Cortex-M4
Structural Core Self-Test Library

Test Specification

Rev. 1.0.3
1 September 2017

Contents

1 Introduction 4
2 Software Unit Test Specification 4
M4_SCST_UT_T01 4
M4_SCST_UT_T02 5
M4_SCST_UT_T03 5
M4_SCST_UT_T05 6
M4_SCST_UT_T06 6
M4_SCST_UT_T07 7
M4_SCST_UT_T09 8
M4_SCST_UT_T10 9
M4_SCST_UT_T11 9
M4_SCST_UT_T12 10
M4_SCST_UT_T13 11
M4_SCST_UT_T14 12
M4_SCST_UT_T15 13
M4_SCST_UT_T16 14
M4_SCST_UT_T17 15

1 Introduction

This document contains the specification of tests to be performed for the Cortex-M4 SCST Library.

2 Software Unit Test Specification

Table 2-1. M4_SCST_UT_T01 Specification

Test ID	**M4_SCST_UT_T01**
Requirement ID	REQ_M4_SCST_04, REQ_M4_SCST_05, REQ_M4_SCST_36
Design ID	n/a
Test Type:	Functional, intended functionality
Derivation of Test Patterns:	White box
Test Method:	Analysis of requirements
Test Description:	Check whether core test execution result in fault-free case doesn't depend on register initial content. Execute every core test with different settings of general, configuration registers, and then check the test execution result and the global variable that stores its execution result.
Test Procedure:	1. Initialize general purpose registers (R4-R12), configuration registers (FAULTMASK, PRIMASK, BASEPRI, VTOR, CCR, SHPR1, SHPR2, SHPR3, SHCSR, CONTROL) according to different sets containing unique numbers. Note, that the CONTROL register is tested for two different bit1 configurations (MSP, PSP). 2. Set breakpoint in the m4_scst_execute_core_tests() function where every single atomic test execution returns. 3. Execute atomic tests one after another and for each test perform the verification steps 4 - 6. 4. Verify stop position. Verification point: execution should stop at breakpoint in step 2. 5. Verify execution result. Verification point: actual execution result should be equal to the documented test result. 6. Verify value of the m4_scst_accumulated_signature variable by comparing it to the corresponding documented result. Verification point: value of this global variable is equal to the expected accumulated result.
Test pass / fail criteria:	Test passes if: - Breakpoint in step 2 was reached. - Core test execution result and the global variable are as documented for different settings of initial register content. Fails otherwise.

Table 2-2. M4_SCST_UT_T02 Specification

Test ID	**M4_SCST_UT_T02**
Requirement ID	REQ_M4_SCST_12, REQ_M4_SCST_36
Design ID	
Test Type:	Functional, intended functionality
Derivation of Test Patterns:	White box
Test Method:	Analysis of requirements
Test Description:	Check whether core tests store / restore initial register content correctly. Execute every core test with different settings of general, configuration registers and check register content.
Test Procedure:	1. Initialize general purpose registers (R4-R12), configuration registers (FAULTMASK, PRIMASK, BASEPRI, VTOR, CCR, SHPR1, SHPR2, SHPR3, SHCSR, CONTROL) according to different sets containing unique numbers. 2. Store values of non-volatile, dedicated, configuration and control registers (GPRs (R4-R12), FAULTMASK, PRIMASK, BASEPRI, VTOR, CCR, SHPR1, SHPR2, SHPR3, SHCSR, CONTROL) in script variables. 3. Set breakpoint in the m4_scst_execute_core_tests() function where every single atomic test execution returns. 4. Execute atomic tests one after another and for each test perform the verification steps 5 - 6. 5. Verify stop position. Verification point: execution should stop at breakpoint in step 3. 6. Verify values of non-volatile, dedicated, configuration and control registers by comparing their contents to the stored values in step 2. Verification point: registers restored their content correctly.
Test pass / fail criteria:	Test passes if: - Breakpoint in step 3 was reached. - Registers content verified in step 6 are the same as the values stored in step 2. Fails otherwise.

Table 2-3. M4_SCST_UT_T03 Specification

Test ID	**M4_SCST_UT_T03**
Requirement ID	n/a
Design ID	n/a
Test Type:	Functional, unintended functionality
Derivation of Test Patterns:	White box
Test Method:	Error guessing
Test Description:	Check whether the code of core tests does not perform unintended read /

	write accesses to the memory. Intention of this test is to assure that core self-tests do not destroy data of other SW components and do not depend on data which does not belong to them. Pre-conditions: Allocate RAM memory for SCST library.
Test Procedure:	1. Set breakpoint in the m4_scst_execute_core_tests() function where every single atomic test execution returns. 2. Set READ/WRITE breakpoint to the RAM memory region which does not belong to the Cortex M4 SCST library. 3. Let the code execute further. 4. Verify stop position. Verification point: execution should stop at breakpoint in step 1, which means breakpoint in step 2 was not to be reached.
Test pass / fail criteria:	Test passes if none of the breakpoints in step 2 was reached. Fails otherwise.

Table 2-5. M4_SCST_UT_T05 Specification

Test ID	**M4_SCST_UT_T05**
Requirement ID	REQ_M4_SCST_14
Design ID	n/a
Test Type:	Functional, intended functionality
Derivation of Test Patterns:	White box
Test Method:	Fault injection
Test Description:	This test case to check whether SCST library triggers an exception by means of executing illegal opcode when execution control is incorrectly passed to the SCST library.
Test Procedure:	1. Set breakpoint at the usage fault handler which handles the event of executing an illegal opcode. 2. Execute an illegal opcode in the m4_scst_test_shell() function. - Modify value of variable “icst_cfm” to a different value compared to the one set by the m4_scst_test_shell() function. - Let the code execute further. 3. Verify stop position: Verification point: execution should stop at breakpoint in step 1, where handler of an illegal opcode is executed.
Test pass / fail criteria:	Test passes if execution control reached the breakpoints in step 1. Fails otherwise.

Table 2-6. M4_SCST_UT_T06 Specification

Test ID	**M4_SCST_UT_T06**
Requirement ID	REQ_M4_SCST_20
Design ID	n/a

Test Type:	Functional, unintended functionality
Derivation of Test Patterns:	White box
Test Method:	Analysis of requirements
Test Description:	This test case checks the mechanism of error injection into execution of SCST. Note: Error injection is not applied in case test. Execution is interrupted.
Test Procedure:	1. Set breakpoint in application where execution control returns from executing SCST. 2. Inject an error into the core test by setting the m4_scst_fault_inject_test_index variable to the core test index before calling the m4_scst_execute_core_tests() function. 3. Continue execution to reach breakpoint in step 1. 4. Verify result of executed test in case error injected by checking return value. Verification point: the content of the m4_scst_fault_inject_value variable is XOR-ed to the actual result returned by the executed test as well as to the redundant m4_scst_accumulated_signature variable. 5. Re-execute this atomic test without fault injection. 6. Verify execution result. Verification point: actual execution result as well as the content of the redundant m4_scst_accumulated_signature variable should be equal to the expected test result.
Test pass / fail criteria:	Test passes if: - The actual return value is equal to the XOR-ed operator of the m4_scst_fault_inject_value variable and the corresponding documented result. - The next SCST execution runs again normally without fault injection. Fails otherwise.

Table 2-7. M4_SCST_UT_T07 Specification

Test ID	**M4_SCST_UT_T07**
Requirement ID	REQ_M4_SCST_10, REQ_M4_SCST_11
Design ID	
Test Type:	Functional, intended functionality
Derivation of Test Patterns:	White box
Test Method:	Analysis of requirements
Test Description:	Check whether the code of core tests, in case of no unexpected exception/interrupt, does not modify the m4_scst_test_was_interrupted variable to a non-zero value. Execute all atomic tests individually one after another without triggering unexpected exception/interrupt.
Test Procedure:	1. For all atomic tests repeat steps 2-3. 2. Request execution of single atomic test.

	3. Verify the m4_scst_test_was_interrupted variable. Verification point: value of the m4_scst_test_was_interrupted variable must be zero.
Test pass / fail criteria:	Test passes if the m4_scst_test_was_interrupted variable was not changed to a non-zero value. Fails otherwise.

Table 2-9. M4_SCST_UT_T09 Specification

Test ID	**M4_SCST_UT_T09**
Requirement ID	REQ_M4_SCST_08, REQ_M4_SCST_11
Design ID	
Test Type:	Functional, intended functionality
Derivation of Test Patterns:	White box
Test Method:	Analysis of requirements
Test Description:	Check whether the code of core tests, in case of no unexpected exception/interrupt, correctly sets the m4_scst_test_was_interrupted variable and the m4_last_executed_test_number variable for user application. Request execution of multiple atomic tests without occurrence of unexpected interrupt.
Test Procedure:	1. Set breakpoint in application where the m4_scst_execute_core_test() function returns. 2. Set breakpoint at the address of the last requested atomic test. 3. Request execution of multiple atomic tests in which the first requested test will generate an interrupt inside its code. 4. Verify stop position. Verification point: the execution should stop at breakpoint in step 2. 5. Continue to execute 6. Verify stop position. Verification point: the execution should stop at breakpoint in step 1. 7. Verify setting of the m4_scst_test_was_interrupted variable and the m4_scst_last_executed_test_number variable. Verification point: Value of the m4_scst_test_was_interrupted variable must be zero. Value of the m4_last_executed_test_number variable must be equal to the test index of the last requested test in step 3. 8. Verify execution result. Verification point: the return value of executing the m4_scst_execute_core_tests() function must not be equal to the M4_SCST_TEST_WAS_INTERRUPTED constant.
Test pass / fail criteria:	Test passes if: - The breakpoints in step 2 and step 1 were reached. - Value of the m4_scst_test_was_interrupted variable must be zero. - Value of the m4_last_executed_test_number must be equal to the test index of the last requested atomic test. - The test should not return the value of the

	M4_SCST_TEST_WAS_INTERRUPTED constant. Fails otherwise.

Table 2-10. M4_SCST_UT_T10 Specification

Test ID	**M4_SCST_UT_T10**
Requirement ID	REQ_M4_SCST_02
Design ID	n/a
Test Type:	Functional, intended functionality
Derivation of Test Patterns:	White box
Test Method:	Analysis of requirements
Test Description:	Check whether the m4_scst_execute_core_tests() function performs correctly checking of correctness of input parameters. Check whether the m4_scst_execute_core_tests() function shall return value of the M4_SCST_WRONG_RANGE constant in case of the start and end parameters are incorrect. - Index of the first atomic test (start) greater than index of the last atomic test (end). - Index of the last atomic test (end) greater than the available number of atomic tests.
Test Procedure:	1. Set breakpoint in application where the m4_scst_execute_core_test() function returns. 2. Set breakpoint at the address of the first requested test. 3. Pass wrong input parameters to the m4_scst_execute_core_tests() function. Let the code execute further. 4. Verify stop position. Verification point: execution should stop at breakpoint in step 1. 5. Verify return value of the m4_scst_execute_core_tests() function. Verification point: return value of the m4_scst_execute_core_tests() function should be equal to the M4_SCST_WRONG_RANGE constant.
Test pass / fail criteria:	Test passes if: - Breakpoint in step 1 is reached. - The return value of the m4_scst_execute_core_tests() function should be equal to the M4_SCST_WRONG_RANGE constant. - No core test was executed. Fails otherwise.

Table 2-11. M4_SCST_UT_T11 Specification

Test ID	**M4_SCST_UT_T11**
Requirement ID	REQ_M4_SCST_08
Design ID	n/a
Test Type:	Functional, intended functionality

Derivation of Test Patterns:	White box
Test Method:	Analysis of requirements
Test Description:	Check whether the m4_scst_execute_core_tests() function performs correctly checking of correctness of input parameters. Verify whether the m4_scst_execute_core_tests() function returns correct number of the last executed test number in case multiple tests were requested for execution.
Test Procedure:	1. Set breakpoint in application where the m4_scst_execute_core_test() function returns. 2. Set breakpoint at the address of the first requested test. 3. Pass correct input parameters to the m4_scst_execute_core_tests() function to request multiple atomic tests. Let the code execute further. 4. Verify stop position. Verification point: execution should stop at breakpoint in step 2. 5. Continue to execute. 6. Verify stop position. Verification point: execution should stop at breakpoint in step 1. 7. Verify the m4_scst_last_executed_test_number variable. Verification point: the m4_scst_last_executed_test_number variable must be equal to the test index of the last requested test in step 3.
Test pass / fail criteria:	Test passes if: - Breakpoint in step 2 and step 1 was reached in turn. - The m4_scst_last_executed_test_number variable was set to the test index of the last requested test. Fails otherwise.

Table 2-12. M4_SCST_UT_T12 Specification

Test ID	**M4_SCST_UT_T12**
Requirement ID	REQ_M4_SCST_06, REQ_M4_SCST_08
Design ID	n/a
Test Type:	Functional, intended functionality
Derivation of Test Patterns:	White box
Test Method:	Analysis of requirements
Test Description:	Check whether the m4_scst_execute_core_tests() function returns the execution result of the core tests correctly and correctly sets the m4_scst_last_executed_test_number variable. Request execution of both single and multiple atomic tests. Do not apply fault injection - let the core tests produce expected results.
Test Procedure:	1. Set breakpoint in application where the m4_scst_execute_core_test() function returns. 2. Request execution of a single or multiple atomic tests. Let the code execute further. 3. Verify stop position. Verification point: Execution should stop at breakpoint in step 1.

	4. Verify execution result of the single/multiple requested atomic test. Verification point: - In case of single atomic test was requested, the returned result as well as the content of the redundant m4_scst_accumulated_signature variable should be equal to the corresponding documented result. - In case of multiple atomic tests were requested, the returned result as well as the content of the redundant m4_scst_accumulated_signature variable should be the XOR-ed value of documented results of the requested tests in step 2. 5. Verify setting of the m4_scst_last_executed_test_ number variable. Verification point: the m4_scst_last_executed_test_number variable must contain the test end index of the inputted range in step 2.
Test pass / fail criteria:	Test passes if: - Breakpoint in step 1 was reached - In step 4, the returned value is as expected in both cases. - The m4_scst_last_executed_test_number variable was set to the test end index of the requested range in step 2. Fails otherwise.

Table 2-13. M4_SCST_UT_T13 Specification

Test ID	**M4_SCST_UT_T13**
Requirement ID	REQ_M4_SCST_03, REQ_M4_SCST_08
Design ID	n/a
Test Type:	Functional, intended functionality
Derivation of Test Patterns:	White box
Test Method:	Analysis of requirements
Test Description:	Check whether the m4_scst_execute_core_tests() function returns execution result of the core tests correctly and correctly sets the m4_scst_last_executed_test_number variable. Request execution of multiple atomic tests. Corrupt the result returned by the first core test.
Test Procedure:	1. Set breakpoint in application where the m4_scst_execute_core_test() function returns. 2. Set breakpoint in the m4_scst_execute_core_tests() function where every single atomic test returns. 3. Request execution of multiple atomic tests. 4. Verify stop position. Verification point: execution should stop at breakpoint in step 2. 5. When execution reaches the breakpoint in step 2, replace the correct result by an incorrect result. Let the code execute further. 6. Verify stop position. Verification point: execution should stop at breakpoint in step 1. 7. Verify execution result. Verification point: the actual returned result should be the intended incorrect result.

	8. Verify setting of the m4_scst_last_executed_test_number variable. Verification point: the m4_scst_last_executed_test_number variable must be equal to the test start index of the requested range in step 3.
Test pass / fail criteria:	Test passes if: - Breakpoints in step 2 and step 1 were reached. - In step 7, the intended incorrect result was returned. - The m4_scst_last_executed_test_ number variable was set to the index of the corrupted test in step 3. Fails otherwise.

Table 2-14. M4_SCST_UT_T14 Specification

Test ID	**M4_SCST_UT_T14**
Requirement ID	REQ_M4_SCST_22, REQ_M4_SCST_23
Design ID	n/a
Test Type:	Functional, unintended functionality
Derivation of Test Patterns:	White box
Test Method:	Error guessing
Test Description:	Check whether M4 SCST tests which can be executed in unprivileged mode do not access memory intended for privileged mode access only. Pre-conditions: Place privileged SCST tests code in memory region which configured for privileged access only. Place unprivileged SCST tests code in memory region which configured to can be accessed by both privileged and unprivileged mode (full access). Execute each M4-SCST test in two cases. - Request execution of the unprivileged SCST tests in unprivileged mode to check whether these tests ever access to the memory region which allows privileged access only. - Request execution of all M4-SCST tests in privileged mode to verify that no exception happens caused by an unprivileged access.
Test Procedure:	1. Enabling module Memory Protected Unit (MPU) then switching processor state to unprivileged-mode. 2. Set breakpoint at the memory manage handler – which handles the event of accessing privileged memory region while executing in unprivileged mode. 3. Set breakpoint in the m4_scst_execute_core_tests() function where every single atomic test returns. 4. Execute unprivileged tests one after another and for each test perform the verification step 5. 5. Verify stop position Verification point: execution should stop at breakpoint in step 3. 6. Switch processor to privileged-mode then re-execute all SCST tests to check that no exception happens caused by an unprivileged access.

	7. Verify stop position. Verification point: execution should stop at breakpoint in step 3.
Test pass / fail criteria:	Test passes if: - Unprivileged SCST tests do not access memory region which dedicated for privileged access only. - All SCST tests were successfully executed in privileged mode. Fails otherwise

Table 2-15. M4_SCST_UT_T15 Specification

Test ID	**M4_SCST_UT_T15**
Requirement ID	REQ_M4_SCST_12, REQ_M4_SCST_36
Design ID	n/a
Test Type:	Functional, intended functionality
Derivation of Test Patterns:	White box
Test Method:	Analysis of requirements
Test Description:	Trigger external interrupt on every assembly instruction of the SCST test and check whether the content of non-volatile and dedicated registers is restored correctly.
Test Procedure:	1. Go to the start address of the individual SCST test. 2. Read the content of the non-volatile general purpose registers (R4-R12) and dedicated registers (CCR, SHPR1, SHPR2, SHPR3, BASEPRI). Read content of CONTROL register and both stack pointers (PSP, MSP). 3. Initialize the non-volatile registers (R4-R12) with a set of unique numbers (see step 15). 4. Place breakpoint on the m4_scst_execute_core_tests() function where every atomic test returns. 5. Continue execution till the *n*-th instruction of the test is reached (see the step 14). 6. Trigger the external interrupt. 7. Continue execution till the IRQ handler corresponding to the external interrupt triggered in the step 6. 8. Verify stop position. Verification point: Execution should stop at the IRQ handler corresponding to the external interrupt triggered in the step 6. 9. Verify restoring of the dedicated registers. Verification point: The registers were correctly restored (i.e. contains initial values read in the step 2). 10. Let the code execute further till the breakpoint set in the step 4 is reached. 11. Verify stop position. Verification point: Execution should stop at the breakpoint in step 4. 12. Verify restoring of the non-volatile general purpose registers (R4-R12). Verify restoring of the CONTROL register and both stack pointers (PSP, MSP).

	Verification point: The registers were correctly restored (i.e. contains initial values written in the step 3). 13. Restore the original content of non-volatile registers (R4-R12) obtained in the step 2. 14. Repeat the steps 1-13 for $n = 1, 2, \ldots N$ (see the step 5), where N is the total count of the instructions executed within a given SCST test. 15. Repeat the steps 1-14 for two mutually inverted sets applied in the step 3 so that each bit in each initialized register is written with both 0 and 1. 16. Repeat the steps 1-15 for all of atomic tests. 17. Run steps 1-16 for both stack pointers configurations (PSP, MSP).
Test pass / fail criteria:	Test passes if: - Execution reaches the external IRQ handler - Dedicated registers are restored properly - Execution reaches the breakpoint in step 4 - Non-volatile registers (R4-R12) were correctly restored - The CONTROL register and stack pointers (PSP, MSP) were restored correctly. Fails otherwise.

Table 2-16. M4_SCST_UT_T16 Specification

Test ID	**M4_SCST_UT_T16**
Requirement ID	REQ_M4_SCST_09, REQ_M4_SCST_10, REQ_M4_SCST_11, REQ_M4_SCST_36
Design ID	n/a
Test Type:	Functional, intended functionality
Derivation of Test Patterns:	White box
Test Method:	Analysis of requirements
Test Description:	Verify that the SCST Library exceptions which were entered by tail-chaining the user application interrupts were not serviced by the user application interrupt service routines. Furthermore, verify that the user application interrupts were forwarded to the user application service routines.
Test Procedure:	1. Prepare user application to have timer interrupt with highest priority. Configure this interrupt to occur at every stick count during SCST execution. 2. Declare a global variable which counts the number of events when there are two exceptions with the same priority at the same time (one exception from the SCST library and the second one from the timer configured in step 1).

	3. Declare a global variable which counts number of SCST interrupts which were wrongly serviced by the user application interrupt services routines. 4. Declare a global variable which counts the number of wrong returned SCST signatures (other than documented signature or M4_SCST_TEST_WAS_INTERRUPTED signature) 5. Request execution of each SCST atomic test in long time then perform the verification step 6 – 8. 6. Verify the global variable in step 2. Verification point: Value of this counter variable must not be zero (it proves that the worst situation occurred several times (the external interrupt and the SCST exception were triggered at the same time). 7. Verify the global variable in step 3. Verification point: Value of this counter variable must be zero (it proves that every exception which was triggered by the SCST Library is handle by the SCST library, not by the user IRQ handler). 8. Verify the global variable in step 4. Verification point: Value of this counter variable must be zero (it proves that SCST library always returns the documented signature). 9. Repeat the steps 1-8 for all of atomic tests which intentionally trigger exceptions during their execution. 10. Run steps 1-9 for both stack pointers MSP, PSP configurations.
Test pass / fail criteria:	Test passes if: - Value of the counter variable in step 6 is not zero. - Value of the counter variable in step 7 is zero. - Value of the counter variable in step 8 is zero. Fails otherwise.

Table 2-17. M4_SCST_UT_T17 Specification

Test ID	**M4_SCST_UT_T17**
Requirement ID	REQ_M4_SCST_09, REQ_M4_SCST_36
Design ID	n/a
Test Type:	Functional, intended functionality
Derivation of Test Patterns:	White box
Test Method:	Analysis of requirements
Test Description:	Check whether the SCST library is able to handle the occurrence of nested interrupt during its execution.
Test Procedure:	1. Prepare user application to have timer interrupt with highest priority. Configure this interrupt to occur at every stick count during SCST execution. 2. Declare three global counter variables. The first variable is updated before the occurrence of nested-interrupt. The second variable is updated inside the user IRQ handler of the nested interrupt. The last one is update after nested-interrupt is served. 3. Catch the event of there is another pending exception which is

	trigged by the SCST library inside the user timer IRQ handler. 4. Request execution of the SCST library and wait until the situation in step 3 is reached, then perform the nested-interrupt triggering in step 5. 5. Trigger Non-Maskable Interrupt to emulate the occurrence of nested interrupt. Continue to execute… 6. Let the SCST library to execute in long time to be sure that nested interrupt (described in steps 3 – 5) was triggered several times. Perform the verification in step 7. 7. Verify content of the three variables declared in step 2. Verification point: All three variables have the same value and none of these variables is zero. 8. Repeat the steps 1-7 for all of atomic tests which intentionally trigger exceptions during their execution. 9. Run steps 1-8 test for both stack pointers MSP, PSP configurations.
Test pass / fail criteria:	Test passes if: - All three global variables have the same value and none of these variables is zero to be sure that nested interrupt was triggered several times and the SCST Library successfully forwarded nested interrupts to its interrupt service routines. Fails otherwise.

How to Reach Us:

Home Page:
www.nxp.com

Web Support:
www.nxp.com/support