A121 SW Integration

User Guide

A121 SW Integration

User Guide

Author: Acconeer AB

Version:a121-v1.13.0

Acconeer AB March 26, 2026

© 2026 by Acconeer AB - All rights reserved

Contents

© 2026 by Acconeer AB - All rights reserved

1 Acconeer SDK Documentation Overview

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

Table 1: SDK document overview.

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.	- Working with Raspberry Pi
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
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)	

© 2026 by Acconeer AB - All rights reserved

README	Various target specific information and links	- After SDK download

© 2026 by Acconeer AB - All rights reserved

2 Introduction

This document aims to provide help and support for customers that wish to integrate A121 towards an MCU or processor that is not included in Acconeer's module offering. This document covers the software needed to integrate Acconeer's libraries with MCU specific drivers. Hardware integration is covered in the document "Hardware and physical integration guideline" that is available for download at the developer site.

The A121 pulse coherent radar sensor is dependent on a software stack running on a host MCU. The host software handles low-level configuration of the radar sensor and pre-processing of radar data. All low-level sensor configurations are uploaded to the sensor at startup, meaning that no firmware or configuration parameters are stored in the sensor permanently.

Acconeer have selected a few MCUs to verify our software against for each release. Some of them have also been included in different versions of our EVKs or modules.

3 Acconeer EVK and Modules

Acconeer has designed EVKs and Modules based on the following MCU/Processors. All of which can be bought on Digi-Key.

- XC120 + XE121 A121 EVK
- Raspberry Pi 4: Our standard EVK (XE121), running Raspberry Pi OS.
- STM32L431CB: An ARM Cortex M4 MCU that is featured in our XM125 Entry+ module.

Documentation, including schematics and BOM, for above HW is available from developer site.

4 Acconeer Software Delivery

Acconeer provides software deliveries to the EVKs and Modules on our developer site. It contains example programs and fully implemented HAL for respective hardware for an easy start of development.

Acconeer also provides an SDK for ARM Cortex M0, M4 and M7 based MCUs. The SDK contains the libraries for respective MCU and example programs to show how to use the different services in Radar System Software (RSS). Acconeer also provides a reference implementation of the Hardware Abstraction Layer (HAL) for MCUs from ST Microelectronics and this implementation can be used for a quick start with STM32CubeIDE.

Instructions on how to integrate STM32 MCUs from ST Microelectronics and set up a project in STM32CubeIDE can be found on the developer site. For other MCUs a Hardware Abstraction Layer (HAL) integration must be written according to the guidelines in this document.

5 HW Integration Overview

Hardware integration is covered in the document "Hardware and physical integration guideline" that is available for download at the developer site. The purpose of this section is only to define the pin names and connections necessary for the software integration.

© 2026 by Acconeer AB - All rights reserved

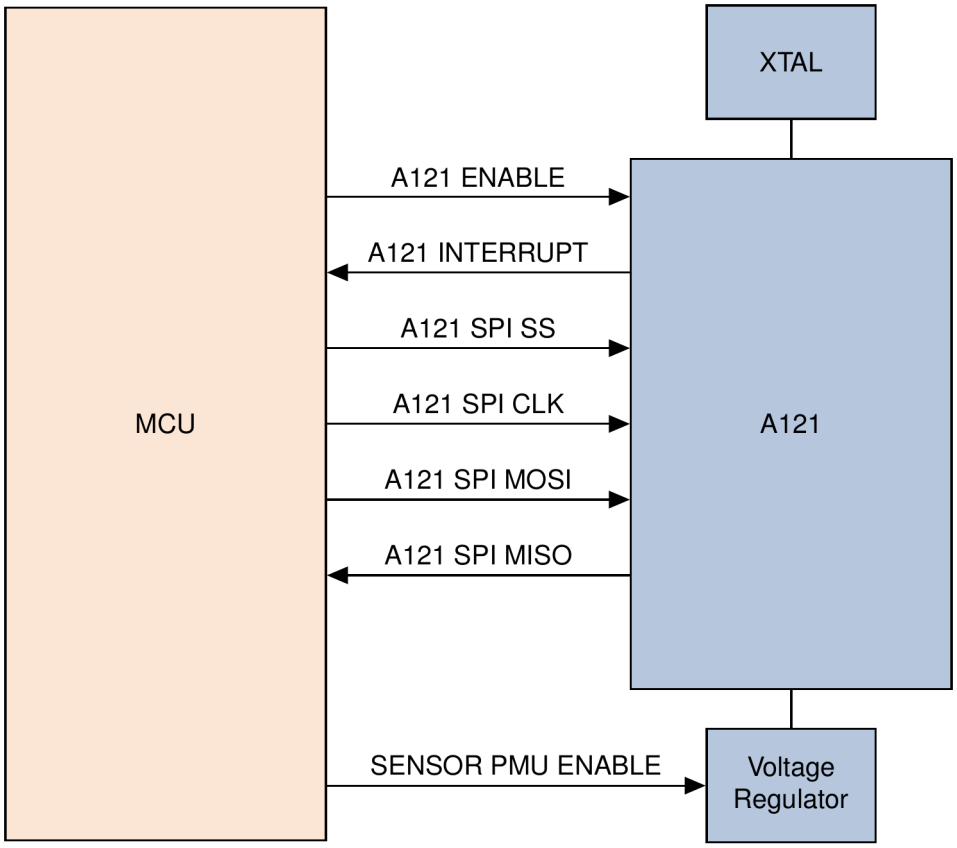

Figure 1: Pin names and connections between A121 and MCU

5.1 GPIO Control Signals

The Acconeer Sensor SW package uses two GPIO control signals: A121_ENABLE and A121_INTERRUPT. A121_ENABLE signal is used to turn the A121 sensor on and off. The A121_ENABLE signal is active high. The A121_INTERRUPT signal is used to signal the host MCU that the internal buffer memory contains new measurement data ready to be transferred via the SPI interface. The A121_INTERRUPT signal is active high.

5.2 SPI Bus

The A121 communicates with the host MCU via a 4-line SPI interface. The maximum supported SPI clock frequency is 50 MHz. The A121 is an SPI slave device. The SPI signals are named A121_SPI_CLK, A121_SPI_MOSI, A121_SPI_MISO, and A121_SPI_SS_N. Note that the slave select signal, A121_SPI_SS_N, is active low and often controlled by a GPIO on the MCU.

5.3 Power Control

Some hardware designs include a Power Management Unit (PMU) or a power switch that allows the MCU to completely shut off the power to the A121 sensor.

5.4 Crystal

The A121 radar sensor needs a crystal or an external clock source for the 24 MHz clock reference.

6 Prototype Integrations

For prototype integrations with an MCU development board we recommend using the XE121. It is a breakout board with an A121 radar sensor, crystal and pin-headers that matches the layout of Nucleo-64, Nucleo-144, Arduino and Raspberry Pi.

© 2026 by Acconeer AB - All rights reserved

7 Software Integration

The Acconeer software delivery consists of an SDK with pre-compiled RSS libraries, headers and example programs to show how to use the different services and detectors.

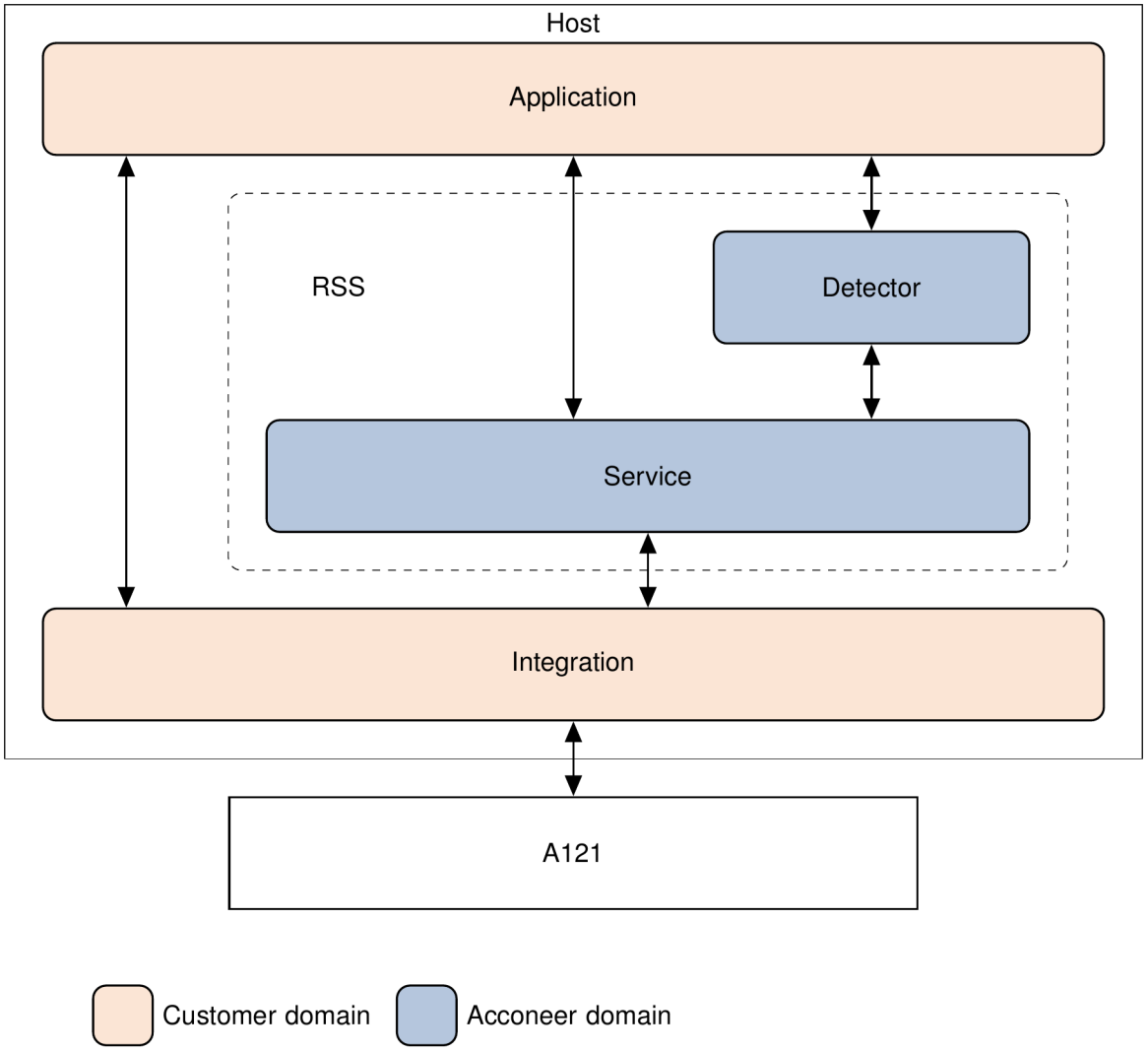

Figure 2: Overview of Radar System Software

Radar System Software (RSS) is the software that will help you interact with the A121 radar sensor. To properly function, this software utilizes MCU specific functions to handle memory and communication with the sensor. Each integration of the sensor to a hardware requires a specific pin configuration and different drivers from the MCU. This is solved by a user-written Hardware Abstraction Layer (HAL). The HAL is a glue layer between RSS and the MCU drivers.

Before the RSS can be used a HAL must be registered. The HAL struct is passed as an argument to the function acc_rss_hal_register(). The HAL struct contains properties and function pointers that RSS use in its communication with the hardware. The HAL struct and the function pointer types are declared in the header file acc_hal_definitions_a121.h.

Acconeer provide fully verified implementations of a HAL for our EVKs and Modules in the respective software deliveries. Reference implementations are also provided for STM32 MCUs in the Cortex M0, Cortex M4 and Cortex M7 packages, see for example the file acc_hal_integration_stm32cube_xe121_single_sensor.c

© 2026 by Acconeer AB - All rights reserved

7.1 HAL Struct

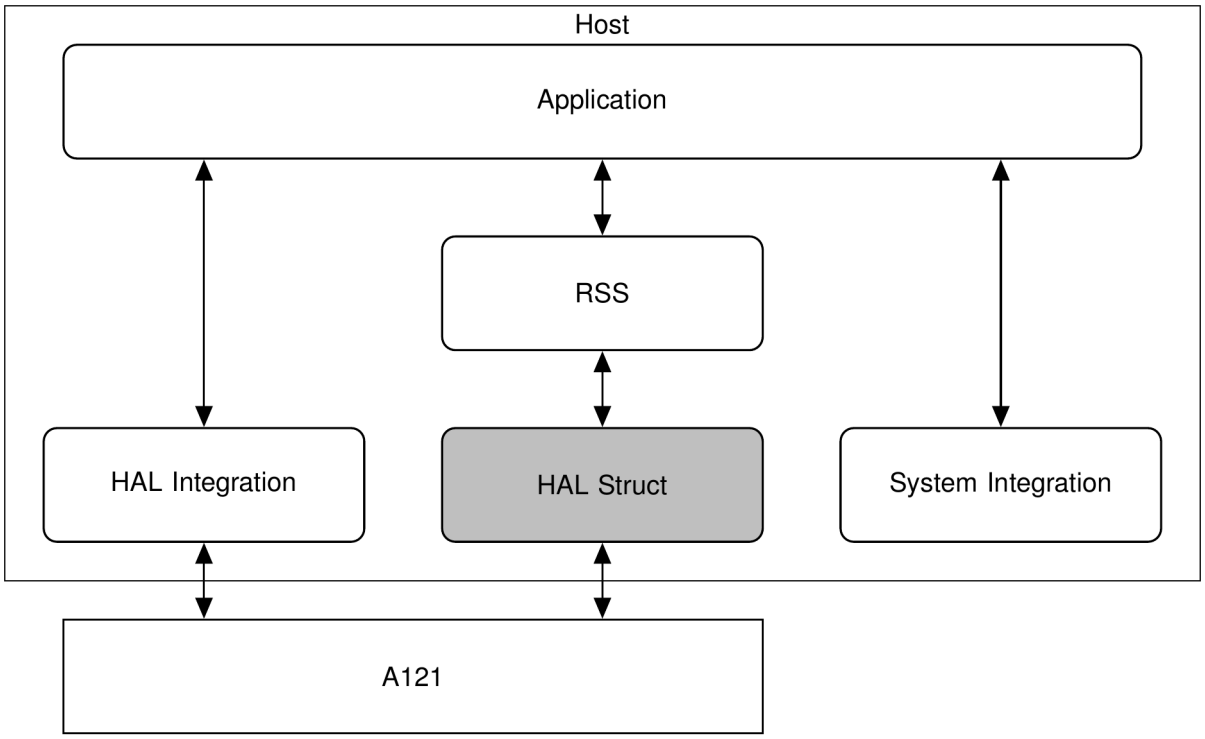

Figure 3: HAL Struct Integration

The following sub-chapters describe how to set the properties and write the functions that are included in the HAL struct.

The **HAL Struct** is defined in *acc_hal_definitions_a121.h* and it is typically declared by and returned from the function **acc_hal_rss_integration_get_implementation** in the file *acc_hal_integration_.c*, for example *acc_hal_integration_stm32cube_xe121_single_sensor.c*.

The functions in the HAL struct are the only external functions outside the C standard library that are called by RSS.

7.1.1 The max_spi_transfer_size Property

The max_spi_transfer_size should be set to the maximum buffer size that can be transferred over the SPI bus. If there is no restriction, the limit should be set to SIZE_MAX.

7.1.2 Mem-alloc Function

Return pointer to memory, NULL is seen as failure. Allocated memory should be naturally aligned.

7.1.3 Mem-free Function

Free memory which is previously allocated.

7.1.4 Sensor Transfer Function

The sensor transfer function is called by RSS to transfer data to and from the radar sensor over the SPI bus. The maximum buffer size can be limited by setting the property max_spi_transfer_size in the HAL struct. The buffer size is always a multiple of two and the buffer is guaranteed to be 16 bit aligned even though the transfer function is defined with an 8-bit array. It is beneficial from performance perspective to utilize DMA if available and as big buffer transfer size as possible.

The function should do the following:

© 2026 by Acconeer AB - All rights reserved

- Set A121_SPI_SS_N Low
- Send and receive SPI buffer
- Set A121_SPI_SS_N High

7.1.5 Log Function

The purpose of the log function is to format log messages and to print them to e.g. the console or debug UART.

The function below shows an example of how the log formatting can be implemented:

```
#define LOG_BUFFER_MAX_SIZE 150
#define LOG_FORMAT "%02u:%02u:%02u.%03u (%c) (%s): %s\n"

void acc_hal_integration_log(acc_log_level_t level, const char *module,
    const char *format, ...)
{
    char    log_buffer[LOG_BUFFER_MAX_SIZE];
    va_list ap;

    va_start(ap, format);

    int ret = vsnprintf(log_buffer, LOG_BUFFER_MAX_SIZE, format, ap);
    if (ret >= LOG_BUFFER_MAX_SIZE)
    {
        log_buffer[LOG_BUFFER_MAX_SIZE - 4] = '.';
        log_buffer[LOG_BUFFER_MAX_SIZE - 3] = '.';
        log_buffer[LOG_BUFFER_MAX_SIZE - 2] = '.';
        log_buffer[LOG_BUFFER_MAX_SIZE - 1] = 0;
    }

    uint32_t time_ms = acc_hal_integration_get_current_time();
    char     level_ch;

    unsigned int timestamp    = time_ms;
    unsigned int hours        = timestamp / 1000 / 60 / 60;
    unsigned int minutes      = timestamp / 1000 / 60 % 60;
    unsigned int seconds      = timestamp / 1000 % 60;
    unsigned int milliseconds = timestamp % 1000;

    level_ch = (level <= ACC_LOG_LEVEL_DEBUG) ? "EWIVD"[level] : '?';

    printf(LOG_FORMAT, hours, minutes, seconds, milliseconds, level_ch,
       module, log_buffer);

    va_end(ap);
}
```

7.1.6 16-bit Sensor Transfer Function

The 16-bit sensor transfer function is optional and can be implemented on supported integrations to optimize the SPI data transfer time. It will do the same as the normal transfer function except that it will transfer data over SPI to the sensor in 16 bit chunks.

© 2026 by Acconeer AB - All rights reserved

7.2 HAL Integration Functions

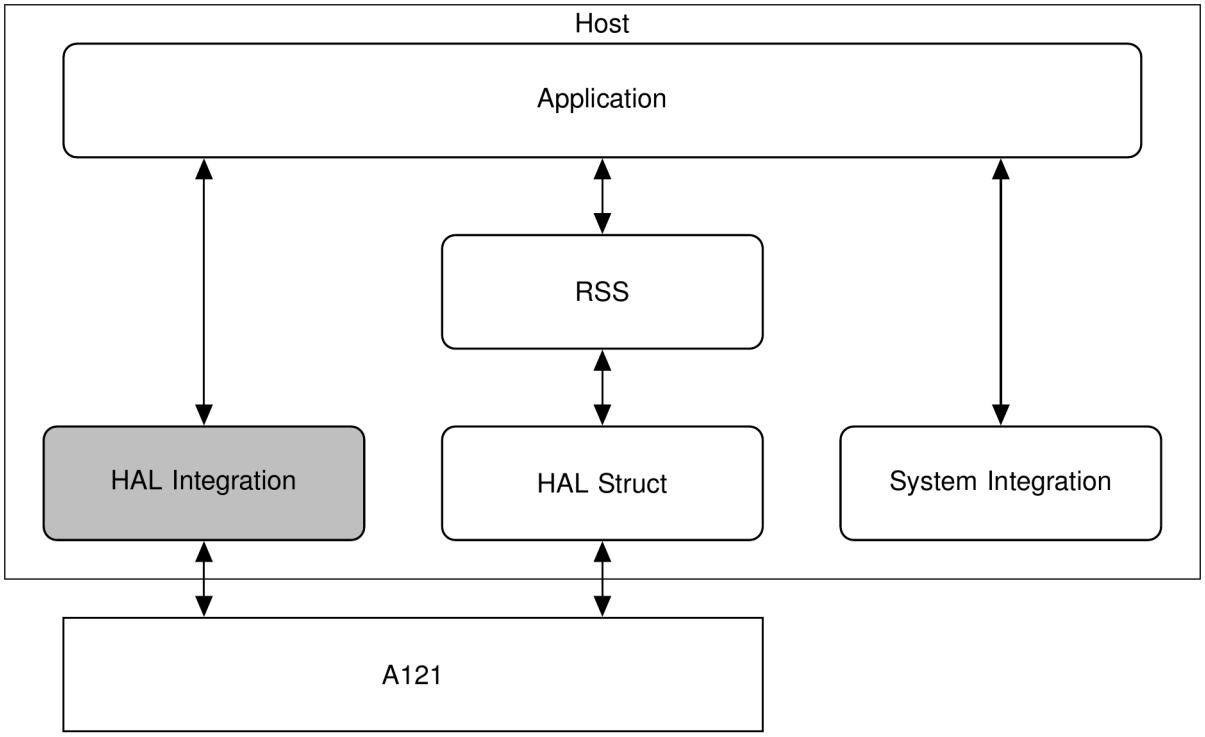

Figure 4: HAL Integration

The HAL Integration is used to define a common API for sensor control from the example and reference applications and it contains functionality for sensor control, such as power on/off and enable/disable. These functions are not used by the RSS library. It is the responsibility of the application to handle power supply, control the A121_ENABLE pin, and wait for interrupt signals. The HAL Integration functions described in this section are intended to help applications to perform this functionality.

The **HAL Integration** functions are defined in *acc_hal_integration_a121.h* and they are typically declared and implemented in the file *acc_hal_integration_.c*, for example *acc_hal_integration_stm32cube_xe121_single_sensor.c*.

The following sub-chapters describe how to write the integration functions that are used by example code to control the sensor.

7.2.1 Sensor Supply On Function

The supply on function is called to enable the supply to the sensor. On some boards, like the XE121, the sensor is always powered. In that case this function is empty.

When the sensor is powered the MCU is allowed to drive the GPIO signals towards the A121 sensor high, for example the A121_SPI_SS_N.

7.2.2 Sensor Supply Off Function

The supply off function is called to disable the supply to the sensor. On some boards, like the XE121, the sensor is always powered. In that case this function is empty.

All GPIO signals from the MCU to the sensor that is driven by the MCU, for example the A121_SPI_SS_N, must be set LOW before disabling power to the sensor.

7.2.3 Sensor Enable Function

The sensor enable function is called to enable the sensor.

The function should do the following:

© 2026 by Acconeer AB - All rights reserved

- Prerequisite: The sensor is powered on and A121_SPI_SS_N is set to high
- Set the A121_ENABLE signal High.
- Wait for sensor crystal to stabilize. This time depends on the used integration and crystal, see “A121 Datasheet” for more information.

7.2.4 Sensor Disable Function

The sensor disable function is called to disable the sensor.

The function should do the following:

- Set the A121_ENABLE signal Low.
- Wait 2 ms

7.2.5 Wait for Interrupt Function

This function shall wait at most timeout_ms for the A121_INTERRUPT pin to become active and then return true. If the A121_INTERRUPT pin is not high after timeout_ms the function shall return false.

A simple and portable way of implementing the wait for interrupt function is to use polling. The function should then repeatedly do the following until the interrupt pin is high or a timeout has occurred:

- Check the A121_INTERRUPT pin and return true if interrupt pin is HIGH
- Sleep a short time
- Return false if timeout
- Start over and check the interrupt pin again.

While polling is great for maximum portability an optimized implementation may take advantage of HW interrupt support in the MCU to reduce latency and power consumption.

7.2.6 Sensor Count Function

This function should return the number of sensors connected in the system.

© 2026 by Acconeer AB - All rights reserved

7.3 System Integration Functions

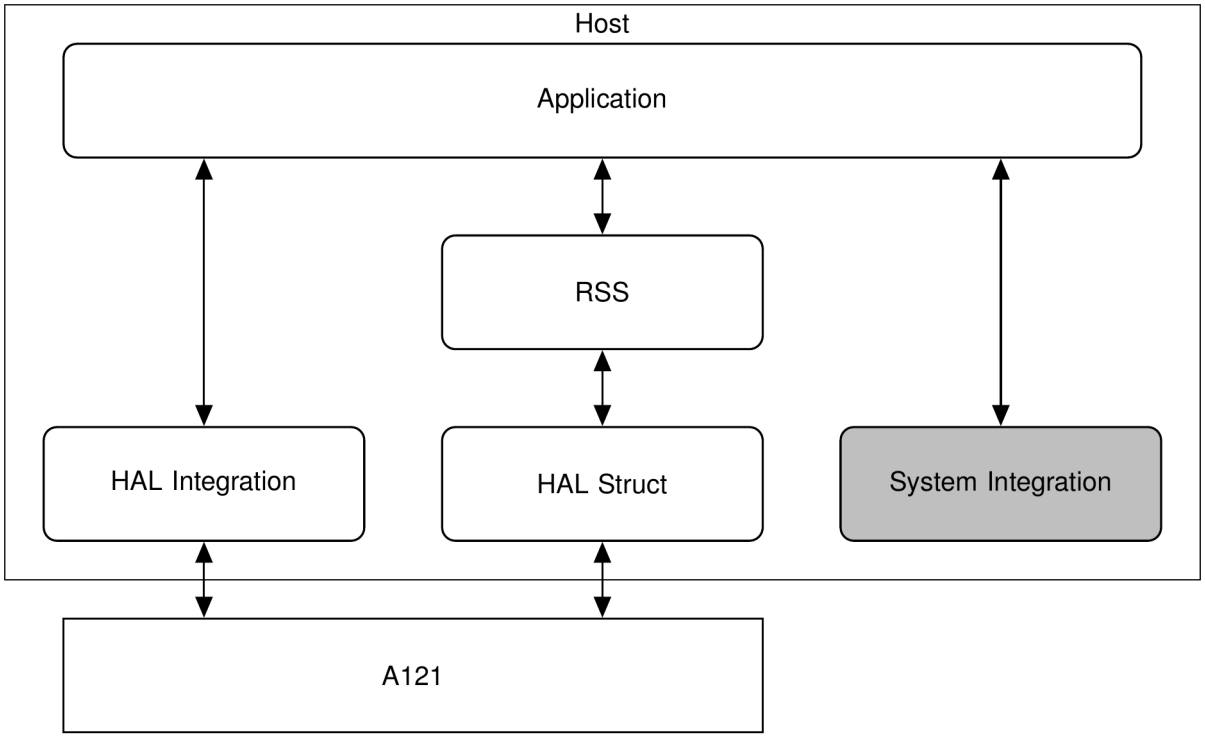

Figure 5: System Integration

The System Integration contains functionality for timekeeping and memory allocations. It is used to make the reference applications and example applications portable between different hardware platforms.

The **System Integration** functions are defined in *acc_integration.h* and they are typically declared and implemented in the file *acc_integration_.c*, for example *acc_integration_stm32.c*.

7.3.1 Timekeeping

These functions are used for timekeeping in the host system.

Get Current Time in Milliseconds

```
uint32_t acc_integration_get_time(void)
```

Millisecond Sleep

```
void acc_integration_sleep_ms(uint32_t time_msec)
```

Microsecond Sleep

```
void acc_integration_sleep_us(uint32_t time_usec)
```

7.3.2 Memory Handling

These functions are used to allocate and free memory.

Allocate Memory

```
void *acc_integration_mem_alloc(size_t size)
```

Allocate and Clear Memory

```
void *acc_integration_mem_calloc(size_t nmemb, size_t size)
```

Free Memory

© 2026 by Acconeer AB - All rights reserved

```
void acc_integration_mem_free(void* ptr)
```

7.3.3 Low Power Functionality

These functions are used by the low power examples. The *acc_integration_set_periodic_wakeup* function will setup the wakeup interval and the *acc_integration_sleep_until_periodic_wakeup* function will put the host system into its lowest power state and sleep until the periodic wakeup time is due.

Note that these functions are only available for targets that include the low power example and reference applications, such as the XM125 module.

Set Periodic Wakeup

```
void acc_integration_set_periodic_wakeup(uint32_t time_msec)
```

Sleep Until Periodic Wakeup

```
void acc_integration_sleep_until_periodic_wakeup(void)
```

7.4 Hibernate

The sensor supports hibernation where the program memory in the sensor is retained when the sensor is disabled. This will allow very low power consumption without the penalty of having to completely re-initialize the sensor. Re-initialization of the sensor includes transfer of the whole sensor program from the host to the sensor over SPI.

Hibernate is entered by first calling acc_sensor_hibernate_on() to enable retention of the program memory. Then, the sensor disable function is called:

```
bool enter_hibernate(acc_sensor_t *sensor)
{
    bool status = true;

    if (!acc_sensor_hibernate_on(sensor))
    {
        printf("acc_sensor_hibernate_on failed\n");
        status = false;
    }

    acc_hal_integration_sensor_disable(SENSOR_ID);
    return status;
}
```

Hibernate is exited by first calling the sensor enable function followed by a call to acc_sensor_hibernate_off() to disable retention of the program memory:

```
static bool exit_hibernate(acc_sensor_t *sensor)
{
    bool status = true;

    acc_hal_integration_sensor_enable(SENSOR_ID);
    if (!acc_sensor_hibernate_off(sensor))
    {
        printf("acc_sensor_hibernate_off failed\n");
        status = false;
    }

    return status;
}
```

8 References, List of Documentation

- Hardware and physical integration guideline
- STM32CubeIDE User Guide

© 2026 by Acconeer AB - All rights reserved

- A121 Datasheet

All Documents referred to can be found on the developer site.

© 2026 by Acconeer AB - All rights reserved

9 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