

A121 Assembly Test

User Guide

A121 Assembly Test

User Guide

Author: Acconeer AB

Version: a121-v1.14.0

Acconeer AB Nov 04, 2026

© 2026 by Acconeer AB - All rights reserved

Contents

1 Acconeer SDK Documentation Overview 3
2 Introduction 5
3 Setting up the Assembly Test 6
3.1 Initializing the System 6
3.2 Allocate Buffer 6
3.3 Create Test 6
3.4 Enable / Disable Tests 6
4 Running the assembly test 8
5 Get the assembly test result 9
6 Deactivation and Destroy 10
7 Assembly test descriptions 11
7.1 Basic Read Test 11
7.2 Communication Test 11
7.3 Enable Pin Test 11
7.4 Interrupt Test 11
7.5 Clock & Supply Test 11
7.6 Sensor Calibration Test 11
8 Disclaimer 12

© 2026 by Acconeer AB - All rights reserved

1 Acconeer SDK Documentation Overview

To better understand which SDK document to use, a summary of the documents is shown in the table below.

Name	Description	When to use
RSS API documentation (html)		
rss_api	The complete C API documentation.	• RSS application implementation • Understanding RSS API functions
User guides (PDF)		
A121 Assembly Test	Describes the Acconeer assembly test functionality.	• Bring-up of HW/SW • Production test implementation
A121 Breathing Reference Application	Describes the functionality of the Breathing Reference Application.	• Working with the Breathing Reference Application
A121 Distance Detector	Describes usage and algorithms of the Distance Detector.	• Working with the Distance Detector
A121 SW Integration	Describes how to implement each integration function needed to use the Acconeer sensor.	• SW implementation of custom HW integration
A121 Presence Detector	Describes usage and algorithms of the Presence Detector.	• Working with the Presence Detector
A121 Smart Presence Reference Application	Describes the functionality of the Smart Presence Reference Application.	• Working with the Smart Presence Reference Application
A121 Sparse IQ Service	Describes usage of the Sparse IQ Service.	• Working with the Sparse IQ Service
A121 Tank Level Reference Application	Describes the functionality of the Tank Level Reference Application.	• Working with the Tank Level Reference Application
A121 Touchless Button Reference Application	Describes the functionality of the Touchless Button Reference Application.	• Working with the Touchless Button Reference Application
A121 Parking Reference Application	Describes the functionality of the Parking Reference Application.	• Working with the Parking Reference Application
A121 Vibration Example Application	Describes the functionality of the Vibration Example Application.	• Working with the Vibration Example Application
A121 STM32CubeIDE	Describes the flow of taking an Acconeer SDK and integrate into STM32CubeIDE.	• Using STM32CubeIDE
A121 Raspberry Pi Software	Describes how to develop for Raspberry Pi (aarch64).	• Working with Raspberry Pi (64-bit application)
A121 Raspberry Pi armv7l Software	Describes how to develop for Raspberry Pi (armv7l).	• Working with Raspberry Pi (32-bit application)
A121 Ripple	Describes how to develop for Ripple.	• Working with Ripple on Raspberry Pi
A121 ESP32 User Guide	Describes how to develop with A121 and ESP32 targets.	• Working with ESP32 targets
XM125 Software	Describes how to develop for XM125.	• Working with XM125
XM126 Software	Describes how to develop for XM126.	• Working with XM126

© 2026 by Acconeer AB - All rights reserved

Name	Description	When to use
I2C Distance Detector	Describes the functionality of the I2C Distance Detector Application.	• Working with the I2C Distance Detector Application
I2C Presence Detector	Describes the functionality of the I2C Presence Detector Application.	• Working with the I2C Presence Detector Application
I2C Breathing Reference Application	Describes the functionality of the I2C Breathing Reference Application.	• Working with the I2C Breathing Reference Application
I2C Cargo Example Application	Describes the functionality of the I2C Cargo Example Application.	• Working with the I2C Cargo Example Application
A121 Radar Data and Control (PDF)		
A121 Radar Data and Control	Describes different aspects of the Acconeer offer, for example radar principles and how to configure	• To understand the Acconeer sensor • Use case evaluation
Readme (txt)		
README	Various target specific information and links	• After SDK download

Table 1: SDK document overview.

© 2026 by Acconeer AB - All rights reserved

2 Introduction

The assembly test can be used for two different purposes which both aim to verify the assembly of the sensor into a custom PCB:

- Flexible and accurate production test to maintain good quality in PCB assembly of the sensor
- A tool for easy bring-up of HW/SW to decrease the time-to-market

The assembly test contains several tests, where each test is focused on testing a particular functionality. The test purpose and interpretation of the test result are described in this document. The tests are structured to provide clear feedback on what in the assembly, implementation, and/or integration is failing.

Acconeer provides an example of how to use the assembly test in a bring-up context: `example_bring_up.c`

© 2026 by Acconeer AB - All rights reserved

3 Setting up the Assembly Test

3.1 Initializing the System

The HAL (Hardware Abstraction Layer) must be registered to the Radar System Software (RSS) before any other calls are done. The registration requires a pointer to an `acc_hal_a121_t` struct which contains information on the hardware integration and function pointers to hardware driver functions that are needed by RSS. See the document "HAL Integration" user guide for more information on how to integrate the driver layer and populate the hal struct.

In Acconeer's example integrations there is a function `acc_hal_rss_integration_get_implementation()` to obtain the hal struct.

```
const acc_hal_a121_t *hal = acc_hal_rss_integration_get_implementation();

if (!acc_rss_hal_register(hal))
{
    /* Handle error */
}
```

3.2 Allocate Buffer

Before the assembly test can be executed, a memory buffer must be allocated for the test.

```
void *buffer = acc_integration_mem_alloc(ACC_RSS_ASSEMBLY_TEST_MIN_BUFFER_SIZE);
if (buffer == NULL)
{
    /* Handle error */
}
```

3.3 Create Test

```
acc_rss_assembly_test_t *assembly_test = acc_rss_assembly_test_create(SENSOR_ID, buffer,
ACC_RSS_ASSEMBLY_TEST_MIN_BUFFER_SIZE);
if (assembly_test == NULL)
{
    /* Handle error */
}
```

3.4 Enable / Disable Tests

All tests are enabled by default but the application can choose to run a subset of the tests by disabling tests in the configuration.

Functions to enable or disable all or individual tests are provided by `acc_rss_a121.h`.

```
/* Enable or disable all tests */
acc_rss_assembly_test_enable_all_tests(assembly_test);
acc_rss_assembly_test_disable_all_tests(assembly_test);

/* Enable or disable Basic Read Test */
acc_rss_assembly_test_enable(assembly_test, ACC_RSS_ASSEMBLY_TEST_ID_BASIC_READ);
acc_rss_assembly_test_disable(assembly_test, ACC_RSS_ASSEMBLY_TEST_ID_BASIC_READ);

/* Enable or disable Communication Test */
acc_rss_assembly_test_enable(assembly_test, ACC_RSS_ASSEMBLY_TEST_ID_COMMUNICATION);
acc_rss_assembly_test_disable(assembly_test, ACC_RSS_ASSEMBLY_TEST_ID_COMMUNICATION);

/* Enable or disable Enable Pin Test */
acc_rss_assembly_test_enable(assembly_test, ACC_RSS_ASSEMBLY_TEST_ID_ENABLE_PIN);
acc_rss_assembly_test_disable(assembly_test, ACC_RSS_ASSEMBLY_TEST_ID_ENABLE_PIN);

/* Enable or disable Interrupt Test */
acc_rss_assembly_test_enable(assembly_test, ACC_RSS_ASSEMBLY_TEST_ID_INTERRUPT);
acc_rss_assembly_test_disable(assembly_test, ACC_RSS_ASSEMBLY_TEST_ID_INTERRUPT);
```

© 2026 by Acconeer AB - All rights reserved

```
/* Enable or disable Clock and Supply Test */
acc_rss_assembly_test_enable(assembly_test, ACC_RSS_ASSEMBLY_TEST_ID_CLOCK_AND_SUPPLY);
acc_rss_assembly_test_disable(assembly_test, ACC_RSS_ASSEMBLY_TEST_ID_CLOCK_AND_SUPPLY);

/* Enable or disable Sensor Calibration Test */
acc_rss_assembly_test_enable(assembly_test, ACC_RSS_ASSEMBLY_TEST_ID_SENSOR_CALIBRATION);
acc_rss_assembly_test_disable(assembly_test, ACC_RSS_ASSEMBLY_TEST_ID_SENSOR_CALIBRATION);
```

© 2026 by Acconeer AB - All rights reserved

4 Running the assembly test

The sensor needs to be powered on and enabled before the test is started.

The `acc_rss_assembly_test_execute()` function has to be called multiple times until it returns `ACC_RSS_TEST_STATE_COMPLETE`.

```
acc_hal_integration_sensor_supply_on(sensor_id);
acc_hal_integration_sensor_enable(sensor_id);

acc_rss_test_state_t              assembly_test_state = ACC_RSS_TEST_STATE_ONGOING;
acc_rss_test_integration_status_t integration_status  = ACC_RSS_TEST_INTEGRATION_STATUS_OK;

do
{
    assembly_test_state = acc_rss_assembly_test_execute(assembly_test, integration_status);

    switch (assembly_test_state)
    {
        case ACC_RSS_TEST_STATE_TOGGLE_ENABLE_PIN:
            acc_hal_integration_sensor_disable(sensor_id);
            acc_hal_integration_sensor_enable(sensor_id);
            integration_status = ACC_RSS_TEST_INTEGRATION_STATUS_OK;
            break;
        case ACC_RSS_TEST_STATE_WAIT_FOR_INTERRUPT:
            if (!acc_hal_integration_wait_for_sensor_interrupt(sensor_id, SENSOR_TIMEOUT_MS))
            {
                /* Wait for interrupt failed */
                integration_status = ACC_RSS_TEST_INTEGRATION_STATUS_TIMEOUT;
            }
            else
            {
                integration_status = ACC_RSS_TEST_INTEGRATION_STATUS_OK;
            }

            break;
        default:
            integration_status = ACC_RSS_TEST_INTEGRATION_STATUS_OK;
            break;
    }
} while (assembly_test_state != ACC_RSS_TEST_STATE_COMPLETE);

acc_hal_integration_sensor_disable(sensor_id);
acc_hal_integration_sensor_supply_off(sensor_id);
```

© 2026 by Acconeer AB - All rights reserved

5 Get the assembly test result

The different sub tests in the Assembly test aim to verify the integration of A121 into a custom PCB. The assembly test does not test the actual sensor but focuses on testing the interaction. The assembly test returns an array with the result for each individual test and a variable containing the number of tests. The array should be iterated to check the result of each test and all tests should pass to confirm that the sensor and HAL have been integrated properly.

```
uint16_t nbr_of_test_results = 0U;

const acc_rss_assembly_test_result_t *test_results = acc_rss_assembly_test_get_results(assembly_test,
&nbr_of_test_results);

bool all_passed = true;

for (uint16_t idx = 0; idx < nbr_of_test_results; idx++)
{
    printf("Assembly test: '%s' [%s]\n", test_results[idx].test_name, test_results[idx].test_result ?
"PASS" : "FAIL");
    if (!test_results[idx].test_result)
    {
        all_passed = false;
    }
}

if (all_passed)
{
    printf("Assembly test: All tests passed\n");
}
```

© 2026 by Acconeer AB - All rights reserved

6 Deactivation and Destroy

After the test is finished, the assembly test needs to be destroyed.

```
acc_rss_assembly_test_destroy(assembly_test);
```

The allocated buffer also needs to be freed.

```
if (buffer != NULL)
{
    acc_integration_mem_free(buffer);
}
```

© 2026 by Acconeer AB - All rights reserved

7 Assembly test descriptions

7.1 Basic Read Test

SPI bus communication is tested by reading data from the sensor. This basic read test is useful in bring-up to reassure that the SPI driver and HAL registration are working properly. If the SPI bus is monitored using a logic analyzer, this is the pattern that should be observed (first 32 MISO bits may be integration dependent).

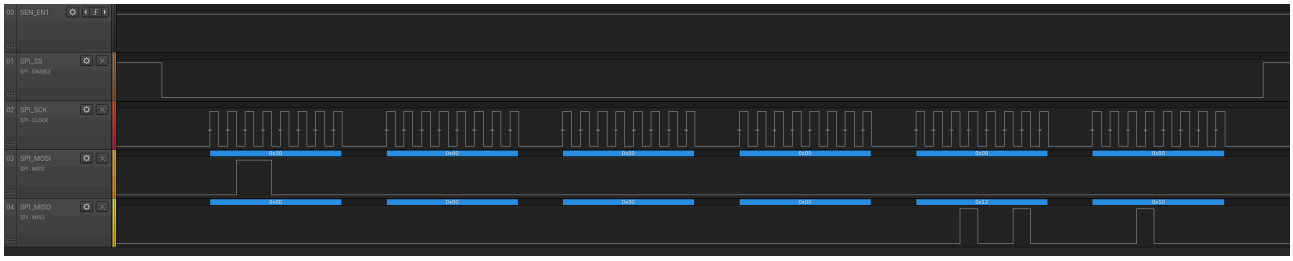


Figure 1: Assembly test, Basic Read Test SPI pattern

7.2 Communication Test

The communication test checks that the communication with the sensor is functioning.

SPI bus communication is tested by writing and reading data to and from the sensor. The test contains a longer sequence of random bit patterns which is good to use for detecting glitches and runt pulses in the HW implementation that could increase the risk of communication failure. This test is also useful in bring-up to investigate the maximum stable SPI bit rate. The log also contains information on effective bit-rate, which can be used as a guide to whether the SPI driver is working properly.

7.3 Enable Pin Test

The enable pin is tested by setting up the sensor in a known state and then resetting the sensor by toggling the enable pin. This test is useful in bring-up to reassure that the Enable pin is correctly connected and that the integration functions `acc_hal_integration_sensor_enable()` and `acc_hal_integration_sensor_disable()` are working.

7.4 Interrupt Test

The interrupt test checks that the interrupt pin functionality is correct. This test is useful in bring-up to ensure that the interrupt connectivity and pin selection in the HAL is correct. It also tests that the integration function `acc_hal_integration_wait_for_sensor_interrupt()` is working. In addition, the test also verifies the bandwidth requirement on the interrupt path to ensure that no sensor interrupts are lost.

The test log contains information on the latency of the host interrupt handling.

7.5 Clock & Supply Test

The clock and supply test will check the reference clock and the quality of the voltage supplies. The log contains information about the test result for the clock and supply tests.

The clock test checks that the reference clock for the sensor is valid. If the HW implementation includes an externally generated reference clock, the test also ensures that the clock control connectivity and pin selection in the HAL is correct. If this test fails, it could be that the crystal doesn't oscillate at the correct frequency, for example due to incorrect crystal tuning capacitor values. The crystal frequency could also vary due to crosstalk from signals that have accidentally been routed too close to the crystal or its traces.

The supply test will check that the sensor is properly powered. This test can be used in bring-up to ensure that all the power supplies are valid during operation.

7.6 Sensor Calibration Test

The calibration step will use "maximum" current and will therefore be a good indication of whether the output current of the 1.8V power supply is sufficient.

© 2026 by Acconeer AB - All rights reserved

8 Disclaimer

The information herein is believed to be correct as of the date issued. Acconeer AB (“Acconeer”) will not be responsible for damages of any nature resulting from the use or reliance upon the information contained herein. Acconeer makes no warranties, expressed or implied, of merchantability or fitness for a particular purpose or course of performance or usage of trade. Therefore, it is the user’s responsibility to thoroughly test the product in their particular application to determine its performance, efficacy and safety. Users should obtain the latest relevant information before placing orders

Unless Acconeer has explicitly designated an individual Acconeer product as meeting the requirement of a particular industry standard, Acconeer is not responsible for any failure to meet such industry standard requirements.

Unless explicitly stated herein this document Acconeer has not performed any regulatory conformity test. It is the user’s responsibility to assure that necessary regulatory conditions are met and approvals have been obtained when using the product. Regardless of whether the product has passed any conformity test, this document does not constitute any regulatory approval of the user’s product or application using Acconeer’s product.

Nothing contained herein is to be considered as permission or a recommendation to infringe any patent or any other intellectual property right. No license, express or implied, to any intellectual property right is granted by Acconeer herein.

Acconeer reserves the right to at any time correct, change, amend, enhance, modify, and improve this document and/or Acconeer products without notice.

This document supersedes and replaces all information supplied prior to the publication hereof.

© 2026 by Acconeer AB - All rights reserved