short(out8) should be keep off, as CAN be disabled at node3.
        - It should takes about 60ms delay to off from node1 disable to link_direct_timeout_short(out12) off.
        - It should takes about 300ms delay to off from node1 disable to The link_direct_timeout_long(out10) off.
3. Link timeout test with disable CAN and enable Radio.
    a) Settings all 3 nodes disable CAN and enable Radio.
    b) Start the 3 units. Use oscilloscope measured node1 TP4, node3 out2 and out4.

- The can_timeout_long(out6) and can_timeout_short(out8) should be keep off, as CAN be disabled at node3.
- Both link_direct_timeout_short(out12) and link_timeout_short(out4) should takes about 60ms delay to off from node1 disable to they off.
- Both link_direct_timeout_long(out10) and link_timeout_long(out2) should takes about 300ms delay to off from node1 disable to They off.

4. Link timeout test with enable CAN and disable Radio.
    a) Settings all 3 nodes enable CAN and disable Radio.
    b) Start the 3 units. All 6 timeout should still keep on, as CAN also include radio memory.

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>

Follow up the up point 2 test, Measured the time length takes about 61ms delay to off from node1 disable to link_direct_timeout_short(out12) off, and takes about 297ms delay to off from node1 disable to The radio_timeout_long(out10) off.

Follow up test point 3. Measured link_direct_timeout_short(out12) is delay about 63.3ms to turn off and link_timeout_short(out4) is about 66.1ms to off. And both link_direct_timeout_long(out10) and link_timeout_long(out2) takes about 300ms delay to off from node1 disable to They off.

Module Test #49 – check that ”allow repeat” on/off works.

Test Specification:

Number of units used	3
Config file	stressTest2.ssplog alternative 8 node 1,2,3,
Alternative used	8

Perform a check of the code that checks allow repeat works. The purpose is to verify that this code will behave correctly with handler allow repeat or no-repeat works.

Note that for alternative 8, all 3 nodes mem1 configure as “nopreat,” and mem2 configure default repeat:

mem1=(1,nozero,safe,norepeat,fast),
mem2=(2,nozero,safe,fast),

Note that for the alternative 8 node 3 configure as below:

out2 = node[1].mem1
out4 = node[1].mem2
out6 = node[2].mem1
out8 = node[2].mem2

```
system:
    slot_time = 32          # 32*62.5 us = 2ms
    can_interval = 64       # 64*62.5 us = 4ms
    can_baudrate = 250      # 250 kbit/s
    ossd_low = 1            # 1*62.5 us
    radio_timeout_short = 10    #   50*num_nodes*(slot_time*62.5us). Also used for CAN timeouts.
    radio_timeout_long = 50    #   50*num_nodes*(slot_time*62.5us). Also used for CAN timeouts.
```

Procedure:

1. In Sconstruct, enable SAFETY_TEST_INJECT = 0x60 test. Recompile and flash node1 CPU1 via USB. Note that CPU1 should disable(3s)/enable(3s) send out radio packets after five seconds, but send out the radio data to CPU2. Keep node2 and node3 with normal works well SW. And TP4 output high when disable send radio packets.
2. Enable node1 and node2 both CAN and Radio, node3 disable CAN but enable Radio.
3. The out2 should be delay to keep off when Node1 disable send out radio packets. This indicate that the no-repeat works.
4. The out4 should be keep on even though Node1 not send out radio packets. As node3 able to get the CPU1 data via node2.

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>

Follow up the up test, The out2(SIO2) delay about 59.3ms to turn off, but out4(SIO4) keep high when 3 nodes link up.

Module Test #50 – check that timeout from radio and not send data to CPU2 test

Test Specification:

Number of units used	3
Config file	stressTest2.ssplog
Alternative used	7

Perform a check of the code that checks the timeout from radio and CAN and not send data to CPU2. The purpose is to verify that this code will behave correctly check with six timeout of link_timeout_long, link_timeout_short,
can_timeout_long, can_timeout_short, link_direct_timeout_long, link_direct_timeout_short.

Note that for the alternative 7 node3 configure as below:

```
    dummy1 = link_timeout_long(node = const 1)
    dummy2 = link_timeout_short(node = const 1)
    dummy3 = can_timeout_long(node = const 1)
```

```
    dummy4 = can_timeout_short(node = const 1)
    dummy5 = link_direct_timeout_long(node = const 1)
    dummy6 = link_direct_timeout_short(node = const 1)

    #out1 =  s1s
    out2 =  dummy1
    out4 =  dummy2
    out6 =  dummy3
    out8 =  dummy4
    out10 = dummy5
    out12 = dummy6

system:
  slot_time = 32          # 32*62.5 us = 2ms
  can_interval = 64       # 64*62.5 us = 4ms
  can_baudrate = 250      # 250 kbit/s
  ossd_low = 1            # 1*62.5 us
  radio_timeout_short = 10    #   10*num_nodes*(slot_time*62.5us). Also used for CAN timeouts.
  radio_timeout_long = 50    #   50*num_nodes*(slot_time*62.5us). Also used for CAN timeouts.
```

Procedure:

1. In Sconstruct, enable SAFETY_TEST_INJECT = 0x61 test. Recompile and flash node1 CPU1 via USB. Note that CPU1 should disable(3s)/enable(3s) send out radio packets after five seconds, but send out the radio data to CPU2. Keep node2 and node3 with normal works well SW. And TP4 output high when disable send radio packets.
2. Link direct timeout test.
    a) Settings node1 and node 2 with enable CAN and Radio, and set node3 with enable Radio but disable CAN.
    b) Start the 3 units. Use oscilloscope measured node1 TP4, node3 out10 and out12.
        - The link_timeout_long(out2) and link_timeout_short(out4)should be keep on, because repeater is enable, node3 should able to get node1 data from node 2.
        - The can_timeout_long(out6) and can_timeout_short(out8) should be keep off, as CAN be disabled at node3.
        - It should takes about 60ms delay to off from node1 disable to link_direct_timeout_short(out12) off.
        - It should takes about 300ms delay to off from node1 disable to The link_direct_timeout_long(out10) off.
3. Link timeout test with disable CAN and enable Radio.
    a) Settings all 3 nodes disable CAN and enable Radio.
    b) Start the 3 units. Use oscilloscope measured node1 TP4, node3 out2 and out4.
        - The can_timeout_long(out6) and can_timeout_short(out8) should be keep off, as CAN be disabled at node3.

- Both link_direct_timeout_short(out12) and link_timeout_short(out4) should takes about 60ms delay to off from node1 disable to they off.
- Both link_direct_timeout_long(out10) and link_timeout_long(out2) should takes about 300ms delay to off from node1 disable to They off.

4. Link timeout test with enable CAN and disable Radio.
   a) Settings all 3 nodes enable CAN and disable Radio.
   b) Start the 3 units. All 6 timeout should still keep on, as CAN also include radio memory.

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>

Follow up the up point 2 test, Measured the time length takes about 61ms delay to off from node1 disable to link_direct_timeout_short(out12) off, and takes about 297ms delay to off from node1 disable to The radio_timeout_long(out10) off.

Module Test #51 – check that timeout from CAN

Test Specification:

Number of units used	Node1,2,3
Config file	stressTest2.ssplog
Alternative used	7

Perform a check of the code that checks the timeout from radio and CAN. The purpose is to verify that this code will behave correctly check with six timeout of link_timeout_long, link_timeout_short, can_timeout_long, can_timeout_short, link_direct_timeout_long, link_direct_timeout_short.

Note that for the alternative 7 node3 configure as below:

```
dummy1 = link_timeout_long(node = const 1)
dummy2 = link_timeout_short(node = const 1)
dummy3 = can_timeout_long(node = const 1)
dummy4 = can_timeout_short(node = const 1)
dummy5 = link_direct_timeout_long(node = const 1)
dummy6 = link_direct_timeout_short(node = const 1)

#out1 =  s1s
out2 =  dummy1
out4 =  dummy2
out6 =  dummy3
out8 =  dummy4
out10 = dummy5
out12 = dummy6
```

```
system:
  slot_time = 32          # 32*62.5 us = 2ms
  can_interval = 64       # 64*62.5 us = 4ms
  can_baudrate = 250      # 250 kbit/s
  ossd_low = 1            # 1*62.5 us
  radio_timeout_short = 10    #   10*num_nodes*(slot_time*62.5us). Also used for CAN timeouts.
  radio_timeout_long = 50    #   50*num_nodes*(slot_time*62.5us). Also used for CAN timeouts.
```

Procedure:

1. In Sconstruct, enable SAFETY_TEST_INJECT = 0x70 test. Recompile and flash node1 CPU1 via USB. Note that CPU1 should disable(3s)/enable(3s) send out radio packets after five seconds, but send out the radio data to CPU2. Keep node2 and node3 with normal works well SW. And TP4 output high when disable send radio packets.
2. Link timeout test with enable CAN and disable Radio.
    a) Settings node1 and node 2 with enable CAN and Radio, and set node3 with enable CAN but disable Radio.
    b) Start the 3 units. Use oscilloscope measured node1 TP4, node3 out2/4/6/8/10/12
    c) Start the 3 units. The 3 nodes relays keep on at the first 5 seconds. And then all 3 timeout of short should turn off about delay 60ms, also all 3 timeouts of long should turn off about delay 300ms.
3. Link timeout test with enable CAN and disable Radio.
    a) Settings all 3 nodes enable CAN and disable Radio.
    b) Start the 3 units. The 3 nodes relays keep on at the first 5 seconds. And then all 3 timeout of short should turn off about delay 60ms, also all 3 timeouts of long should turn off about delay 300ms.

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>

Follow up the up point 2,3 test, Measured the time length of all 3 timeout of short is 61ms , also all 3 timeouts of long is 299ms.

Module Test #52 – check that timeout from CAN and not send CAN data to CPU2 test

Test Specification:

Number of units used	Node1,2,3
Config file	stressTest2.ssplog
Alternative used	7

Perform a check of the code that checks the timeout from radio and CAN and not send data to CPU2. The purpose is to verify that this code will behave correctly check with six timeout of link_timeout_long, link_timeout_short,
can_timeout_long, can_timeout_short, link_direct_timeout_long, link_direct_timeout_short.

Note that for the alternative 7 node3 configure as below:

```
  dummy1 = link_timeout_long(node = const 1)
dummy2 = link_timeout_short(node = const 1)
dummy3 = can_timeout_long(node = const 1)
dummy4 = can_timeout_short(node = const 1)
dummy5 = link_direct_timeout_long(node = const 1)
dummy6 = link_direct_timeout_short(node = const 1)

#out1 =  s1s
out2 =  dummy1
out4 =  dummy2
out6 =  dummy3
out8 =  dummy4
out10 = dummy5
out12 = dummy6
```

```
system:
  slot_time = 32          # 32*62.5 us = 2ms
  can_interval = 64       # 64*62.5 us = 4ms
  can_baudrate = 250      # 250 kbit/s
  ossd_low = 1            # 1*62.5 us
  radio_timeout_short = 10    #   10*num_nodes*(slot_time*62.5us). Also used for CAN timeouts.
  radio_timeout_long = 50    #   50*num_nodes*(slot_time*62.5us). Also used for CAN timeouts.
```

Procedure:

1. In Sconstruct, enable SAFETY_TEST_INJECT = 0x71 test. Recompile and flash node1 CPU1 via USB. Note that CPU1 should disable(3s)/enable(3s) send out radio packets after five seconds, but send out the radio data to CPU2. Keep node2 and node3 with normal works well SW. And TP4 output high when disable send radio packets.
2. Link timeout test with enable CAN and disable Radio.
    a) Settings node1 and node 2 with enable CAN and Radio, and set node3 with enable CAN but disable Radio.
    b) Start the 3 units. Use oscilloscope measured node1 TP4, node3 out2/4/6/8/10/12
    c) Start the 3 units. The 3 nodes relays keep on at the first 5 seconds. And then all 3 timeout of short should turn off about delay 60ms, also all 3 timeouts of long should turn off about delay 300ms.
3. Link timeout test with enable CAN and disable Radio.
    a) Settings all 3 nodes enable CAN and disable Radio.
    b) The unit shall enter CPU2 fatal error MEMORY_MISMATCH (141) about five seconds delay when SIO output high few, the extern LED board display the correspond error number, and the two board LEDs should quick blink orange.

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>

Follow up the up point 2,3 test, Measured the time length of all 3 timeout of short is 61ms , also all 3 timeouts of long is 299ms.

Module Test #53 – Board button long time pressed test.

Test Specification:

Number of units used	1
Config file	stressTest2.ssplog
Alternative used	1

Perform a check of the board button be pressed long time. The purpose is to verify that this code will behave correctly to detect board button pressed, and if give out fatal error if button still pressed.

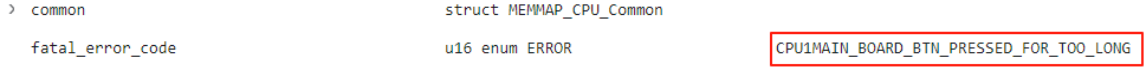
Procedure:

1. Start up the unit.
2. CPU1 should enter fatal error CPU1MAIN_BOARD_BTN_PRESSED_FOR_TOO_LONG (1128) if still press the board button more than 60 seconds.

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>

Following the up test, CPU1 able to enter fatal error CPU1MAIN_BOARD_BTN_PRESSED_FOR_TOO_LONG (1128) about delay five second when power-up the units. Attached pictures as below:



Module Test #54 – RF clock check at CPU1

Test Specification:

Number of units used	2 nodes and test at node2

Config file	stressTest2.ssplog
Alternative used	2

Perform a check of the RF clock be watched at CPU1. The purpose is to verify that this code will behave correctly to detect RF clock at CPU2, and compare RF clock changed at both CPUs.

Procedure:

1. Start the unit. Run “j2.py inject -i 1 -t s 2101 -r” via usb
2. CPU2 should enter fatal error CPU2MAIN_RADIO_CLOCK_COUNT_BAD (2101) after 5 seconds.

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>

Following the up test, CPU1 able to enter fatal error RF_CLK_MISMATCH (35) about delay five second when power-up the units. Attached pictures as below:

Module Test #55 – AD noraml works following check.

Test Specification:

Number of units used	2
Config file	stressTest2.ssplog
Alternative used	2

Perform a check of the code that checks the AD convert following. The purpose is to verify that this code will behave correctly check with AD convert following, to check if AD converted finished if want to start it, and also check if unexpected AD convert occurred if not start AD convert. IA real AD convert following error cannot be triggered, so the code has to be inject a fake dealing to trigger the error occurring after five seconds.

Procedure:

1. In Sconstruct, enable SAFETY_TEST_INJECT = 0x80 test. Recompile and flash CPU1 via USB.
2. Start the unit. The unit should enter fatal error AD_STARTED (24) after five seconds.
3. In Sconstruct, enable SAFETY_TEST_INJECT = 0x81 test. Recompile and flash CPU1 via USB.
4. Start the unit again, The unit should enter fatal error AD_NOT_START (25) after five seconds.

Test Execution:

Date	

Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>

Follow up the up test, The unit able to enter fatal error AD_STARTED (24) when test point 2, and able to enter fatal error AD_NOT_START (25) when test point 4.

Module Test #56 – 1Khz works following check.

Test Specification:

Number of units used	2
Config file	stressTest2.ssplog
Alternative used	2

Perform a check of the code that checks the 1Khz IRQ following. The purpose is to verify that this code will behave correctly check with1Khz following, it should be trigged from fast IRQ, and should check if have run 1Khz IRQ when need trigger again. A real 1Khz works following error cannot be triggered, so the code has to be inject a fake dealing to trigger the error occurring after five seconds.

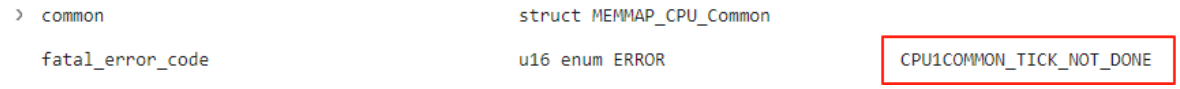
Procedure:

1. Start the unit. Run “j2.py inject -i 1 -t s 11364-r” via usb
2. The unit should enter fatal error CPU1COMMON_TICK_NOT_DONE (11364) .

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>

Follow up the up test, The unit able to enter fatal error LOW_IRQ_MISMATCH (67) after five seconds. Attached picture as below.



Module Test #57 – Adjust measured Plus 12V for test at CPU2

Test Specification:

Number of units used	1
Config file	stressTest2.ssplog
Alternative used	1

Perform a check of the code that checks the 3.3V voltage limit. The purpose is to verify that this code will behave correctly to verify the 3.3V range, it should be in [3.1, 3.5]

Procedure:

1. Tested with 12V voltage input. Use adapter(220V12V,1000mA) MK120P100PHT
2. HW adjust the CPU1 3.3V to 3.1, And the power up the unit, CPU2 should able to give output fatal error CPU1_3V3.
3. HW adjust the CPU1 3.3V to 3.5, And the power up the unit, CPU2 should able to give output fatal error CPU1_3V3.
4. HW adjust the CPU2 3.3V to 3.1, And the power up the unit, CPU1 should able to give output fatal error CPU2_3V3.
5. HW adjust the CPU2 3.3V to 3.5, And the power up the unit, CPU1 should able to give output fatal error CPU2_3V3.

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>

Following the up test, able to give out fatal error when the 3.3V not in [3.1,3.5]

Module Test #58 – CAN memories test.

Test Specification:

Number of units used	16
Config file	stressTest2.ssplog
Alternative used	1

Perform a check of the CAN send out 2 memories of itself. The purpose is to verify that code able to send out itself memories via CAN. For the stress test need, we set up with 16 nodes works together, then the 2*16= 32 memories should sent out in one CAN_INTERVAL time.

Note that alternative 1 set CAN baud as 250Kbits and can_interval as 512(32ms). And for easy to see the memories data send via CAN, disable radio at all nodes.

Use USBCANII to sniffer all the CAN packets. Note that set the PC Ecantools to filter out the CAN debug packet for easy to view the memories packet.

Node that for 11bits CAN ID define as PPCCCCCNNNN. PP is priority, 00 for memory packet, 11 for debug packet. CCCCC for count, it should increase 1 at memory packet, and keep 0 at debug packet. NNNN is the node ID.

Procedure:

1. Set-up 16 nodes to works just CAN.

2. Connect USBCANII to CAN bus to sniffer the CAN datas.
3. Open Ecantools and set as 250Kbits. And also filter out the CAN debug packet.
4. Power up the unit.
5. We can see the all node relays are open, This indicate that all node link up and works well.
6. Save the CAN data as excel file. Then analyse the excel file. All 16 nodes memories data should send out one time per 32ms.

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>

Followed up test. 16 nodes memories data able to send out in 32MS. Attached as below:
System set up as below:

Module Test #59 – C2C send packet with same packet count test.

Test Specification:

Number of units used	Set up with 16 Nodes works, and tested at node 5.
Config file	stressTest2.ssplog
Alternative used	1

Perform a check of the code that checks the CPU1 not handler the packets with same packet count. The purpose is to verify that this code will behave correctly if CPU1 receive CPU2 with same packet counts. CPU1 should not handler the data which have same packet count with pre packet. CPU2 code has to fake to send out data with same packets to trigger CPU1 enter fatal error CPU1COMMON_C2C_BAD_PACKET_COUNTER (11306).

Procedure:

1. Start the unit. Run “j2.py inject -i 2 -t s 11306” via usb
2. The unit should enter fatal error CPU1COMMON_C2C_BAD_PACKET_COUNTER (11306). after about five seconds.

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>

The unit into fatal error and indicate CPU1COMMON_C2C_BAD_PACKET_COUNTER (11304) error about delay 5 seconds.

Module Test #60 – C2C communicate missed packets from CPU2 test.

Test Specification:

Number of units used	Set up with 16 Nodes works, and tested at node 5.
Config file	stressTest2.ssplog
Alternative used	1

Perform a check of the code that checks the CPU1 missed packets from CPU2. The purpose is to verify that this code will behave correctly if CPU1 receive CPU2 with lost packets. The CPU1 allow to lost about 20 packets from CPU2. A real lost packets more than 20 packets is difficult to happen, so the CPU2 code has to fake to miss 20 packets one time to trigger CPU1 enter fatal error CPU1COMMON_C2C_MISSED_TOO_MANY_PACKETS (11304).

Procedure:

1. Start the unit. Run “j2.py inject -i 2 -t s 11304” via usb
2. The unit should enter fatal error CPU1COMMON_C2C_MISSED_TOO_MANY_PACKETS (11304) after about five seconds.

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>

Following up procedure test, CPU1 able to enter fatal error C2C_MISSED_TO_MANY_PACKETS (39) after five seconds.

Module Test #61 – STACK overflow checking.

Test Specification:

Number of units used	Set up with 16 Nodes works, and tested at node 5.
Config file	stressTest2.ssplog
Alternative used	1

Perform a check of the code that checks the stack if overflow. The purpose is to verify that this code will behave correctly when stack overflow happen. A real stack overflow is difficult to occurred, so we reduce the STACK_SIZE to trigger it to test.

Procedure:

1. In CPU1 “startup.h”, Modify STACK_SIZE = “0x200”. Recompile and flash CPU1 via USB.
2. Start the unit.
3. The unit should enter fatal error CPU1COMMON_STACK_OVERFLOW (11277) during system running.

4. CPU1 “startup.h”, Return STACK_SIZE = “0xC00”. Recompile and flash CPU1 via USB.
5. Start the unit.
6. The unit should works normal gain.

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>

Following up procedure test, CPU1 able to enter fatal error CPU1COMMON_STACK_OVERFLOW (11277) during running. And able to works normal when return to set STACK_SIZE to 0xC00.

Module Test #62 – C2C respond time delay check.

Test Specification:

Number of units used	Set up with 16 Nodes works, and tested at node 5.
Config file	stressTest2.ssplog
Alternative used	1

Perform a check of the code that checks the CPU2 respond CPU1 time delay. The purpose is to verify that this code will behave correctly if CPU2 respond CPU1 time delay checking. It the respond delay over than 20 packets(ms) CPU1 should trigger fatal error. A real 20ms delay is difficult to happened, so CPU2 code has to fake to send out data with same packets to trigger CPU1 enter fatal error CPU2_RESPOND (40).

Procedure:

1. In CPU2 Sconstruct, enable the SAFETY_TEST_INJECT = 13 test. Recompile and flash CPU2 via USB.
2. Start the unit.
3. The unit should enter fatal error CPU2_RESPOND (40)after about five seconds.

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>

Following up procedure test, CPU1 able to enter fatal error CPU2_RESPOND (40) after five seconds. Attached picture as below.

Module Test #63 – Check flash memory test

Test Specification:

Number of units used	1
Config file	stressTest2.ssplog
Alternative used	None

Perform a check of the crc check of the flash memory. The purpose of this test is to verify the mechanism in the code that shall stop execution and keep in the bootloader if the runtime calculated crc differs from the compile time stored crc, and should able to update SW via Trabus.

Procedure:

1. Flash SRC021-00Xv0xpre_noCRC.srec to CPU1 via USB.
2. When the CRC is calculated in CPU1, it will have a different result from what is stored in flash, and the unit will not start, it should keep in bootloader and the two board LEDs should lit red.
3. Update SRC021-00Xv0xpre.srec via j2, and the unit re-works normal again.

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>

The two board LED lit Red to indicate still in bootloader, and then it support to re-flash the correct SW via trabus and re-works normal.

Software module tests for CPU2

Module Test #1 - Power-up stable test

Test Specification:

Number of units used	Set up with 16 Nodes works, and tested at node 5.
Config file	stressTest2.ssplog
Alternative used	1

Perform a check of the power-up stable, The purpose is to verify if enter fatal error at power-up.
Procedure:

1. Start the unit.
2. The unit should restart per 5 seconds all the time, and not enter any fatal error. Note that if enter fatal error, the unit still keep in fatal error mode, not restart gain.

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>

Check the power-up stable about 12 hours, the system restart per 5 seconds all the time stable, not enter fatal error.

Module Test #2 - Runtime CPU2CPU RX DMA stable test

Test Specification:

Number of units used	Set up with 16 Nodes works, and tested at node 5.
Config file	stressTest2.ssplog
Alternative used	1

Perform a check of the code that checks the CPU2CPU DMA RX stable with bad data. The purpose is to verify that this code will behave correctly if wrong data is detected runtime, and the system should still works normal.

Procedure:

1. Start the unit. Run “j2.py inject -i 1 -t s 11304” via usb
2. After five seconds, CPU2 should flush out one byte per 50ms. It can filter out incorrect data and discard received data packets. If the number of data packets exceeds 20, an error will be triggered.

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>

If the number of data packets exceeds 20, an error will be triggered.

Module Test #3 - Runtime CPU2CPU TX DMA with bad CRC test

Test Specification:

Number of units used	Set up with 16 Nodes works, and tested at node 5.
Config file	stressTest2.ssplog
Alternative used	1

Perform a check of the code that checks the CPU2CPU DMA TX stable with bad CRC. The purpose is to verify that this code will behave correctly if some unexpected data with CRC error, and the unit should still works normal.

Procedure:

1. Start the unit. Run “j2.py inject -i 3 -t s 11264” via usb
2. Quickly short pressed board button to flush out one receive buffer to destroy the data. The system should still works normal, but CRC error should data should be detected at CPU1.

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>

if button pressed in board one time. The num_c2c_not_plus1_pkt_count will add one. Attached the value as below:

REMOVE Module Test #4 - Runtime CPU2CPU TX DMA with destroy Tx length

Test Specification:

Number of units used	Set up with 16 Nodes works, and tested at node 5.
Config file	stressTest2.ssplog
Alternative used	1

this test is not valid anymore due to using fixed size packets

Perform a check of the code that checks the CPU2CPU DMA TX stable with destroy Tx length. The purpose is to verify that this code will behave correctly if unexpected data is detected runtime.

Procedure:

1. In Sconstruct, enable the SAFETY_TEST_INJECT = 4 test. Recompile and flash CPU2 via USB. Note that the SW will send out two packets with unexpected length data per 256 packets, one with len+1, the other one with len-1.
2. Start the unit.
3. Run “sdd --view CPU2CPU --filter c2cInfo.dbg.” to watch rxCpu2cpuCRCerrors and rxSearchPreambleUnexpects should be detected.

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	

Config file GIT commit ID	
Result	

Module Test #5 - Inject fatal error test .

Test Specification:

Number of units used	Set up with 16 Nodes works, and tested at node 5.
Config file	stressTest2.ssplog
Alternative used	1

Perform a check of the code that fatal error works. The purpose is that verify fatal error able to works, and display the fatal error number at the extern LED board. The unit also able to restart if hold pressed board button or cap button when unit enter fatal error.

Procedure:

1. Start the unit. Run “j2.py inject -i 4 -t s 12288” via usb
2. The unit should enter fatal error after about five seconds. All SIOs, SIO power, relay power, plus 12V should be off. Note that the SIO16 LED lit to indicate is CPU2 fatal error.

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>

When power up the unit, The uint enter fatal error CPU2COMMON_NO_ERROR (11264) after five seconds. All SIOs, SIO power, relay power, plus 12V be turned off at fatal error status.

Module Test #6 - Runtime CPU2CPU not send packets 20ms test

Test Specification:

Number of units used	Set up with 16 Nodes works, and tested at node 5.
Config file	stressTest2.ssplog
Alternative used	1

Perform a check of the code that checks the CPU2CPU TX not send 20ms. packets The purpose is to verify that this code will behave correctly into CPU2CPU_FAILURE_AT_RUNNING when CPU1 not received packet from CPU2 more than 20ms.

Procedure:

1. Start the unit. Run “j2.py inject -i 1 -t s 12328” via usb
2. The unit should enter fatal error CPU2COMMON_C2C_MISSED_TOO_MANY_PACKETS after about five seconds.

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>

Enter fatal error CPU2COMMON_C2C_MISSED_TOO_MANY_PACKETS. Note that CPU2 also send out some debug data to CPU1, but the time out just reload with the normal packets.

Module Test #7 - Runtime CPU2CPU RX timeout test.

Test Specification:

Number of units used	Set up with 16 Nodes works, and tested at node 5.
Config file	stressTest2.ssplog
Alternative used	1

Perform a check of the code that checks the CPU2CPU RX timeout. The purpose is to verify how long to time out about not received packet from CPU1 to indicate fatal error. A real RX time out is difficult to trigger error at HW, so the code has to fake not reload the RX timing after five seconds.

Procedure:

1. Start the unit. Run “j2.py inject -i 7 -t s 11264” via usb
2. Note that for easy to measured the time lenght, the SW use PA13 high pulse lenght to indicate the time RX time out. PB4 should be set high after unit power-up 5 seconds, and then reset to lowif SW detect timeout. Note that for CPU2, it just reset itself, so the CPU1 shoud give out fatal error for time out CPU2CPU_FAILURE_AT_RUNNING.
3. Start the unit.
4. The unit should enter fatal error CPU2CPU_FAILURE_AT_RUNNING. after about five seconds.

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>

Module Test #8 - CPU2CPU TX overrun test.

Test Specification:

Number of units used	Set up with 16 Nodes works, and tested at node 5.
Config file	stressTest2.ssplog
Alternative used	1

Perform a check of the code that checks the CPU2CPU TX overrun. The purpose is to verify that this code will behave correctly check with CPU2 TX overrun. A real DMA TX overrun is difficult to trigger error, so the code has to fake to send out many data at the same time after five seconds.

Procedure:

1. Start the unit. Run “j2.py inject -i 1 -t s 12331” via usb
2. The unit should enter fatal error CPU2COMMON_C2C_DMA_NOT_READY (12331) after about five seconds.

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>

Following up procedure test, CPU2 able to enter fatal error CPU2COMMON_C2C_DMA_NOT_READY (12331) after five seconds

Module Test #9 - CPU2CPU TX unexpect test.

Test Specification:

Number of units used	Set up with 16 Nodes works, and tested at node 5.
Config file	stressTest2.ssplog
Alternative used	1

Perform a check of the code that checks the CPU2CPU TX at disable state. The purpose is to verify that this code will behave correctly check with CPU2CPU TX at disable should be forbid and trigger error. A real DMA TX at disable state is difficult to trigger error, so the code has to fake to send out C2C packet before enable send it out.

Procedure:

1. Start the unit. Run “j2.py inject -i 2 -t s 12331” via usb
2. The unit should enter fatal error CPU2COMMON_C2C_DMA_NOT_READY (12331)

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	

Config file GIT commit ID	
Result	<pass>

Following up procedure test, CPU1 able to enter fatal error CPU2COMMON_C2C_DMA_NOT_READY (12331) immediately when power up.

Module Test #10 - Fast IRQ follow control test.

Test Specification:

Number of units used	Set up with 16 Nodes works, and tested at node 5.
Config file	stressTest2.ssplog
Alternative used	1

Perform a check of the code that check the fast IRQ follow control. The purpose is to verify the following control at fast IRQ run correct. A real fast IRQ follow control error cannot be triggered, so the code has to be modified to “think” there is an error occurring.

Procedure:

1. Start the unit. Run “j2.py inject -i 1 -t s 12308 -r” via usb
2. The unit should enter fatal error CPU2COMMON_FL_CTRL_FTICK (12308) after about five seconds.

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>

Module Test #11 – 1Khz IRQ follow control

Test Specification:

Number of units used	Set up with 16 Nodes works, and tested at node 5.
Config file	stressTest2.ssplog
Alternative used	1

Perform a check of the code that check the 1Khz IRQ follow control. The purpose is to verify the following control at 1Khz IRQ run correct. A real 1Khz IRQ follow control error cannot be triggered, so the code has to be modified to “think” there is an error occurring.

Procedure:

1. Start the unit. Run “j2.py inject -i 1 -t s 12310 -r” via usb
2. The unit should enter fatal error CPU2COMMON_FL_CTRL_1KHZ_IRQ (12310) after about five seconds.

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>

The unit into CPU2 fatal error and indicate FL_CTL_1KHZ_IRQ (66) error about delay 5 seconds.

Module Test #12 - Mainloop follow ctl test

Test Specification:

Number of units used	Set up with 16 Nodes works, and tested at node 5.
Config file	stressTest2.ssplog
Alternative used	1

Perform a check of the code that check the mainloop follow control. The purpose is to verify the following control at mainloop run correct. A real mainloop follow control error cannot be triggered, so the code has to be modified to “think” there is an error occurring.

Procedure:

1. Start the unit. Run “j2.py inject -i 1 -t s 12309 -r” via usb
2. The unit should enter fatal error CPU2COMMON_FL_CTRL_1KHZ_IRQ (12309) after about five seconds.

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>

Module Test #13 – check that startup-check for memories work.

Test Specification:

Number of units used	Set up with 16 Nodes works, and tested at node 5.
Config file	stressTest2.ssplog
Alternative used	1

Perform a check of the code that checks the all ram memory works at start up. The purpose is to verify that all ram from 0x20000000 to 0x20018000 able to write and read out correct by word. A real RAM memory error cannot be triggered, so the code has to be modified to “think” there is an error occurring.

Procedure:

1. Start the unit. Run “j2.py inject -i 1 -t s 12348 -r” via usb
2. The unit shall enter fatal error CPU2COMMON_RAM_TEST_START_UP (12348), the extern LED board display the correspond error number, and the two board LEDs should quick blink orange.

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>

The unit can enter CPU2 fata error CPU2COMMON_RAM_TEST_START_UP (12348), the extern LED board display the correspond error number, and the two board LEDs should quick blink orange

Module Test #14 - Continuous ram test

Test Specification:

Number of units used	Set up with 16 Nodes works, and tested at node 5.
Config file	
Alternative used	1

Perform a check of the code that checks the RAM memory. The purpose is to verify that this code will behave correctly if a RAM error is detected runtime. A real RAM memory error cannot be triggered, so the code has to be modified to “think” there is an error occurring.

Procedure:

1. Start the unit. Run “j2.py inject -i 1 -t s 12349 -r” via usb
2. The unit shall enter CPU2 fatal error CPU2COMMON_RAM_TEST_RUNNING (12349), the extern LED board display the correspond error number, and the two board LEDs should quick blink orange.

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	

Config file GIT commit ID	
Result	<pass>

The unit can enter CPU2 fatal error CPU2COMMON_RAM_TEST_RUNNING (12349), the extern LED board display the correspond error number, and the two board LEDs should quick blink orange.

Module Test #15 – Continue RAM check timeout test

Test Specification:

Number of units used	Set up with 16 Nodes works, and tested at node 5.
Config file	stressTest2.ssplog
Alternative used	1

Perform a check of the code that checks the RAM memory running check . The purpose is to verify that this code will behave correctly if a RAM running checking not run about 60 seconds. A real RAM memory error cannot be triggered, so the code has to be modified to “think” there is an error occurring.

Procedure:

1. Start the unit. Run “j2.py inject -i 1 -t s 12350 -r” via usb
2. The unit shall enter CPU2 fatal error CPU2COMMON_RAM_TEST_TIMEOUT (12350), the extern LED board display the correspond error number, and the two board LEDs should quick blink orange.

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>

The unit can enter CPU2 fatal error CPU2COMMON_RAM_TEST_TIMEOUT (12350) after about 60 seconds, the extern LED board display the correspond error number, and the two board LEDs should quick blink orange

Module Test #16 – Flash test during running.

Test Specification:

Number of units used	Set up with 16 Nodes works, and tested at node 5.
Config file	
Alternative used	1

Perform a check of the code that checks the flash memories integrity. The purpose is to verify that this code will behave correctly if the flash memory broken. A real flash memory broken is difficult to happend, so the code has to fake to miss one word of flash to calculate CRC to trigger fatal error.

Procedure:

1. Start the unit. Run “j2.py inject -i 1 -t s 12302 -r” via usb
2. The unit should enter CPU2 fatal error CPU2COMMON_FLASH_CRC_INVALID (12302) after about five seconds.

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>

Following up procedure test, CPU2 able to enter fatal error CPU2COMMON_FLASH_CRC_INVALID (12302) after five seconds.

Module Test #17 - Fast irq between time limit test

Test Specification:

Number of units used	Set up with 16 Nodes works, and tested at node 5.
Config file	stressTest2.ssplog
Alternative used	1

Perform a check of the code that checks the max between two fast IRQ time limit . The purpose is to verify that this code will behave correctly if one of fast IRQ between time over than limit. A real fast IRQ between time cannot be over than the limit 75us, so the code has to be modified fast IRQ period to 77.5us to trigger the error occurring after five seconds.

Procedure:

1. Start the unit. Run “j2.py inject -i 1 -t s 12489 -r” via usb
2. The unit shall enter fatal error CPU2COMMON_FTICK_BETWEEN_TIME (12489) after five seconds, the extern LED board display the correspond error number, and the two board LEDs should quick blink orange.

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>

Module Test #18 - Fast irq average time limit test

Test Specification:

Number of units used	Set up with 16 Nodes works, and tested at node 5.
Config file	stressTest2.ssplog
Alternative used	1

Perform a check of the code that checks the fast IRQ average time limit. The purpose is to verify that this code will behave correctly if fast IRQ average time over than limit. A real fast IRQ between time cannot be over than the limit 65us, so the code has to be modified fast IRQ period to 67.5us to trigger the error occurring after five seconds.

Procedure:

1. Start the unit. Run “j2.py inject -i 1 -t s 12490 -r” via usb
2. The unit shall enter fatal error CPU2COMMON_FTICK_AVG_TIME (12490) after five seconds, the extern LED board display the correspond error number, and the two board LEDs should quick blink orange.

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>

Module Test #19 - Fast irq run for time test

Test Specification:

Number of units used	Set up with 16 Nodes works, and tested at node 5.
Config file	stressTest2.ssplog
Alternative used	1

Perform a check of the code that checks the fast IRQ for time limit. The purpose is to verify that this code will behave correctly if fast IRQ for time over than limit. A real fast IRQ for time cannot be over than the limit 40us, so the code has to be inject a delay about 15us to trigger the error occurring after five seconds.

Procedure:

1. Start the unit. Run “j2.py inject -i 1 -t s 12488 -r” via usb
2. The unit shall enter fatal error CPU2COMMON_FTICK_MAX_TIME (12488) after five seconds, the extern LED board display the correspond error number, and the two board LEDs should quick blink orange.

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>

Module Test #20- 1KHZ irq between time limit test

Test Specification:

Number of units used	Set up with 16 Nodes works, and tested at node 5.
Config file	stressTest2.ssplog
Alternative used	1

Perform a check of the code that checks the max between two 1Khz IRQ time limit . The purpose is to verify that this code will behave correctly if one of 1Khz IRQ between time over than limit. A real 1Khz IRQ between time cannot be over than the limit 1120 us, so the code has to be modified 1Khz IRQ period to trigger the error occurring after five seconds.

Procedure:

1. Start the unit. Run “j2.py inject -i 1 -t s 12499 -r” via usb
2. The unit shall enter fatal error CPU2COMMON_IRQ_1KHZ_BETWEEN_TIME (12499) after five seconds, the extern LED board display the correspond error number, and the two board LEDs should quick blink orange.

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>

Module Test #21 - 1KHZ irq average time limit test

Test Specification:

Number of units used	Set up with 16 Nodes works, and tested at node 5.
Config file	stressTest2.ssplog
Alternative used	1

Perform a check of the code that checks the 1Khz IRQ average time limit. The purpose is to verify that this code will behave correctly if 1Khz IRQ average time over than limit. A real 1Khz IRQ between time

cannot be over than the limit 1010us, so the code has to be modified 1010 IRQ period to 1062.5us to trigger the error occurring after five seconds.

Procedure:

1. Start the unit. Run “j2.py inject -i 1 -t s 12499 -r” via usb
2. The unit shall enter fatal error CPU2COMMON_IRQ_1KHZ_AVG_TIME (12499) after five seconds, the extern LED board display the correspond error number, and the two board LEDs should quick blink orange.

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>

Module Test #22 - 1KHZ irq run for time limit test

Test Specification:

Number of units used	Set up with 16 Nodes works, and tested at node 5.
Config file	stressTest2.ssplog
Alternative used	1

Perform a check of the code that checks the fast 1Khz for time limit. The purpose is to verify that this code will behave correctly if 1Khz IRQ for time over than limit. A real 1Khz IRQ for time cannot be over than the limit 800us, so the code has to be inject a delay to trigger the error occurring after five seconds.

Procedure:

1. Start the unit. Run “j2.py inject -i 1 -t s 12498 -r” via usb
2. The unit shall enter fatal error CPU2COMMON_IRQ_1KHZ_FOR_TIME (12498) after five seconds, the extern LED board display the correspond error number, and the two board LEDs should quick blink orange.

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>

Module Test #23 – Main loop average time limit test

Test Specification:

Number of units used	Set up with 16 Nodes works, and tested at node 5.
Config file	stressTest2.ssplog
Alternative used	1

Perform a check of the code that checks the main loop average time limit. The purpose is to verify that this code will behave correctly if main loop average time over than limit. A real main loop average time cannot be over than the limit 1000us, so the code has to be inject a delay about 600us in the main loop to trigger the error occurring.

Procedure:

1. Start the unit. Run “j2.py inject -i 1 -t s 12510 -r” via usb
2. The unit shall enter fatal error CPU2COMMON_MAIN_LOOP_AVG_TIME (12510) after five seconds, the extern LED board display the correspond error number, and the two board LEDs should quick blink orange.

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>

Module Test #24 - Main loop between time limit test

Test Specification:

Number of units used	Set up with 16 Nodes works, and tested at node 5.
Config file	stressTest2.ssplog
Alternative used	1

Perform a check of the code that checks the max between time limit. The purpose is to verify that this code will behave correctly if one of main loop between time over than limit. A real mainloop between time cannot be over than the limit 5ms, so the code has to be insert a 5ms delay to trigger the error occurring after five seconds.

Procedure:

1. Start the unit. Run “j2.py inject -i 1 -t s 12508 -r” via usb
2. The unit shall enter fatal error CPU2COMMON_MAIN_LOOP_BETWEEN_TIME (12508) after five seconds, the extern LED board display the correspond error number, and the two board LEDs should quick blink orange.

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>

Module Test #25 - Main loop run for time limit test

Test Specification:

Number of units used	Set up with 16 Nodes works, and tested at node 5.
Config file	stressTest2.ssplog
Alternative used	1

Perform a check of the code that checks the main loop for time limit. The purpose is to verify that this code will behave correctly if main loop for time over than limit. A real main loop for time cannot be over than the limit 3000us, so the code has to be inject a delay about 3000us to trigger the error occurring after five seconds.

Procedure:

1. Start the unit. Run “j2.py inject -i 1 -t s 12509 -r” via usb
2. The unit shall enter fatal error CPU1COMMON_MAIN_LOOP_FOR_TIME (12509) after five seconds, the extern LED board display the correspond error number, and the two board LEDs should quick blink orange.

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>

Module Test #26 – RF clock check at CPU2

Test Specification:

Number of units used	2 nodes and test at node2
Config file	stressTest2.ssplog
Alternative used	2

Perform a check of the RF clock be watched at CPU2. The purpose is to verify that this code will behave correctly to detect RF clock at CPU2, and compare RF clock changed at both CPUs. A real RF clock lost detect at CPU2 cannot be happend, so the code has to be inject a fake to not update the RF clock to trigger the error occurring after five seconds.

Procedure:

3. Start the unit. Run “j2.py inject -i 1 -t s 2101 -r” via usb
4. CPU2 should enter fatal error CPU2MAIN_RADIO_CLOCK_COUNT_BAD (2101) after 5 seconds.

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>

Module Test #27 – RF clock not changed and tell CPU1 Test

Test Specification:

Number of units used	2 nodes and test at node2
Config file	stressTest2.ssplog
Alternative used	2

Perform a check of the RF clock be watched at CPU1. The purpose is to verify that this code will behave correctly to detect RF clock at CPU1, and compare RF clock changed at both CPUs. A real RF clock lost detect at CPU1 cannot be happend, so the code has to be inject a fake to not changed and tell CPU1 to trigger the error occurring after five seconds.

Procedure:

1. Start the unit. Run “j2.py inject -i 1 -t s 2101 -r” via usb
2. CPU2 should enter fatal error CPU2MAIN_RADIO_CLOCK_COUNT_BAD (2101) after 5 seconds.

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>

Module Test #28 – CAN TX CTL pin force to high test

Test Specification:

Number of units used	2 nodes and test at node2
Config file	stressTest2.ssplog

Alternative used	2

Perform a check of the CAN TX CTL pin force to high at CPU2. The purpose is to verify that this code will behave correctly to detect CAN TX CTL pin at CPU2 when TX CTL pin as output high or as input pull down.

Procedure:

1. Start the unit. Run “j2.py inject -i 28 -t s 12288” via usb
2. The system should still works normal, but CAN should not work.

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>

Module Test #29 – REL lost test

Test Specification:

Number of units used	2 nodes and test at node2
Config file	stressTest2.ssplog
Alternative used	2

Perform a check of the REL pulse. The purpose is to verify that this code will behave correctly to detect REL pulse lost. A real REL lost cannot be happened, so the code has to be injected a fake to disable output REL pulse after 5 seconds.

Procedure:

1. Start the unit. Run “j2.py inject -i 2 -t s 11322 -r” via usb.
2. The unit shall enter fatal error CPU1COMMON_RELAY_STATUS_MISMATCH (11322).

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>

Following the up test, CPU1 able to enter fatal error CPU1COMMON_RELAY_STATUS_MISMATCH (11322) about delay five second when power-up the units and radio link up.

Module Test #30 – SIOs analogue compare test

Test Specification:

Number of units used	1
Config file	stressTest2.ssplog
Alternative used	Alternative 6 for test output, and 9 for test input.

Perform a check of the code that checks SIO analogue compare status. The purpose is to verify that this code will behave correctly check with compare the SIO analogue value, and be able to trigger fatal error if each SIO AD mismatch. The SIO analogue both CPU compare should satisfy all of below condition, if one of then fail to meet the conditions, there should be an error occurred. The Max difference time is 200ms.

- ➔ At output, CPU1 SIO voltage should be in [0.75*powerSuply : powerSupply], if not record as error type 1
- ➔ At output, CPU1 SIO voltage should less than CPU2, and the min difference should more than 1/32 of CPU2 SIO voltage.
- ➔ At output, the max difference of CPU2 SIO voltage between SIO power should less than 0.75V when SIO power voltage less than 24V, and less than 1/16 of SIOpower voltage when over than 24V. If not record as error type 3.
- ➔ At input, the max difference of SIO CPU1 and CPU2 should less than 0.75V when SIO power voltage less then 24V, and less then 1/16 of SIOpower voltage when over than 24V. If the high voltage not correct, record as error type 4, else record error type as 5.

Note that we just need check the code able to detect the up difference error type and able to give out fatal error if still fail more than 200ms here, don't need to check all SIO HW should be check or not. Please refer to the HW module test for the detail SIO test if need. So just test with SIO2 here.

Procedure:

1. Test with 24V. And use the alternative 6 for test. Note that for output. the real ad detected voltage should be 24/(22+(1.1*10/(1.1+10)))*(1.1*10/(1.1+10))*(22+1.1)/1.1 = 21.724V.
2. Adjust R213 to 1.4K, when turn SIO, then read out the analogureErrorLog data we should able to see it record the error type 1 and give out fatal error SIO_ANLG_TOO_UNSTABLE(148). If we adjust to 1.4K: 24/(22+(1.4*10/(1.4+10)))*(1.4*10/(1.4+10))*21 = 26.6465256 bigger than 24V.
3. Adjust R211 to 1.2K, when turn on SIO, then read out the analogureErrorLog data we should able to see it record the error type 3 and give out fatal error SIO_ANLG_TOO_UNSTABLE(148). If we adjust to 1.2K: 24/(24+1.2)*1.2 * (24+1.1)/1.1=26.078.
4. Test with 24V, and use the alternative 9 for test.
5. Adjust R211 to 1.2K, and connect 24V input to SIO2, then read out analogureErrorLog data we should able to see it record the error type 4 or 5 and output fatal error.

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>

According to above procedure to test, The fatal error SIO_ANLG_TOO_UNSTABLE(148) should be triggered with error type in range 1 to 5.

Module Test #31 - Check flash memory test

Test Specification:

Number of units used	1
Config file	stressTest2.ssplog
Alternative used	None

Perform a check of the crc check of the flash memory. The purpose of this test is to verify the mechanism in the code that shall stop execution and keep in the bootloader if the runtime calculated crc differs from the compile time stored crc, and should able to update SW via USB.

Procedure:

1. Flash SRC022-00Xv0xpre_noCRC.srec to CPU2 via USB.
2. When the CRC is calculated in CPU2, it will have a different result from what is stored in flash, and the unit will not start, it should keep in bootloader and unit enter fatal error CPU1MAIN_CHECK_CPU2_FAIL (1231). Note that for CPU2 keep in bootloader and not able to respond the CMD from CPU1, So the unit should enter CPU1 fatal error CPU1MAIN_CHECK_CPU2_FAIL (1231).
3. Update SRC022-00Xv0xpre.srec to CPU2 via USB, and the unit re-works normal again.

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>

The unit enter faltal CPU1MAIN_CHECK_CPU2_FAIL (1231), and then it support to re-flash the correct SW via USB and re-works normal.

Module Test #32– STACK overflow checking.

Test Specification:

Number of units used	Set up with 16 Nodes works, and tested at node 5.
Config file	stressTest2.ssplog
Alternative used	1

Perform a check of the code that checks the stack if overflow. The purpose is to verify that this code will behave correctly when stack overflow happen. A real stack overflow is difficult to occurred, so we reduce the STACK_SIZE to trigger it to test.

Procedure:

1. In CPU2 startup.h, Modify STACK_SIZE = “0x100”. Recompile and flash CPU2 via USB.
2. Start the unit.
3. The unit should enter fatal error CPU2COMMON_STACK_OVERFLOW (12301) during system running.
4. CPU2 “startup.h”, Return STACK_SIZE = “0x200”. Recompile and flash CPU2 via USB.
5. Start the unit.
6. The unit should works normal gain.

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>

Following up procedure test, CPU2 able to enter fatal error CPU2COMMON_STACK_OVERFLOW (12301) during running. And able to works normal when return to set STACK_SIZE to 0x200.

Module Test #33 – CPU1 SW version check.

Test Specification:

Number of units used	Set up with 16 Nodes works, and tested at node 5.
Config file	stressTest2.ssplog
Alternative used	1

Perform a check of the code that checks the CPU2 SW version. The purpose is to verify that this code will behave correctly if CPU2 SW version error. It shall enter fatal error CPU2 SW version if CPU2 flash a mismatch SW version.

Procedure:

1. Flash both CPUs SW to let unit work normal.
2. Start the unit. The unit shall works normal, not give out fatal error.
3. Flash CPU1 with uncorrect SW version.
4. Restart the unit
5. The unit should enter fatal error INVALID_CPU1_VERSION(75).

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>

Following up procedure test, CPU1 able to enter fatal error INVALID_CPU1_VERSION(75) . Attached picture as below.

Module Test #34 – Logic valid check

Test Specification:

Number of units used	1
Config file	stressTest2.ssplog
Alternative used	None

Perform a check of the code that check without logic, invalid logic CRC and others kind parameters invalid should be detect. The purpose is to verify that this code will behave correctly if logic not exist, logic CRC invalid or other parameter not correct. The code check the logic header CRC, flash CRC, flash size, headerVersion, magicVal, ramDataAddress, slotTime, mainLoopFun valid, init valid, compilerGeneration and init() to init logic successful. A real logic invalid is difficult to trigger an logic error, so the SW has to inject a fake to trigger error.

Procedure:

1. Header CRC check. Start the unit. Run “j2.py sfe -e 22531 -s 1 -r” via usb, the unit should eneter fatal error CPU2LOGIC_HEADER_BAD_CRC (22531). Note that the code revert (~(LOGIC_HEADER.crc)) to campare with the code running calculated.
2. Flash CRC check. Start the unit. Run “j2.py sfe -e 22533 -s 1 -r” via usb, the unit should eneter fatal error CPU2LOGIC_BAD_CODE_CRC (22533).Note that the code revert ~LOGIC_HEADER.flashCRC to campare with the code running calculated.
3. Flash size check. Start the unit. Run “j2.py sfe -e 22534 -s 1 -r” via usb, the unit should eneter fatal error CPU2LOGIC_BAD_FLASH_SIZE (22534). Note that the code revert flashSize != ~0x4000 to campare with the header flash size.
4. headerVersion check. Start the unit. Run “j2.py sfe -e 22532 -s 1 -r” via usb, the unit should eneter fatal error CPU2LOGIC_HEADER_BAD_VERSION (22532).Note that the code revert

”(__LOGIC_HEADER__.common_header.header_version != ~MM_CONST_LOGIC_HEADER_VERSION)” to campare with the headerVersion.

5. magicVal check. Start the unit. Run “j2.py sfe -e 22530 -s 1 -r” via usb, the unit should eneter fatal error CPU2LOGIC_HEADER_BAD_MAGIC (22530). Note that the code revert ”(__LOGIC_HEADER__.common_header.magic_val != ~MM_CONST_MAGIC_MAIN_VAL)” to campare with the magicVal.
6. ramDataAddress check. Start the unit. Run “j2.py sfe -e 22539 -s 1 -r” via usb, the unit should eneter fatal error CPU2LOGIC_BAD_RAM_DATA_ADDRESS (22539). Note that the code revert ” __LOGIC_HEADER__.common_header.ram_data_address != ~__ram_logic_origin” to campare with the ramDataAddress.
7. slotTime check. Start the unit. Run “j2.py sfe -e 22540 -s 1 -r” via usb, the unit should eneter fatal error CPU2LOGIC_BAD_SLOT_TIME (22540). Note that the code use ”(__LOGIC_HEADER__.cfg_params.slottime_ms >= 1)” to campare with the slotTime.
8. Ram size valid check. In CPU2 Sconstruct, enable the SAFETY_TEST_INJECT = 0x98 test. Recompile and flash CPU1 via USB. Start the unit, the unit should eneter fatal error CONFIGURE_ERROR_9(169). Note that the code use ”(LOGIC_HEADER.sizeRAMData > 1024)” to fake ram size error, it shall 2048.
9. init valid check. Start the unit. Run “j2.py sfe -e 22541 -s 1 -r” via usb, the unit should eneter fatal error CPU2LOGIC_NO_INIT_FUNCTION (22541). Note that the code use ”(LOGIC_HEADER.common_header.init !=NULL)” to fake init valid error.
10. compilerGeneration check. Start the unit. Run “j2.py sfe -e 22543 -s 1 -r” via usb, the unit should eneter fatal error CPU2LOGIC_BAD_COMPILER_GENERATION (22543). Note that the code use ”(LOGIC_HEADER.common_header.compiler_generation == 0)” to compare with the revert compilerGeneration value.
11. Init() inited successful check. Start the unit. Run “j2.py sfe -e 22533 -s 1 -r” via usb, the unit should eneter fatal error CPU2LOGIC_INIT_FUNCTION_FAIL (22535). Note that the code use ”(LOGIC_HEADER.common_header.init() == 0)” to fake fatal error if init the logic successful.

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>

Module Test #35 – Cpu1 to Cpu2 init data check

Test Specification:

Number of units used	1
Config file	stressTest2.ssplog
Alternative used	1

Perform a check of the code that checks the initial data pack from CPU1 to CPU2. The purpose is to verify that this code will behave correctly check with verify the serial number,myNodeNr and networkID . It should be able to give out fatal error if there have one of invalid data.

Procedure:

1. Invalid node number. Start the unit. Run “j2.py sfe -e 12320 -s 1 -r” via usb, start the unit, the unit should enter fatal error CPU2COMMON_MY_ID_INVALID (12320). Note that CPU1 send out node number 0.
2. Invalid network ID. Run “j2.py sfe -e 12321 -s 1 -r” via usb, start the unit, the unit should enter fatal error CPU2COMMON_NETWORK_ID_INVALID (12321). Note that CPU1 send out network ID is 0xffffffff.
3. serialNumbers[0] not equal with networkID. Run “j2.py sfe -e 12322 -s 1 -r” via usb, start the unit, the unit should enter fatal error CPU2COMMON_MY_ID_MISMATCH (12322). Note that CPU1 send out network ID is ~settings.serialNumbers->serialNumbers[0];.

Test Execution:

Date	
Target SW CPU1	
Target SW CPU2	
Compiler SW	
Config file GIT commit ID	
Result	<pass>

Following the up test. The unit able to enter correspond CPU2 fatal error to indicate difference fatal error.