

A121 STM32CubeIDE

User Guide

A121 STM32CubeIDE

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 Getting Started with STM32CubeIDE ... 6
3.1 MCU/Board Selection ... 6
3.2 Project Setup ... 7
3.3 Pin Configuration ... 8
3.3.1 Pin Configuration with XE121 ... 11
3.4 Interrupt Configuration ... 12
3.5 GPIO Configuration ... 12
3.6 XE121 Sensor Selection ... 13
3.7 XE121 Single Sensor Setup ... 13
3.8 SPI Configuration ... 13
3.9 Code Generation ... 14
4 Configuring Project for Acconeer Software ... 16
4.1 Adding Acconeer Software ... 16
4.1.1 Source-files ... 16
4.1.2 Header-files ... 16
4.1.3 Libraries ... 16
4.2 Project Settings ... 17
4.3 Adding Print Functionality with UART/USART ... 17
4.3.1 Find STM32 Board COM port ... 17
4.3.2 Start and Configure PuTTY ... 18
5 Using STM32CubeIDE 2.x and STM32CubeMX 2 (CMake Projects) ... 21
5.1 Creating the Project ... 21
5.2 Adding Acconeer Software to the CMake Build ... 21
5.2.1 Libraries and Link Order ... 22
5.3 Project Settings ... 22
5.4 Heap and Stack Configuration ... 23
5.5 Adding Print Functionality ... 23
5.6 Devices Using the New-Generation STM32 HAL ... 23
5.7 User Code and Running the Examples ... 24
6 HAL Integration File ... 25
6.1 Selecting the Appropriate HAL-integration File ... 25
6.2 A121_SPI_HANDLE ... 25
7 Running a Radar Sensor Example ... 26
7.1 Running the Program ... 26
8 Running the Exploration Server ... 27
8.1 STM32CubeMX ... 27
8.2 STM32CubeIDE ... 28
8.2.1 Running the Program ... 29
9 Troubleshooting and FAQ ... 30
9.1 Example Fails ... 30
9.1.1 Sensor Creation Returns NULL ... 30
9.1.2 Config Creation Hardfaults ... 30
9.2 The Program is Stuck in HAL_Delay ... 30
9.3 Troubleshooting SPI Communication ... 30
9.4 UART Problems ... 31
9.5 Link Errors ... 31
9.6 Heap Memory Corruption ... 31
9.7 Floating point equality checking errors ... 32
10 Disclaimer ... 34

© 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

In this document there will be a short guide, with an example, on how to generate a project and setup the Acconeer software in STM32CubeIDE.

The MCU board used as an example in this guide is a Nucleo-L476RG. We will show how to connect an XE121, including the A121 radar sensor. There is some extra logic on the XE121 board to support multiple sensors, which is typically not present on boards with only one sensor. To show how a typical single sensor integration can be done, we have also included some notes on how to use the XE121 in a single sensor setup.

STM32CubeIDE can be downloaded from the ST website at: https://www.st.com/en/development-tools/stm32cubeide.html

This guide has been verified in Windows with STM32CubeIDE 1.19.0.

Starting with STM32CubeIDE 2.0, projects are generated by STM32CubeMX 2 as CMake projects instead of native Eclipse projects. If you are using STM32CubeIDE 2.x, see section 5 Using STM32CubeIDE 2.x and STM32CubeMX 2 (CMake Projects) for how to add the Acconeer software to the project.

© 2026 by Acconeer AB - All rights reserved

3 Getting Started with STM32CubeIDE

This section will cover how to set up a project in STM32CubeIDE, and make sure that the code works with the Acconeer software.

Start STM32CubeIDE and click “Start new STM32 project”. The option is also available under “File → New → STM32Project”.

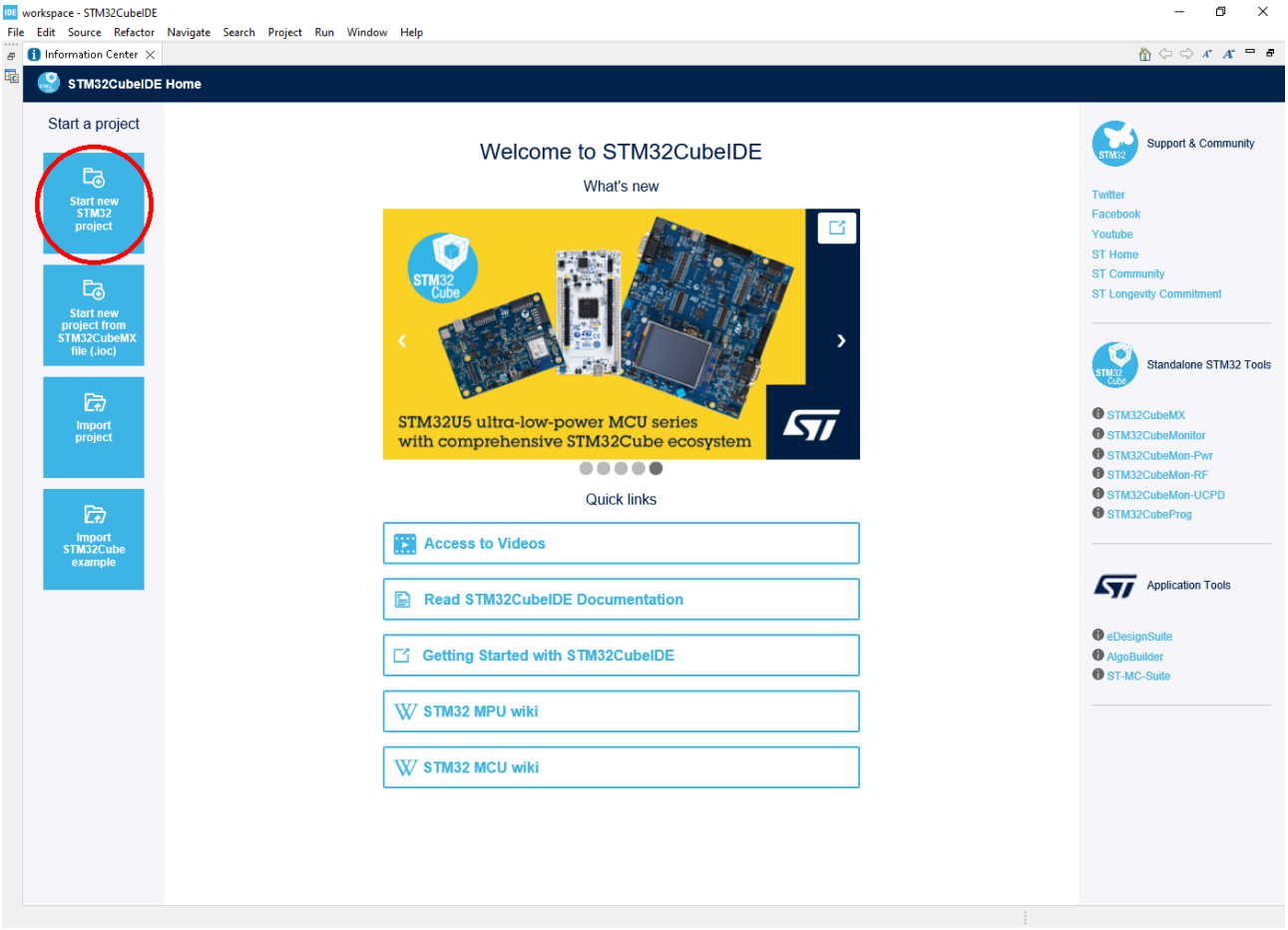


Figure 1: Start new STM32CubeIDE project

When running on Linux/Ubuntu, you might be asked about Connection Parameters, generally, you can skip this part by selecting “No proxy”.

3.1 MCU/Board Selection

Search for the MCU or Board you are working with in the MCU Selector/Board Selector tab. In the example in this document, we use the board “NUCLEO-L476RG”. Start off by searching for “NUCLEO-L47” in the “Part Number Search”-option at the top left in the Board Selector tab.

The board will show up in the “Boards list” at the bottom of the page. Click it. When clicking the board or MCU, you will be given some information about it.

Make sure the “Nucleo-L476RG” board is selected and press the “Next”-button at the bottom of the page.

© 2026 by Acconeer AB - All rights reserved

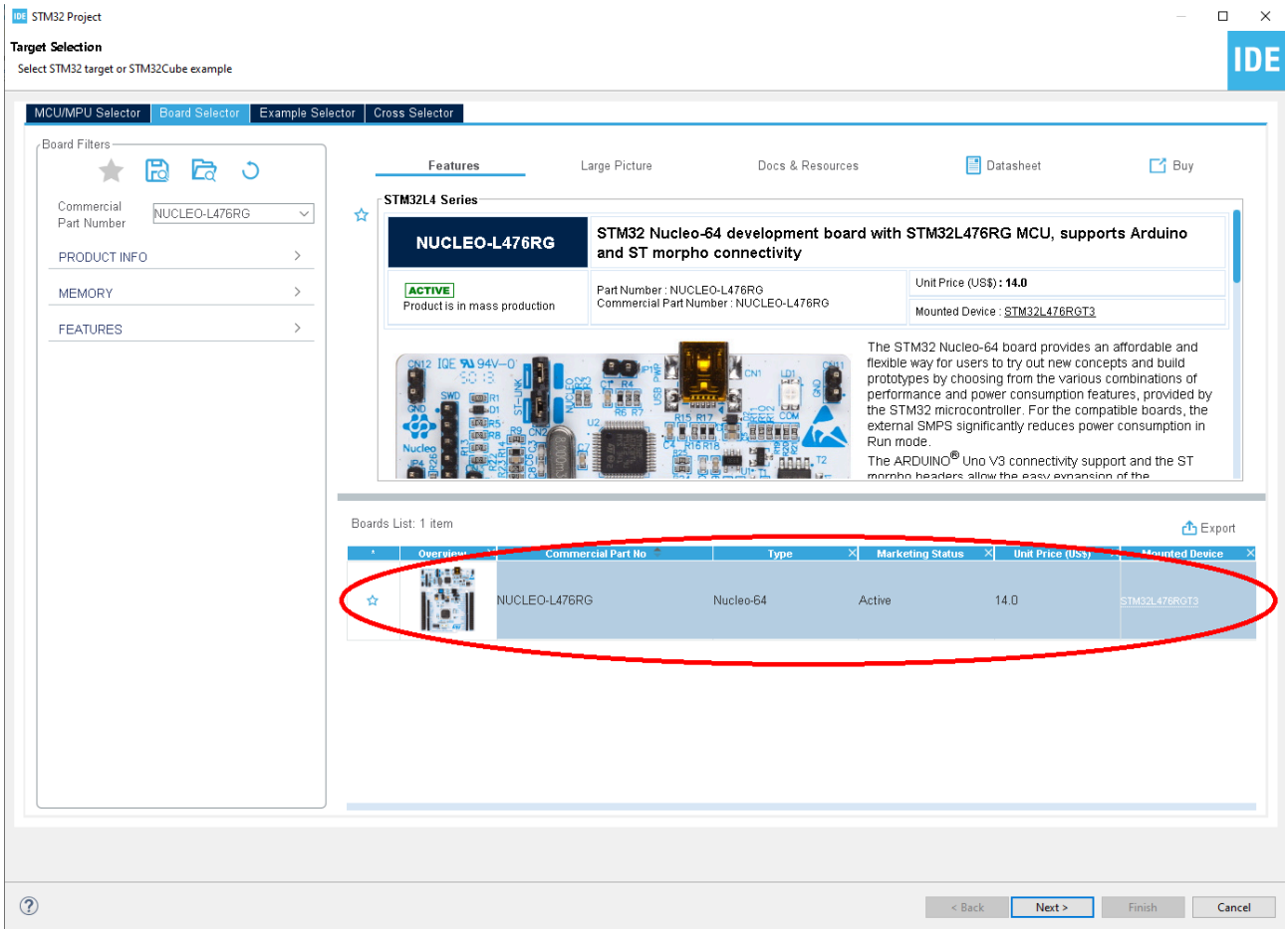


Figure 2: Target Selection

3.2 Project Setup

Select a name and location for your project and select the following options:

- Target language: C
- Target Binary type: Executable
- Target Project Type: STM32Cube

Finally, press the “Finish”-button at the bottom of the page.

© 2026 by Acconeer AB - All rights reserved

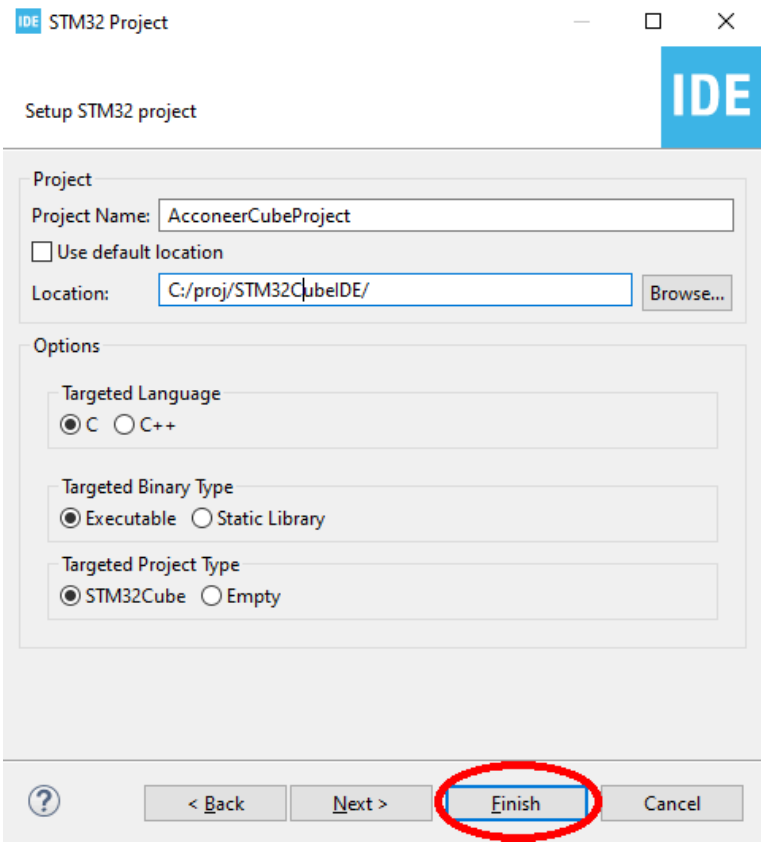


Figure 3: Project Setup

Press “Yes” when you are asked if you want to initialize peripherals to their default mode.

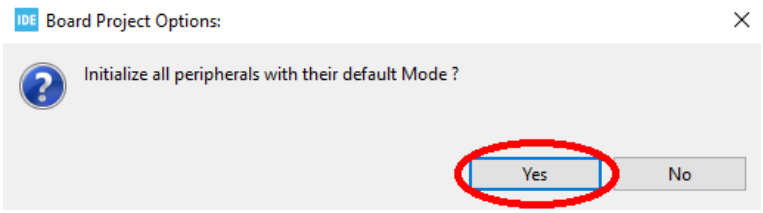


Figure 4: Initialize Peripherals

Press “Yes” when you are asked if you want to open the STM32CubeMx perspective.

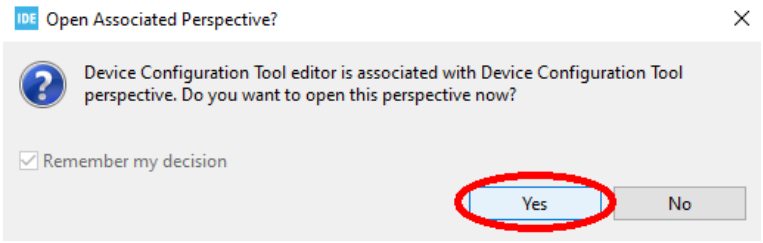


Figure 5: Open STM32CubeMX perspective

3.3 Pin Configuration

The Pinout is flexible – however, it is important that the pins communicating with the radar have the right user labels and that SPI is configured with “Full-Duplex Master”-mode.

In order to perform pin configuration, you need to have the “.ioc”-file open that is named after your project.

The “LD2 [green Led]“ on the Nucleo board shares functionality with “SPI1_CLK“. Before we add SPI functionality, we need to remove the “LD2 [green Led]“ configuration from PA5. The pin configuration is removed by doing a left mouse click on PA5 / “LD2 [green Led]“ and then selecting “Reset_State“.

© 2026 by Acconeer AB - All rights reserved

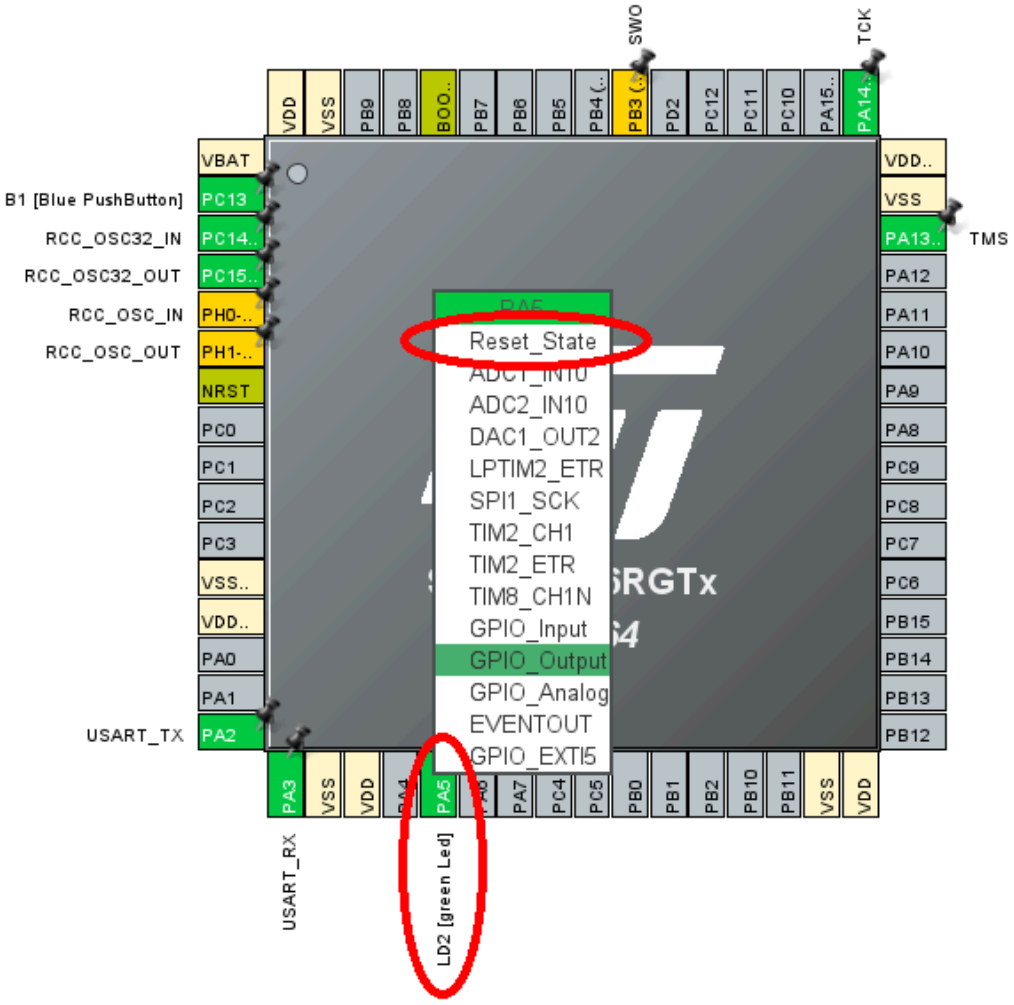


Figure 6: Remove LD2 pin

The “B1 [Blue PushButton]“ on the Nucleo board is not used in our examples and can be removed. The pin configuration is removed by doing a left mouse click on PC13 / “B1 [Blue PushButton]“ and then selecting “Reset_State“.

© 2026 by Acconeer AB - All rights reserved

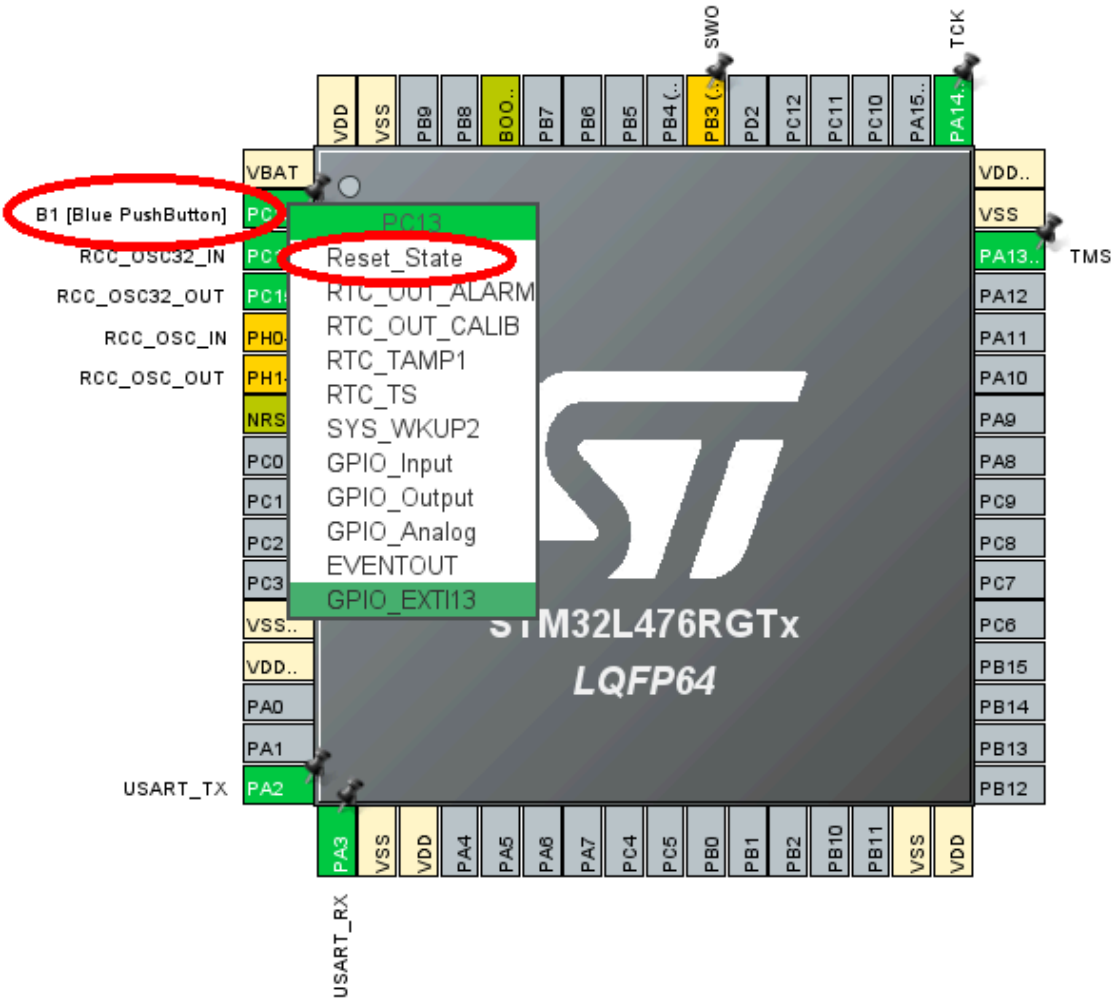


Figure 7: Remove B1 pin

In our example, we activate SPI1 with “Full-Duplex Master”-mode by going into “Connectivity” in the “Pinout and Configuration”-tab. Then by pressing “SPI1” the option of selecting mode is available, “Full-Duplex Master” is selected. When doing this, we get the SPI GPIOs:

© 2026 by Acconeer AB - All rights reserved

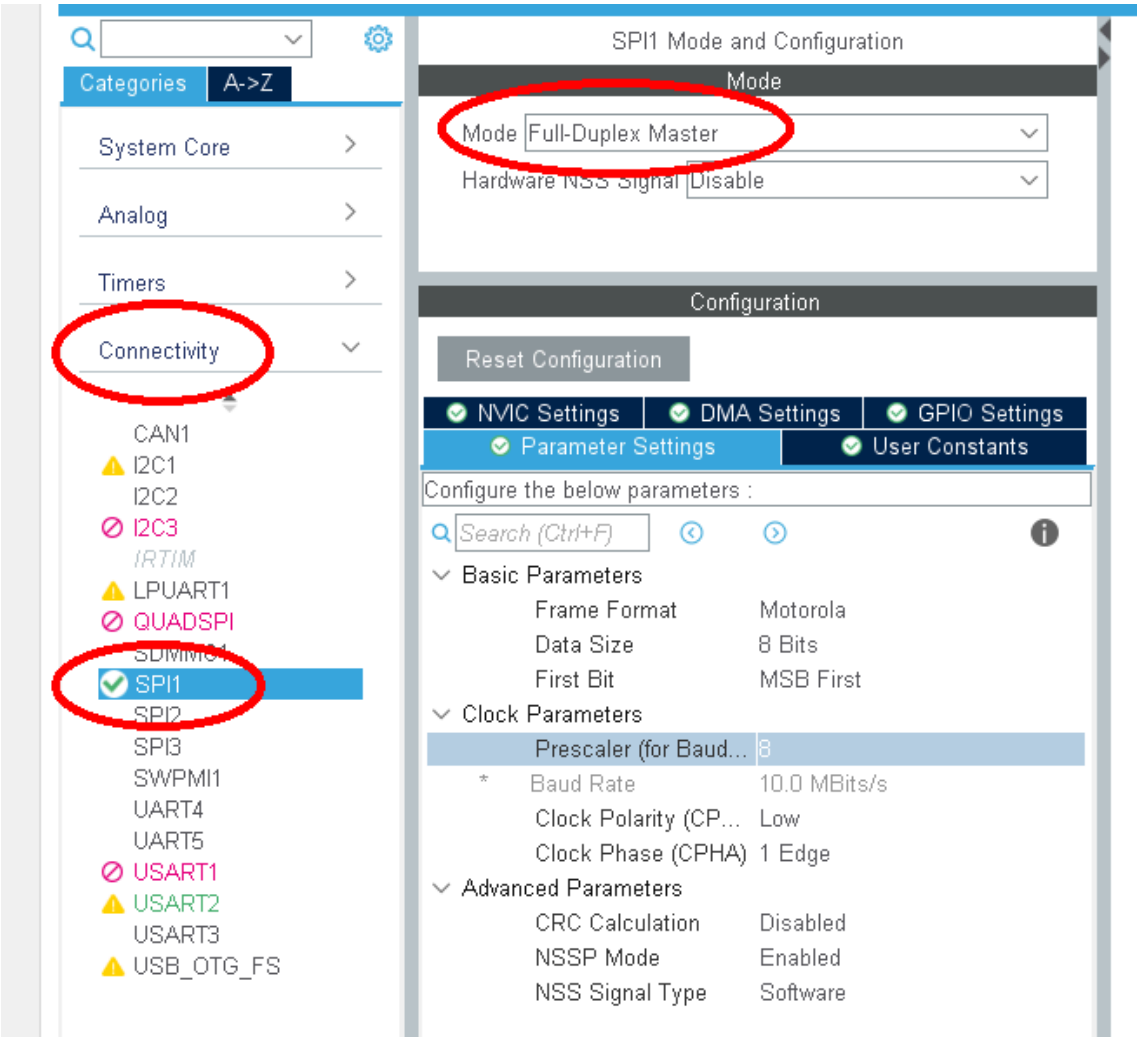


Figure 8: SPI Setup

USER_LABEL	NUCLEO PIN
SPI1_MOSI	PA7
SPI1_MISO	PA6
SPI1_SCK	PA5

3.3.1 Pin Configuration with XE121

In order to set new GPIOs, you can left click the desired pin and select the desired function of the pin.

The table below shows how the XE121 and the NUCLEO-board can be connected:

USER_LABEL	NUCLEO PIN	GPIO TYPE	COMMENT
SEN_EN1	PB0	GPIO_Output	
SEN_EN2	PB5	GPIO_Output	Only needed for multi-sensor
SEN_EN3	PB3	GPIO_Output	Only needed for multi-sensor
SEN_EN4	PA10	GPIO_Output	Only needed for multi-sensor
SEN_EN5	PC1	GPIO_Output	Only needed for multi-sensor
SPI1_MOSI	PA7	SPI1_MOSI	
SPI1_MISO	PA6	SPI1_MISO	
SPI1_SCK	PA5	SPI1_SCK	

© 2026 by Acconeer AB - All rights reserved

USER_LABEL	NUCLEO PIN	GPIO TYPE	COMMENT
A121_SPI_SS	PB6	GPIO_Output	
SPI_SEL0	PA4	GPIO_Output	
SPI_SEL1	PA1	GPIO_Output	
SPI_SEL2	PA0	GPIO_Output	
SEN_INT1	PA9	GPIO_EXTI9	
SEN_INT2	PA8	GPIO_EXTI8	Only needed for multi-sensor
SEN_INT3	PB10	GPIO_EXTI10	Only needed for multi-sensor
SEN_INT4	PB4	GPIO_EXTI4	Only needed for multi-sensor
SEN_INT5	PC0	GPIO_EXTI0	Only needed for multi-sensor

3.4 Interrupt Configuration

The NVIC interrupt for the SEN_INT1, SEN_INT2, SEN_INT3, SEN_INT4, SEN_INT5 pins should be enabled.

1. Select the **GPIO** item under the **System Core**.
2. Select the **NVIC** tab.
3. Tick the checkbox **EXTI line0 interrupt**.
4. Tick the checkbox **EXTI line4 interrupt**.
5. Tick the checkbox **EXTI line[9:5] interrupt**.
6. Tick the checkbox **EXTI line[15:10] interrupt**.

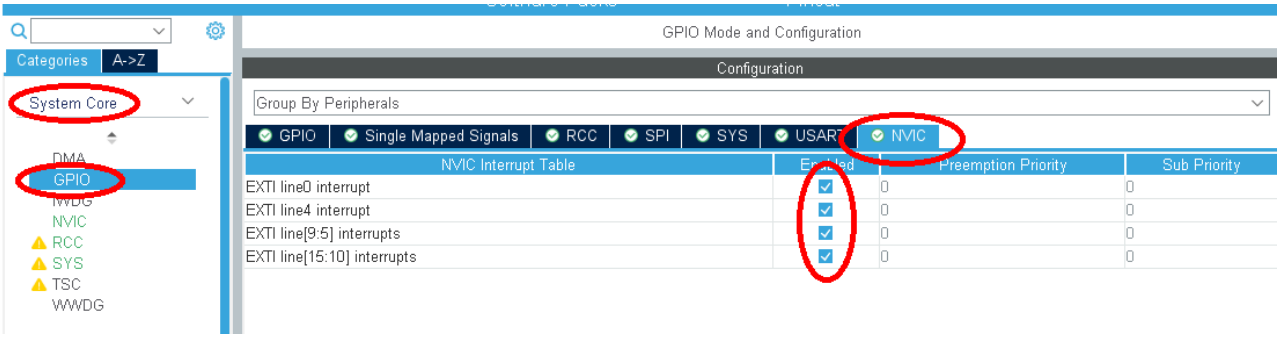


Figure 9: NVIC Configuration

3.5 GPIO Configuration

The GPIO signals from the processor to the sensor should be set up as outputs with very high drive strength.

1. Select the **GPIO** item under the **System Core**.
2. Select the **GPIO** tab.
3. Configure the GPIO pins according to the table below.

USER_LABEL	NUCLEO PIN	GPIO Output Level	Maximum Output speed
SEN_EN1	PB0	Low	Very High
SEN_EN2	PB5	Low	Very High
SEN_EN3	PB3	Low	Very High
SEN_EN4	PA10	Low	Very High
SEN_EN5	PC1	Low	Very High
A121_SPI_SS	PB6	High	Very High
SPI_SEL0	PA4	Low	Very High
SPI_SEL1	PA1	Low	Very High
SPI_SEL2	PA0	Low	Very High

© 2026 by Acconeer AB - All rights reserved

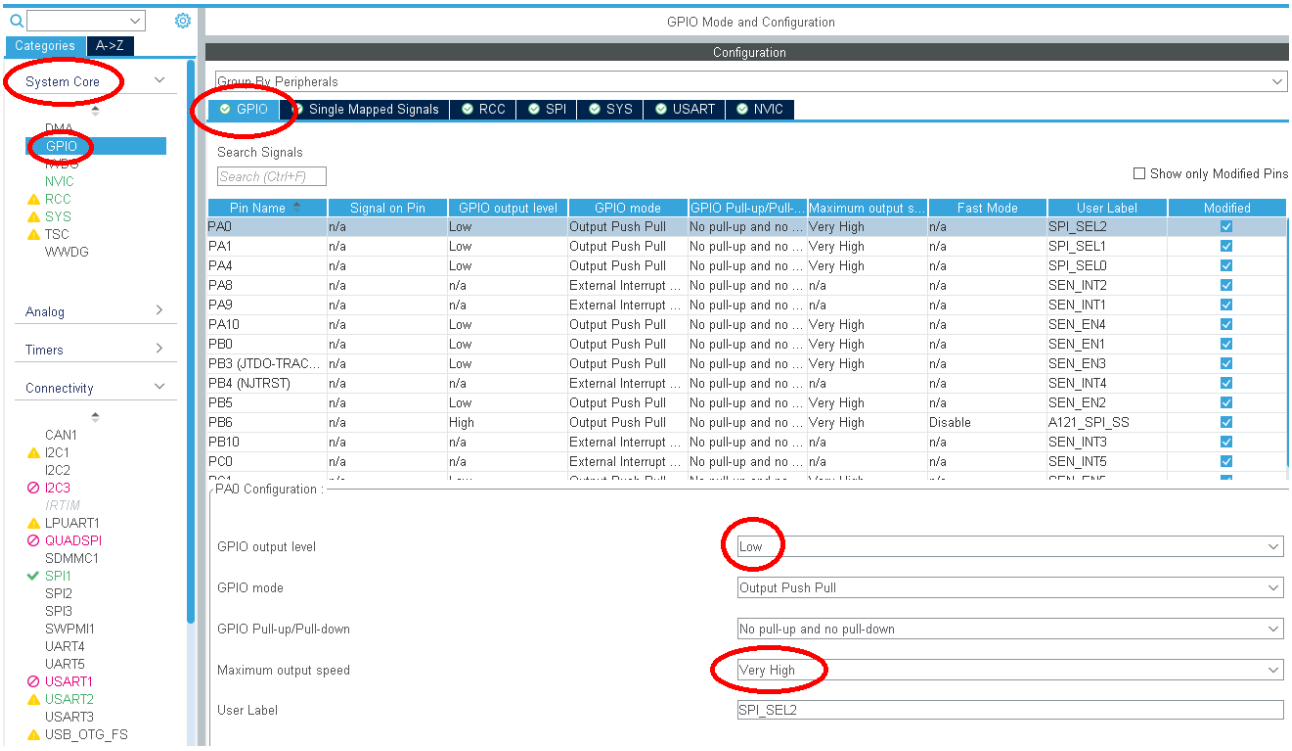


Figure 10: GPIO Configuration

3.6 XE121 Sensor Selection

The XE121 board has support for adding up to 4 extra A121 sensors by using FFC cables and XS121 boards. The selection of which sensor to communicate with is done in the `acc_hal_integration_xe121_multi_sensor.c` file.

SPI_SEL2	SPI_SEL1	SPI_SEL0	Sensor Selection
LOW	LOW	LOW	Sensor 1 (XE121)
LOW	LOW	HIGH	Sensor 2 (XS121 S2)
LOW	HIGH	LOW	Sensor 3 (XS121 S3)
LOW	HIGH	HIGH	Sensor 4 (XS121 S4)
HIGH	LOW	LOW	Sensor 5 (XS121 S5)

3.7 XE121 Single Sensor Setup

If the XE121 board is used in a single sensor setup, there is no need to connect the following signals: SEN_INT2, SEN_INT3, SEN_INT4, SEN_INT5, SEN_EN2, SEN_EN3, SEN_EN4 and SEN_EN5.

Please note that SEN_SEL0, SEN_SEL1 and SEN_SEL2 need to be kept low to select the sensor onboard the XE121.

The `acc_hal_integration_xe121_multi_sensor.c` should be replaced with the `acc_hal_integration_xe121_single_sensor.c` file in the STM32CubeIDE project.

3.8 SPI Configuration

To make the SPI interface work properly with the Acconeer software, you might need to set the Prescaler (for Baud Rate) and Data Size.

Press the SPI you are using and under the Configuration menu you can change parameters. Under “Basic Parameters” you can find that “Data Size” is set to 4 Bits by default, change this to 8 Bits.

Under “Clock Parameters” you will find “Prescaler”. This controls the frequency of the SPI bus. The higher the prescaler value is, the lower the frequency will be. Depending on your setup, you might have to use different prescaler values to fit your project. The A121 sensor supports SPI frequencies up to 50 MHz. In experimental configurations where the sensor and MCU are not mounted on the same PCB, the maximum SPI frequency is often significantly lower and typically around 10 MHz.

© 2026 by Acconeer AB - All rights reserved

It is recommended to use the highest prescaler to begin with in order to make sure the SPI-communication is stable. Once the communication works properly, you can start trying lower prescalers in order to increase the frequency.

You might have to save the project and restart the program in order to access these settings.

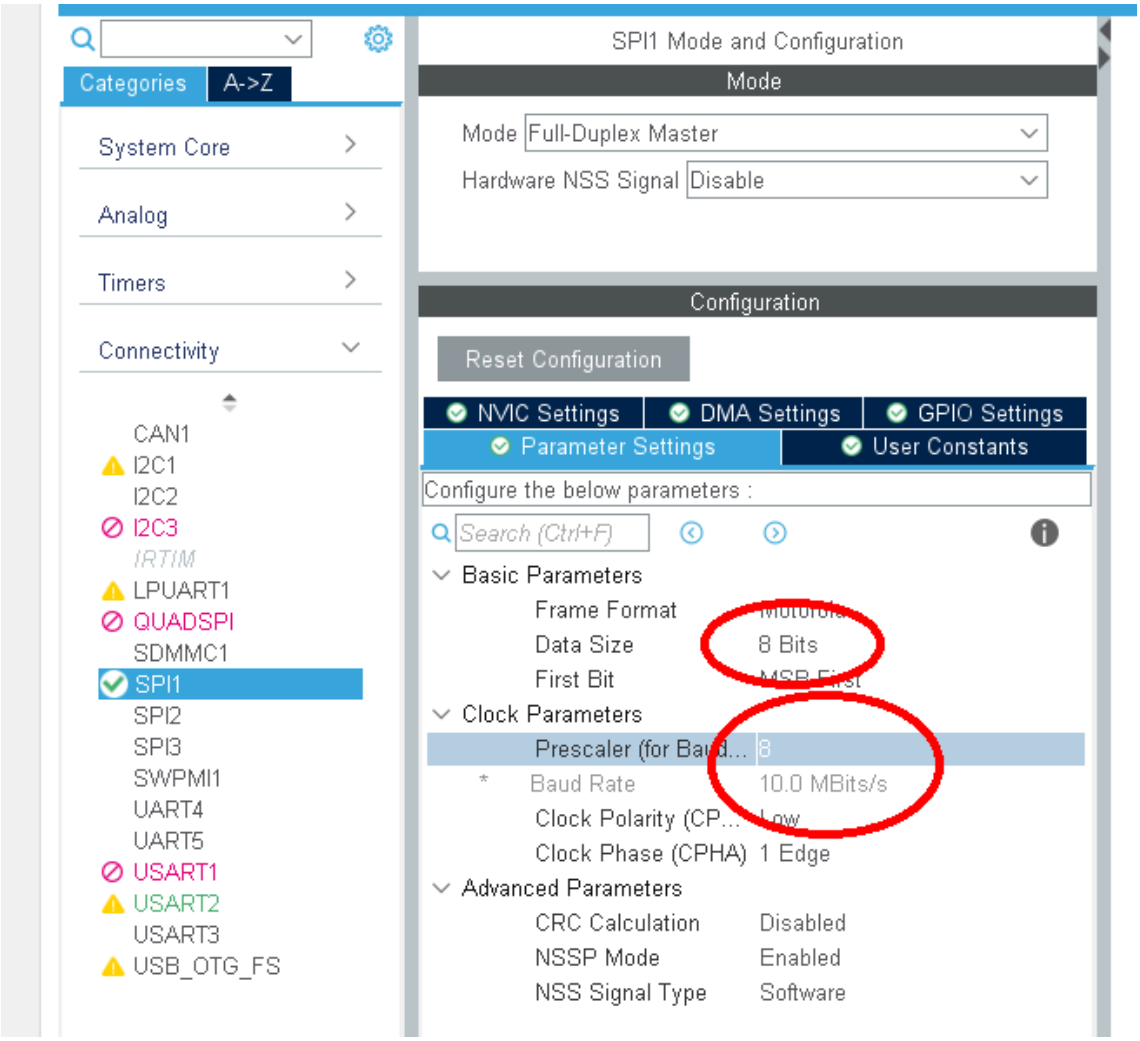


Figure 11: SPI Master Config

3.9 Code Generation

Select Project/Generate Code to generate the driver and MCU configuration.

© 2026 by Acconeer AB - All rights reserved

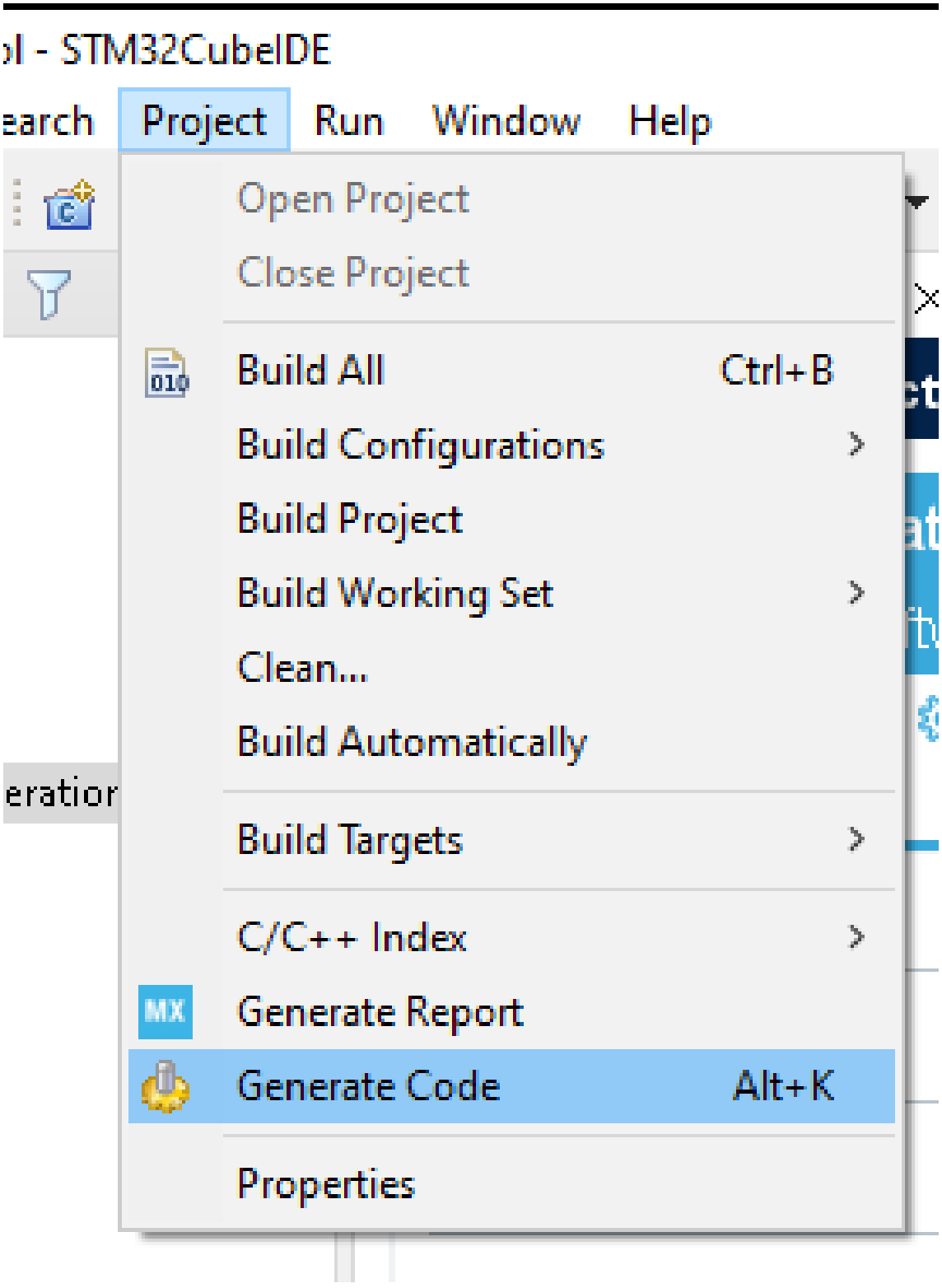


Figure 12: Generate code

© 2026 by Acconeer AB - All rights reserved

4 Configuring Project for Acconeer Software

This section applies to the Eclipse-based projects created with STM32CubeIDE 1.x. If you are using STM32CubeIDE 2.x together with STM32CubeMX 2, the project is CMake-based and configured as described in section 5 Using STM32CubeIDE 2.x and STM32CubeMX 2 (CMake Projects) instead.

In this guide the folder unpacked from the SDK zip is called “cortex_mXX_gcc”. Replace it with the name of the SDK folder for your target, for example “cortex_m4_gcc” or “cortex_m33_gcc”.

4.1 Adding Acconeer Software

There are different folders in the SDK zip:

- algorithms
- doc
- examples
- exploration_server
- integration
- rss
- use_cases

In the “doc”-folder you can find reference documentation of the Acconeer software. The “examples”-folder contains examples of how to use the RSS API. The “use_cases”-folder contains example and reference apps for specific use cases. The “rss”-folder contains two subfolders called “include” and “lib”. The “lib”-folder contains the Radar System Software (RSS) and the “include”-folder contains the header-files needed to use RSS. The “integration”-folder contains files which connect RSS with the drivers generated by STM32CubeIDE. The “exploration_server”-folder contains the files needed to build the exploration server application.

Start off by unpacking the zip-file into your project. Refresh the project by right clicking the project in the Project Explorer and click “Refresh”. Make sure you can see the folder in your project before continuing.

4.1.1 Source-files

Now the integration-file which you want to use needs to be selected from the “cortex_m4_gcc/integration”-folder,

Copy/move the “acc_hal_integration_stm32cube_xe121_multi_sensor.c”-file you have selected into “Core/Src”-folder. Also, copy content from “cortex_mXX_gcc/algorithms” to “Core/Src”-folder. If you want any example or reference apps, copy them from “cortex_mXX_gcc/examples” or from “cortex_mXX_gcc/use_cases” to “Core/Src”-folder. Some apps are split in two files, for example “example_cargo.c” and “example_cargo_main.c”, copy both. Apps that include “acc_control_helper.h” or “acc_processing_helpers.h” also need “acc_control_helper.c” and “acc_processing_helpers.c” from the “cortex_mXX_gcc/examples/helper”-folder.

Also move the “acc_integration_stm32.c“ and the “acc_integration_log.c“ files from the “cortex_mXX_gcc/integration”-folder to the “Core/Src”-folder.

For building the exploration server application you need to move the file “acc_exploration_server_stm32.c“ from the “cortex_mXX_gcc/exploration_server”-folder to the “Core/Src”-folder.

For this guide, we select “example_service.c” and move it to the “Core/Src”-folder.

4.1.2 Header-files

1. Select your project in the “Project Explorer”
2. Go into “Project → Properties → C/C++ General → Paths and Symbols → Includes”
3. Press “Add...” and then “Workspace...”
4. Select the “cortex_mXX_gcc/rss/include”-folder in your project

Repeat this procedure for the “cortex_mXX_gcc/integration”-folder and the “cortex_mXX_gcc/exploration_server”-folder. The headers of the apps are in subfolders of “cortex_mXX_gcc/examples” and “cortex_mXX_gcc/use_cases”, add the subfolder of each app you use, for example “cortex_mXX_gcc/examples/getting_started” for “example_service.h” and “cortex_mXX_gcc/examples/helper” for the helpers.

4.1.3 Libraries

In order to set the path for the libraries, do the following:

© 2026 by Acconeer AB - All rights reserved

1. Select your project in the “Project Explorer”
2. Go into “Project → Properties → C/C++ General → Paths and Symbols → Library Paths”
3. Press “Add...” and then “Workspace...”
4. Select the “cortex_mXX_gcc/rss/lib”-folder in your project

Once the path is set, you can add the specific libraries as follows:

1. Go into “Project → Properties → C/C++ General → Paths and Symbols → Libraries”
2. Click “Add...”
3. Write “acconeer_a121”
4. Click “OK”

If you want to add the “acc_detector_distance_a121” or “acc_detector_presence_a121” library, simply repeat the procedure above and exchange “acconeer_a121” for “acc_detector_distance_a121” or “acc_detector_presence_a121”. Make sure that the detector is being added before the “acconeer_a121”-library by moving “acconeer_a121” down using the “Move Down” button when “acconeer_a121” is selected. If you want to build the exploration server application, you also have to add the “acconeer_exploration_server_a121” library by repeating the procedure above and exchanging “acconeer_a121” for “acconeer_exploration_server_a121”. This library also needs to be added before the “acconeer_a121”-library.

4.2 Project Settings

Set the project to gnu99-compiler by going into “Project → Properties → C/C++ Build → Settings → Tool Settings → MCU GCC Compiler → General”.

If you want to build the exploration server application, you also have to add the linker flag “-u `_printf_float`” in order for the exploration server to correctly format floats in the JSON result.

1. Go to “Project → Properties → C/C++ Build → Settings → Tool Settings → MCU GCC Linker → Miscellaneous → Other flags”.
2. Add “-u `_printf_float`”

4.3 Adding Print Functionality with UART/USART

If a UART/USART has been added when setting up the project in the STM32CubeMX perspective, and you want to use it for prints, then you can simply add the following code to your project between the comments “Private user code USER CODE BEGIN 0” and “USER CODE END 0” in the file “Core/Src/main.c”:

```
int _write(int file, char *ptr, int len)
{
  (void)file;
  HAL_UART_Transmit(&huart2, (uint8_t *)ptr, len, 0xFFFF);
  return len;
}
```

A terminal emulator, for example PuTTY, can be used to view the UART prints from the board.

Download PuTTY from https://putty.org and install it on your computer.

4.3.1 Find STM32 Board COM port

If using Windows, go to Device Manager to locate the COM port used by your STM32 Board.

© 2026 by Acconeer AB - All rights reserved

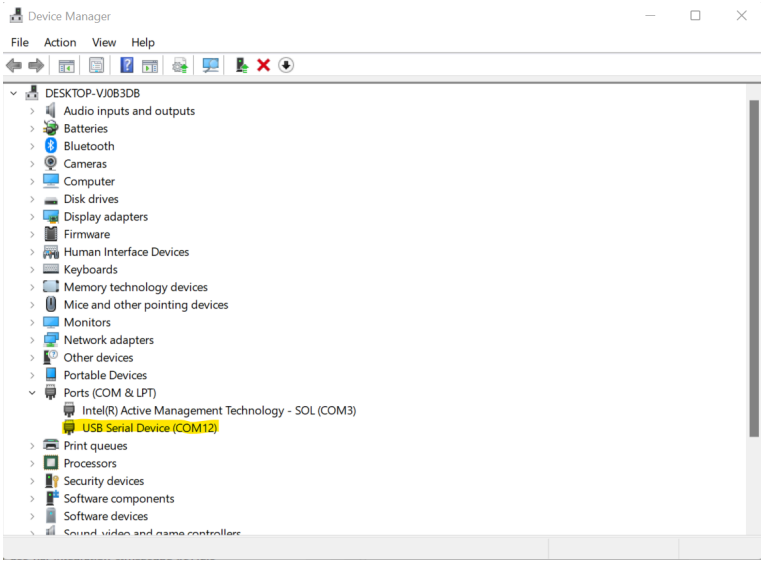


Figure 13: Windows Device Manager

4.3.2 Start and Configure PuTTY

1. Start the PuTTY application
2. Select **Session** in the **Category** window.
3. Set the **Connection Type** to **Serial**
4. Type the STM32 Board COM port in the **Serial line** text box.
5. Use the same **Speed** as used for the UART in the STM32CubeIDE project, default 115200.
6. Select **Terminal** in the **Category** window.
7. Tick the **Implicit CR in every LF** tickbox
8. Click on **Open** to start the terminal

© 2026 by Acconeer AB - All rights reserved

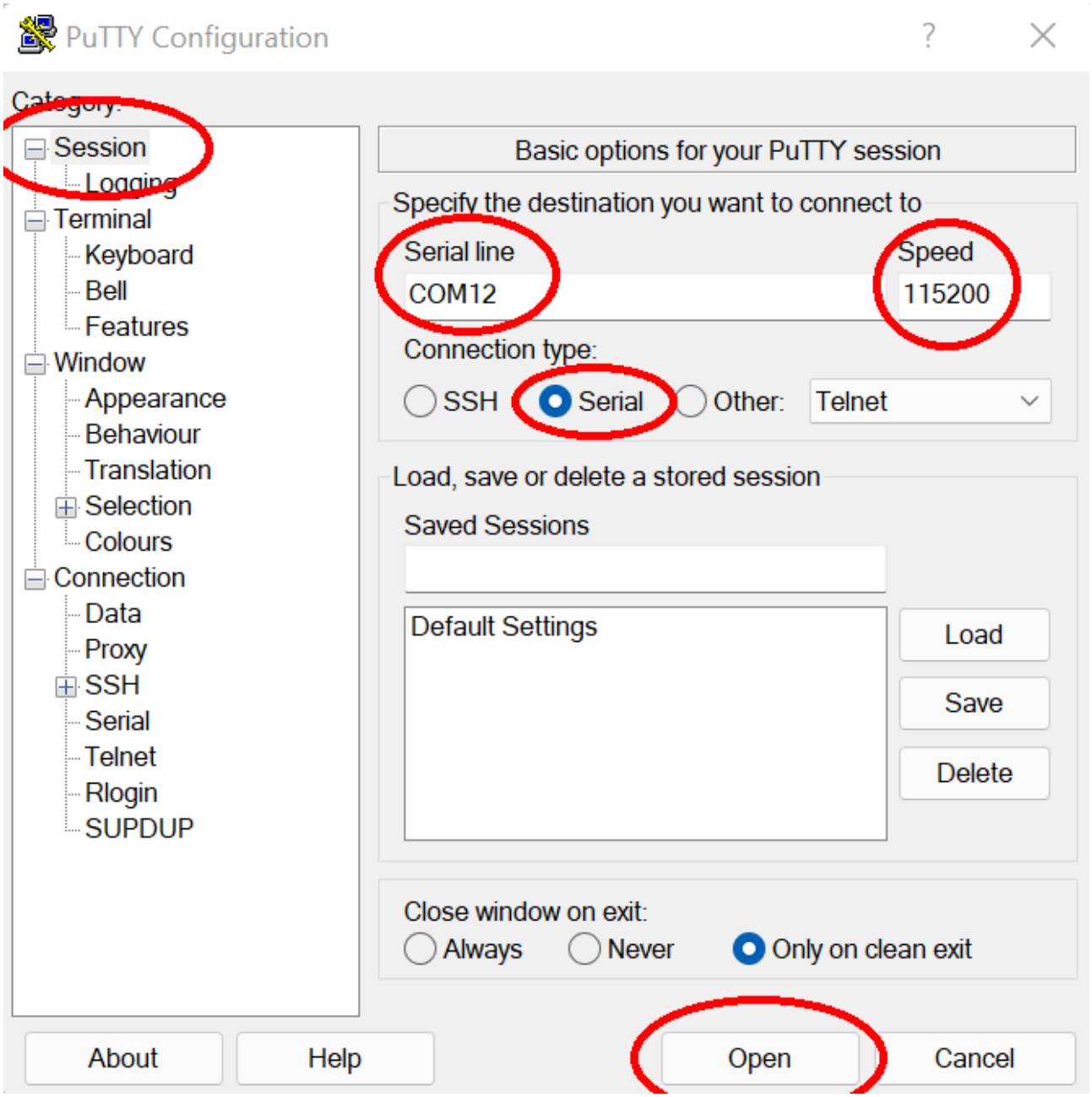


Figure 14: PuTTY Session

© 2026 by Acconeer AB - All rights reserved

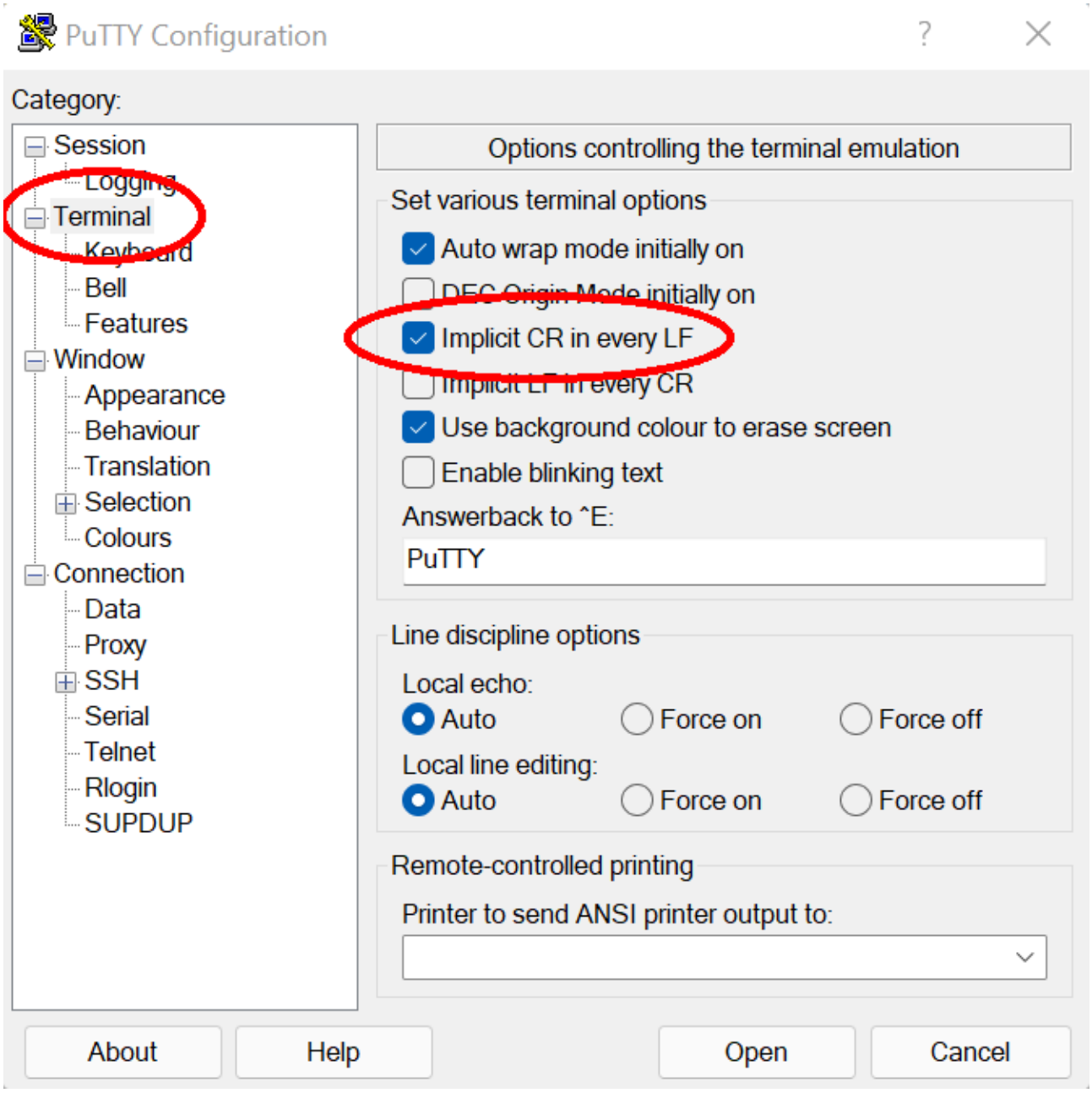


Figure 15: PuTTY Configuration

© 2026 by Acconeer AB - All rights reserved

5 Using STM32CubeIDE 2.x and STM32CubeMX 2 (CMake Projects)

Starting with STM32CubeIDE 2.0, projects are generated by the STM32CubeMX 2 tool as CMake projects instead of native Eclipse projects. This changes how the Acconeer software is added to the project. The IDE dialogs used in section 4 Configuring Project for Acconeer Software, such as “Paths and Symbols”, “Libraries” and “Tool Settings”, are no longer available. The corresponding settings are instead made in the CMake files of the project.

This section describes the differences when using the CMake-based tools. The configuration itself is the same: the pin assignments, user labels, SPI settings, interrupt settings and GPIO settings described in section 3 Getting Started with STM32CubeIDE apply also when the project is created with STM32CubeMX 2. How to enter them in STM32CubeMX 2 differs from the STM32CubeMX perspective shown in this guide and is not covered here; please refer to the ST documentation for the tool version you are using.

Please note that the exact layout of the generated project can differ between tool versions. Consult the ST documentation for the version you are using if the project does not match the description below.

5.1 Creating the Project

Create the project with STM32CubeMX 2 and configure the pins, interrupts, GPIOs and SPI according to section 3 Getting Started with STM32CubeIDE. In “Project Manager → IDE Project”, select CMake as format and GCC as CMake toolchain, then generate the project.

Import the generated project into STM32CubeIDE 2.x with “File → New → STM32 Project → STM32 Project Create/Import” and select “STM32CubeMX2 project”. On the “Project with existing CMake sources” page, enter a project name and browse to the generated folder, then select the GCC toolchain and the same MCU as in the STM32CubeMX 2 project and press “Finish”.

Please note that importing the project with “Open Projects from File System” does not work. The project is then imported without CMake support, no CMake configuration is run, and the “Build Targets” view stays empty.

The generated project separates tool-owned files from user-owned files:

- Tool-owned files, for example the generated initialization code and the CMake files that list the generated source files, are overwritten every time the project is regenerated and must not be modified.
- User-owned files, for example the top-level “CMakeLists.txt” and the source files containing “USER CODE” sections, are kept when the project is regenerated. All modifications described in this section are made in these files.

5.2 Adding Acconeer Software to the CMake Build

Unpack the SDK zip into the root of your project so that the SDK folder, in this guide called “cortex_mXX_gcc”, is located next to the top-level “CMakeLists.txt”.

Unlike in the Eclipse-based projects, the source files do not need to be copied into the project source folder. Instead, every source file, include path and library is listed explicitly in the top-level “CMakeLists.txt”. Open the top-level “CMakeLists.txt” and locate the name of the application target, i.e. the name used in the “add_executable” command. In the projects generated by STM32CubeMX 2 the application target is typically available as `${CMAKE_PROJECT_NAME}`, which is used in the example below.

The generated top-level “CMakeLists.txt” already contains empty stub sections (“target_sources”, “target_include_directories” and so on, marked with “Add additional … here” comments) intended for user additions. Fill them in as follows:

```
# Path to the unpacked Acconeer SDK
set(ACCONEER_SDK_DIR ${CMAKE_CURRENT_SOURCE_DIR}/cortex_mXX_gcc)

# Acconeer source files
target_sources(${CMAKE_PROJECT_NAME} PRIVATE
    ${ACCONEER_SDK_DIR}/integration/acc_hal_integration_stm32cube_xe121_single_sensor.c
    ${ACCONEER_SDK_DIR}/integration/acc_integration_stm32.c
    ${ACCONEER_SDK_DIR}/integration/acc_integration_log.c
    ${ACCONEER_SDK_DIR}/algorithms/acc_algorithm.c
    ${ACCONEER_SDK_DIR}/examples/getting_started/example_bring_up.c
)

# Acconeer include paths
```

© 2026 by Acconeer AB - All rights reserved

```
target_include_directories(${CMAKE_PROJECT_NAME} PRIVATE
    ${ACCONEER_SDK_DIR}/rss/include
    ${ACCONEER_SDK_DIR}/integration
    ${ACCONEER_SDK_DIR}/algorithms
    ${ACCONEER_SDK_DIR}/examples/getting_started
)

# Acconeer library path
target_link_directories(${CMAKE_PROJECT_NAME} PRIVATE
    ${ACCONEER_SDK_DIR}/rss/lib
)

# Acconeer libraries
target_link_libraries(${CMAKE_PROJECT_NAME}
    acconeer_a121
)
```

Use "acc_hal_integration_stm32cube_xe121_multi_sensor.c" instead of "acc_hal_integration_stm32cube_xe121_single_sensor.c" for a multi-sensor setup, see section 6 HAL Integration File.

If you want to use other examples, reference apps or example apps, add their source files to "target_sources" and their folders to "target_include_directories" in the same way.

5.2.1 Libraries and Link Order

The same library link order as described in section 4.1.3 Libraries applies. In CMake the link order is given by the order in "target_link_libraries". The "acc_detector_distance_a121", "acc_detector_presence_a121" and "acconeer_exploration_server_a121" libraries must be listed before the "acconeer_a121" library:

```
target_link_libraries(${CMAKE_PROJECT_NAME}
    acc_detector_distance_a121
    acc_detector_presence_a121
    acconeer_a121
)
```

Please note that "target_link_libraries" must be called with the plain signature (without the "PUBLIC" keyword) as shown above. The generated "cmake/components.cmake" file uses the plain signature, and CMake does not allow mixing the plain and keyword signatures for the same target, even though the generated stub in the top-level "CMakeLists.txt" suggests the "PUBLIC" keyword.

5.3 Project Settings

The Acconeer software requires the C99 standard or later. The projects generated by the CMake-based tools already use a newer C standard by default, so the "Set the project to gnu99-compiler" step from section 4 Configuring Project for Acconeer Software is normally not needed. If your project explicitly selects an older standard, set it to at least gnu99 in the top-level "CMakeLists.txt":

```
set_target_properties(${CMAKE_PROJECT_NAME} PROPERTIES
    C_STANDARD 99
    C_EXTENSIONS ON
)
```

If you want to build the exploration server application you also have to add the linker flag "-u _printf_float" in order for the exploration server to correctly format floats in the json result:

```
target_link_options(${CMAKE_PROJECT_NAME} PRIVATE
    -u _printf_float
)
```

The Acconeer libraries are built without Link Time Optimization (LTO). Do not enable LTO (the "-flto" compiler flag) in the project.

© 2026 by Acconeer AB - All rights reserved

The compiler flags of the project must match the flags that were used when building the Acconeer libraries, listed in the "BUILDINFO.txt" file in the "doc"-folder of the SDK. In particular the CPU and float ABI flags, for example "-mcpu=cortex-m33 -mfloat-abi=hard -mfpu=fpv5-sp-d16", must match. The compiler flags of the generated project are found in the CMake files in the "cmake"-folder of the project.

5.4 Heap and Stack Configuration

The linker script generated by STM32CubeMX 2, found under `user_modifiable/Device/<mcu>/` in the project, is user-owned and must be adjusted before running the Acconeer software. The generated defaults are too small:

```
HEAP_SIZE = 0x200;
STACK_SIZE = 0x400;
```

Unlike the classic STM32CubeIDE linker scripts, where the stack is placed at the end of RAM and the heap can grow into all unused memory, this linker script reserves a fixed-size heap region and places the stack directly after it. With the default values the heap is limited to 512 bytes and the remaining RAM is unused. The Acconeer software allocates its working memory from the heap and requires considerably more.

Typical symptoms of the default values are the log message "Memory allocation failed" or a hard fault as soon as the sensor is used. Increase the values in the linker script, for example for a device with 128 kB of RAM:

```
HEAP_SIZE = 0x1C000;
STACK_SIZE = 0x1400;
```

As a guideline, use at least 0x1400 (5 kB) of stack and give the heap as much of the remaining RAM as possible. The linker reports an error if the total exceeds the available RAM. Please note that the build system does not track changes to the linker script, so force a re-link (for example by deleting the ".elf" file or rebuilding from scratch) after editing it.

5.5 Adding Print Functionality

The "_write" retarget described in section 4.3 Adding Print Functionality with UART/USART works differently in the projects generated by STM32CubeMX 2, since the UART driver functions have new signatures and the UART handle is accessed through a generated get function instead of a global variable. With a UART enabled in STM32CubeMX 2, for example USART2 which is connected to the ST-LINK virtual COM port on the Nucleo boards, add the following to "main.c":

```
#include "mx_usart2.h"

int _write(int file, char *ptr, int len)
{
  (void)file;
  HAL_UART_Transmit(mx_usart2_uart_gethandle(), ptr, len, 0xFFFF);
  return len;
}
```

5.6 Devices Using the New-Generation STM32 HAL

Some newer STM32 series, for example the STM32C5, use ST's new-generation HAL drivers with a changed API. The HAL integration files delivered in the SDK target the classic HAL and do not compile against the new-generation HAL without adaptation. The function names are largely unchanged, but types and constants differ. The most important changes for the integration files are:

Classic HAL	New-generation HAL
`SPI_HandleTypeDef`	`hal_spi_handle_t`
`HAL_StatusTypeDef`	`hal_status_t`
`GPIO_TypeDef *`	`hal_gpio_t`
`GPIO_PIN_SET` / `GPIO_PIN_RESET`	`HAL_GPIO_PIN_SET` / `HAL_GPIO_PIN_RESET`
`<label>_GPIO_Port` / `<label>_Pin`	`<label>_PORT` / `<label>_PIN`
`extern SPI_HandleTypeDef hspi1`	`mx_spi1_gethandle()`

© 2026 by Acconeer AB - All rights reserved

The SPI transfer functions take `const void *` / `void *` data pointers and a 32-bit element count, and "HAL_SPI_TransmitReceive_DMA" is only available when the DMA channels have been configured for the SPI in STM32CubeMX 2. "HAL_Delay", "HAL_GetTick", "HAL_GPIO_WritePin" and "HAL_GPIO_ReadPin" keep their names, so "acc_integration_stm32.c" and "acc_integration_log.c" compile unchanged; only the `acc_hal_integration_stm32cube_xe121_<single/multi>_sensor.c` file needs to be adapted. Note that the generated code no longer provides an "Error_Handler()" function, so the adapted integration file needs to provide its own error handler.

5.7 User Code and Running the Examples

The generated source files still contain "USER CODE" sections, and user code is added in the same way as in the Eclipse-based projects:

- Define "A121_SPI_HANDLE" in "main.h" as described in section 6 HAL Integration File.
- Include "example_bring_up.h" and call "acc_example_bring_up()" from "main.c", in the same way as shown for the radar example in section 7 Running a Radar Sensor Example.

The bring-up example verifies the HAL integration step by step (SPI communication, enable pin, sensor interrupt and sensor calibration) and is the recommended starting point. Once all bring-up tests pass, continue with "example_service.c" as described in section 7 Running a Radar Sensor Example.

Build the project using the build presets generated by STM32CubeMX 2, for example the "debug" preset, either from within STM32CubeIDE or from the command line with `cmake --build`. Flashing and debugging with the ST-LINK debugger works in the same way as described in section 7 Running a Radar Sensor Example.

© 2026 by Acconeer AB - All rights reserved

6 HAL Integration File

6.1 Selecting the Appropriate HAL-integration File

First off, you need to pick which HAL integration file to use. The functions in the HAL integration file act as glue between the RSS radar stack and the drivers generated by the device configuration tool (STM32CubeMX perspective). Your hardware setup determines which HAL integration file to select as a starting point.

The files "`acc_hal_integration_stm32cube_xe121_<single/multi>_sensor.c`" are HAL integrations prepared by Acconeer to handle the XE121 EVK board together with an STM32 processor.

The file "`acc_hal_integration_stm32cube_xe121_single_sensor.c`" will only handle the onboard sensor on the XE121. The file "`acc_hal_integration_stm32cube_xe121_multi_sensor.c`" will handle the onboard sensor on the XE121 as well as the possibility to use XS121 sensors connected in slots S2, S3, S4 and S5.

6.2 A121_SPI_HANDLE

Define "A121_SPI_HANDLE" as hspi1 between the comments "USER CODE BEGIN Private defines" and "USER CODE END Private defines" in the file "Core/Inc/main.h".

```
/* USER CODE BEGIN Private defines */
#define A121_SPI_HANDLE hspi1
/* USER CODE END Private defines */
```

© 2026 by Acconeer AB - All rights reserved

7 Running a Radar Sensor Example

As a first radar example program to run, we selected the program “`example_service.c`”.

To run the radar example program that we have chosen, simply include the header file “example_service.h” in the user code includes field in your “main.c”-file in the following manner:

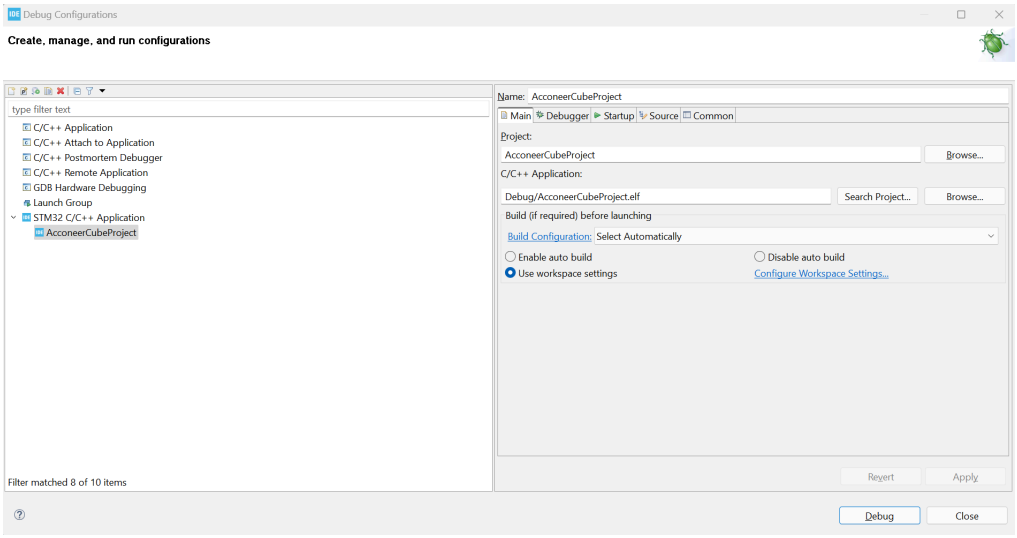
```
/* USER CODE BEGIN Includes */
#include "example_service.h"
/* USER CODE END Includes */
```

After including the header-file, you can call the function from the “main.c”-file in the user code 2 field by the following call:

```
/* USER CODE BEGIN 2 */
acc_example_service(0, NULL);
/* USER CODE END 2 */
```

7.1 Running the Program

Build the software by pressing “Ctrl-B” and then start debugging by right-clicking on the project “AcconeerCube-Project → Debug As → STM32 Cortex-M C/C++ Application”. This will open the “Debug Configurations” dialog and there you can choose which debugger to use, “Debugger → Debug Probe”, and select “ST-LINK”. Click “Debug”. This will automatically flash the board and execute the program until the “main()” function.



© 2026 by Acconeer AB - All rights reserved

8 Running the Exploration Server

To be able to run the Exploration Server application on your STM32 Board you need to assign a UART with Flow Control. For Nucleo-L476RG we will use USART3.

8.1 STM32CubeMX

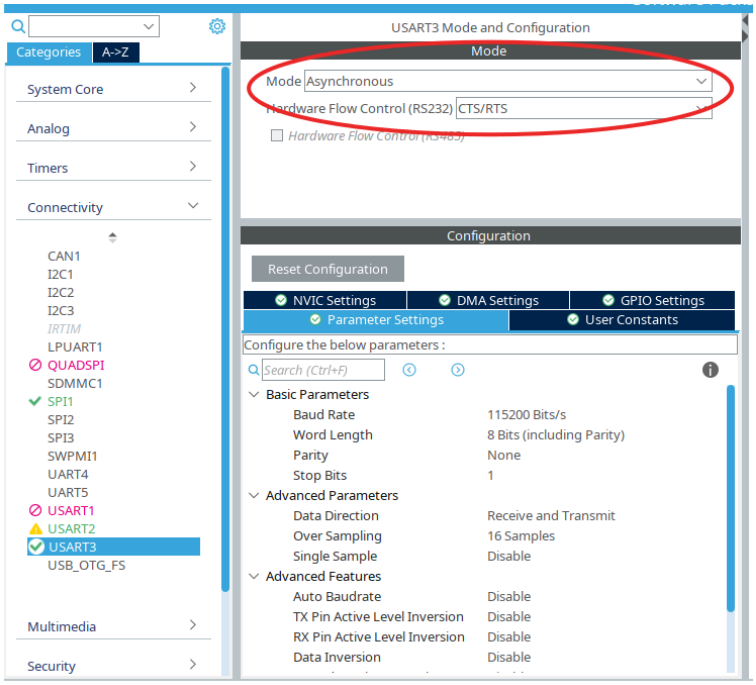


Figure 17: Enable USART3

Make sure that the USART3 signals use the pins in the table below. If STM32CubeMX has placed a signal on another pin, left click the pin in the table and select the signal. The exploration server waits for the host by reading the RX pin through the user label “UART_RX”, so right click PC5, select “Enter User Label” and enter “UART_RX”.

USER_LABEL	NUCLEO PIN	GPIO TYPE	CN10 PIN
	PC4	USART3_TX	34
UART_RX	PC5	USART3_RX	6
	PB13	USART3_CTS	30
	PB1	USART3_RTS	24

You also need to use DMA for both RX and TX as well as enable global interrupts for USART3:

© 2026 by Acconeer AB - All rights reserved

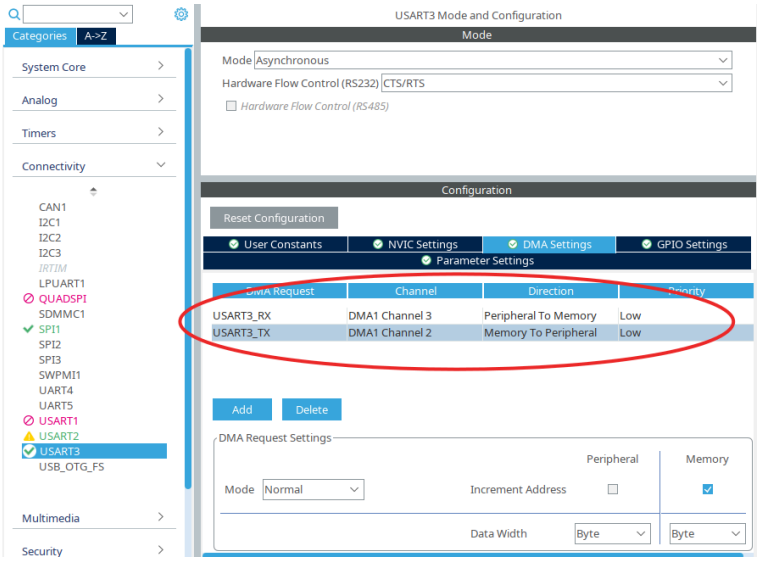


Figure 18: Enable DMA for USART3

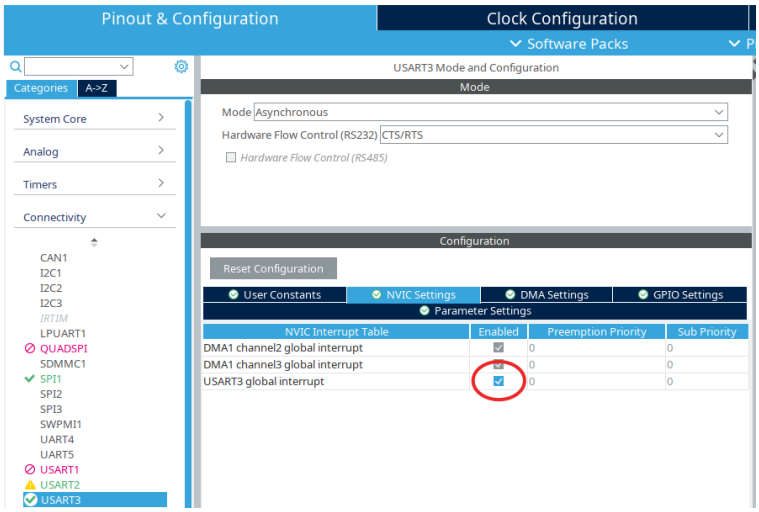


Figure 19: Enable Global Interrupts for USART3

You need to generate the code again after doing these changes.

8.2 STM32CubeIDE

Define “EXPLORATION_SERVER_UART_HANDLE” as huart3 between the comments “USER CODE BEGIN Private defines” and “USER CODE END Private defines” in the file “Core/Inc/main.h”.

```
/* USER CODE BEGIN Private defines */
#define EXPLORATION_SERVER_UART_HANDLE huart3
/* USER CODE END Private defines */
```

The file “acc_exploration_server_stm32.c” finds the RX pin through the “UART_RX” user label, so it needs no changes.

To run the Exploration Server application, simply declare the main function in the user code includes field in your “main.c”-file in the following manner:

```
/* USER CODE BEGIN Includes */
extern int acc_exploration_server_stm32(int argc, char *argv[]);
/* USER CODE END Includes */
```

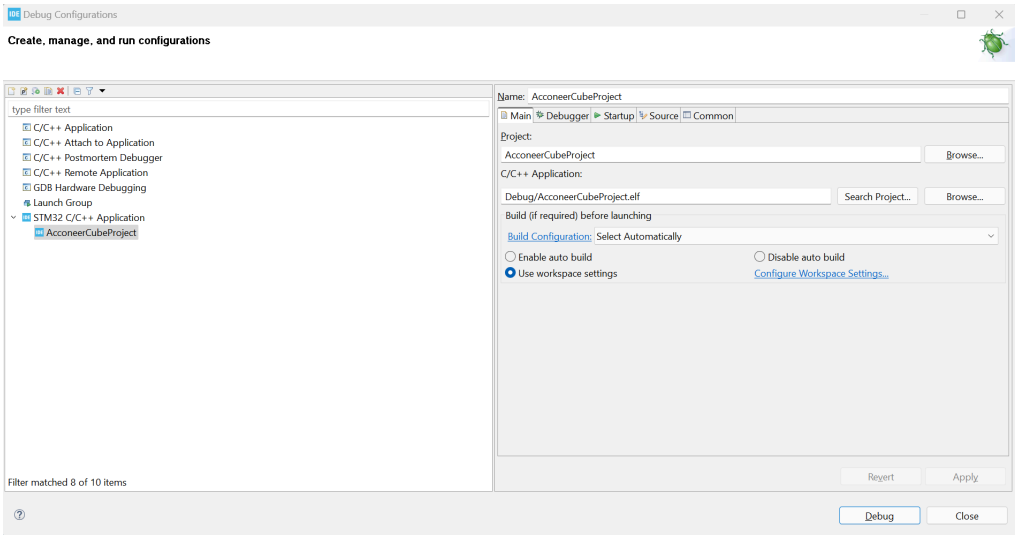
After declaring the main function, you can call the function from the “main.c”-file in the user code 2 field by the following call:

© 2026 by Acconeer AB - All rights reserved

```
/* USER CODE BEGIN 2 */
acc_exploration_server_stm32(0, NULL);
/* USER CODE END 2 */
```

8.2.1 Running the Program

Build the software by pressing “Ctrl-B” and then start debugging by right-clicking on the project “AcconeerCube-Project → Debug As → STM32 Cortex-M C/C++ Application”. This will open the “Debug Configurations” dialog and there you can choose which debugger to use, “Debugger → Debug Probe”, and select “ST-LINK”. Click “Debug”. This will automatically flash the board and execute the program until the “main()” function.



Connect the host through a USB to serial adapter with RTS/CTS and 3.3 V logic levels to the USART3 pins on the connector CN10 of the Nucleo-L476RG board, see the table in section 8.1 STM32CubeMX. Cross the signals: the adapter’s TX goes to PC5, RX to PC4, RTS to PB13 and CTS to PB1. Connect GND to CN10 pin 20. If the adapter takes its I/O voltage from the target, as for example the FTDI TTL-232RG-VIP cable does, connect that pin to +3.3V on CN7 pin 5. Otherwise leave the adapter’s supply pin unconnected.

Until the adapter drives the RX pin, the exploration server prints “Waiting for host line break to end...” on the UART used for prints. Then connect the Exploration Tool to the serial port of the adapter.

© 2026 by Acconeer AB - All rights reserved

9 Troubleshooting and FAQ

9.1 Example Fails

The example program can fail for different reasons. Here are a few common ones.

9.1.1 Sensor Creation Returns NULL

The function `acc_sensor_create` returns NULL. This is most likely because the pins have been connected incorrectly, or because of some other pin fault, and it is usually due to an SPI communication problem. It could be caused by incorrectly connected pins, incorrect drivers, or signals sent to the radar in the wrong order. See section 9.3 Troubleshooting SPI Communication for more information.

9.1.2 Config Creation Hardfaults

The function `acc_config_create` hardfaults. Most likely due to memory problems. Depending on the memory of the MCU, heap and stack might overwrite each other. Or there is simply not enough memory.

In CMake-based projects generated by STM32CubeMX 2, a “Memory allocation failed” log message or a hard fault as soon as the sensor is used is usually caused by the default heap size in the generated linker script, see section 5.4 Heap and Stack Configuration.

Are you using FREERTOS? Make sure that the thread that is handling the Acconeer software has enough stack size to be able to run the software.

9.2 The Program is Stuck in HAL_Delay

If the program keeps entering HAL_Delay() or seems to be “stuck” there for longer periods of time, it might be because the interrupt pin is not connected or malfunctioning.

9.3 Troubleshooting SPI Communication

The following function can be used to find problems in the SPI communication with the radar sensor.

```
#include "acc_hal_integration_a121.h"

bool hal_test_spi_read_chipid(void)
{
  const uint32_t sensor = 1;

  const acc_hal_a121_t *hal = acc_hal_rss_integration_get_implementation();

  uint8_t buffer[6] = {0x30, 0x0, 0x0, 0x0, 0x0, 0x0 };

  acc_hal_integration_sensor_supply_on(sensor);
  acc_hal_integration_sensor_enable(sensor);

  hal->transfer(sensor, buffer, sizeof(buffer));

  acc_hal_integration_sensor_disable(sensor);
  acc_hal_integration_sensor_supply_off(sensor);

  if (buffer[4] == 0x12 && buffer[5] == 0x10)
  {
    printf("Test OK !\n");
    return true;
  }

  printf("Cannot read chip id !\n");
  return false;
}
```

When the program is executed, the signals to the A121 should look as in Figure 21.

© 2026 by Acconeer AB - All rights reserved

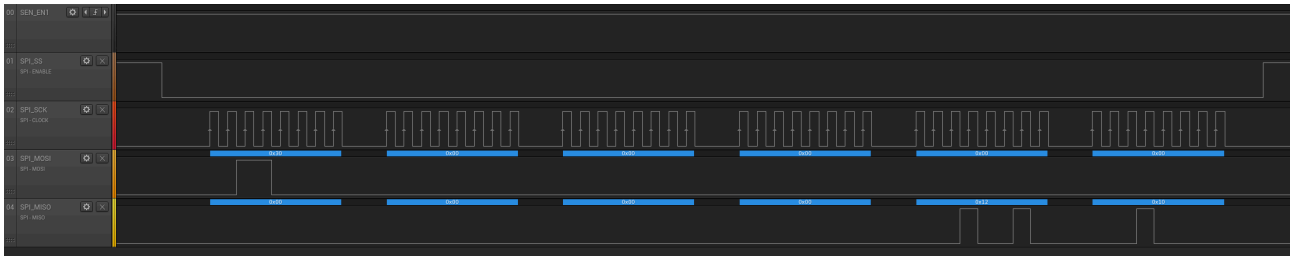


Figure 21: SPI transfer example

Note that the A121 enable signal must be set high at least 2 ms before the SPI transfer.



Figure 22: A121 Enable Signal

9.4 UART Problems

In order to verify the prints over UART, we use picocom in Ubuntu:

```
$ picocom --imap lfcrlf --baud 115200 /dev/ttyACM0
```

We also had to make sure, in main.c, that the baudrate and word length is correct:

```
huart2.Init.BaudRate = 115200;
huart2.Init.WordLength = UART_WORDLENGTH_8B;
```

The linker might tell you that you have multiple definitions of the function "`_write`". If it happens, remove the implementation in "syscalls.c" and compile/link again.

9.5 Link Errors

Some users have experienced that STM32CubeIDE forgets the link order of the libraries. Please check that the RSS libraries are listed in order stated in section 4.1.3 Libraries.

In CMake-based projects, an "undefined reference" error for a function that is defined in one of the SDK source files, for example "acc_example_service", means that the source file has not been added to the CMake build. Make sure every used SDK source file is listed in "target_sources" as described in section 5.2 Adding Acconeer Software to the CMake Build.

9.6 Heap Memory Corruption

When asking for more heap, the sbrk function will increase the heap towards the stack. However, it is imperative that the heap and stack never meet or overwrite each other. Unfortunately, the automatically generated code is using the stack pointer as border between them when it should use the maximum needed stack. This means that the heap might use some memory which is later overwritten when the stack is growing.

In the file "Src/sysmem.c" remove the line:

```
register char * stack_ptr asm("sp");
```

After removing the above line, change the function called "`_sbrk`" so that it looks like this:

```
caddr_t _sbrk(int incr)
{
  extern char end asm("end");
```

© 2026 by Acconeer AB - All rights reserved

```
  extern char estack asm("_estack");
  extern char min_stack_size asm("_Min_Stack_Size");
  char *stack_limit = (char*)(&estack - &min_stack_size);
  static char *heap_end;
  char *prev_heap_end;

  if (heap_end == 0)
    heap_end = &end;

  prev_heap_end = heap_end;
  if (heap_end + incr > stack_limit)
  {
    errno = ENOMEM;
    return (caddr_t) -1;
  }

  heap_end += incr;

  return (caddr_t) prev_heap_end;
}
```

9.7 Floating point equality checking errors

If you get run time errors when running `example_bring_up`, `example_diagnostics` or during calibration, often related to floating point equality checking, such as the following:

```
(assembly_test_a121) Basic read: Test
Bring-up: 'Basic read test' [PASS]
(assembly_test_a121) Communication test: Test
(assembly_test_a121) Communication test: Effective bitrate = 1103kbps
Bring-up: 'Communication test' [PASS]
(assembly_test_a121) Enable pin test: Setup
(assembly_test_a121) Enable pin test: Test
Bring-up: 'Enable pin test' [PASS]
(assembly_test_a121) Interrupt test: Reset
(assembly_test_a121) Interrupt test: Setup
(cpd_pll_divisors) Invalid reference frequency 24000000.000000 Hz
(cpd_pll_divisors) Invalid reference frequency 24000000.000000 Hz
(assembly_test_a121) Test interrupt setup failed
Bring-up: 'Interrupt test' [FAIL]
Bring-up: Interrupt test failed
```

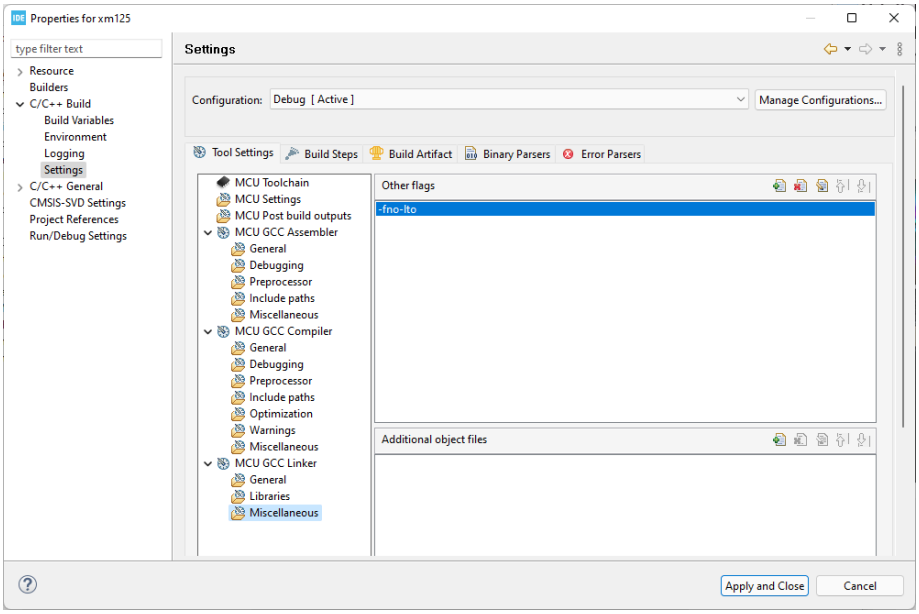
there might be a mismatch between the compiler used to produce the libraries in the SDK from Acconeer and the compiler used to build the project in STM32CubeIDE. Make sure that both are using the same toolchain. The toolchain used to build the Acconeer libraries can be found in the file VERSION in the SDK.

If you're using STM32CubeIDE on Windows, there is a known problem with the LTO wrapper in that version of the GCC compiler used.

To explicitly disable LTO (link-time optimizations):

1. Go to "Project → Properties → C/C++ Build → Settings → Tool Settings → MCU GCC Linker → Miscellaneous → Other flags".
2. Add "-fno-lto"

© 2026 by Acconeer AB - All rights reserved



If you get an error that the program will not fit in flash you need to change the optimization level for size:

1. Go to “Project → Properties → C/C++ Build → Settings → Tool Settings → MCU/MPU GCC Compiler → Optimizations → Optimization level”.
2. Change to “Optimize for size (-Os)”

© 2026 by Acconeer AB - All rights reserved

10 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