Table of contents

1.0 Objectives / Purpose 5

2.0 Scope of Document 5

3.0 Testing Planning 5

3.1 Test Items 5

3.2 Test Scope 6

3.2.1 In Scope 6

3.2.2 Out of Scope 6

3.3 Risks, Assumptions and Constraints 6

3.4 Stakeholders and Communication 7

3.5 Resources 7

3.6 Testing Activities 7

3.7 Schedule 7

4.0 Testing Approach and Methodology 7

4.1 Testing Types and Levels 8

4.1.1 Pre-Silicon Test 9

4.1.1.1 Definition 9

4.1.1.2 Input 9

4.1.1.3 Test Techniques 9

4.1.1.4 Test Environment 10

4.1.1.5 Methods for deriving and designing test cases 10

4.1.1.6 Deliverables 10

4.1.1.7 Test case 10

4.1.2 Static Analysis/Verification 10

4.1.2.1 Definition 10

4.1.2.2 Input 10

4.1.2.3 Test Techniques: 10

4.1.2.4 Deliverables: 11

4.1.2.5 Methodology 11

4.1.3 RFE Unit Test 12

4.1.3.1 Definition 12

4.1.3.2 Input 12

4.1.3.3 Test Techniques 12

4.1.3.4 Methods for deriving and designing test cases 12

4.1.3.5 Deliverables 12

4.1.3.6 Methodology 12

4.1.3.7 Test Items 12

4.1.3.8 Test case 13

4.1.4 Integration Test 13

4.1.4.1 Definition 13

4.1.4.2 Input 13

4.1.4.3 Test Techniques 13

4.1.4.4 Methods for deriving and designing test cases: 13

4.1.4.5 Deliverables: 13

4.1.4.6 Methodology 13

4.1.4.7 Test Cases 14

4.1.5 Qualification Test 14

4.1.5.1 Definition 14

4.1.5.2 Input 14

4.1.5.3 Test technique 14

4.1.5.4 Methods for deriving and designing test cases 14

4.1.5.5 Deliverables 14

4.1.5.6 Cascading Test 14

4.1.5.7 Methodology 15

4.1.5.7.1 HW Connection 15

4.1.5.7.2 Test Platform: MATLAB/Python 15

4.1.5.7.3 Test Code and Configuration Files 16

4.1.5.8 Test Results 17

4.1.5.9 RFE System Test 17

4.1.5.10 Functional Safety (FUSA) Test 18

4.1.5.11 Customer Specific Use Cases 19

4.1.6 Regression Test 19

4.1.6.1 Definition 19

4.1.6.2 Input 20

4.1.6.3 Test Techniques 20

4.1.6.4 Methods for deriving and designing test cases 20

4.1.6.5 Deliverables 20

4.1.6.6 Test Frequency 20

4.1.6.7 Methodology 20

4.1.7 Coverage Test 22

4.1.7.1 Input 22

4.1.7.2 Deliverables 22

4.1.7.3 Methodology 22

4.1.7.4 Statement Coverage Test 22

4.1.7.5 Branch Coverage Test 22

4.1.7.6 Function Coverage Test 23

4.1.7.7 MC/DC Coverage Test 23

4.1.7.8 Call Coverage 23

4.1.8 Black Duck Scan 24

4.1.8.1 Definition 24

4.1.8.2 Methodology 24

4.1.8.3 Test Items 24

4.2 Test Design Techniques 24

4.3 Test Environment Requirements 25

5.0 Acceptance Reviews and Approvals 27

6.0 Annexes 28

6.1.1 Requirements Based Testing 28

6.1.2 Equivalence Classes 28

6.1.3 Boundary Testing 28

6.1.4 Static Analysis 28

6.1.5 Fault Injection 28

6.1.6 Error Guessing 29

6.1.7 Performance Test 29

6.1.8 Verification 29

6.1.9 Validation 29

7.0 Document Information 30

7.1 References 30

7.2 Terms/Acronyms and Definitions 31

7.3 Revision History 31

# 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 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 our effort within this test architecture is going to serve the core idea.

# Scope of Document

This test plan is aligned with the concepts about RFE One-Chip and Remote variant that is defined in the STRX RFE SW Architectural Specification [7]. STRX Shark is not considered in this documentation. Only the RFE relevant requirements and tests are considered in this document.

# Testing Planning

## 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[Unit name]) includes all the analog and digital IP HW blocks and their drivers of the RFE subsystem
* RFE SW units: (rfeSw [Unit name] all Software units like, BIST, Startup Calibration, Clocking, Main State Machine, Power Manager, etc.
* RFE units: (rfe[Unit name] 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(A53/APP-M7), it provides hooks between Abstract API and RFE FW, and therefore enable the RFE control from A53/APP-M7.

3. Autosar Driver – SW running on APP M7 for RFE Abstract API, a version for Autosar applications but achieve the same functions as abstract API.

4. RFE tools:

* rfeConfigGenerator: a SW tool to compile rfeConfig file from \*.xml format into \*.c \*.h, and binary formats. It is part of release package to customer.
* rfeRadarCycleScheduleGenerator: a SW tool to compile rfeConfig file timing information from \*.xml format into excel file. It is part of release package to customer

## Test Scope

### In Scope

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 RFE M7 and control cores. For STRX One-chip, control core includes the A53 and AppM7. For STRX Remote, control core is the App M7. RFE Qualification Test will be tested via abstract APIs and the raw radar data is captured for offline analysis. Testing covers the integration, qualification and RFE unit tests, static analysis and code coverage.

STRX One-Chip variant: stand-alone use case, test from RFE-M7, A53, APP-M7, Autosar CDD are in scope.

STRX Remote variant: stand-alone use case, 1 leader and 1 follower cascading use case, test from RFE-M7, APP-M7, Autosar CDD, CSI2 TX DMA on BBE are in scope.

Detailed test cases under this scope can be found in the test specification: <rfe repo>\Test\_rfe\integration\generic\src, and <rfe repo>\Test\_rfe\qualification\generic\src

### Out of Scope

Testing of the actual behaviour of all the HW IP blocks over temperature and over conners are out of Scope.

For STRX One-Chip, test on A53 only focus on the RFE driver, and the features described by the RFE requirements, other features in signal processing cluster, including BBEs are out of scope.

For STRX Remote, test on A53 only focus on the RFE driver and some BBE functions, and the features described by the RFE requirements, other features in signal processing cluster are out of scope.

Test on APP M7 only focus on the RFE driver and Autosar CDD drivers, and the features described by the RFE requirements, other APP M7 and Real-Time-Cluster functions are out of scope.

Test on rfeProxy, which is a SW media between host PC and STRX chip are not part of the SW delivery to the customer, is out of test scope.

## Risks, Assumptions and Constraints

1. Pre-silicon 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. The developers of M3SA at One-Chip and Remote come from different team, there is the risk of miss-understanding of the new developers in Remote project.
2. RFE Qualification Test involves performance tests, the performance assertion is based on the IP validation result instead of the datasheet of STRX product. RFE SW can only help to achieve the best performance of the post-silicon instead of pre-silicon expectation.

## Stakeholders and Communication

Stakeholder information is described in detail in Software Project Management Plan [5].

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.

The Integration and Validation team communicates the activity status during project meetings defined in Software Project Management Plan [5].

## Resources

Resources information is described in detail in Software Project Management Plan [5]*.*

## Testing Activities

Testing Activities are detailed in STRX SW Test plan[1].

## Schedule

STRX One-chip/ Remote variant project schedule is captured in Software Project Management Plan [5].

# Testing Approach and Methodology

Below table provides an overview of the Test techniques and methodology for each test type. More details can be found in the following sections.

|  |  |  |
| --- | --- | --- |
| Test Type | Test Technique | Methodology |
| Static Analysis | Static Analysis on the source code | Trigger Coverity analysis tool, collect result from it |
| Unit Test | White box tests | Create tests with Unity framework, compile Application and get result over UART, test from RFE-M7 |
| Integration Test | White box tests | One-Chip variant: Create tests with Unity framework, compile Application and get result over UART. Test from A53, APP-M7 and Autosar driver.  Remote variant: Create tests with Qualification test framework, compile Application and get result over UART. Test from APP-M7, Autosar driver and local host PC. |
| Qualification Test | Black box tests | Create tests with Qualification test framework through the Control core (A53&AppM7 for One-Chip, AppM7 for Remote) or local host PC |
| Regression Test | Unit + Integration + Qualification Tests | Apply all the test cases from each level onto CI/CD setup, execute for every Pull request and twice a day in develop branch. The test is also executed before each release. |
| Coverage Test | White box tests with 3rd Party tool | Using LDPR tool to log the sections of FW while running tests, check which lines are covered or not in a corresponding way. |
| Black Duck Scan | Static Analysis on the source code | Load RFE source code onto Black Duck Scan server and analyse the report. |

## Testing Types and Levels

In this documentation, the RFE SW tests are classified into unit test, integration test, qualification test, regression test, static analysis, coverage test, regression test and black duck scan. They checked RFE SW from different levels of ASPICE and provide different coverage. The RFE SW A-SPICE Compliant V-Model can be described as below figure. SPICE categorizes aspects of the model on process groups. Of interest here is the Software Engineering process group (SWE) which is further broken down into the typical phases of the development life cycle. The SWEs of interest is for software unit verifications which includes guidelines for unit test, integration test, qualification tests and static analysis.

A diagram of a computer

Description automatically generated

Figure 1 RFE SW A-Spice Compliant V-model

### Pre-Silicon Test

#### Definition

RFE SW starts before the silicon arrives, pre-silicon test is to ensure the SW basic functionality during this period. The test is at unit level; therefore, the test content is highly overlapping with the Unit tests, which are on the SWE.4 in traceability V model.

One-Chip variant: pre-silicon test is running on the SW simulation environment, namely M3SA and Zebu. M3SA is a pure SW simulation application, it enables the tests on register toggle of digital part. Zebu is a FPGA simulation environment; it also simulates analogy behaviours.

Remote variant: pre-silicon test is running on the One-Chip ES2 setup, because remote project is sharing the same RFE design with One-Chip ES2, no difference on RFE part is expected.

#### Input

* RFE Software Unit detailed design

#### Test Techniques

* White box tests

#### Test Environment

One-Chip

* M3SA SW Simulation Environment
* Zebu FPGA Simulation Environment

Remote

* One-Chip ES2 setup

#### Methods for deriving and designing test cases

* Analysis of RFE Software Unit detailed design
* Analysis of structure (statement, decision, multiple condition coverage)

#### Deliverables

* Test Specification
* Test source files: test cases, test suites
* Test Reports

#### Test case

Each unit has their unique criteria, which are defined by unit developers. The unit test code can be found int the repo: <rfe root>\rfem7\code\tests\...

### Static Analysis/Verification

#### Definition

It is an ASPICE requirement to perform static verification of software units by SWE4. The static analysis consists of Cyclomatic Complexity Measurement (CCM), Common Weakness Enumeration (CWE), and standard check for C programming language Motor Industry Software Reliability Association (MISRA C) 2012. Relevant test definition also defined in the [RFE\_FW\_RFE\_GUI\_Quality\_Assurance\_Plan](https://nxp1.sharepoint.com/teams/206_21/Shared%20Documents/Forms/AllItems.aspx?OR=Teams%2DHL&CT=1638376039224&id=%2Fteams%2F206%5F21%2FShared%20Documents%2FTestPlan%2FQualityAssurancePlan&viewid=94a06352%2D64ef%2D47c8%2Dad03%2D520a3815a252) [6]

#### Input

* RFE Software Unit

#### Test Techniques:

* Static Analysis

#### Deliverables:

* CWE Test Reports
* CCM Test Reports
* MISRAC Test Reports

#### Methodology

Static analysis/verification highly depends on 3rd party test tools, like Coverity. Tester needs to deploy the Coverity on CI/CD server first and link the repository of RFE SW to the test tool and configure the tool to run CWE, CCM, and MISRAC tests.

### 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 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.

#### Input

* RFE Software Unit detailed design

#### Test Techniques

* White box tests

#### Methods for deriving and designing test cases

* Analysis of RFE Software Unit detailed design
* Analysis of structure (statement, decision, multiple condition coverage)

#### Deliverables

* Test Specification
* Test source files: test cases, test suites
* Test Reports

#### Methodology

Unit testing is done with the Unity Framework [10]. It provides the interface for automated testing. For each unit a unit test app is created with the Unity framework. The unit test examples can be found in the repo: <rfe root>\rfem7\code\tests\...

#### Test Items

Units of Remote project is aligned with the STRX RFE subsystem components, involves the RFE-Analog, RFE-Digital, and RFE-Access parts, which are defined in the STRX RFE SW Architectural Specification [7].

#### Test case

Each unit has their unique criteria, which are defined by unit developers. The unit test code can be found int the repo: <rfe root>\rfem7\code\tests\...

### Integration Test

#### Definition

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.

#### Input

* RFE Software Architecture Specification

#### 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
* Integration Test Reports
* Traceability report

#### Methodology

Most of the RFE M7 internal integration test methodology of is just the same as the [Unit Test Methodology](#_Methodology_1).

#### Test Cases

The test cases are stored in the repository:

. \Test\_rfe\integration\generic\src

### Qualification Test

#### Definition

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

#### Test technique

* black box tests

#### Methods for deriving and designing test cases

* Analysis of requirements

#### Deliverables

* Test Specification
* Test source files: test cases, test suites.
* Test Reports

#### Cascading Test

In a STRX Remote RFE SW runs a synchronization protocol on top of a GPIO interface between leader and follower RFE to synchronize the radar cycle, chirp sequence and configuration calibrations on each cascaded device.

STRX Remote project has unique cascaded requirement specifications defined in the STRX RFE Requirement Specification [8]. The cascading test for Remote shall use the ethernet to communicate between the local host PC to either Leader/follower at one time. Timing sensitive qualification test shall directly be executed with APP-M7 application, non-timing sensitive tests can be executed from the host PC.

#### Methodology

RFE system tests relay on the Trace32 software and Lauterbach to do communication, local host PC to execute test and process the ADC data, RFE Abstract API to control the finite state machine, and RFE configuration file to update settings. Below methodology is also a test case creation guidance, to help tester with new test cases.

##### HW Connection

At the early stage of test, the Lauterbach is used to do communication, J-TAG connector shall be connected to the STRX board. On the other side of Lauterbach, USB is connected to PC. Lauterbach need an 8V/1A power supply individually for power cycle. While using Lauterbach as media, T32 on PC host shall be installed to drive Lauterbach, T32 for NXP can be downloaded from link of [NXP T32 Official Release](https://nxp1.sharepoint.com/teams/140/product_development/ampapps/Shared%20Documents/Lauterbach_patches/Old_versions/trace32_n_2022_10_000153069_nxp_internal_conf_20221019173534.zip).

Other Communications are advanced options: Ethernet connection, UART connection, and J-Link connections are potential media that can be used in this test. If the driver is available, these connections shall be considered to add to test framework, to reduce the consume more Lauterbach.

##### Test Platform: MATLAB/Python

The test code is either in MATLAB \*.m file or the python \*.py file. The test code is stored in the repo directory: \rfe\_validation\MatlabTester\ . When starting a test, always start from the file ‘STRX\_RFE\_Test\_Main.m’, this file controls the test sequencing, environment configuration, instrument control, and report generation. It acts as the main test sequence interface.

In the test main file, tester can select which test to run by enable the test or disable them by comment them out. Test cases files are collect in the directory of repo:

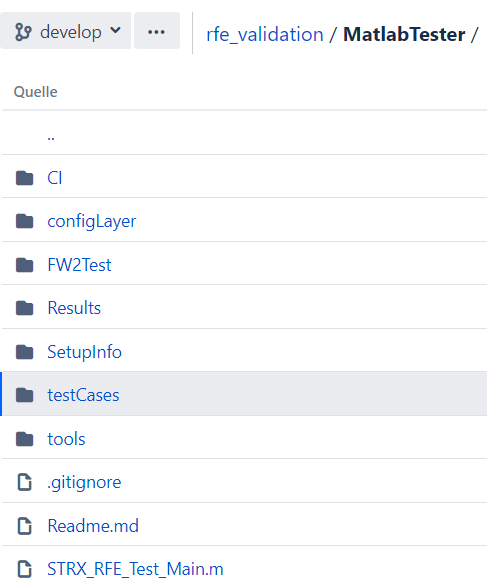


Figure 2 RFE System Test Folder Hierarchy

\rfe\_validation\MatlabTester\testCases\ . Folder Hierarchy is displayed Figure 2 RFE System Test Folder Hierarchy.

Relevant tests are collected as a cluster below the folder of ‘rfeSystemTest’, the names of cluster folders are the same as this documentation organized. Below the cluster folder, there’s folder for each test cases folder. Each test cases folder consists of an execution m file plus a configuration xml file.

While tester want to create a new test, they can copy paste a exist test folder, like the basic ceiling test as shown in the figure and modify the files to specific settings to meet test specification.

##### Test Code and Configuration Files

Each individual test cases/items have own test code with specific pass/fail criteria. The test code consists of instrument control, communication control, config binary file generation, config RFE, trigger radar cycle, and signal processing. Test code example can be found in the \MatlabTester\testCases\.

Test code is responsible to drive the test process as above introduced, but detailed configuration is defined in the rfeConfig\_<TestCaseName>.xml file. Test cases are different from each other on all possible settings in this file, like center frequency, bandwidth, timings, etc. Tester needs to modify the setting according to the test.

#### Test Results

The results are stored in the. \MatlabTester\Results\<TestToken&Date>\ <TestCaseName>\.

Word format test results are stored in the ‘Files’ folder. Picture format results are stored in the ‘Pictures’ folder. And there’s must be a ‘TestResults.txt’ file identify the test ID, test specification, test description, test parameters, and most important one: test Pass/fail result. This text file will be used by CI/CD and manually check both.

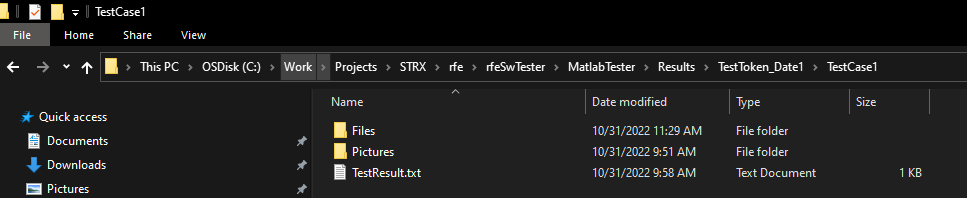


Figure 3 Test Result Folder Example

#### RFE System Test

RFE System test is to repeat customer use cases and check the whole RFE FW integration behaviour in a comprehensive way. This test is also radar front end full chain test, that customer used to do to check the basic radar functionalities: target detection, phase detection, stability, and other radar functions. This test is very necessary, because the impact between multiple components is hard to be detected just on SoC level or IP level, they only show by the whole RFE M7 system working like current overshoot, unstable phase, out of band emissions, etc. 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. The test specification can be found in the repo: \rfe\_validation\testRfe\qualification\generic\src, the test specifications are stored in the C file format.

#### Functional Safety (FUSA) Test

FUSA test is to check our FUSA mechanism in STRX RFE. 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.

|  |
| --- |
| Figure 4 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. These SW safety features shall be measured at unit, integration, and qualification levels, via Fault Injection Tests (FIT).

Unit level: trigger fault from ISM and check SW response.

Integration level: simulate API error and check SW response.

Qualification level: inject fault in safety monitor by using validation hook SW and check SW response; error clearance test over PVT and unmask errors, check if any error occurs; run the full RFE abstract API functions, and check the FUSA status for each state; FUSA error check is required for every Qualification test, no FUSA error is a default test criterion.

#### Customer Specific Use Cases

Some test cases’ definition come from customer, their scenarios shall be considered and merged into SW test scope. Below are some known test cases that we need to implement according to them:

* MCGEN PN\_Test according to: <https://jira.sw.nxp.com/browse/STRX-5263>
* FCM1 x 8 FCM1 Staggers x 2 Power Modes (High\Low)) x (FCM2 x 8 FCM1 Staggers x2 Power Modes (High\Low) Test matrix, settings are from lead customer and the spectrum shall be observed.
* GLDO2 dI/dt tests according to: https://jira.sw.nxp.com/browse/STRX-5336
* MCGEN Bump Test
* PN\_test pin Test
* Init Register Test
* OBW Test
* Lead Customer App Test
* 2xFCMs x8 staggers Test
* FSW on Chirp Pin Test

The test request from customer side shall follow the overall traceability rules, create corresponding requirements in requirement specification or create design decision in the Architecture specification, then create corresponding test specification in the repo.

### Regression Test

#### Definition

Regression tests represent the repeated 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.

#### Input

* Static Verification
* RFE Unit Test Suite
* Integration Test Suite
* Qualification Test Suite

#### Test Techniques

* White box tests and Black box tests
* Continuous integration/Continuous deployment (CI/CD)

#### Methods for deriving and designing test cases

* Use the test cases from Unit Test, integration test, and qualification test.

#### Deliverables

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

#### Test Frequency

* Every Pull Request from developer
* Every Release Candidate
* 2 times every day on develop branch

#### Methodology

Regression test is achieved by CI/CD technology. Jenkins Server shall be deployed for the regression test, this server will trigger all the static analysis, unit test, integration test, and qualification tests daily and repeatedly. Jenkins server is linked to Bitbucket, every pull request from developers will trigger a regression test before it is merged into the develop branch. Only when PR pass a full Regression test, then it can be approved to merge, otherwise ‘Merge’ button is greyed out and impossible to execute more action. Besides this, test shall be executed twice on the develop branch in nightly build to check the stability of the develop branch. For each release candidate, the CI/CD machine shall also trigger full tests of static analysis, unit test, integration test, and qualification tests, to guarantee the quality of the release. Below figure introduces this procedure.

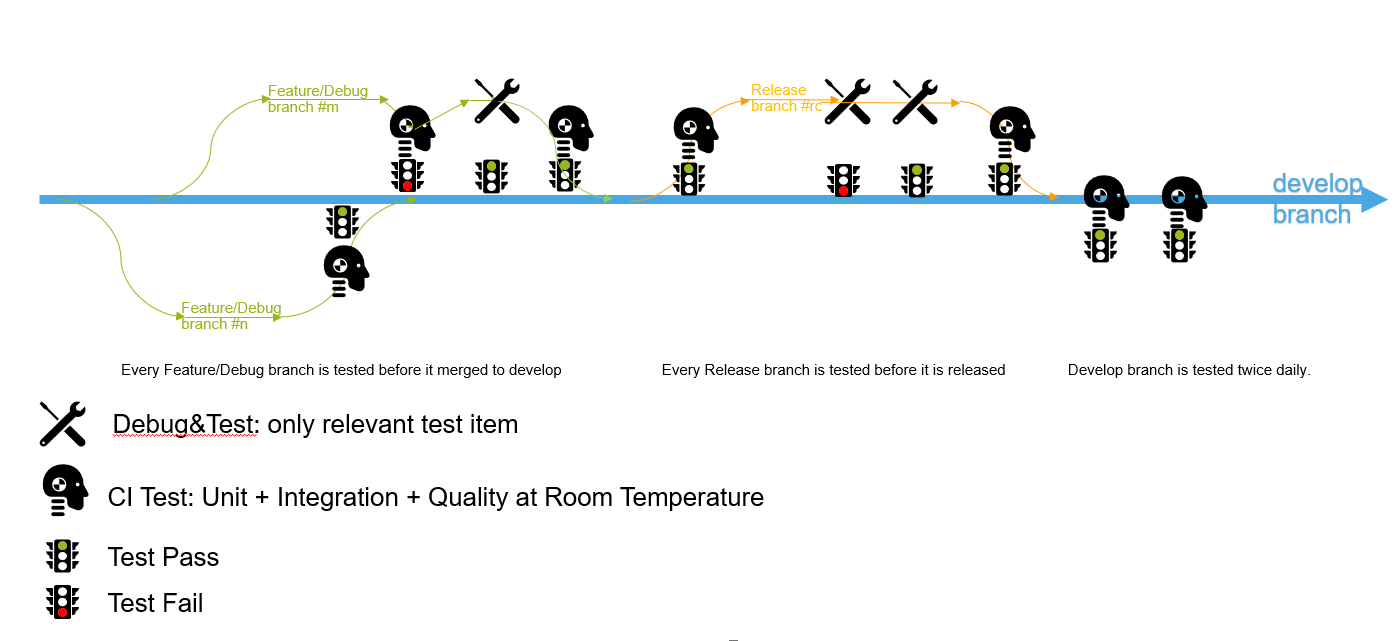


Figure 5 CI/CD Test Trigger Mechanism

### Coverage Test

Coverage Test is a percentage measure of the degree to which the source code of a program is executed when a particular test suite is run. Coverage tests consist of statement coverage, branch coverage, function coverage, MC/DC coverage, and call coverage.

#### Input

* RFE Software Unit Test

#### Deliverables

* Test Reports

#### Methodology

Coverage Test highly depends on 3rd party test tools, like Trace32 or LDRA. Tester needs to deploy the relevant tools on the test PC, configure the environment, then build all test \*.elf files and import them into LDRA. In LDRA tester can select statement, branch, function, MC/DC, and Call coverage to test and export the results.

The LDRA tool is introduced in the reference LDRA\_Coverage\_readme [9]

#### Statement Coverage Test

Statement coverage ensures that every statement in the program has been invoked at least once. Statement in this context means block of source code lines.

#### Branch Coverage Test

Assessing the code coverage when the code is a ‘branch’ statement or decision-making statement is referred to as branch coverage. For example – if-else, do-while switch. All of these are branch statements and branch coverage ensure executing them at least once. Every point of entry and exit in the program has been invoked at least once and every decision in the program has taken on all possible outcomes at least once.

#### Function Coverage Test

Closely similar to branch coverage, the function coverage facilitates the execution of each function while performing code coverage. Additionally, function coverage checks if different types of arguments are passed onto the function to verify the efficient functionality.

#### MC/DC Coverage Test

Modified Condition/Decision Coverage (MC/DC) is a code coverage criterion commonly used in software testing. This type of coverage tests the conditions that are present in the code. A condition coverage technique verifies the accuracy of subexpressions and variables inside all the conditional statements. Every point of entry and exit in the program has been invoked at least once and every decision in the program has taken all possible outcomes at least once. Each condition in a decision is shown to independently affect the outcome of that decision.

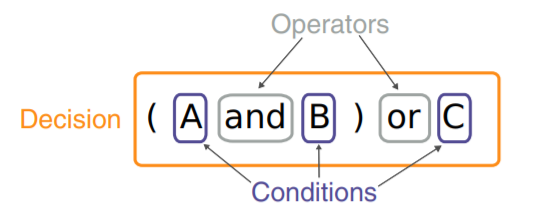


Figure 6 Modified Condition/Decision Coverage

#### Call Coverage

Call Coverage: Every function call has been executed at least once.

### 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, to avoid any potential legal issue.

#### Methodology

Black duck scan needs professional library and database to store and compare the code from public. NXP has bought relevant service and deploy such server by IT, relevant account can be open by IT. The main steps are below.

* 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 Design Techniques

The test design technique for each testing type is introduced in section 4.1.

* Dynamic
  + Analysis of RFE Software architecture for integration
  + Analysis of RFE Software Unit detailed design
  + Analysis of structure (statement, decision, multiple condition coverage
  + Experience based (error guessing, exploratory)
  + Structure based (statement, decision, multiple condition/decision)
* Static
  + Static Analysis (data flow, control flow)
  + Black Duck Scan
  + Reviews

## 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 for radar signal post processing and generating test cases | R2021b or newer |
| Python | Software | Python is good for instrument control, test process control and necessary for rfeProxy Python client test. | V3.8 or newer |
| Trace32 | Software | Trace32 is necessary for Lauterbach driving. | N.2021.12.000142973 or newer released by NXP Lauterbach Supporting team |
| Waveguide boards for testing | Hardware | 8 WR12 24dB gain horn antennas, 2 90-degree twisters, 2 45-degree twisters for each setup | N/A |
| Host platform for testing | Hardware | STRX | N/A |
| Evaluation board | Hardware | STRX evaluation board | WG\_SKT\_V3, V4,  WG\_SILD\_V3,  Dig\_SKT\_V1, V2,  IF\_V1, V2,  CAB |
| Spectrum Analyzer | Hardware | R&S FSW with spectrum and trace analysis functions | FSW81, FSW26+ E band mixer |
| Horn Antenna | Hardware | 24dB gain horn antennas | N/A |
| Anechoic Box | Hardware | 60x60x200cm Anechoic box, internal surface over | N/A |
| Current Probe | Hardware | High Sensitivity 2-channel Current Probe | Keysight N2820A |
| Oscilloscope | Hardware | Four channel Oscilloscope | DSO, MSO-4054 |

# Acceptance Reviews and Approvals

| Name | Role | Location | Date |
| --- | --- | --- | --- |
| Shakti Prasad Shenoy | STRX SW Test Architect | Bangalore, India |  |
| Marthijn de Man | STRX SW Architect | Nijmegen, the Netherlands |  |
| Artur Burchard | STRX RFE SW Architect | Eindhoven, the Netherlands |  |
| Lars van Meurs | STRX RFE SW Architect | Eindhoven, the Netherlands |  |

# Annexes

### Requirements Based Testing

Analysis of requirements leads toprecondition of the test and sequence of steps to be performed by the component. Test vectors definition is not in scope of this technique, typically it is combined with error guessing test technique to select proper test vectors, but error guessing method does not ensure meaningful and measurable coverage of space of test vectors.

### Equivalence Classes

Equivalence class is the partitioning of the input variable’s value set into classes using some equivalence relation. Equivalence Class Partitioning is a software testing technique that divides the input data of a software unit into partitions of equivalent data from which test cases can be derived. In principle, test cases are designed to cover each partition at least once. This technique tries to define test cases that uncover classes of errors, thereby reducing the total number of test cases that must be developed.

### Boundary Testing

Boundary testing is subset of equivalence classes testing, focused on test vector on the border between functional and non-functional equivalence classes. Test preparation starts also with analysis of all API parameters, but then tests are implemented manually, without the automated generator based on pairwise algorithm. Boundary test shall be written for API parameter based on integer type, for enumeration types of boundary test are focused primarily on channel parameter, representing range of available resources. To test this, range a configuration with all available channels is created and then correctness of the first and the last channel is verified.

This test technique is suitable to create test vectors for component testing**.**

### Static Analysis

The source code that is part of the build process to obtain STRX One-chip/Remote variant release candidates shall be statically verified using Coverity central analysis, with checkers enabled per Coding Style. The false positives and accepted violations will be marked as such in Coverity and will not be reported during the next analysis.

Desktop analysis may be used by developers, e.g., for pre-commit verification.

### Fault Injection

Fault-injection tests involve the deliberate introduction of faults to test system robustness and error-handling capabilities. Faults can be introduced directly into the code (compile-time injection) or using software triggers that cause specific scenarios to occur in a running system (runtime injection). It ensures that the system can handle and recover from fault or error conditions and identifies design weaknesses where a single fault could potentially be propagated into a severe error or systemic failure.

### Error Guessing

Error guessing isa test technique in which test cases used to find bugs in programs are established based on experience in prior testing. The scope of test cases usually relies on the software tester involved, who uses experience and intuition to determine what situations commonly cause software failure or may cause errors to appear.

### Performance Test

Testing conducted to evaluate the compliance of a system or component with specified performance requirements. Performance attributes such as minimum or maximum execution times, storage usage (e.g., RAM for stack and heap, ROM for program and data) and the bandwidth of communication links (e.g., data busses) are measured.

### Verification

The verification in this doc means the tests during Pre-Silicon period and the tests without HW. For example, the Pre-Silicon tests on M3SA and Zebu are verification. The static analysis and Black Duck Scan are not HW dependent, they are SW verification process.

### Validation

The validation in this doc means the HW test during post-Silicon period. For example, unit, integration, and qualification tests on ES2 HW are validation. If the same tests are executed during Pre-Silicon period, then they are regard as Verification. For example, Pre-Silicon tests of STRX Remote shall be running on the ES2 HW of One-Chip, even they are HW testing, but still treated as verification, instead of Validation.

# 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> |
| [6] | RFE FW RFE GUI Quality Assurance Plan  https://nxp1.sharepoint.com/teams/206\_21/Shared%20Documents/Forms/AllItems.aspx?OR=Teams%2DHL&CT=1638376039224&id=%2Fteams%2F206%5F21%2FShared%20Documents%2FTestPlan%2FQualityAssurancePlan&viewid=94a06352%2D64ef%2D47c8%2Dad03%2D520a3815a252 |
| [7] | STRX RFE SW Architectural Specification  https://www.collabnet.nxp.com/sf/go/doc403502 |
| [8] | STRX RFE Requirement Specification  https://doorsng.nxp.com/rm/web#action=com.ibm.rdm.web.pages.showArtifact&artifactURI=https%3A%2F%2Fdoorsng.nxp.com%2Frm%2Fresources%2F\_d383e39a562a4815b0051ce989cd65b5&componentURI=https%3A%2F%2Fdoorsng.nxp.com%2Frm%2Frm-projects%2F\_r8ZsYRVKEeiLn-6Dce4MoA%2Fcomponents%2F\_entLQI3hEeqaeff\_JhQ61g&vvc.configuration=https%3A%2F%2Fdoorsng.nxp.com%2Frm%2Fcm%2Fstream%2F\_en8b0I3hEeqaeff\_JhQ61g |
| [9] | LDRA\_Coverage\_readme.docx  https://nxp1.sharepoint.com/teams/206\_21/Shared%20Documents/Forms/AllItems.aspx?OR=Teams%2DHL&CT=1638376039224&id=%2Fteams%2F206%5F21%2FShared%20Documents%2FTestPlan%2FCoverage%20Test&viewid=94a06352%2D64ef%2D47c8%2Dad03%2D520a3815a252 |
| [10] | Unity Framework Documentation  https://docs.unity3d.com/Manual/index.html |

## 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 |
| ASPICE | Automotive Software Performance Improvement and Capability dEtermination |
| CDD | Complex device driver |

## Revision History

| ***Document Author*** | ***Version*** | ***Status*** | ***Date*** | ***Description of Change*** |
| --- | --- | --- | --- | --- |
| Wolfgang Vogel | V0.7.1 | *In Review* | Oct.20.2023 | Update links to new repo structure |
| Dongyu Gao | V0.7 | *In Review* | Apr.13.2023 | Update the test plan according to Lars, Artur, |
| Dongyu Gao | V0.6 | *In Review* | Feb.05.2023 | Update the test plan about STRX Remote and Internal Review Input |
| Dongyu Gao | V0.5 | *In Review* | Dec.20.2022 | Update the Test Plan to the latest Template |
| Dongyu Gao | V0.4 | *In Review* | Nov.30.2022 | Modified after 1st pre-review with Architects |
| Dongyu Gao | V0.3 | *In Review* | OCT.28.2022 | Complete the definition of tests |
| Dongyu Gao | V0.2 | *In Review* | April.29.2022 | Restructure the test plan |
| Biju Ravindran | V0.1 | *Approved* | July.28.2021 | Initial draft version of test plan |