STRX RFE SW Test Architecture

|  |  |  |  |
| --- | --- | --- | --- |
| reversion | Author | Date | Description |
| V0.2 | Dongyu Gao | April.29.2022 |  |
| V0.1 | Biju Ravindran | July.28.2021 | Initial draft version of test plan |

Table of contents

I. Introduction 5

I.1 Objectives / Purpose 5

I.2 Test Scope 5

I.3 Audience 6

II. Test Classification and Coverage 7

II.1 Functional Test 7

II.1.1 RFE Unit Test 7

II.1.1.1 Definition 7

II.1.1.2 Methodology 7

II.1.1.3 Test Items 9

II.1.2 RFE Build-In Self-Test(BIST) 11

II.1.2.1 Definition 11

II.1.2.2 Methodology 11

II.1.2.3 Test Items 11

II.1.2.3.1 TX BIST 11

II.1.2.3.2 RX BIST 11

II.1.2.3.3 ADC Test Pattern 12

II.1.2.3.4 PDC Test Pattern 12

II.1.2.3.5 Packer Test Pattern 12

II.1.2.4 Test Coverage Analysis 12

II.1.3 RFE System Test 13

II.1.3.1 Definition 13

II.1.3.2 Methodology 13

II.1.3.3 Test Items 13

II.1.3.3.1 Single Profile 13

II.1.3.3.2 Multiple Profiles 13

II.1.3.3.3 DDMA 14

II.1.3.3.4 TDMA 14

II.1.3.3.5 CDMA 14

II.1.3.3.6 Frequency Drift 15

II.1.3.3.7 Customers Specific Use Cases 15

II.1.3.4 Test Coverage Analysis 15

II.1.4 RFE Proxy Test 16

II.1.4.1 Definition 16

II.1.4.2 Methodology 16

II.1.4.3 Test Items 16

II.1.4.4 Test Coverage Analysis 16

II.1.5 A53 System Test 17

II.1.5.1 Definition 17

II.1.5.2 Methodology 17

II.1.5.3 Test Items 17

II.1.5.4 Test Coverage Analysis 17

II.1.6 Functional Safety(FUSA) Test 18

II.1.6.1 Definition 18

II.1.6.2 Methodology 18

II.1.6.3 Test Items 18

II.1.6.3.1 Fault Injection 18

II.1.6.3.2 FCCU/ISM Errors 18

II.1.6.4 Test Coverage Analysis 19

II.1.7 Integration Test(TO DO) 21

II.1.8 Qualification Test(TO DO) 22

II.2 Integration Test 23

II.2.1 Definition 23

II.2.2 Methodology 23

II.2.3 Test Items 23

II.2.4 Test Coverage Analysis 23

II.3 Regression Test 24

II.3.1 Definition 24

II.3.2 Methodology 24

II.3.3 Test Items 24

II.3.4 Test Coverage Analysis 24

II.4 Black Duck Check 25

II.4.1 Definition 25

II.4.2 Methodology 25

II.4.3 Test Items 25

II.4.4 Test Coverage Analysis 25

II.5 Overall Test Coverage 27

II.6 Test Environment Requirements 28

III. Test Architecture 29

III.1 Test Items 29

III.2 Risks, Assumptions and Constraints 29

III.3 Stakeholders and Communication 29

III.4 Resources 30

III.5 Testing Activities 30

III.5.1 Test Execution Completion Criteria 30

III.5.2 Test Execution Suspension Criteria and Resumption Requirements 30

III.6 Schedule 30

IV. Acceptance Reviews and Approvals 31

V. Annexes 32

VI. Document Information 33

VI.1 References 33

VI.2 Terms/Acronyms and Definitions 33

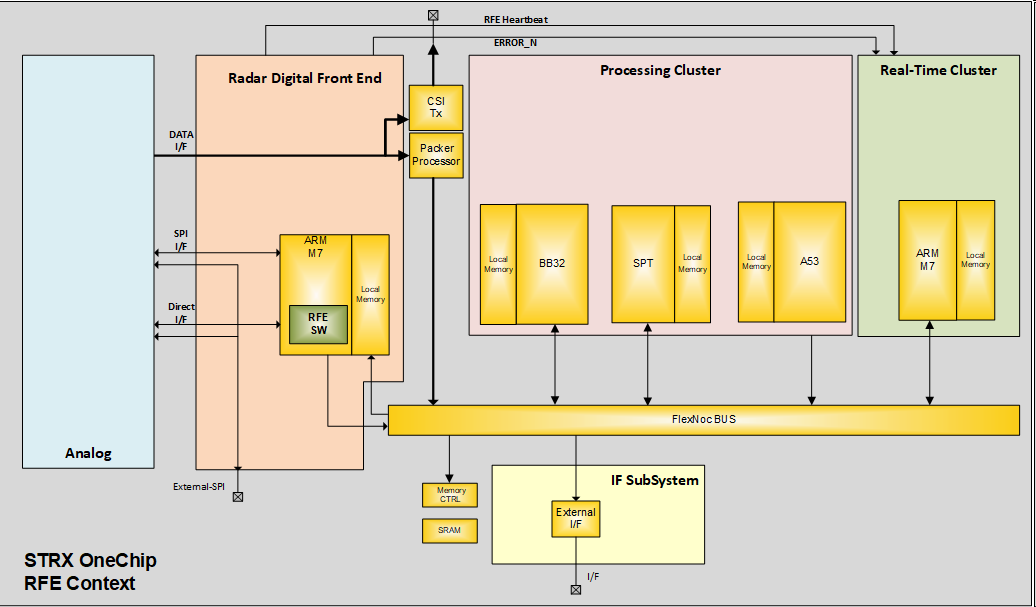
# Introduction

## Objectives / Purpose

This document describes the test architecture of Radar Front End Software (RFE SW) Subsystem. It defines the test classifications, methodology, and the tasks required for the product testing process. It also defines the infrastructure for both software and hardware environment needed during tests development, execution and results reporting. This document provides a guideline to test developer a guideline, to standardize the test code and the appearance of the test result. The core idea of our SW test work is to, filter out the bugs and resolve them within SW team, then guarantee the quality of each release. All of our effort within this test architecture is going to serve the core idea.

## Test Scope

This document is only applied to the RFE SW of STRX OneChip project. See the picture below how the RFE SW fits in the whole STRX OneChip. STRX family also includes STRX Remote, STRX



This section details the testing scope for RFE Software, what is in scope and what is out-of-scope. Below are the list of activities planned for testing RFE software. RFE SW code Build including regression test as part of the build process

RFE SW unit test for all units of rfeFw based on Unity running on RFE-M7*.* RFE SW integration test for all integration test cases/scenarios also based on Unity running on APP-A53.RFE functionality will be tested using use case APIs and the raw radar data is captured for offline analysis in Matlab. APP-A53 to RFE-M7 communication through IPCF will be tested by RFE APIs. RFE-M7 Infra IP drivers will be tested using infra APIs. Testing covers the integration, qualification and RFE reference applications.

Out of scope is the testing of the actual behavior of all the HW IP blocks.

## Audience

The main audience is the FW and test code developer, they can use this document as a tool book or guide book to generate test code in the same standard. The other audience consists of Project Manager, Safety Manager, Software Architect, Configuration Manager and Quality Assurance, help them to understand our test procedure and provide improvement advices.

# Test Architecture

## Test Items

RFE Software: This constitutes of the software which runs on RFE-M7 (rfeFw) and any client software that runs on host core (rfeDriver).

The test items are listed below;

1. rfeFw

* RFE Hw units: (rfeHw[Unitname]) includes all the analog and digital IP HW blocks and their drivers of the RFE subsystem
* RFE Sw units: (rfeSw[Unitname] all Software units like Bist Startup Calibration Clocking Main Statemachine Power Manager etc.
* RFE units: (rfe[Unitname] All software units that interface to the outside world and global supporting libraries. (like DSP and Math function)

2. rfeDriver - SW running on a remote processor for RFE Abstract API

## Risks, Assumptions and Constraints

1. Pre-Zebu testing using M3SA simulation for the RFE Digital Subsystem. . This doesn’t include the RFE Analog IPs. M3SA will come in different versions with incremental functionality as RTL implementation progresses. Simulation speed is a limitation.
2. Pre-silicon testing on ZeBu Emulation platform for RFE Digital Subsystem. This doesn’t include the RFE Analog IPs. Zebu simulation accuracy is a limitation. Simulation speed is also a limitation.

1. C-Simulation environment for RFE simulation of analog (AMS models), RFE digital access and control.
2. RFE Analogue model developed in a simulated environment is provided for testing.

## Stakeholders and Communication

*The Integration and Validation team communicates the activity status during project meetings defined in Software Project Management Plan [040.20\_SmartTRX\_SW\_PMP]*

*Stakeholder information is detailed in Software Project Management Plan [040.20\_SmartTRX\_SW\_PMP]*

*The test team communicates the test results (test reports, defects) to the entire project team (development team, project manager, quality assurance, safety manager), at the end of the test campaign or when necessary.*

## Resources

(To Do)

## Testing Activities

This chapter is intended primarily for testers. It provides chronological listing of all activities performed during test campaign, and it refers all other important information needed by tester.

Please refer STRX overall test plan doc section 7.1

### Test Execution Completion Criteria

Please refer STRX overall test plan doc section 7.1

### Test Execution Suspension Criteria and Resumption Requirements

Please refer STRX overall test plan doc section 7.1

## Schedule

(TO DO)

# Test Classification and Test Environment

There’s mainly 4 types of tests shall be done: functional test, integration test, regression test, and black duck scan. They checked RFE SW from different aspect and provide different coverage. This chapter start with functional test, because it is the basic cell of other test type. Then is the integration and regression test, they are similar in implementation. Black duck is the last type to introduce. By next chapter, the relation of different test type is introduced, to help reader understand the test framework. The last section introduces the test environment.

## Functional Test

Functional Test is a type of software testing that validates the software system against the functional requirements/specifications. The purpose of Functional tests is to test each function of the software application, by providing appropriate input, verifying the output against the Functional requirements. As defined [Test Scope](#_Test_Scope) in previous chapter, our functional tests are limit to the RFE firmware, RFE Proxy and it’s client, and A53 firmware. At IP level we have unit tests, at SoC level we have BIST tests, and we also develop system test to cover customer use cases. Due to FUSA is a group of designed ‘function’, FUSA test is also parts of the functional test.

### RFE Unit Test

#### Definition

RFE SW units are the smallest piece of code that can be logically isolated in RFE SW. RFE SW units are classified into 3 types: HW, SW, and General units. HW units’ are focusing on an individual functional module, they are usually one to one correspond to a HW IP module, with the same module name. SW units’ APIs are used to control functions across multiple functional IP modules. General Units’ APIs doesn’t control any IP module, but used by test code for general purpose, like print debug information, signal processing, and add delay in code.

#### Methodology

Unit testing is done with the Unity Framework (<https://docs.unity3d.com>). This provides the interface for automated testing. For each unit a unit test app is created with the Unity framework.

The automated unit tests will be executed each time code is pushed to repository of a branch.

#include "unity.h"

#include "unity\_fixture.h"

#include "rfeHwCrc.h"

#define REF\_RESULT 0x29058c73

void rfe\_hw\_crc\_tc\_001\_calBuf(void)

{

uint32\_t crc;

uint32\_t \*d32;

uint8\_t data[256];

RFE\_ERROR\_CREATE;

// init 256 byte message

// The 32 bit CRC of this message is REF\_RESULT 0x29058c73

for(int i = 0; i < 256; i++)

{

data[ i ] = i;

}

d32 = (uint32\_t \*)(data);

crc = rfeHwCrc\_calBuf( d32 , (256>>2), RFE\_ERROR\_FUNCTION\_ARGUMENT);

TEST\_ASSERT\_EQUAL\_HEX32( crc, REF\_RESULT);

}

This is a simple example hoe to write a Unity unit test at the top level:

#include "unity.h"

#include "unity\_fixture.h"

#include <stdint.h>

#include "testHwCrc\_internal.h"

TEST\_GROUP( Crc );

TEST\_SETUP( Crc )

{

}

TEST\_TEAR\_DOWN( Crc )

{

}

TEST( Crc, Init )

{

}

/\*!

\* Test of rfeHwCrc\_calBuf function

\*/

TEST( Crc,calBuf )

{

rfe\_hw\_crc\_tc\_001\_calBuf();

}

TEST\_GROUP\_RUNNER( Crc )

{

RUN\_TEST\_CASE( Crc, calBuf );

}

Each test (in this case rfe\_hw\_crc\_tc\_001\_calBuf()) simply ends its code with an assert which communicates the test result to the higher level(s):

#### Test Items

Table in below introduce the modules that shall be covered in the unit test.

|  |  |  |
| --- | --- | --- |
| HW Units | SW Units | General Units |
| ACC | Init | DbgPrintf |
| ATB | Calibration | DspMath |
| Chirp PLL | Configure Manager(CfgMngr) | Wait |
| CRC | Clocking |  |
| CSI2 | MainFSM |  |
| FCCU | OcOTP |  |
| FLEXIO | Power Manager(PwrMngr) |  |
| GBIAS | Utils |  |
| GLDO |  |  |
| LLDO |  |  |
| LOIF |  |  |
| LP UART |  |  |
| MCGEN |  |  |
| NVIC |  |  |
| Packer |  |  |
| PDC |  |  |
| PIT |  |  |
| PPD |  |  |
| RCOSC |  |  |
| Reg |  |  |
| RX |  |  |
| RXADC |  |  |
| RXBIST |  |  |
| TE |  |  |
| TSENSE |  |  |
| TX |  |  |

Each unit has their unique criteria, which are defined by unit developers.

### RFE Build-In Self-Test(BIST, new name to discuss)

#### Definition

As the name of BIST test described, it’s a mechanism that permits a machine to test itself. BIST is an important function safety design in STRX to ensure the chip track its own status on the road, and also plays an important role during validation. Customer may also apply BIST between frames in implementation. BIST are tests at SoC level, which can test the integration behaviour of some IP modules and indicate the issues that unit test cannot achieve.

#### Methodology

Most of the BIST test generates RF signal, IF signal, or test pattern. Data are stored in the shared memory. So there’s a general flow for the BIST tests:

Step 1: Configure and Trigger BIST signal

Step 2: Capture data from shared memory

Step 3: Signal Processing(optional)

Step 4: Assert Results

Test environment: rfeProxy + Python Client + Lauterbach.

#### Test Items

##### TX BIST

TX BIST is a function integrity check for TX chain, it is verified against any random hardware permanent fault.

Test Criteria:

* (TODO)

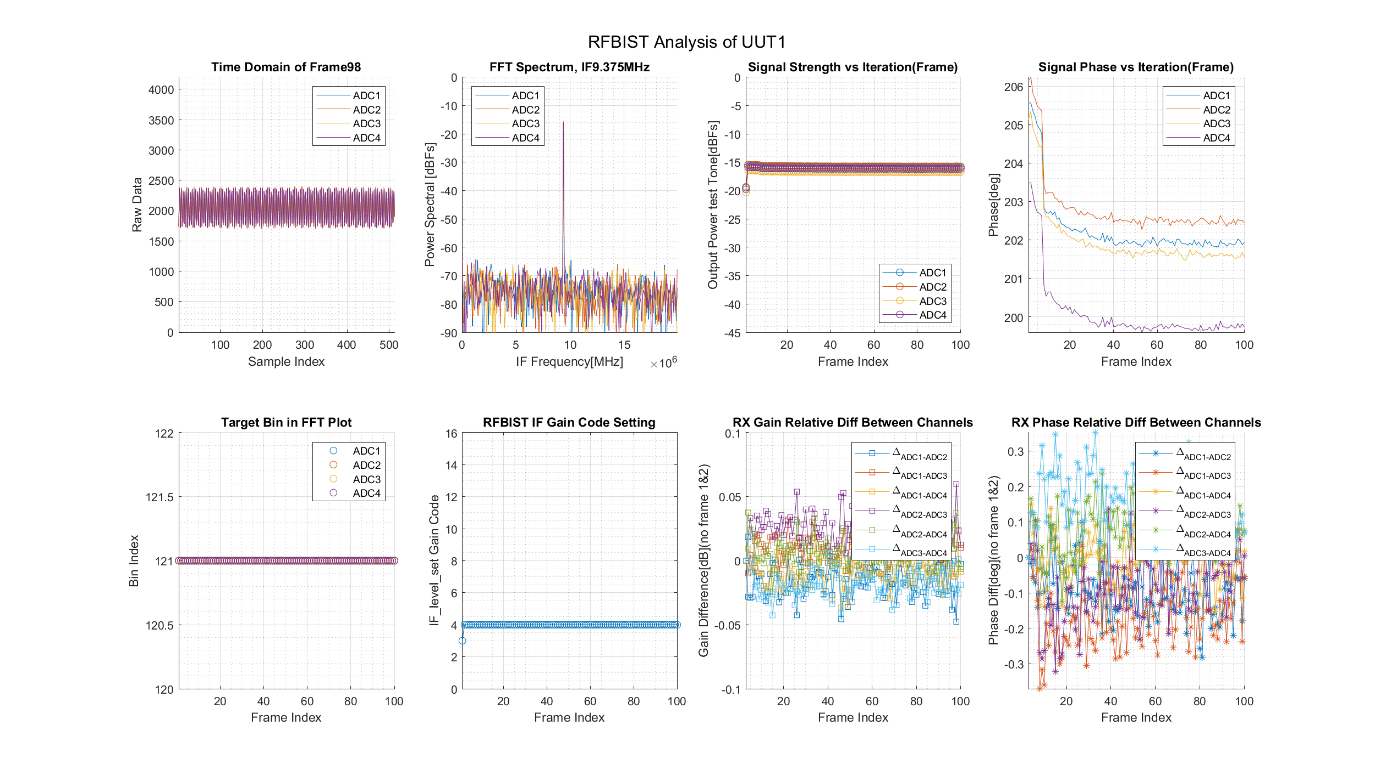
##### RX BIST

RX BIST is a function integrity check for RX chain, it is verified against any random hardware permanent fault.

Test Criteria:

* Test Tone is at target IF frequency.
* 3 mode are as expected: double Sin/cosin wave, double square wave, and DC controlled mode.
* SNR is high enough.
* No second harmonic.
* No FCCU error.

Example:



1. RX BIST Test Result

In example, the RX BIST test result is processed and plotted from multiple aspects, the top left two plots are ‘must have’ for RX BIST test report, other information are logged for further analysis.

##### ADC Test Pattern

(TO DO)

##### PDC Test Pattern

(TO DO)

##### Packer Test Pattern

(TO DO)

#### Test Coverage

(TO DO)

### RFE System Test

#### Definition

System test is to repeat customer use cases, and check the whole RFE FW integration behaviour in a comprehensive way. Most of HW and SW units are utilized in this test, the whole RFE M7 system are configured and working in this test, thus it is called system test. FCCU/ISM errors shall be tracked in every system test.

#### Methodology

(TO DO)

#### Test Items

##### Single Profile

Single Profile test is fundamental case in system test.

Test Criteria:

* 1G/2G/4G BW work without any issue.
* Up-chirp/down-chirp work without any issue.
* Target Detection Correct.
* Phase is stable.
* Target Bin is stable.
* No issue with multiple chirps, sequences, and frames.
* No FCCU error.

##### Multiple Profiles

Multiple profiles are used widely by customer in real scenarios.

Test Criteria:

* VCO switches between profiles cause no issue.
* Centre frequency and BW switch between profiles cause no issue.
* Up-chirp and down-chirp switch between profiles cause no issue.
* Target Detection Correct.
* Phase is stable.
* Target Bin is stable.
* No FCCU error.

##### DDMA

DDMA is Doppler-Division Multiple Access. In SW test, it’s implemented by phase change from chirp to chirp, or from frame to frame.

Test Criteria:

* Init phase update in 8 profiles via 1 TX, absolute phase difference as expected.
* Init phase update 0/45/90/135° in 4 profiles via 4 TX, absolute phase difference as expected.
* Phase Update from chirp to chirp via 1 TX 2.815/19.6875/42.1875/87.1875°, phase update rate from chirp to chirp shall be within ±0.5° tolerance.
* Sequential multimode: 4x4 DDMA(phase change to be discuss) + 4x4 DDMA(phase change to be discuss)
* Sequential multimode: 4x4 DDMA(0/45/90/135°) + 4x4 TX beamforming (constant phase rotator setting)
* No FCCU error.

##### TDMA

TDMA is Time-Division Multiple Access. In SW test, it’s implemented by turning ON/OFF TXs in different Profiles.

Test Criteria:

* 4 TXs can be on/off switched individually in 4 profiles.
* No FCCU error.

##### CDMA

CDMA is Code-Division Multiple Access. In SW test, it’s implemented by changing the phase of chirp or frame, very similar to DDMA test.

Test Criteria:

* Mapping code to absolute phase changes via 8 profiles 1 TXs, chirp sequencing set to be 256 long code info, check the receive signal phase to be expected phase. Example, use TX phase rotation to encode ‘RFE SW TEST TEAM’, by parsing the phase of receiver, the code shall also be ‘RFE SW TEST’
* No FCCU error.

##### Frequency Drift

Frequency drift is a non profile based function, the center frequency keep accumulate itself. The frequency drift will generate a doppler shift, similar to DDMA but much smaller.

Test Criteria:

* Average Phase update rate shall to be expected.
* No FCCU error.

##### Customers Specific Use Cases

Every Customer has their own specific use cases and shall be tested.

Test Criteria:

* No issue with Aptiv use cases
* No issue with Bosch use cases
* No issue with Continental use cases
* No issue with Cubtek use cases
* No issue with Denso use cases
* No issue with Hella use cases

##### RFE FW Test

### RFE Proxy Test

#### Definition

RFE Proxy is an important glue between PC applications and STRX RFE FW. It is used by validation team, designer, and RFE SW test. By RFE Proxy test, we ensure the internal users can use all of the API functions from PC terminal. Both MATLAB and Python client shall be tested.

#### Methodology

(Content)

#### Test Items

##### MATLAB client own function test

MATLAB client has rfeabstract and low level functions, each of them shall be functional.

Test Criteria:

* All rfeabstract and low level functions are effective via both FTDI and Lauterbach
* No rfeProxy self-defined error
* No FCCU error

##### MATLAB client all RFE API test

MATLAB client has rfeapi library, which contains all of the units in RFE SW to call.

Test Criteria:

* All rfeapi functions are effective via both FTDI and Lauterbach
* No rfeProxy self-defined error
* No FCCU error

##### Python client own function test

Python client has rfeabstract and low level functions, each of them shall be functional.

Test Criteria:

* All has rfeabstract and low level functions are effective via both FTDI and Lauterbach
* No rfeProxy self-defined error
* No FCCU error

##### Python client all RFE API test

Python client has rfeapi library, which contains all of the units in RFE SW to call.

Test Criteria:

* All rfeapi functions are effective
* No rfeProxy self-defined error
* No FCCU error

##### SPI Communication test

Due to there’s multiple layers in the communication, from PC client to Proxy, from Proxy to STRX, then back from STRX to Proxy, from Proxy to PC client. Therefore, large delay and potential communication.

Test Criteria:

* One million SPI read/write via Lower level SPI functions correctly through both FTDI and Lauterbach
* One million SPI read/write via rfeapi SPI functions correctly through both FTDI and Lauterbach
* No FCCU error

Important Information to record:

* Total delays of each SPI function test
* Average delays of each SPI function test

##### All Firmware test

Test Criteria:

* rfeIpValFw pass [Python client all RFE API test](#_Python_client_all)
* rfeSysValFw pass [Python client own function test](#_Python_client_own)
* rfeFw pass [MATLAB client own function test](#_MATLAB_client_own) and [MATLAB client all RFE API test](#_MATLAB_client_all)

#### Test Coverage

(Content)

### A53 System Test

#### Definition

(To Do)

#### Methodology

(To Do)

#### Test Items

(To Do)

#### Test Coverage

(To Do)

### Functional Safety(FUSA) Test

#### Definition

FUSA test is to check our FUSA mechanism in STRX RFE and A53. Our FUSA architecture is in line with the ISO26262 standard, titled "Road vehicles – Functional safety", is an international standard for functional safety of electrical and/or electronic systems that are installed in serial production road vehicles (excluding mopeds). This standard request enough qualification of software tools for the intended and actual use.

RFE subsystem is a closed function with software delivered in binary form. Hence, a self-contained functional safety concept is followed for RFE subsystem.

|  |
| --- |
| 1. RFE safety feature overview |

Fig 1 provides an overview of the RFE safety features. The required level of safety integrity is achieved by a combination of hardware and software safety measures.

#### Methodology

There’s no general methodology to all FUSA test, they are described in each individual [Test Items](#_Test_Items).

#### Test Items

##### FUSA APIs

##### Fault Injection & Recovery

(To Do)

##### FCCU/ISM Errors

FCCU errors are tracked when applying other functional tests. Every functional test shall be checked the relevant errors, to filter out the APIs that trigger the functional safety issue.

#### Test Coverage

(To Do)

## Integration Test

### Definition

Integration testing is conducted to evaluate the compliance of a system or component with specified functional requirements. The system test, which are defined in [Functional Test](#_Functional_Test), are most important part of the integration test.

### Methodology

* Continuous Integration test is applied on a specific remote setup for this purpose. The quality and overall issues are checked before merge.
* The functional test is callable by every developer, to enable them debug during development.

### Test Items

* Compile only tests
* SW group of unit test
* [RFE System Test](#_RFE_System_Test)
* [A53 System Test](#_A53_System_Test)
* [Functional Safety(FUSA) Test](#_Functional_Safety(FUSA)_Test)

### Test Coverage

(TO DO)

## Regression Test

### Definition

Regression tests represent the repeated random execution of all relevant test cases, in order to validate bugs fixing, avoid code regression, and make feature enhancements robust.

When a new feature is requested to be implemented or a bug is fixed, all the tests, newly implemented and all previously available, must be re-executed for ensuring that the change made does not have impact on other parts of the software of the RFE SW.

### Methodology

Regression test consist of functional tests, the methodology of test is aligned with their own. If there’s new issues are found, the corresponding test will be added into relevant functional test category, and therefor added into regression test list.

### Test Items

* [RFE Unit Test](#_RFE_Unit_Test)
* [RFE System Test](#_RFE_System_Test)
* [A53 System Test](#_A53_System_Test)
* [Functional Safety(FUSA) Test](#_Functional_Safety(FUSA)_Test)

### Test Coverage

(Content)

## Black Duck Scan

### Definition

Black Duck is a comprehensive solution for managing security, license compliance, and code quality risks that come from the use of open source in applications and containers. Every NXP SW product shall be scanned before release to external customers, in order to avoid any potential legal issue.

### Methodology

* Link the repo of external release package to Black duck scan server and scan.
* Review the report before each release.
* Modify the code or get license if there’s conflict.

### Test Items

Full external release package black duck scan.

Test Criteria:

* NXP has valid license on full release.
* 0 dependency on open source library.

### Test Coverage

* External Release Package
* Violence of software license and risks from using open source code
* Legal Aspect

## Test Environment Requirements

The test environment consists of simulators and physical boards with STRX processors that represent target platforms for RFE SW releases, host computers with installed and configured complete tool chain to generate, build and execute test applications, in the conditions specified by the release notes.

The detail of software and hardware are used in release, will be described in **Table 5**

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Item** | **Type** | **Description** | **Version** |
| S32 Design Studio IDE | Software | The S32 Design Studio is a complimentary Integrated Development Environment (IDE) for automotive and ultra-reliable Power Architecture® (e200 core) and Arm®-based microcontrollers and processors | Version: 3.2 Build id: 190924 (Update 2) with NXP GCC for Arm Release version 9.2 build 1627, SPT assembler version: 200331 and VSPA3 compiler version: 3.00.00.226 |
| NXP GCC for Arm | Software | The Linaro toolchain is cross compiler for recent ARM-based processor | Version: 9.2 |
| MSYS2 | Software | MSYS2 is a software distro and building platform for Windows. | MSYS2 64-bit R.20161025 |
| MATLAB | Software | It is used for simulating Radar model and generating test vectors | R2021b or newer |
| Python | Software |  |  |
| Host platform for testing | Hardware | STRX |  |
| Evaluation board | Hardware | STRX evaluation board |  |
| Dolphin board | Hardware | TEF810x-BEST3 evaluation boards | TEF810x BEST3-S1 |

## ASPICE and Overall Test Coverage

Automotive Software Performance Improvement and Capability Determination (ASPICE) as a standard provides the framework for defining, implementing, and evaluating the process required for system development focused on software and system parts in the automotive industry.

### ASPICE-Integration Test(TO DO)

Integration tests as defined in ASPICE standard is testing the composition of components according to architecture specification (AS). RFE integration testing covers the compliance of integrated software items with the RFE AS, and including the interfaces between the software components and interaction among them.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Test types**  ***SW***  ***Subsystem*** | Fault Injection Tests | Interface Tests | Profiling Tests | Compile only tests | State Transition tests | Integration tests | Regression tests |
| *RFE software* |  |  |  |  |  |  |  |

Table 3 Integration level Test type mapping to SW subsystem

*legend*: EAR milestone

PRC milestone

RFP milestone

**Input:**

* RFE Software Architecture Specification

**Methods:**

* Fault injection tests
* Interface tests(To Discuss)
* Profiling tests: Functional test->System test
* Compile only tests
* State transition tests: Functional test->System test
* Integration tests: Functional test->System test
* Regression tests

**Test Techniques:**

* White box tests

**Methods for deriving and designing test cases**:

* Analysis of RFE Software architecture for integration
* Analysis of structure (statement, decision, multiple condition coverage)

**Deliverables**:

* Test Specification
* Test source files: test cases, test suites, test vectors
* Test Reports
* Profile report
* RAM size report
* Traceability report

### ASPICE-Qualification Test(TO DO)

Qualification testing as defined in ASPICE is to ensure the integrated software is tested according to the RFE SW requirement spec (RS).

**Input:**

* RFE SW Requirements
* RFE use-cases

**Methods:**

* Requirements-based test: Functional test->System test
* Use-case-based test: Functional test->System test

**Test technique:** black box tests

**Methods for deriving and designing test cases**:

* Analysis of requirements
* Analysis of use-cases

**Deliverables:**

* Test Specification
* Test source files: test cases, test suites.
* Test Reports
* Profile report
* RAM size report(To Discuss)
* Traceability report(To Discuss)

# Acceptance Reviews and Approvals

This section describes the process of reviewing and approving this plan. **Remove the example and text in yellow!**

Example:

*Refer to RASCI to define roles who have to review and approve this test plan.*

# Annexes

The annexes (if any) are placed in this chapter (usually one per page).

The reason for having Annexes is that they:

1. can be cited from other documents and
2. can be updated with less fuss than changing the actual document (i.e. can be updated without necessarily re-circulating the whole document for re-approval).

**text in yellow to be verified/checked!**

# Document Information

## References

| ***Item*** | ***Description*** |
| --- | --- |
| [1] | STRX SW Test plan  [doc405691: 105.08\_SmartTRX\_SW\_TP](https://www.collabnet.nxp.com/sf/go/doc405691) |
| [2] | BL-RFP SW Quality Policy  [Collabnet doc356947​​​](https://www.collabnet.nxp.com/sf/docman/do/downloadDocument/projects.blida_subqms_ccb/docman.root.bl_ida_qms_published.software_design.processes/doc356947) |
| [3] | BU-Automotive SW Procedure [NXPOMS-1719007347-3853](https://nxp1.sharepoint.com/sites/OMS/_layouts/15/WopiFrame.aspx?sourcedoc=%7b18C22CC3-44F0-45E0-BAF3-63BDD2F0DDD7%7d&file=BU%20AUTO%20Software%20Procedure%20-%20BCaM7.0.docx&action=default&DefaultItemOpen=1) |
| [4] | Software Configuration Management Plan [SCMP]  <https://www.collabnet.nxp.com/sf/go/doc405541> |
| [5] | Software Project Management Plan[040.20\_SmartTRX\_SW\_PMP]  <https://www.collabnet.nxp.com/sf/go/doc430900> |

## Terms/Acronyms and Definitions

| ***Acronym / Terms*** | ***Definition*** |
| --- | --- |
| RFE | Radar Front End |
| SW RS | Software requirement specification |
| SW AS | Software Architecture specification |
| PPA | Project Planning Approval(phase gate) |
| SCMP | Software configuration management plan |
|  |  |
|  |  |