A121 ESP32

User Guide

A121 ESP32

User Guide

Author: Acconeer AB

Version:a121-v1.13.0

Acconeer AB March 26, 2026

Contents

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)	

README	Various target specific information and links	- After SDK download

2 Introduction

This document will shortly cover how to get started with the different ESP32 SDKs provided by Acconeer. Acconeer currently provides SDKs for three different types of ESP32s:

- ESP32 (LX6 processor), tested with an XE121 + ESP32_DevKitc_V4.
- ESP32C3 (RISC-V processor), tested with an XE121 + ESP32-C3-DevKitM-1 v1.0.
- ESP32S3 (Dual LX7 processor), tested with an XE121 + Ardi-32 from SB-Components.

The SDKs are developed for and tested on ESP-IDF version: v5.4.1

The SDKs have been tested with ESP-IDF extension in VS Code on Windows and using command line on Ubuntu 24.04.

3 Downloading the Acconeer SDK

In order to run Acconeer code with your chosen ESP32 hardware, you will need to download the SDK from the Acconeer developer site. Please choose the SDK corresponding to your ESP32 hardware at: https://developer.acconeer.com/home/a121-docs-software/

Once the zip-file is downloaded, make sure to unzip it at a location with a path with few characters used since the ESP-IDF environment has restrictions on the number of characters in a path.

In our case we use /home/user/acc-esp/ on Ubuntu and “c:\acc-esp\” on Windows.

4 Acconeer ESP32 SDK structure

This chapter shortly covers the SDK structure in order for the user to get a better understanding on how to later modify it.

```
├── doc
├── include
├── lib
├── LICENSES
├── make.sh
├── samples
├── source
└── VERSION
```

The “doc”-folder contains useful documentation for the user to read. The “source”, “include” and “lib” folder contains Acconeer C code, header files and libraries. “make.sh”, is a short shell script which simply cleans and builds all sample projects in the “samples”-folder.

The samples folder in the SDK contains several example projects which can be used as a basis for your own project.

```
samples
├── example_bring_up
├── example_control_helper
├── example_detector_distance
├── example_detector_presence
└── example_service
```

Each example folder in the samples folder is an ESP-IDF project and uses Acconeer source code, header files and libraries from the “source”, “include” and “lib”-folders in the SDK.

Each sample folder contains the following:

```
example_bring_up
├── build
├── CMakeLists.txt
├── main
│   └── CMakeLists.txt
├── sdkconfig
└── sdkconfig.defaults
```

- CMakeLists.txt - essentially contains the project name.
- build folder - contains the output from building the sample project using the command “idf.py build”.
- main folder - contains a CMakeLists.txt specifing what source files, libraries and include path to use when building the project.
- sdkconfig - contains the ESP-IDF SDK configuration generated when building.
- sdkconfig.defaults - contains modifications to default configurations.

5 Example Board Connection

Each ESP32 SDK has a different example integration. The following section will cover the pin connections between specific boards and XE121 for each SDK.

In all cases, since XE121 is used, it is important to also connect 5V, 3.3V and as many GND as possible

5.1 Pin connection for the ESP32 SDK

Hardware used: XE121 + ESP32_DevKitc_V4

This example does not use dedicated IO_MUX for SPI because strapping pins are connected to the dedicated SPI pins.

XE121 pin	ESP32 pin	Function	Commment
EN1	23	GPIO Output	
EN2	N/A	N/A	Only needed for multi-sensor, not used by default
EN3	N/A	N/A	Only needed for multi-sensor, not used by default
EN4	N/A	N/A	Only needed for multi-sensor, not used by default
EN5	N/A	N/A	Only needed for multi-sensor, not used by default
MOSI	3	SPI2	
MISO	21	SPI2	
SCK	2	SPI2	
CS	19	SPI2	
SEL0	18	GPIO Output	
SEL1	5	GPIO Output	
SEL2	17	GPIO Output	
INT1	4	External Interrupt	GPIO input + GPIO_INTR_POSEDGE
INT2	N/A	N/A	Only needed for multi-sensor, not used by default
INT3	N/A	N/A	Only needed for multi-sensor, not used by default
INT4	N/A	N/A	Only needed for multi-sensor, not used by default
INT5	N/A	N/A	Only needed for multi-sensor, not used by default

5.2 Pin connection ESP32C3 SDK

Hardware used: XE121 + ESP32-C3-DevKitM-1

XE121 pin	ESP32C3 pin	Function	Comment
EN1	3	GPIO Output	
EN2	N/A	N/A	Only needed for multi-sensor, not used by default
EN3	N/A	N/A	Only needed for multi-sensor, not used by default
EN4	N/A	N/A	Only needed for multi-sensor, not used by default
EN5	N/A	N/A	Only needed for multi-sensor, not used by default
MOSI	7	SPI2	Dedicated IO_MUX for SPI2
MISO	2	SPI2	Dedicated IO_MUX for SPI2
SCK	6	SPI2	Dedicated IO_MUX for SPI2
CS	10	SPI2	Dedicated IO_MUX for SPI2
SEL0	9	GPIO Output	
SEL1	8	GPIO Output	
SEL2	18	GPIO Output	
INT1	5	External Interrupt	GPIO input + GPIO_INTR_POSEDGE
INT2	N/A	N/A	Only needed for multi-sensor, not used by default
INT3	N/A	N/A	Only needed for multi-sensor, not used by default
INT4	N/A	N/A	Only needed for multi-sensor, not used by default
INT5	N/A	N/A	Only needed for multi-sensor, not used by default

5.3 Pin connection for ESP32S3 SDK

Hardware used: XE121 + Ardi-32

XE121 pin	ESP32S3 pin	Function	Commment
EN1	7	GPIO Output	
EN2	N/A	N/A	Only needed for multi-sensor, not used by default
EN3	N/A	N/A	Only needed for multi-sensor, not used by default
EN4	N/A	N/A	Only needed for multi-sensor, not used by default
EN5	N/A	N/A	Only needed for multi-sensor, not used by default
MOSI	11	SPI2	Dedicated IO_MUX for SPI2
MISO	13	SPI2	Dedicated IO_MUX for SPI2
SCK	12	SPI2	Dedicated IO_MUX for SPI2
CS	10	SPI2	Dedicated IO_MUX for SPI2
SEL0	6	GPIO Output	
SEL1	5	GPIO Output	
SEL2	4	GPIO Output	
INT1	47	External Interrupt	GPIO input + GPIO_INTR_POSEDGE
INT2	N/A	N/A	Only needed for multi-sensor, not used by default
INT3	N/A	N/A	Only needed for multi-sensor, not used by default
INT4	N/A	N/A	Only needed for multi-sensor, not used by default
INT5	N/A	N/A	Only needed for multi-sensor, not used by default

6 Build and Running Acconeer Samples

In order to build and run the Acconeer examples, you will need to install ESP-IDF on Windows or Linux.

In this guide we will run “example_bring_up” on XE121+Ardi-32 and cover two options of several options to run the example:

- ESP-IDF Terminal on Ubuntu 24.04
- ESP-IDF Extension for VS Code in Windows 11

6.1 ESP-IDF Terminal on Ubuntu 24.04

At the time of writing this guide, the current way of installing ESP-IDF is the following:

```
$ mkdir -p ~/esp
$ cd ~/esp
$ git clone -b \espidfversion{} --recursive https://github.com/espressif/esp
   -idf.git
$ cd ~/esp/esp-idf
```

For the following command, you can exchange the argument “all” with for example “esp32s3” if you only want to install the tools needed for “esp32s3”.

```
$ ./install.sh all
```

In your user home directory, open the file “.bashrc” and enter:

```
alias get_idf='. $HOME/esp/esp-idf/export.sh'
```

```
$ source ~/.bashrc
```

Writing “get_idf” in the terminal will now allow you to idf.py commands. Enter “get_idf” in the terminal.

If these instructions are outdated or do not work, please visit Espressif for the best instructions on installing their tools: https://docs.espressif.com/projects/esp-idf/en/stable/esp32/get-started/linux-macos-setup.html

6.1.1 Open an Acconeer example project

Open terminal in the Acconeer SDK folder.

```
$ cd samples/example_bring_up
```

6.1.2 Build an example

To build the example_bring_up example, run the following commands from the example project folder:

```
$ idf.py fullclean
$ idf.py build
```

6.1.3 Flash the device

Connect your device to your PC/laptop and run the following command from the example project folder after building:

```
$ idf.py flash
```

6.1.4 Capture logs

Run the following command to capture the logs from the device:

```
$ idf.py monitor
```

6.2 ESP-IDF Extension for VS Code in Windows 11

When following the Espressif instruction, make sure to select ESP-IDF version v5.4.1, since this is the version currently used by Acconeer. For the best instructions on how to install ESP-IDF Extension in VS code, it is adviced to follow the instructions on the Espressif Github page.

6.2.1 Open an Acconeer example project

Go to "File → Open Folder", Select the "example_bring_up" folder in the samples folder of the SDK.

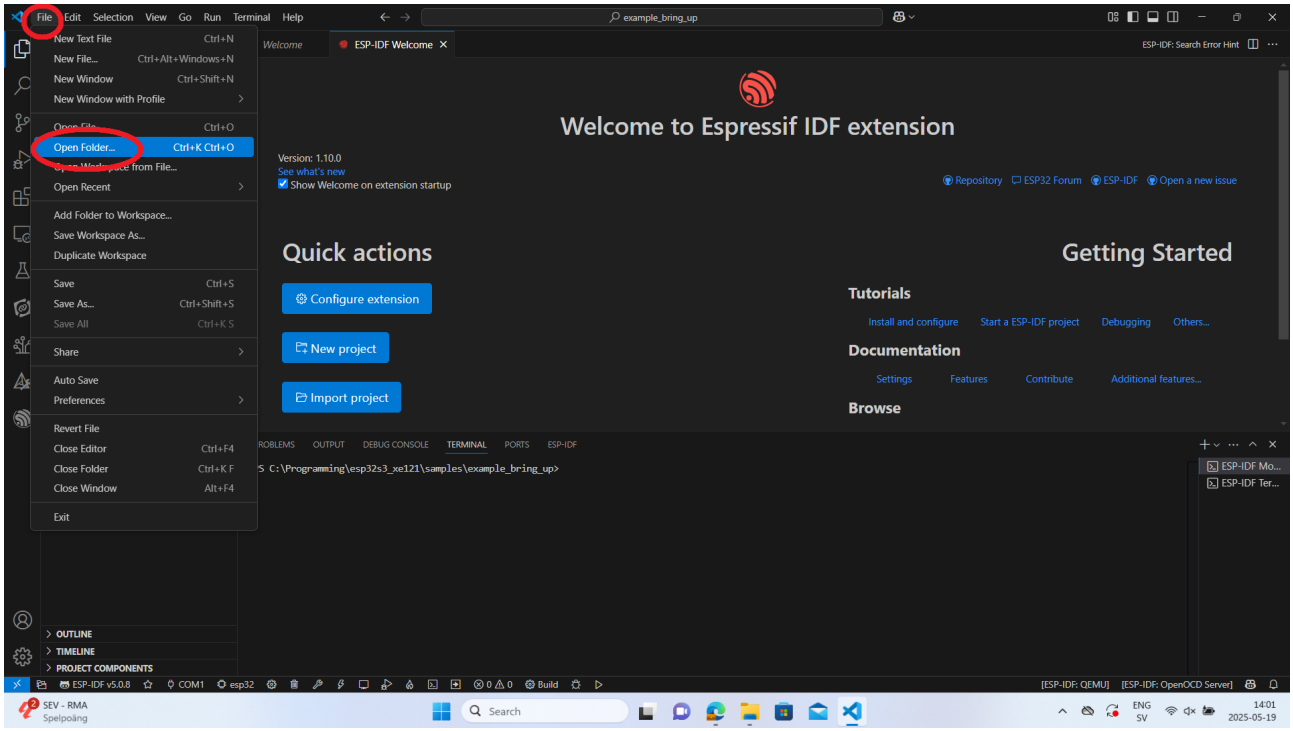

Figure 1: Open Folder

Click select folder.

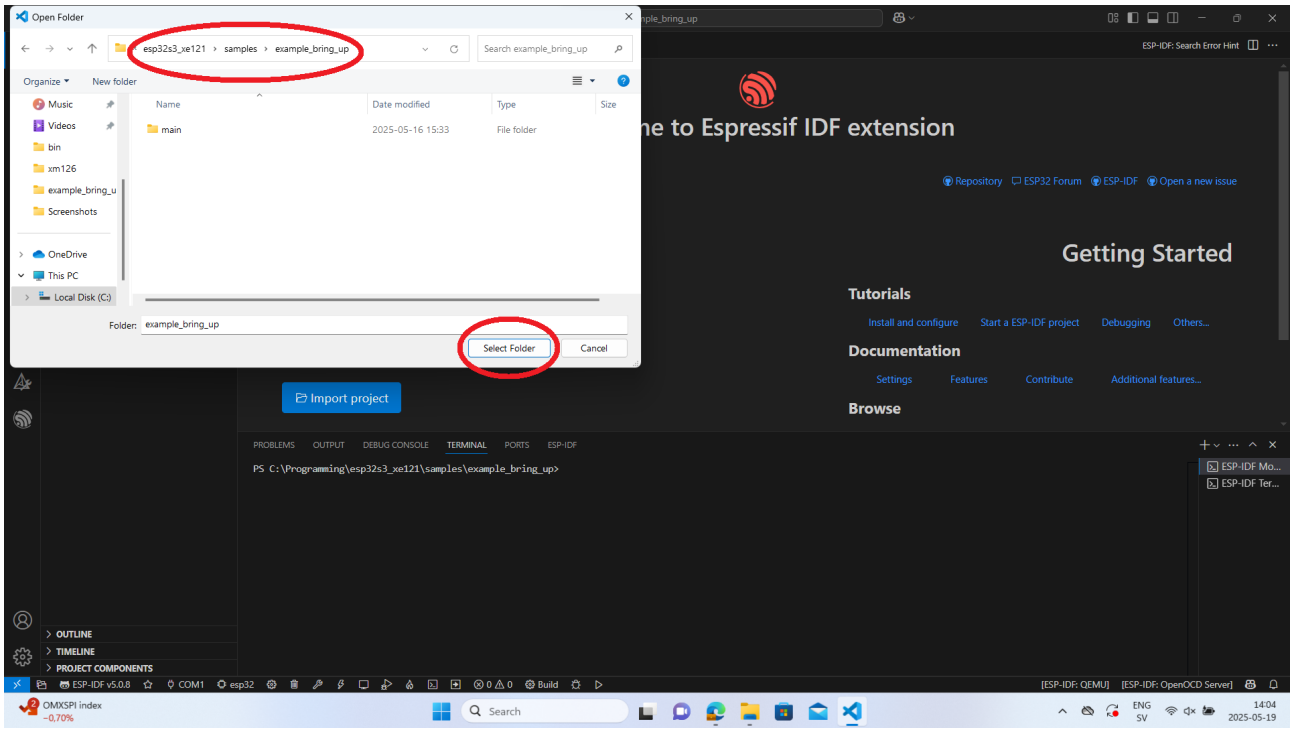

Figure 2: Select folder

6.2.2 Build the example

Click ESP32, “Set Espressif Device Target (IDF_TARGET)”, select your device.

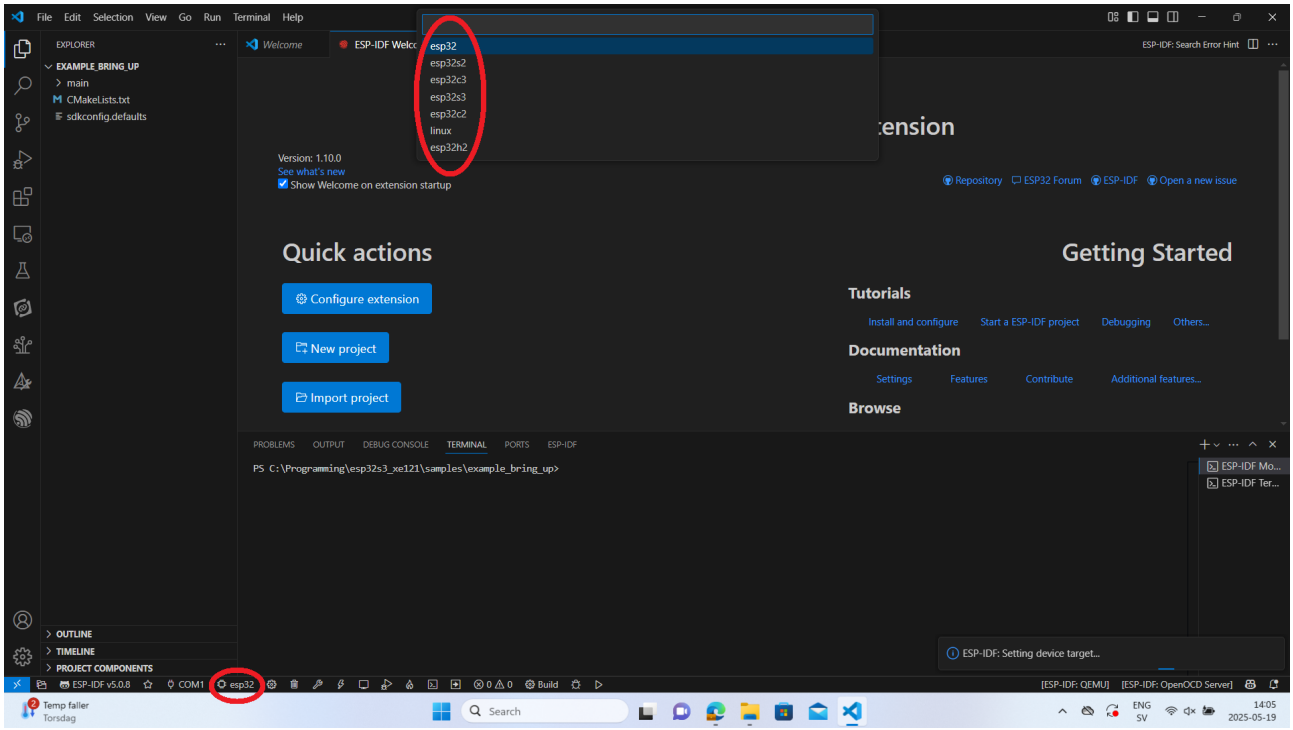

Figure 3: Select Device

Select “OpenOCD Configuration File Path”.

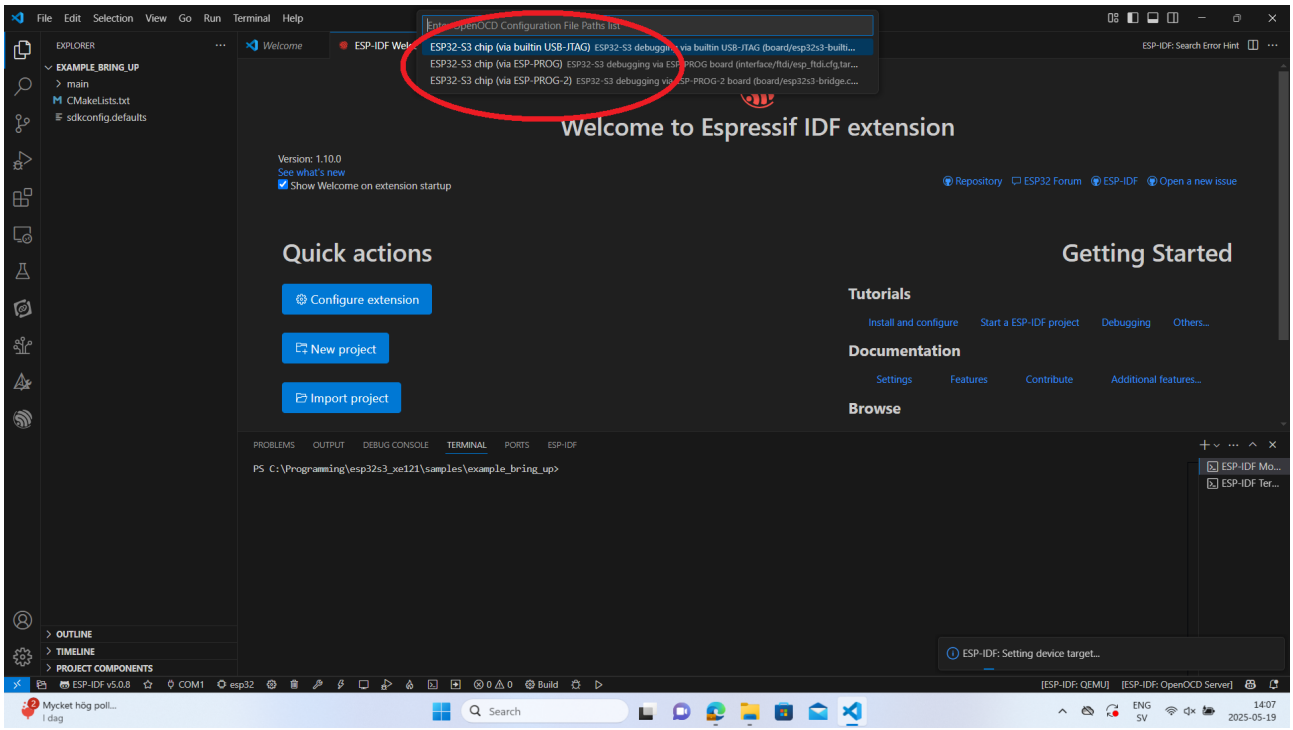

Figure 4: Select OpenOCD

Once selected the tool should generate needed project files, sdkconfig from sdkconfig.defaults and .vscode folder.

Click “Build Project”

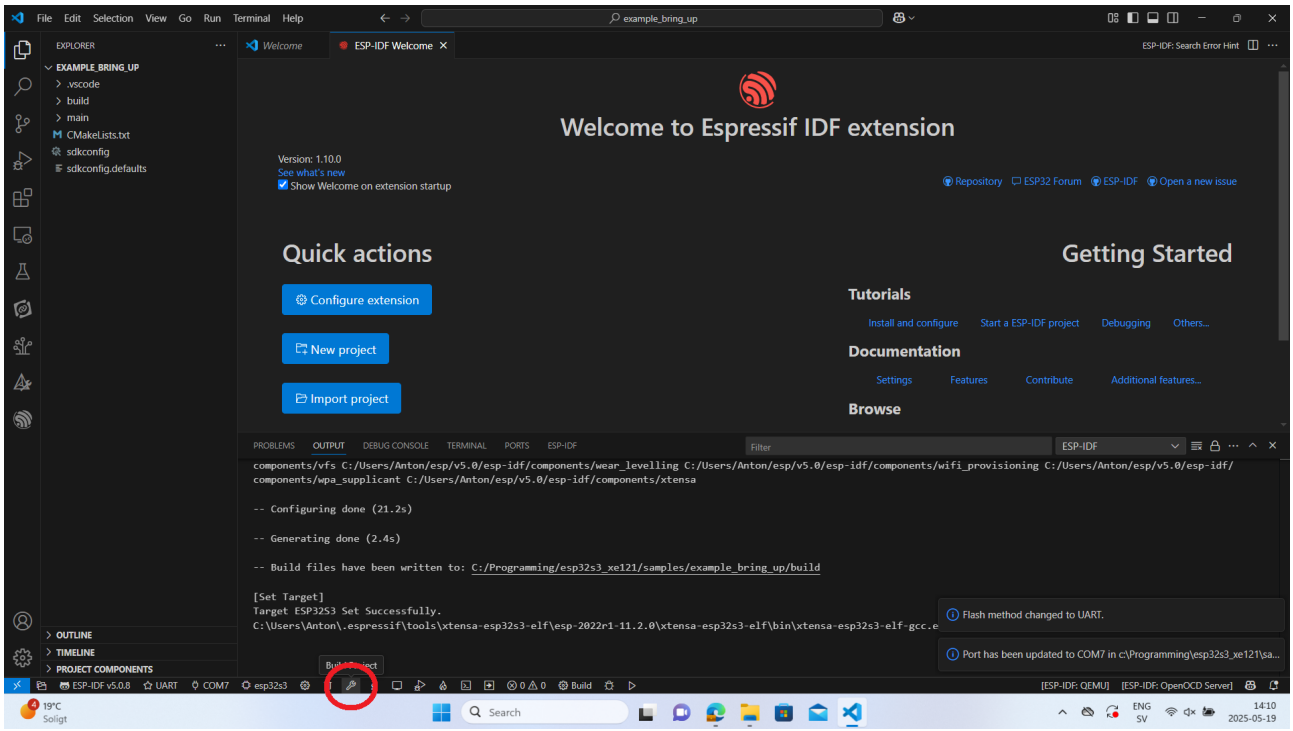

Figure 5: Build Project

6.2.3 Run the example

Connect your ESP32 device to your PC. Select COM.

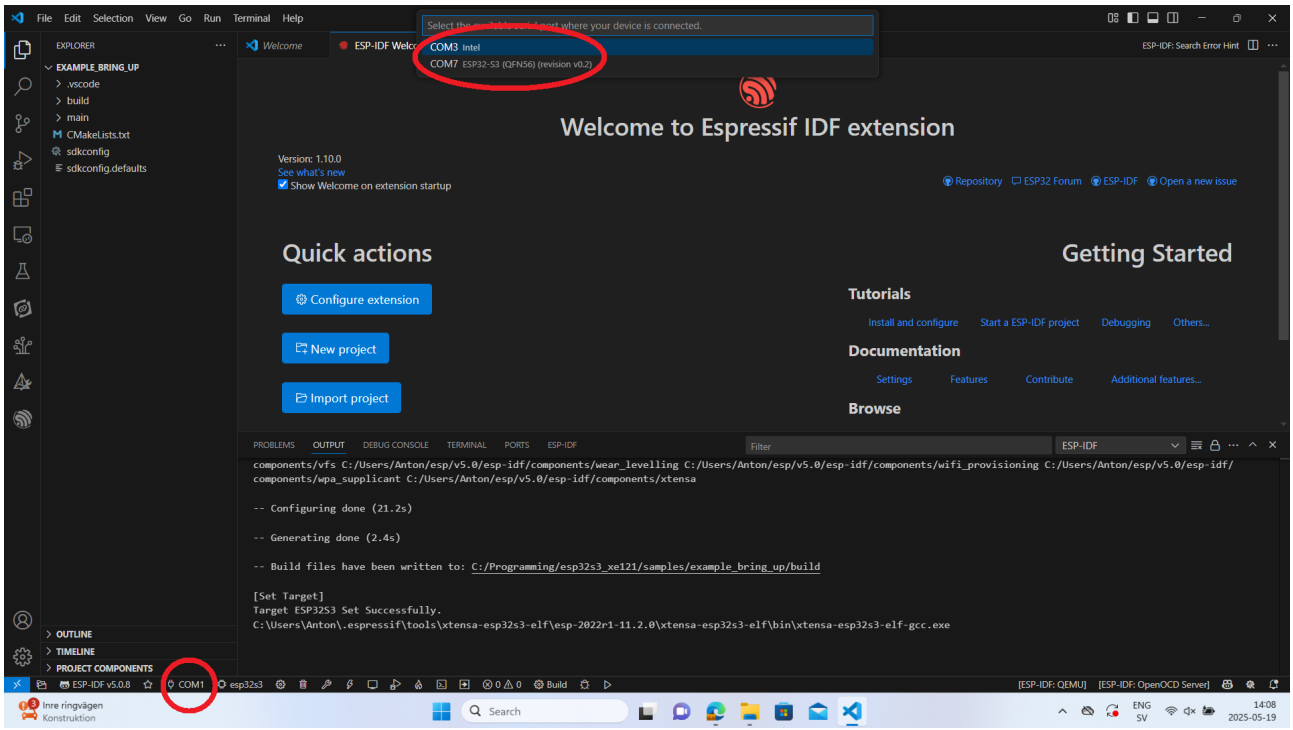

Figure 6: Select COM port

Click “ESP-IDF:Select Flash Method”

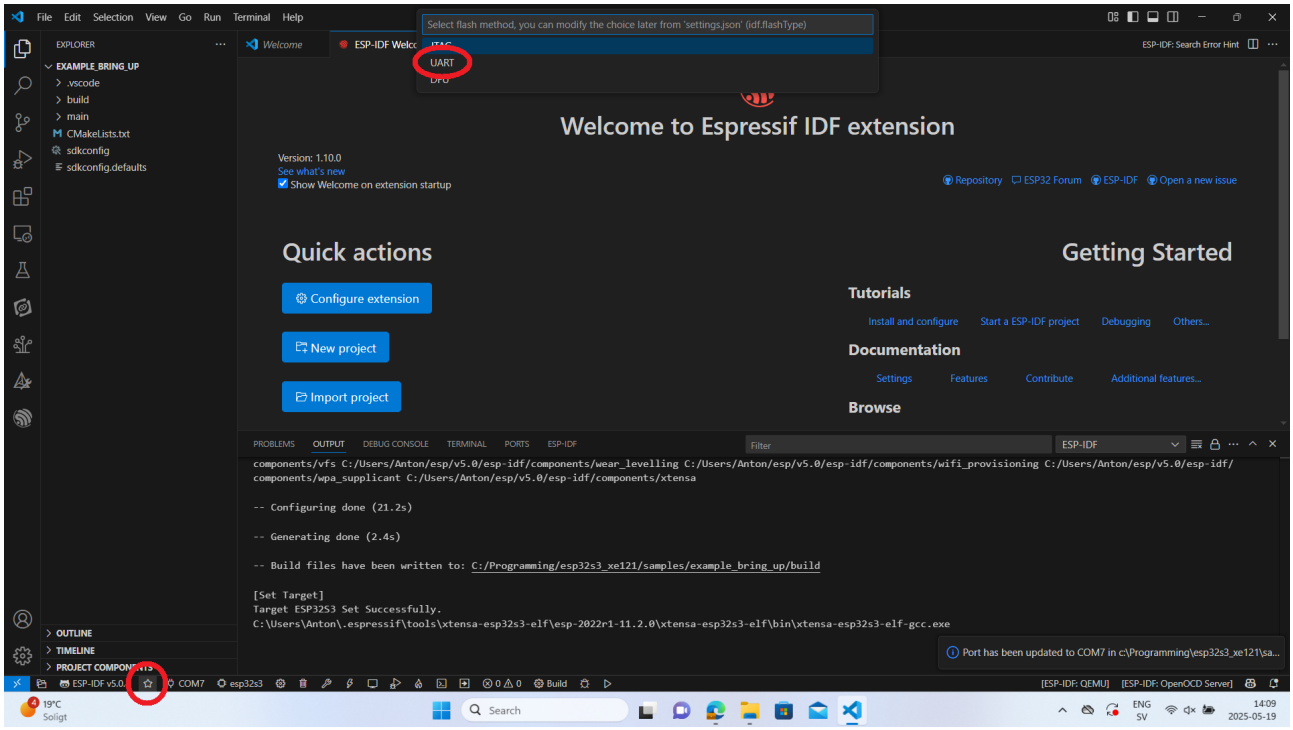

Figure 7: Select Flash Method

Click “Flash Device”

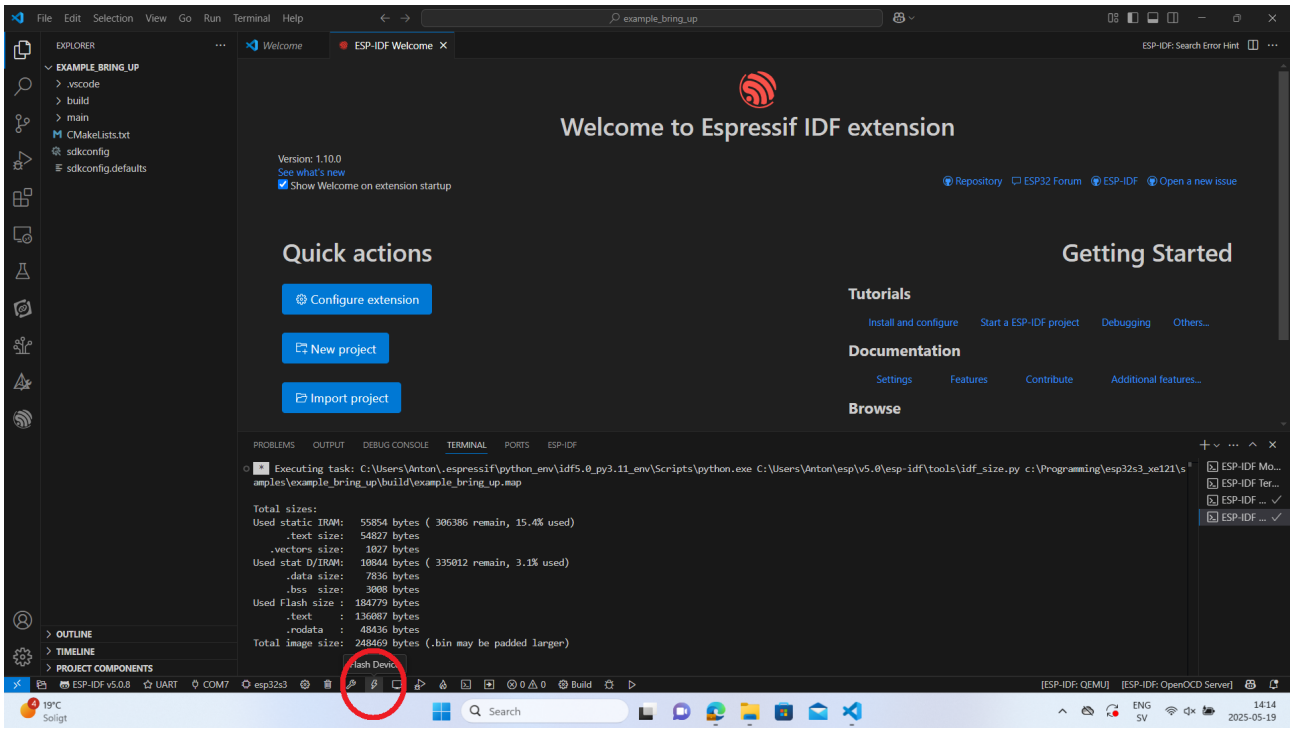

Figure 8: Flash Device

6.2.4 Capture logs

Click “Monitor Device”

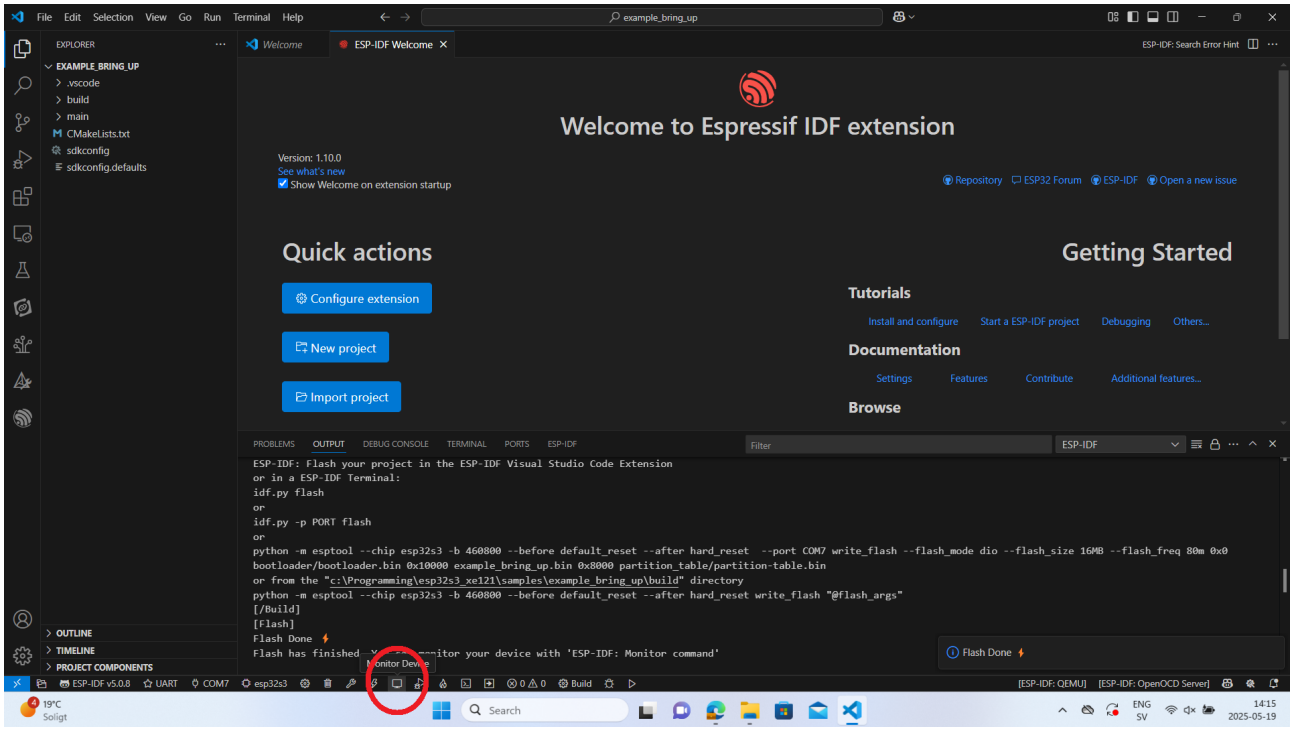

Figure 9: Monitor Device

7 Modifying a sample project

Some users might want to modify one of the default samples or change anything in the integration such as which pins to use or change the SPI frequency.

This section aims to help the user create their own project using the Acconeer sample projects and help locate common things which the user may want to modify.

7.1 Create your own sample project

Copy the “samples/example_bring_up” folder into the “samples”-folder and rename it to the project name you want.

For example “samples/my_example_project”.

Go into the “samples/my_example_project” folder and modify the CMakeLists.txt file. Change:

```
project(example_bring_up)
```

To:

```
project(my_example_project)
```

Now a new project folder has been created.

7.2 Change the example source file used

In the main folder of your project, you will find another CMakeLists.txt file which specifices sources used.

```
idf_component_register(SRCS "${ACCONEER_ROOT}/source/example_bring_up.c"
                       SRCS "${ACCONEER_ROOT}/source/acc_algorithm.c"
                       SRCS "${ACCONEER_ROOT}/source/acc_control_helper.c"
                       SRCS "${ACCONEER_ROOT}/source/
                          acc_hal_integration_espidf_xe121.c"
                       SRCS "${ACCONEER_ROOT}/source/acc_integration_esp32.c
                          "
                       SRCS "${ACCONEER_ROOT}/source/acc_integration_log.c"
                       SRCS "${ACCONEER_ROOT}/source/acc_processing_helpers.
                          c"
                       INCLUDE_DIRS "${ACCONEER_ROOT}/include")
```

If you instead want to build the “waste level” example. You can remove example_bring_up.c then add the source files example_waste_level.c and example_waste_level_main.c:

```
idf_component_register(SRCS "${ACCONEER_ROOT}/source/example_waste_level.c"
                       SRCS "${ACCONEER_ROOT}/source/
                          example_waste_level_main.c"
                       SRCS "${ACCONEER_ROOT}/source/acc_algorithm.c"
                       SRCS "${ACCONEER_ROOT}/source/acc_control_helper.c"
                       SRCS "${ACCONEER_ROOT}/source/
                          acc_hal_integration_espidf_xe121.c"
                       SRCS "${ACCONEER_ROOT}/source/acc_integration_esp32.c
                          "
                       SRCS "${ACCONEER_ROOT}/source/acc_integration_log.c"
                       SRCS "${ACCONEER_ROOT}/source/acc_processing_helpers.
                          c"
                       INCLUDE_DIRS "${ACCONEER_ROOT}/include")
```

7.3 Change pins used

In some cases the user might need to change pins used. The pins used together with the A121 can be found in the file “acc_hal_integration_espidf_xe121.c” in the “source”-folder.

In the ESP32S3 SDK we find them as:

```
#define GPIO_SEL0 6
#define GPIO_SEL1 5
#define GPIO_SEL2 4

#define GPIO_ENABLE    7
#define GPIO_INTERRUPT 47
#define GPIO_SCLK      12
#define GPIO_MOSI      11
#define GPIO_MISO      13
#define GPIO_CS        10
```

Simply change the pin numbers to the pin numbers you want to use on your device.

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.