

TYPE 2AB EVB Developer Manual

Qorvo

Release QM33SDK-1.1.1

© 2025 Qorvo US, Inc.
Qorvo® Proprietary Information

Contents

1 Revision History	1
2 Glossary	3
3 Introduction	8
3.1 Overview	8
3.2 Package Content	9
4 Board Connections	11
4.1 Type 2AB EVB	11
5 Building and flashing	12
5.1 Overview	12
5.2 Setup	12
5.2.1 Required tools	12
5.2.2 Setup environment	14
5.2.3 Hardware tools	15
5.3 Visual Studio Code	15
5.3.1 Setup	15
5.3.2 Tasks	20
5.4 Standalone building and flashing	30
5.4.1 Building SDK Firmware	30
5.4.2 Flashing the development kit	32
6 Projects	35
7 CLI - Command Line Interface	37
7.1 Overview	37
7.2 Setup	37
7.3 Anytime Commands	38
7.3.1 Overview	38
7.3.2 HELP	39
7.3.3 STAT	40
7.3.4 STOP	41
7.3.5 THREAD	41
7.4 Service Commands	41
7.4.1 RESTORE	42
7.4.2 LCFG	42
7.4.3 DIAG	43
7.4.4 DECAID	44
7.4.5 SAVE	44
7.4.6 SETAPP	45
7.4.7 GETOTP	45
7.5 IDLE time Commands	46
7.5.1 Overview	46
7.5.2 UART	46

7.5.3	CALKEY	47
7.5.4	LISTCAL	47
7.6	Application Commands	48
7.6.1	SDK Applications	48
7.6.2	FiRa applications: INITF/RESPF	48
7.6.3	UWB sniffer: LISTENER	52
8	Qorvo One TWR GUI	55
8.1	Overview	55
8.2	User Manual	55
8.2.1	Installation	55
8.2.2	Welcome screen	60
8.2.3	Setup	61
8.2.4	Real-Time Location	66
8.2.5	Trend Over Time	69
8.2.6	Locate My Device	70
8.2.7	Device List	71
8.2.8	Calibration	71
8.2.9	Geofencing	75
8.2.10	Floor Plan	78
8.2.11	Grid	79
8.2.12	Logging	80
9	UWB Qorvo Tools	81
9.1	Overview	81
10	UCI - UWB Command Interface	82
10.1	Overview	82
11	SDK Runtime	83
11.1	Always running Threads	83
11.1.1	Default Task	83
11.1.2	Control Task	83
11.1.3	Flush Task	84
11.1.4	Logger Task	84
11.1.5	Tmr Svc Task	84
11.1.6	Idle Task	84
11.2	Application specific Threads	84
11.2.1	Listener Task	84
11.2.2	LLHW MCPS Task	84
11.2.3	Qworkqueue Task	85
11.2.4	UCI task	85
11.3	Threads stack usage	85
11.3.1	FreeRTOS	85
12	SDK SW Architecture	86
12.1	Overview	86
12.2	Linker sections	86
12.2.1	Flash sections	87
12.2.2	RAM sections	88
12.3	NVM configuration	88
12.3.1	Overview	88
12.3.2	Initialization sequence during powerup	89
12.4	Folders structure	90
12.5	Layers	90
12.5.1	AppConfig API	90
12.5.2	Applications layer (Apps)	91

12.5.3 BLE Folder	92
12.5.4 Board Abstraction Layer (Boards)	94
12.5.5 Hardware Abstraction Layer (HAL)	96
12.5.6 OSAL folder	103
12.5.7 Projects folder	103
12.5.8 SDK BSP Folder	105
12.5.9 Libs Folder	105
13 Customizing the firmware	106
13.1 CMake flags customization	106
13.1.1 Project CMake Configuration	106
13.1.2 UWB-STACK CMake Configuration	110
13.1.3 DWT UWB Drivers CMake Configuration	110
13.2 Adding a new CLI command	111
13.2.1 CLI Overview	111
13.2.2 Implementation of the new command	113
13.2.3 Implementation of the new subcommand	113
13.3 Porting Platform	114
13.3.1 Memory requirements	114
13.3.2 Porting HAL	122
13.3.3 Porting BAL	125
13.3.4 Porting to other OS	126
13.3.5 Porting Project	129
13.3.6 Porting to other Board	130
14 Calibration and Configuration	131
14.1 Overview	131
14.1.1 Introduction	131
14.1.2 Understanding the Calibration and Configuration	132
14.2 Pushing calibration to a device	133
14.2.1 Pushing the calibration over UCI	133
14.2.2 Pushing the calibration over GUI	133
14.3 Calibration and Configuration keys dictionary	134
14.4 Erase calibration	134
14.4.1 How to reset calibration	134
14.4.2 How to erase calibration	134
14.5 Distance and Angle Calibration	135
14.5.1 Distance	136
14.5.2 Angle	136
15 OTP Memory Map	138
15.1 Murata Type 2AB EVB specific OTP	139
15.2 OTP Customer part description	140
16 Licenses in this SDK	141
17 Getting support and contacting Qorvo®	142
18 Important Notice	143
Index	144

© 2025 Qorvo US, Inc.
Qorvo® Proprietary Information

1 Revision History

Version	Date	Comment
DW3_QM33_SDK_1.1.0	2025-08-13	<ul style="list-style-type: none"> Added <i>Glossary</i> chapter to provide a list of terms and abbreviations used in the documentation. Updated <i>Predefined tasks</i> subsection to the <i>Building and Flashing</i> chapter, providing examples of how to utilize predefined tasks in the SDK. Added <i>Projects</i> chapter including the following projects: <ul style="list-style-type: none"> * <i>HelloWorld</i> - demonstrating basic interaction with the UWB transceiver and the Qorvo UWB stack. * <i>CLI</i> - demonstrating basic functionality of the uwb-stack over command line interface (CLI). * <i>UCI</i> - providing a way to fully control FiRa uwb-stack over UCI interface. * <i>QANI</i> - showcasing the QANI (Qorvo Nearby Interaction) to enable proximity interaction with Apple's Nearby Interaction protocol, leveraging both BLE and UWB technologies. Introduced following changes in the <i>SDK SW Architecture</i> chapter: <ul style="list-style-type: none"> Updated <i>Layers</i> subchapter with <i>BLE Folder</i> section, providing explanations of the structure, responsibilities, and integration of the BLE folder within the SDK. Removed obsolete sections: <i>HAL GPIO</i>, <i>HAL BUTTON</i>, <i>HAL LED</i>, <i>HAL SPI</i>, <i>HAL RTC</i>, <i>HAL TIMER</i>, <i>HAL UWB</i>, and <i>HAL SLEEP</i>, as these components have been removed from the SDK. Added <i>HAL CPU</i> section describing the structure and API of the HAL CPU component. Added <i>Libs Folder</i> section detailing the structure and contents of the <i>Libs</i> folder used in the SDK. Introduced following changes in the <i>Customizing the firmware</i> chapter: <ul style="list-style-type: none"> Expanded <i>CMake flags customization</i> section with information about the 'Example and Board specific CMake flags. Added <i>Memory requirements</i> section to <i>Porting Platform</i> subchapter, detailing memory requirements of each specific firmware. Added <i>Low power consideration</i> subsection to <i>Porting HAL</i> subchapter, describing timers and their role in power management, as well as listing hardware requirements to enable porting the power consumption optimization solution based on RTC to other platforms. Added <i>Index</i> chapter to provide a comprehensive list of terms and their corresponding location in the documentation.
DW3_QM33_SDK_1.0.0	2024-08-22	<p>© 2025 Qorvo US, Inc. Qorvo® Proprietary Information</p> <ul style="list-style-type: none"> Restructured documentation into individual document for each supported development kit. Updated <i>Board Connections</i> chapter with information on communication interfaces. Updated building and flashing description: <ul style="list-style-type: none"> Merged <i>Flashing SDK Firmware</i> and <i>Building SDK Firmware</i> chapters into one - <i>Building and flashing</i>.
August 2025 Subject to change without notice		<p>2 of 145</p> <p>Release QM33SDK-1.1.1 www.qorvo.com</p>

2 Glossary

ACM

Abstract Control Model

AES

Advanced Encryption Standard

ANIS

Apple Nearby Interaction Service

AoA

Angle of Arrival

API

Application Programming Interface

ARM

Advanced RISC Machine

BAL

Board Application Layer

BLE

Bluetooth Low Energy

BPRF

Base Pulse Repetition Frequency

CDC

Communication Device Class

CFO

Clock Frequency Offset

CIR

Channel Impulse Response

CLI

Command Line Interface

COM

Communication Port

CRC

Cyclic Redundancy Check

DEVKIT

Development Kit

DS-TWR

Double-Sided Two-Way Ranging

EVB

Evaluation Board

© 2025 Qorvo US, Inc.
Qorvo® Proprietary Information

FCC

Federal Communications Commission

FiRa

Fine Ranging

FoM

Figure of Merit

FP

First Path

FPU

Floating Point Unit

FSL

First Signal Level

FW

Firmware

GCC

GNU Compiler Collection

GDB

GNU Debugger

GPIO

General Purpose Input Output

GUI

Graphical User Interface

HAL

Hardware Abstraction Layer

IDE

Integrated Development Environment

IPATOV

Name given to preamble/sync sequences used in UWB

ISR

Interrupt Service Routine

IV

Initialization Vector

JSON

JavaScript Object Notation

L1

Layer One

LL

Link Layer

LLHW

Low Level Hardware

LNA

Low Noise Amplifier

LUT

Lookup Table

MAC

Medium Access Control

MCU

MicroController Unit

NA

Not Applicable

NI

Nearby Interaction

NVM

Non Volatile Memory

OOB

Out of Band

OS

Operating System

OSAL

Operating System Abstraction Layer

OTP

One Time Programmable

PAC

Peripheral Access Controller

PA

Power Amplifier

PCTT

PHY Conformance Test Tool

PDoA

Phase Difference of Arrival

PHR

Physical Layer Header

PHY

Physical Layer

PLL

Phase Locked Loop

PRF

Pulse Repetition Frequency

PSDU

PHY Service Data Unit

PSR

Preamble Symbol Repetitions

QANI

Qorvo Apple Nearby Interaction

QHAL

Qorvo Hardware Abstraction Layer

QNIS

Qorvo Nearby Interaction Service

QOSAL

Qorvo Operating System Abstraction Layer

RAM

Random Access Memory

RFRAME

Ranging Frame

RFU

Reserved for Future Use

RRU

Radio Resource Unit

RR

Ranging Round

RSL

Received Signal Level

RSP

Response Phase

RSSI

Received Signal Strength Indicator

RSTU

Ranging Scheduling Time Unit

RTOS

Real-Time Operating System

RTT

Real Time Transfer

RX

Receive

SFD

Start of Frame Delimiter

SOC

System On a Chip

SPI

Serial Peripheral Interface

SS-TWR

Single-Sided Two-Way Ranging

STS

Scrambled Timestamp Sequence

SW

Software

SWD

Serial Wire Debug

TWR

Two Way Ranging

TX

Transmit

UART

Universal Asynchronous Receiver Transmitter

UCI

UWB Subsystem Command Interface

USB

Universal Serial Bus

UWB

Ultra-Wide Band

UWBS

Ultra Wide-Band Subsystem

UWBMAC

Ultra-Wide Band Medium Access Control

XTAL

Crystal Oscillator

3 Introduction

3.1 Overview

The SDK package is designed to support developers with comprehensive resources, including a Developer Manual and a Quick Start Guide, to help working with the QM33 SDK. Below is a structured breakdown of the package contents and associated documentation:

Quick Start Guide

The Quick Start Guide, available at [SDK/Documentation/QuickStartGuide](#), includes essential guidelines for getting started with the SDK package.

Developer Manual

The Developer Manual, available at [SDK/Documentation/DeveloperManual](#), contains comprehensive information about the SDK package.

Tools

- The UWB Qorvo Tools are provided as a set of Python scripts to demonstrate key features of the UCI.
- The Qorvo One GUI is provided as a visualization application for ranging capabilities of the device.

Binary and Source Code

The package includes a precompiled binary for the target and source code, enabling developers to start developing their solutions immediately.

Additional Documentation

The SDK package also includes extensive documentation for various components and APIs, as listed below:

- **DW3xxx/QM33xxx Device Driver Application Programming Interface (API) Guide** - refer to [SDK/Documentation/Drivers/QM33XXX_DW3XXX_Software_API_Guide-<x>p<y>.pdf](#)
- **UWB FiRa Protocol Documentation** - refer to [SDK/Documentation/uwb-stack/uwb-fira-protocol-R<x.x.x-y>.pdf](#)
- **UWB L1 API Documentation** - refer to [SDK/Documentation/uwb-stack/uwb-l1-api-R<x.x.x-y>.pdf](#)
- **UWB-Stack L1 Configuration Documentation** - refer to [SDK/Documentation/uwb-stack/uwb-l1-configuration-R<x.x.x-y>.pdf](#)
- **UWB QHAL API Documentation** - refer to [SDK/Documentation/uwb-stack/uwb-qhal-api-R<x.x.x-y>.pdf](#)
- **UWB QOSAL API Documentation** - refer to [SDK/Documentation/uwb-stack/uwb-qosal-api-R<x.x.x-y>.pdf](#)
- **UWB Qplatform API Documentation** - refer to [SDK/Documentation/uwb-stack/uwb-qplatform-api-R<x.x.x-y>.pdf](#)
- **UWB UCI Message API Documentation** - refer to [SDK/Documentation/uwb-stack/uwb-uci-messages-api-R<x.x.x-y>.pdf](#)

- **UWB UWBMAC API Documentation** - refer to [SDK/Documentation/uwb-stack/uwb-uwbmac-api-R<x.x.x-y>.pdf](#)

Note: For anything related to hardware design and custom board creation using Qorvo UWB transceivers, it is recommended to read the documentation provided in [APH301 Hardware Design Guide for DW3000 and QM33100 Series ICs¹](#). This document will provide you with the necessary guidelines and instructions for designing and manufacturing your own custom hardware.

For better understanding the Angle of Arrival (AoA) feature, it is advisable to refer to the documentation in [APH511 UWB AoA Antenna Fundamentals²](#). These resources will provide you with the necessary answers to commonly asked questions on antennas to implement or use the AoA functionality.

By following the guidelines and recommendations provided in these documents, you will be able to get the best performance from your own hardware.

3.2 Package Content

The following table shows SDK package content once unzipped.

Table 3.1: SDK package content

Folder name	Folder description
SDK/Binaries	Executable files to flash the board (UCI and CLI).
SDK/Documentation	Documentation of libraries, SDK Developer Manual, SDK Quick Start Guide, release notes.
SDK/Firmware	SDK source code.
SDK/Tools	Qorvo One GUI application and UWB Qorvo Tools.

Warning: Windows operating system has a known limitation of the maximum path length. It can cause issues when using the SDK package. It is recommended to enable Long Path support, otherwise you may face the following issues:

- When extracting the zipped package inside the Firmware directory, the error can occur: “Error 0x80010135: Path too long”.
- When building the firmware, the error can occur: “No rule to make target <long_path_to_sdk_file>, needed by...”.

To resolve those issues, you can either:

- Run following PowerShell command from a terminal window with elevated privileges:

```
New-ItemProperty -Path "HKLM:\SYSTEM\CurrentControlSet\Control\FileSystem" -Name
    ↳ "LongPathsEnabled" -Value 1 -PropertyType DWORD -Force
```

- Change the following registry key: Computer\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\FileSystem\LongPathsEnabled in the Registry Editor opened with elevated privileges.

If the error persists despite these changes, it is required to use WinZip to extract the package. Alternatively, you can extract the zipped directory at some short path, e.g. C:\sdk instead of inside the Firmware directory.

For more information, refer to the following sections:

- [Building SDK Firmware](#) - Instructions on building the firmware.

¹ <https://www.qorvo.com/products/p/QM33120W#documents>

² <https://www.qorvo.com/products/p/QM33120W#documents>

- [*SDK SW Architecture*](#) - Details of the source code.
- [*Qorvo One TWR GUI*](#) - Information on the Qorvo One GUI.
- [*UWB Qorvo Tools*](#) - Information on Ultra-Wideband Qorvo Tools.

© 2025 Qorvo US, Inc.
Qorvo® Proprietary Information

4 Board Connections

4.1 Type 2AB EVB

To communicate with the board, connect both micro-USB cables to the board.

For more information refer to [muRata® UWB Module\(2AB\) EVB User Manual³](#).

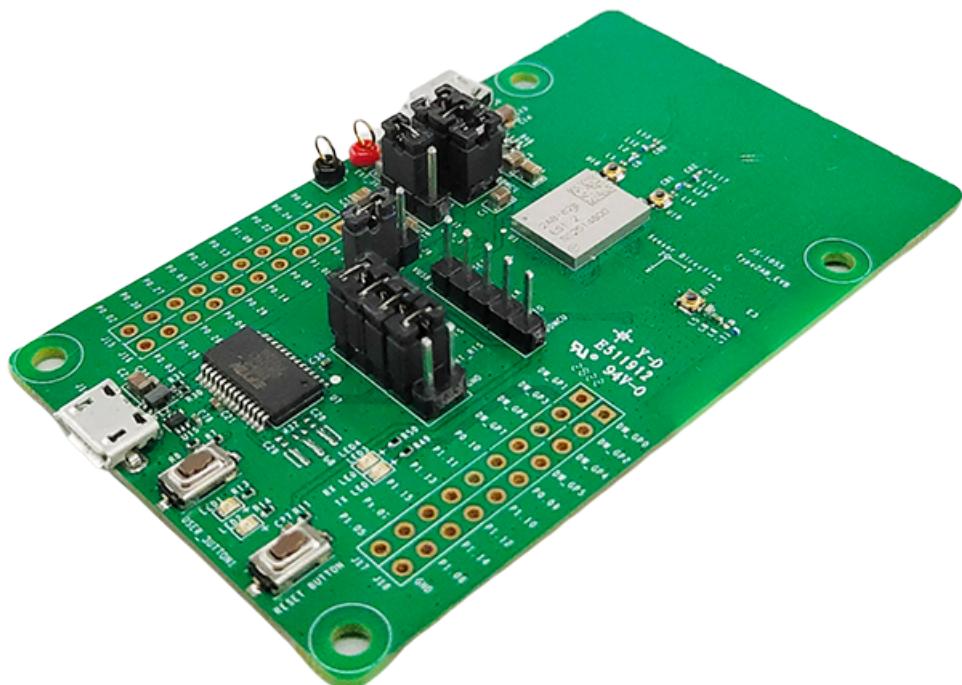


Figure 4.1: Type 2AB EVB overview

To get instructions about connections needed to flash the device, report to [Flashing the development kit](#)

³ <https://www.murata.com/en-us/products/connectivitymodule/ultra-wide-band/qorvo#datasheet-and-documentations>

5 Building and flashing

5.1 Overview

This chapter offers detailed instructions on building firmware and flashing development kit boards.

For development, we recommend utilizing the integrated building and flashing functionalities provided by [Visual Studio Code](#) environment. Nevertheless, the SDK is built upon CMake and Make, ensuring compatibility with various IDEs. If you prefer to use an alternate development environment, please follow [Standalone building and flashing](#) for guidance on executing a standalone build.

Warning: Please be aware that the [Setup](#) subchapter is essential and must be completed prior to proceeding with either the integrated or standalone build approach.

5.2 Setup

5.2.1 Required tools

Before you can build and run the examples, you need to have the following tools installed on your system. Please ensure you install each one before proceeding.

Note: VS Code tool is not mandatory. However, please keep in mind that provided automated tasks for building, flashing and debugging are only supported on VS Code.

5.2.1.1 Windows

- **ARM Toolchain:**
 - Download [arm-none-eabi-gcc-10.3-2021.10-win32](#) from the [ARM download page](#)⁴.
 - Install it inside: `C:\GnuToolsArmForEmbedded\gcc-arm-none-eabi-10.3-2021.10\bin`.
 - Add `C:\\GnuToolsArmForEmbedded\\gcc-arm-none-eabi-10.3-2021.10\\bin` to your system/user environment variables.
- **CMake:**
 - Download CMake from the [CMake download page](#)⁵. The required version is higher than 3.23.
 - Install CMake to a directory of your choice, referred to as `<cmake_path>` in these instructions.
 - Add `<cmake_path>/bin` to your system/user environment variables.

⁴ <https://developer.arm.com/downloads/-/gnu-rm/10-3-2021-10>

⁵ <https://cmake.org/download/>

- **Python:**

- Download Python from the [Python download page⁶](https://www.python.org/downloads/). The required version is 3.10 or higher.
- Install Python and make sure to select the option “Add Python to PATH” during the installation.

- **MinGW and make:**

- MinGW can be downloaded from the [MinGW download page⁷](https://sourceforge.net/projects/mingw/).
- Install MinGW in the directory of your choice.
- Using the MinGW Installation Manager, install mingw32-base to a directory of your choice, referred to as <mingw_path> in these instructions.
- Create a copy of <mingw_path>/bin/mingw32-make.exe and name it <mingw_path>/bin/make.exe.
- Add <mingw_path>/bin to your system/user environment variables.

- **SEGGER J-Link:**

- Install J-Link Software and Documentation Pack from the [Segger downloads page⁸](https://www.segger.com/downloads/jlink/). Recommended version is higher than V7.96j.

- **VS Code:**

- Install VS Code from [VS code download page⁹](https://code.visualstudio.com/Download).

5.2.1.2 Linux

- **ARM Toolchain:**

- Create an install directory:

```
sudo mkdir /opt/gcc
```

- Go to the install directory:

```
cd /opt/gcc
```

- Download [gcc-arm-none-eabi-10.3-2021.10-x86_64-linux](https://developer.arm.com/-/media/Files/downloads/gnu-rm/10.3-2021.10/gcc-arm-none-eabi-10.3-2021.10-x86_64-linux.tar.bz2) from the [ARM page¹⁰](https://www.arm.com/page/10.3-2021.10).

- Extract the tarball:

```
sudo tar -xvf gcc-arm-none-eabi-10.3-2021.10-x86_64-linux.tar.bz2
```

- Remove the tarball:

```
sudo rm gcc-arm-none-eabi-10.3-2021.10-x86_64-linux.tar.bz2
```

- Open `~/.bashrc` and add this line `export PATH="/opt/gcc/gcc-arm-none-eabi-10.3-2021.10/bin:$PATH"`.

- **Make:**

- Install Make:

```
sudo apt-get install build-essential
```

- **CMake:**

⁶ <https://www.python.org/downloads/>

⁷ <https://sourceforge.net/projects/mingw/>

⁸ <https://www.segger.com/downloads/jlink/>

⁹ <https://code.visualstudio.com/Download>

¹⁰ https://developer.arm.com/-/media/Files/downloads/gnu-rm/10.3-2021.10/gcc-arm-none-eabi-10.3-2021.10-x86_64-linux.tar.bz2

- Download CMake from the [CMake github¹¹](#). The required version is higher than 3.23.
- Create a cmake directory in usr/bin:

```
sudo mkdir /usr/bin/cmake
```

- Execute command:

```
sudo ./cmake-<cmake_version>-Linux-x86_64.sh --skip-license --prefix=/usr/bin/cmake
```

- Open `~/.bashrc` and add this line `export PATH="/usr/bin/cmake/bin:$PATH"`.

- **Python:**

- Install Python. The required version is 3.10 or higher:

```
sudo apt-get install -y python3 python3-pip python3-venv
```

- **SEGGER J-Link:**

- Install J-Link Software and Documentation Pack from the [Segger downloads page¹²](#). Recommended version is higher than V7.96j.

- **VS Code:**

- Install VS Code from the [VS code download page¹³](#).

5.2.2 Setup environment

This step is applicable to both Windows and Linux operating systems. Please execute the following commands in your terminal (on Windows please use PowerShell), within the root directory of the project.

1. Create a virtual environment:

```
python -m venv .venv
```

2. Activate the virtual environment:

- On Linux

```
source .venv/bin/activate
```

- On Windows

```
.\venv\Scripts\Activate.ps1
```

3. Install all requirements:

```
pip install -r requirements.txt
```

¹¹ <https://github.com/Kitware/CMake/releases/>

¹² <https://www.segger.com/downloads/jlink/>

¹³ <https://code.visualstudio.com/Download>

5.2.3 Hardware tools

The following hardware tools are needed to flash the binary into the target board over SWD interface:

- J-Link debug/flash probe¹⁴

5.3 Visual Studio Code

SDK provides support to build, flash and debug firmware directly from VS Code. Below you can find brief description of required setup and possible functionalities.

If you wish to use different IDE/code editor, please refer to [Standalone building and flashing](#).

5.3.1 Setup

5.3.1.1 Package location

Warning: Paths containing spaces will lead to errors during the execution of tools and scripts provided within the package, such as Visual Studio Code tasks. To ensure proper functionality, always place this package in a directory path without spaces.

5.3.1.2 Open workspace

To make use of all VS Code features please remember to open project as a workspace:

1. Pick *File* → *Open Workspace from File...* from the menu on top.

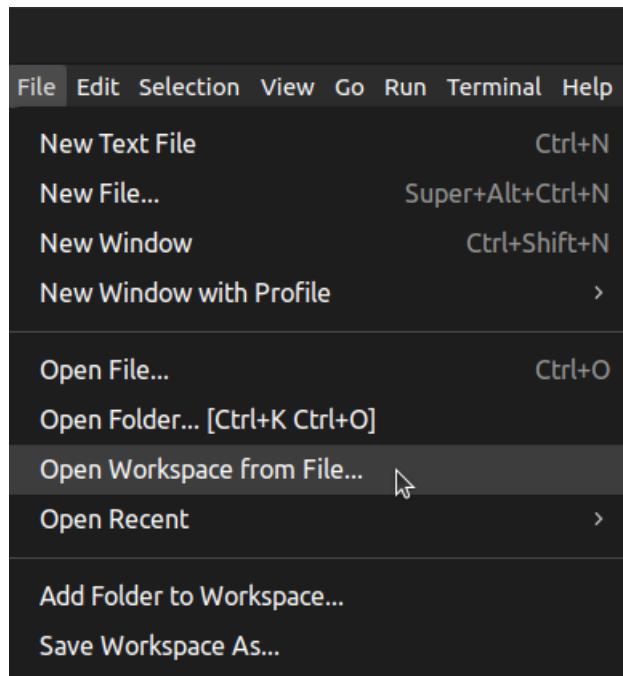


Figure 5.1: VS Code: Open workspace from file.

¹⁴ <https://www.segger.com/products/debug-probes/j-link/>

2. Find and open *DW3_QM33_SDK.code-workspace*.

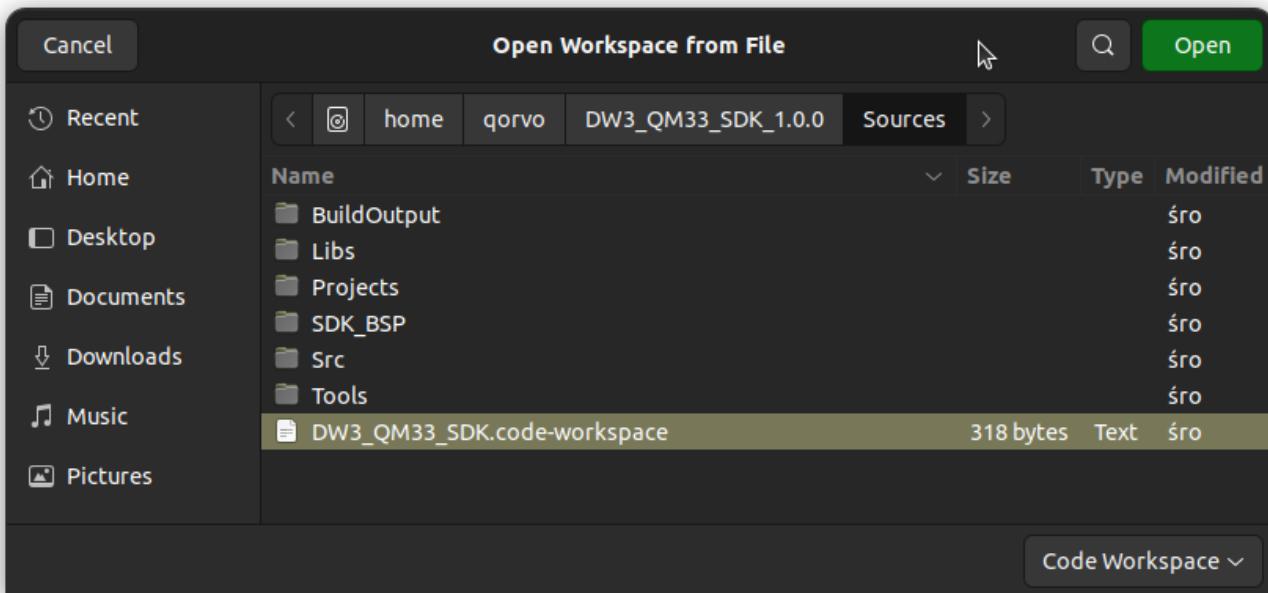


Figure 5.2: VS Code: Find workspace file.

5.3.1.3 Install extensions

To build and run the firmware directly from VS Code, please install all the recommended extensions.

1. Navigate to the Extensions tab located on the left side of the window, or use the keyboard shortcut **Ctrl+Shift+X**.

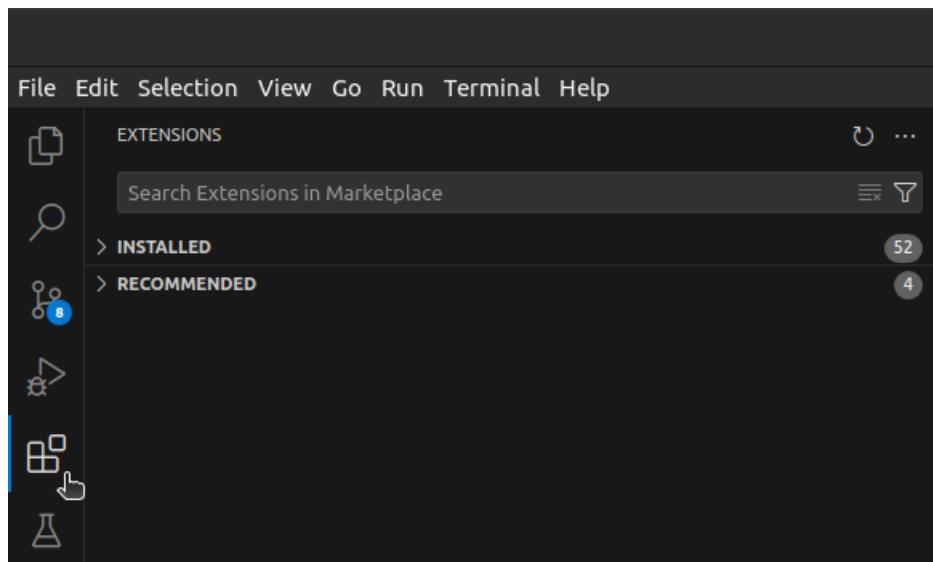


Figure 5.3: VS Code: Extensions tab.

2. Click on the **Filter extensions** button (next to search field).

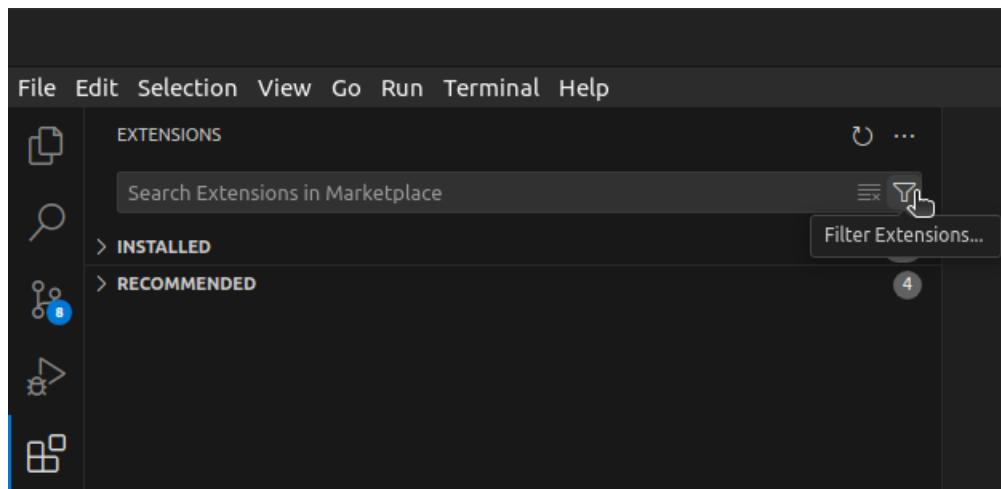


Figure 5.4: VS Code: Filter extensions.

3. Pick the **Recommended** category.

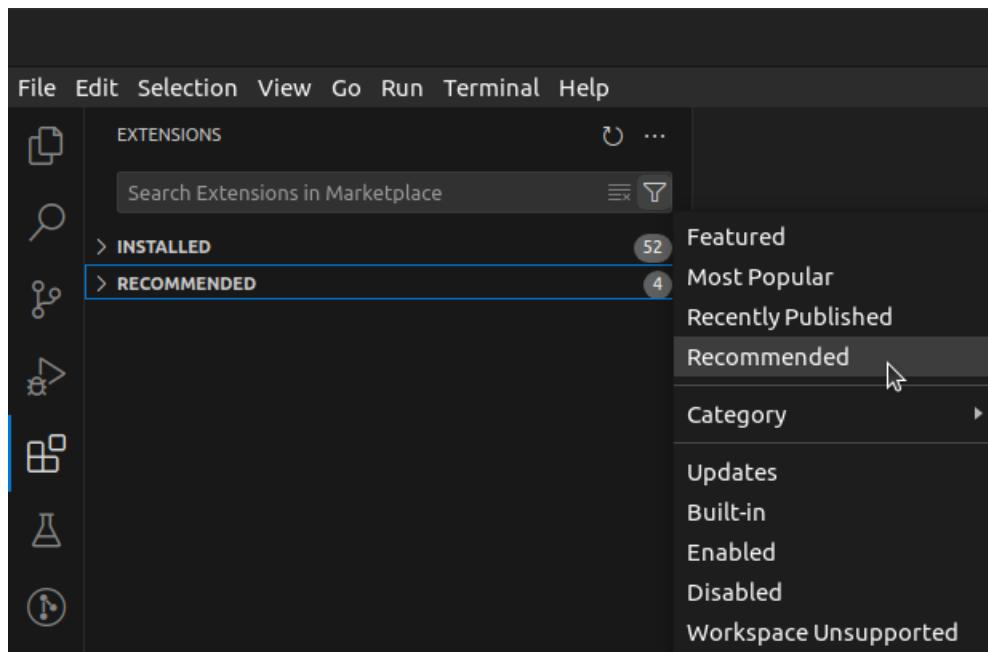


Figure 5.5: VS Code: Filter recommended extensions.

4. Click on the cloud icon located just below to install all recommended extensions.

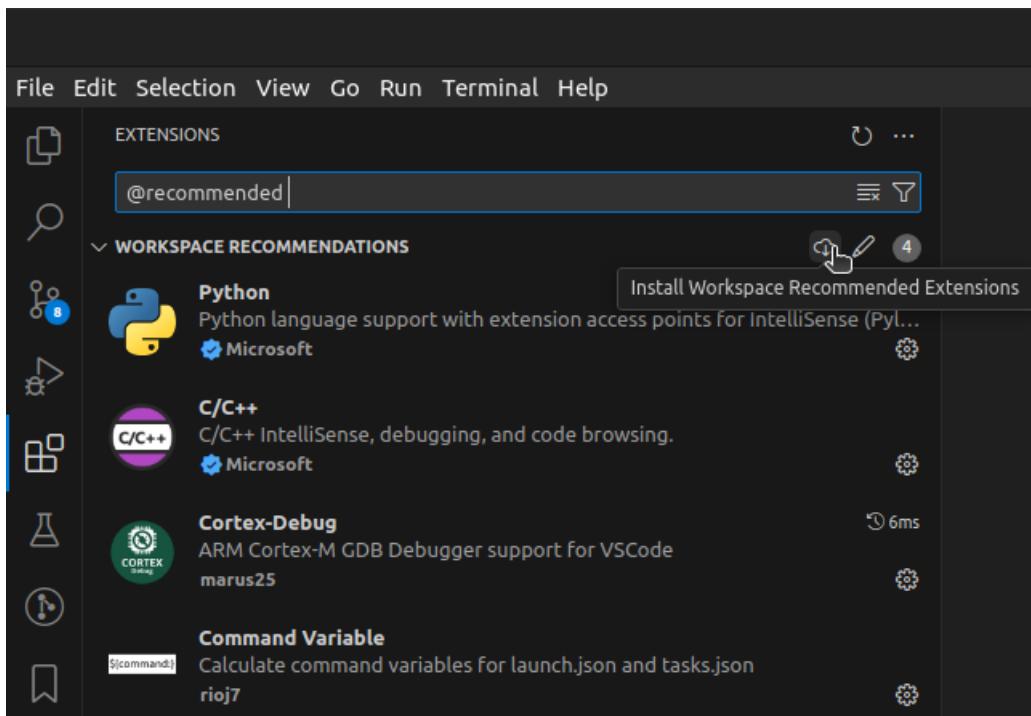


Figure 5.6: VS Code: Install all extensions.

If you wish to install extensions manually, here you can find the list of recommended extensions:

- **ms-vscode.cpptools** (C language support)
- **marus25.cortex-debug** (debugging firmware on target)
- **rioj7.command-variable** (automated tasks for building and debugging)
- **ms-python.python** (Python language support)

Note: We encourage you to use the latest version of VS Code and its extensions. However, if you encounter issues, please revert to the specific extension versions:

- ms-vscode.cpptools **v1.20.5**
- marus25.cortex-debug **v1.12.1**
- rioj7.command-variable **v1.63.0**
- ms-python.python **v2024.6.0**

For compatibility with these extensions, use Visual Studio Code version **1.89.1**.

5.3.1.4 Other

VS Code automatically finds and picks Python interpreter from virtual environment. However, if your global settings were changed for other projects, please make sure to pick the proper interpreter.

1. Open **Command Palette** (*View → Command Palette...*) or use the keyboard shortcut **Ctrl+Shift+P**.

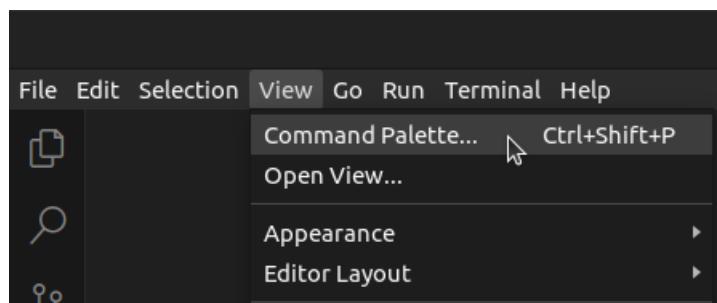


Figure 5.7: VS Code: Open command palette.

2. Write Python: Select interpreter.

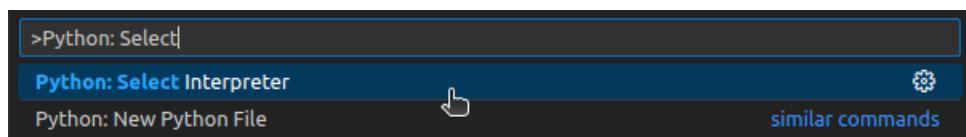


Figure 5.8: VS Code: Choose Python interpreter.

3. Choose path to your local env, e.g. `./.venv/bin/python`.

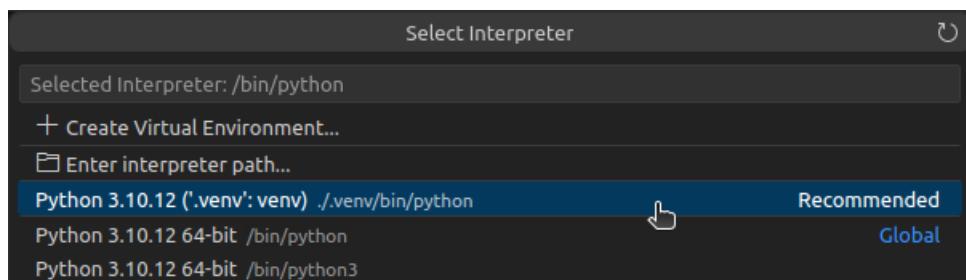


Figure 5.9: VS Code: Pick venv interpreter.

Warning: On Windows, the recommended and working Terminal Default Profile is PowerShell. Git Bash and other may not be working properly because of differences in path resolution.

1. Open **Command Palette** (*View → Command Palette...*) or use the keyboard shortcut **Ctrl+Shift+P**
2. Write Terminal: Select Default Profile and choose **PowerShell**.
3. Choose **PowerShell** from the list.

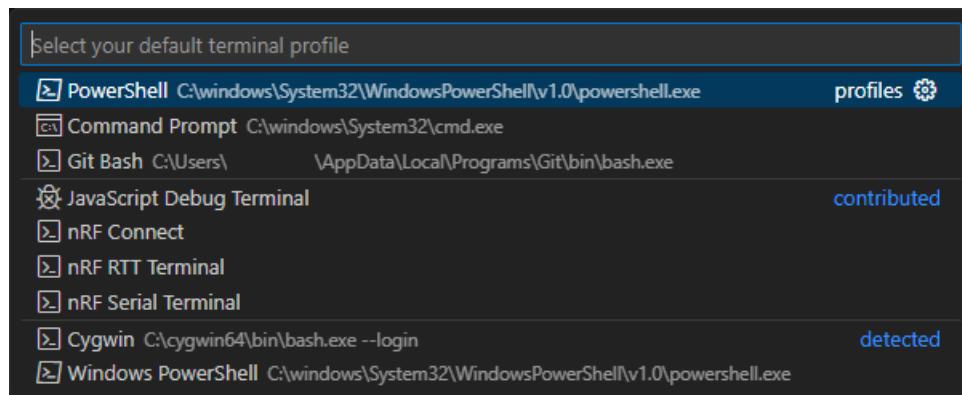


Figure 5.10: VS Code: Pick PowerShell on Windows.

5.3.2 Tasks

VS Code built-in tasks and launch configuration can be utilized to build, flash and debug the firmware. You can find short description of available tasks below:

Build:

- **Build the firmware** - build the firmware (compile only changed files).
- **Build the clean firmware** - rebuild the firmware (delete output files and compile everything again).
- **Flash the target** - flash the target with recently built firmware and reset the target.
- **Build & flash the target** - *Build the firmware + Flash the target*.

Launch:

- **Debug the firmware** - runs GDB sessions with target.
- **Build & debug the firmware** - builds firmware if needed and runs GDB sessions with target.

To provide a short list of tasks, we utilize **Command Variable** extension. This tool enables the passing of arguments between tasks and retains those arguments even after restarting VS Code.

5.3.2.1 Choose configuration

You can choose persistent build configuration which will be saved and used for all other tasks.

1. Open the **Run Build Task** dialog (*Terminal → Run Build Task...*) or use the keyboard shortcut **Ctrl+Shift+B**.

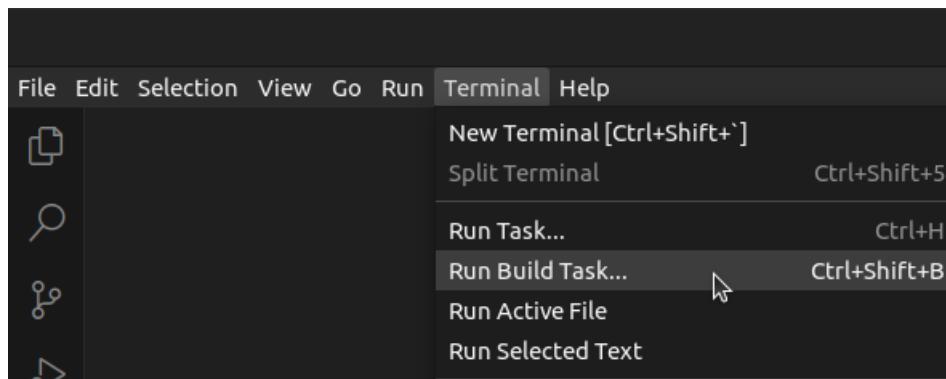


Figure 5.11: VS Code: Open build tasks.

2. Pick **Choose a configuration** and VS Code will prompt about possible configurations.

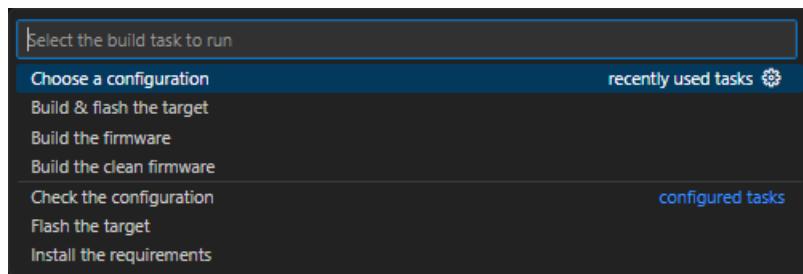


Figure 5.12: VS Code: Choose a configuration.

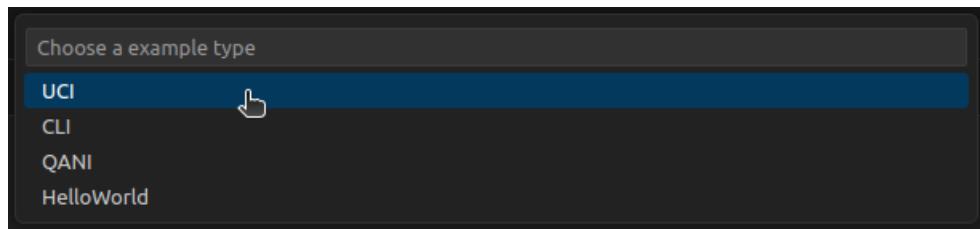


Figure 5.13: VS Code: Choose example configuration.

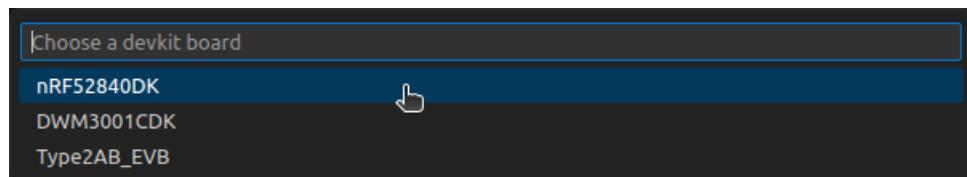


Figure 5.14: VS Code: Choose board configuration.

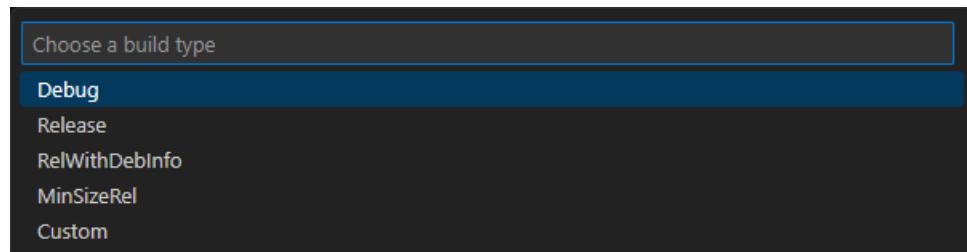


Figure 5.15: VS Code: Choose build configuration.

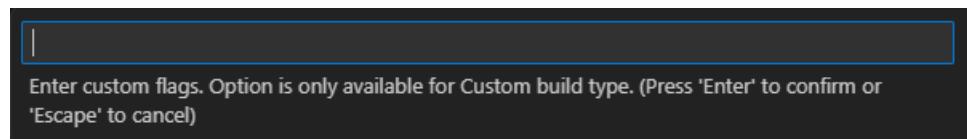


Figure 5.16: VS Code: Choose build flags configuration.

5.3.2.2 Build firmware

1. Open the Run Build Task dialog (*Terminal* → *Run Build Task...*) or use the keyboard shortcut **Ctrl+Shift+B**.
2. To build a target you can use **Build the firmware**.

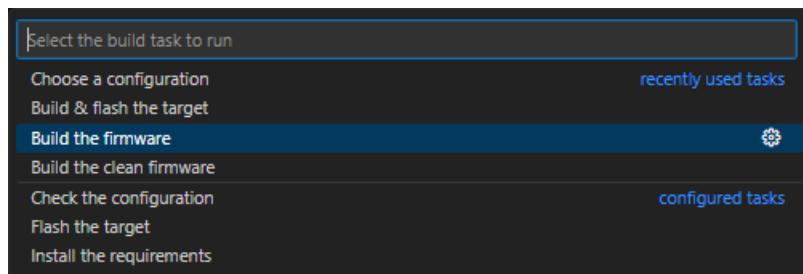


Figure 5.17: VS Code: Pick Build the firmware.

3. New terminal tab will appear, after compilation completes you can see following message.

```
[ 96%] Building C object EventManager/CMakeFiles/EventManager.dir/EventManager.c.obj
[ 97%] Building C object uci/CMakeFiles/uci.dir/uci transport/src/uci transport.c.obj
[ 97%] Building C object uci/CMakeFiles/uci.dir/uci parser.c.obj
[ 98%] Building C object uci/CMakeFiles/uci.dir/uwbtmac helper/src/uwbtmac helper dw3000.c.obj
[ 98%] Linking C static library libEventManager.a
[ 98%] Built target EventManager
[ 98%] Linking C static library libuci.a
[ 98%] Built target uci
[ 99%] Building C object CMakeFiles/nRF52840DK-DW3_QM33_SDK_UCI-FreeRTOS.elf.dir/home/qorvo/DW3_QM33_SDK 1.0.0/Sources/Projects/DW3_QM33_SDK/FreeRTOS/DW3_QM33_SDK-FreeRTOS-Common/hooks.c.obj
[ 99%] Building C object CMakeFiles/nRF52840DK-DW3_QM33_SDK_UCI-FreeRTOS.elf.dir/home/qorvo/DW3_QM33_SDK 1.0.0/Sources/Projects/DW3_QM33_SDK/FreeRTOS/DW3_QM33_SDK-FreeRTOS-Common/main uci.c.obj
[100%] Linking C executable nRF52840DK-DW3_QM33_SDK_UCI-FreeRTOS.elf
Memory region           Used Size Region Size %age Used
  RAM:      130816 B     256 KB   49.90%
  FLASH:    532716 B     1016 KB  51.20%
  CALIB SHA:    32 B       4 KB   0.78%
  CALIB:      4 KB       4 KB  100.00%
[100%] Built target nRF52840DK-DW3_QM33_SDK_UCI-FreeRTOS.elf
* Terminal will be reused by tasks, press any key to close it.
```

Figure 5.18: VS Code: Firmware build successfully.

4. Alternatively, you can pick **Build the clean firmware** to perform build from scratch.

5.3.2.3 Flash firmware

1. Make sure the firmware has been already built.
2. Open the Run Build Task dialog (*Terminal* → *Run Build Task...*) or use the keyboard shortcut **Ctrl+Shift+B**.
3. To flash the target, pick **Flash the target**.

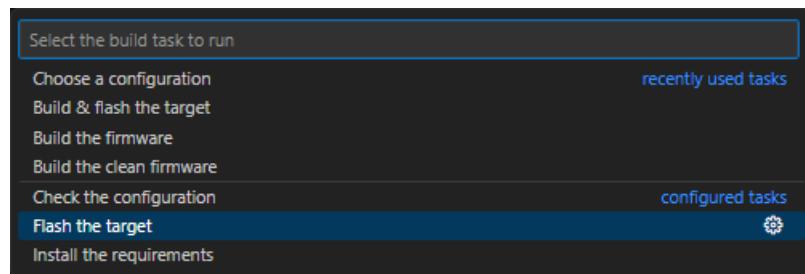


Figure 5.19: VS Code: Pick Flash the target.

4. If you have more than one development kit connected to your PC, you will be prompted to choose which one should be flashed.

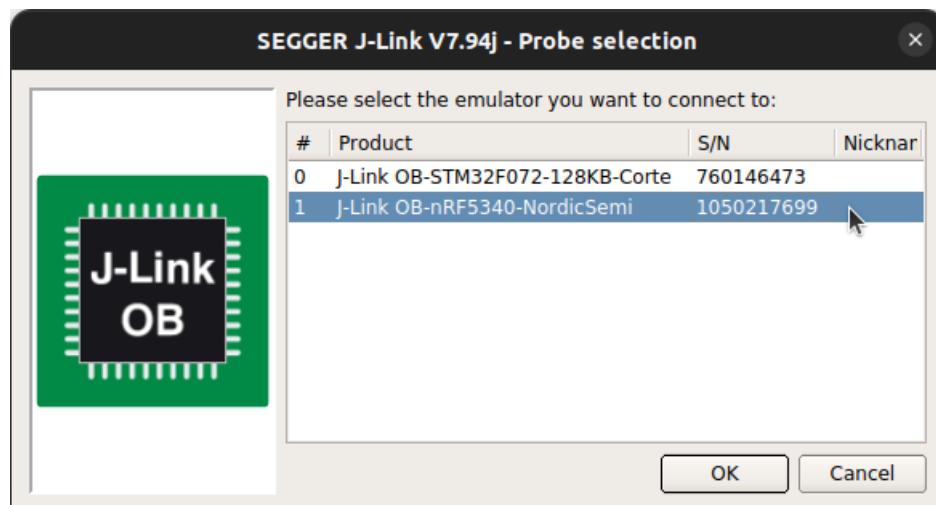


Figure 5.20: VS Code: Pick proper J-Link probe.

5. After this, the flashing progress bar will show up.

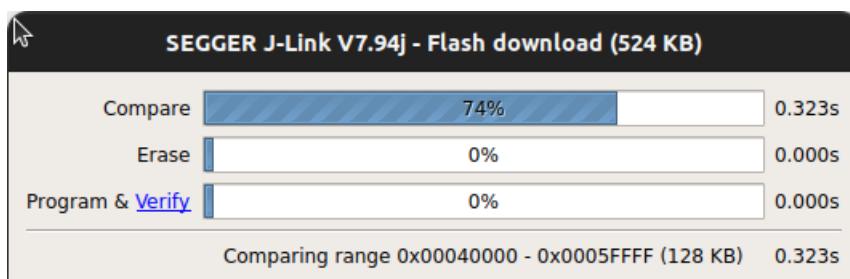
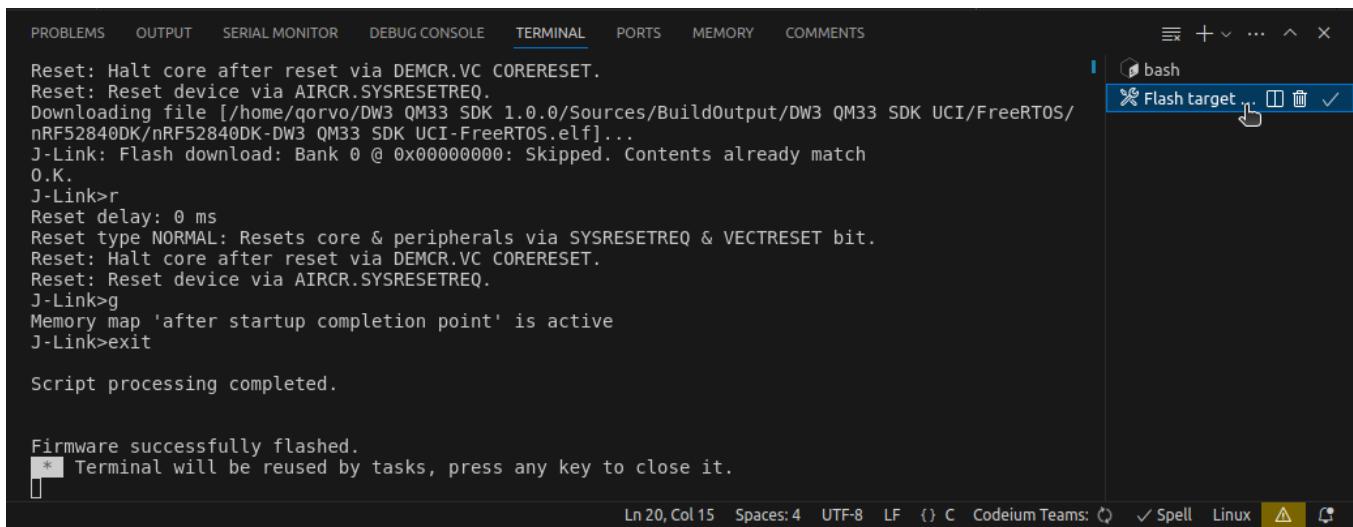


Figure 5.21: VS Code: Flashing progress window.

6. When flashing completed successfully, you will see following message in a dedicated terminal tab. Firmware will now start execution on the device.



PROBLEMS OUTPUT SERIAL MONITOR DEBUG CONSOLE TERMINAL PORTS MEMORY COMMENTS

```

Reset: Halt core after reset via DEMCR.VC CORERESET.
Reset: Reset device via AIRCR.SYSRESETREQ.
Downloading file [/home/qorvo/DW3 QM33 SDK 1.0.0/Sources/BuildOutput/DW3 QM33 SDK UCI/FreeRTOS/nRF52840DK/nRF52840DK-DW3 QM33 SDK UCI-FreeRTOS.elf]...
J-Link: Flash download: Bank 0 @ 0x00000000: Skipped. Contents already match
O.K.
J-Link>r
Reset delay: 0 ms
Reset type NORMAL: Resets core & peripherals via SYSRESETREQ & VECTRESET bit.
Reset: Halt core after reset via DEMCR.VC CORERESET.
Reset: Reset device via AIRCR.SYSRESETREQ.
J-Link>g
Memory map 'after startup completion point' is active
J-Link>exit

Script processing completed.

Firmware successfully flashed.
* Terminal will be reused by tasks, press any key to close it.

```

Ln 20, Col 15 Spaces: 4 UTF-8 LF {} C Codeium Teams: Spell Linux ▲ 🔍

Figure 5.22: VS Code: Flushing succeeded.

7. Alternatively, you can pick **Build & Flash the target** to Build the firmware before flashing.

5.3.2.4 Debug firmware

1. To debug firmware, open **Run and Debug** tab on the left panel, or use the keyboard shortcut **Ctrl+Shift+D**

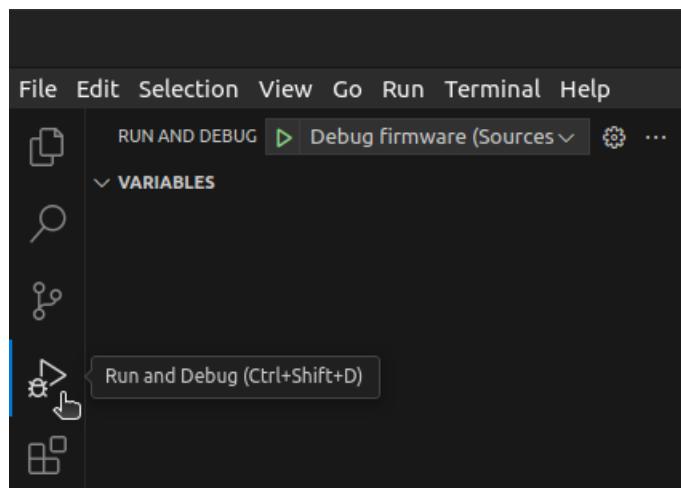


Figure 5.23: VS Code: Open Run and Debug tab.

2. Choose build configuration (If you pick **Debug the firmware**, make sure the firmware has been already built).

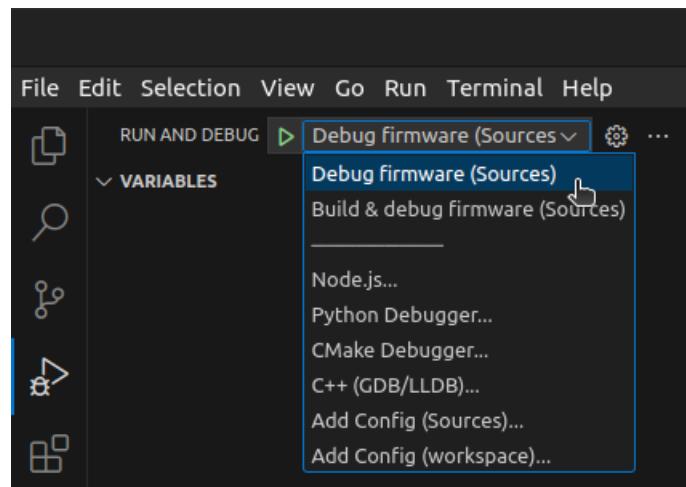


Figure 5.24: VS Code: Pick debugging profile.

3. Start debug session by clicking green *Play* button or use the keyboard shortcut F5

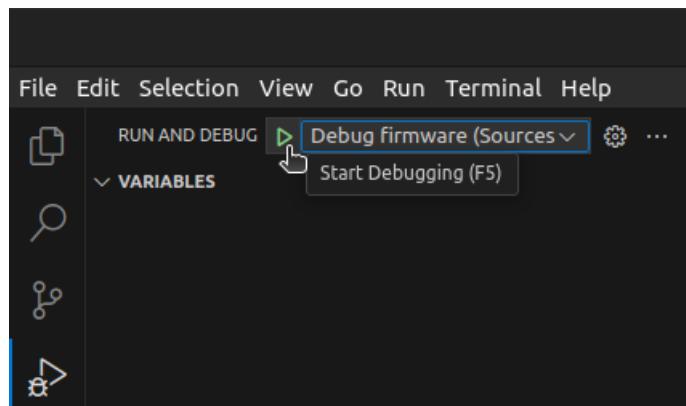


Figure 5.25: VS Code: Start debug session.

4. Debug session will start. Now you can add breakpoints, variables to watch and execute code step-by-step.

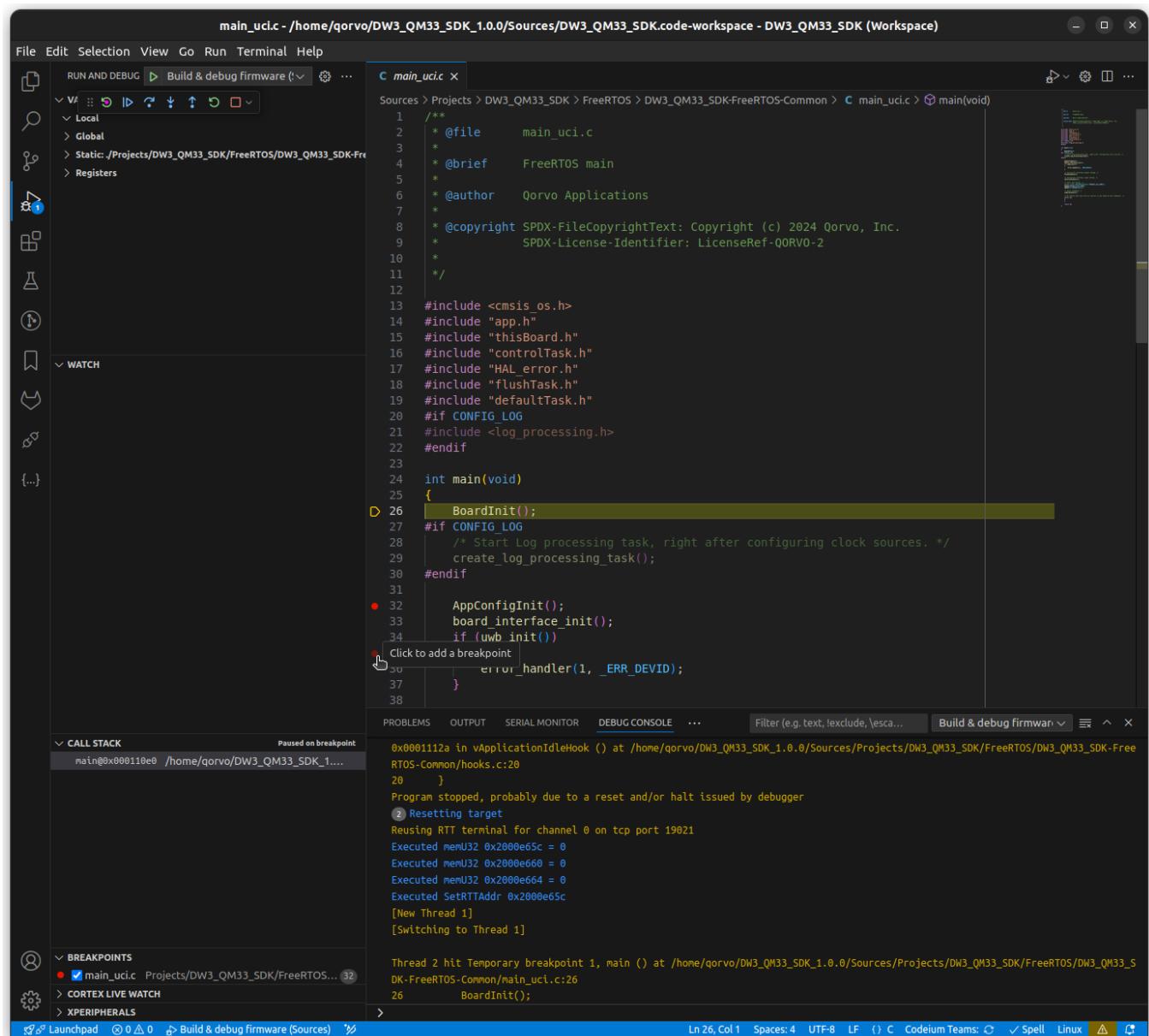


Figure 5.26: VS Code: Window with active debug session.

5. In Terminal tab you can also find RTT console, where you can access logs from the device.

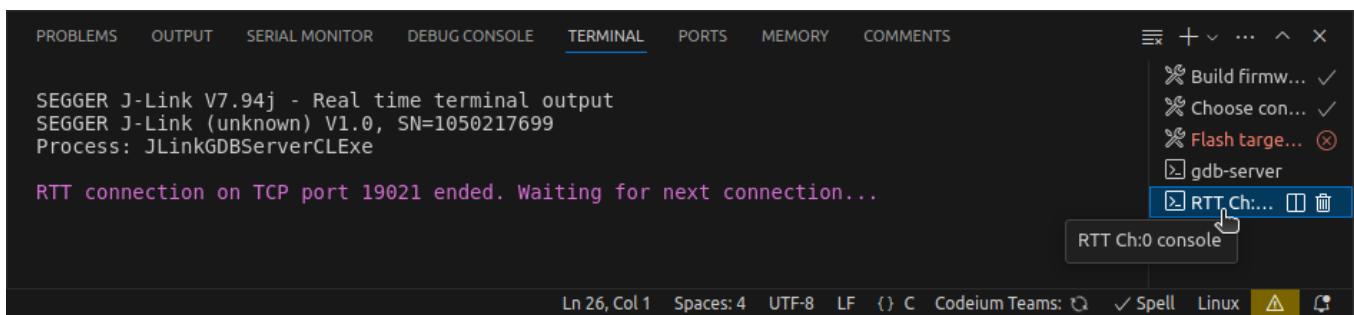


Figure 5.27: VS Code: RTT console.

5.3.2.5 Predefined tasks

If the generic tasks are not suitable or if you encounter issues with the **Command Variable** extension, you can define your own set of predefined tasks. To maintain the same end-functionality, you will need to prepare the appropriate task for your development kit board, example, and build type from a comprehensive list of possible combinations. This ensures that you have the correct configuration for your specific use case.

Setup Instructions

To create your own set of predefined tasks, you will need to remove original content of .vscode directory and set up a new one. This directory should contain the following two files:

1. .vscode/launch.json

This file defines the configurations for launching and debugging your application.

Example content:

```
{
  "version": "0.2.0",
  "configurations": [
    ////////////// DWM3001CDK ///////////
    {
      "name": "UCI | DWM3001CDK | Debug",
      "cwd": "${workspaceRoot}",
      "executable": "BuildOutput/UCI/FreeRTOS/DWM3001CDK/Debug/DWM3001CDK-UCI-FreeRTOS.elf",
      "rtos": "FreeRTOS",
      "request": "launch",
      "type": "cortex-debug",
      "serverType": "jlink",
      "device": "nrf52833",
      "preLaunchTask": "Build UCI | DWM3001CDK | Debug",
      "svdFile": "${workspaceRoot}/SDK_BSP/Nordic/SDK_17_1_0/modules/nrfx/mdk/nrf52833.svd",
      "interface": "swd",
      "runToEntryPoint": "main",
      "presentation": {"hidden": false, "group": "DWM3001CDK"},
      "linux": {
        "serverpath": "/usr/bin/JLinkGDBServerCLExe",
        "armToolchainPath": "/opt/gcc/gcc-arm-none-eabi-10.3-2021.10/bin",
        "gdbPath": "/opt/gcc/gcc-arm-none-eabi-10.3-2021.10/bin/arm-none-eabi-gdb"
      },
      "windows": {
        "serverpath": "C:/Program Files/SEGGER/JLink/JLinkGDBServerCL.exe",
        "armToolchainPath": "C:/GnuToolsArmForEmbedded/gcc-arm-none-eabi-10.3-2021.10/bin",
      },
      "serverArgs": ["-gui"],
      "rttConfig": {
        "enabled": true,
        "address": "auto",
        "decoders": [
          {
            "label": "",
            "port": 0,
            "type": "console"
          }
        ]
      }
    }
  ]
}
```

(continues on next page)

(continued from previous page)

```
    ]
}
```

2. .vscode/tasks.json

This file specifies the tasks for building, flashing, and other operations.

Example content:

```
{
  "version": "2.0.0",
  "tasks": [
    ////////////////////////////////////////////////////////////////// DWM3001CDK ///////////////
    {
      "label": "Build UCI | DWM3001CDK | Debug",
      "type": "shell",
      "command": "make all V=1 -j",
      "dependsOn": "_create UCI DWM3001CDK",
      "group": {
        "kind": "build",
        "isDefault": true
      },
      "options": {
        "cwd": "${workspaceRoot}/BuildOutput/UCI/FreeRTOS/DWM3001CDK/Debug"
      },
      "problemMatcher": [
        "$gcc"
      ],
      "presentation": {
        "group": "build"
      }
    },
    {
      "label": "Build (clean) UCI | DWM3001CDK | Debug",
      "type": "shell",
      "command": "make all -j",
      "dependsOn": "_force create UCI | DWM3001CDK | Debug",
      "group": {
        "kind": "build",
        "isDefault": true
      },
      "options": {
        "cwd": "${workspaceRoot}/BuildOutput/UCI/FreeRTOS/DWM3001CDK/Debug"
      },
      "problemMatcher": [
        "$gcc"
      ],
      "presentation": {
        "group": "build"
      }
    },
    {
      "label": "_create UCI | DWM3001CDK | Debug",
      "type": "shell",
      "group": {
        "kind": "build"
      }
    }
  ]
}
```

(continues on next page)

(continued from previous page)

```

},
"windows": {
    "command": "python ${workspaceRoot}/Projects/FreeRTOS/UCI/DWM3001CDK/CreateTarget.
→py -no-force -build Debug",
},
"linux": {
    "command": "python3 ${workspaceRoot}/Projects/FreeRTOS/UCI/DWM3001CDK/CreateTarget.
→py -no-force -build Debug",
},
"hide": true,
"presentation": {
    "group": "build",
    "clear": true
}
},
{
    "label": "_force create UCI | DWM3001CDK | Debug",
    "type": "shell",
    "group": {
        "kind": "build"
    },
    "windows": {
        "command": "python ${workspaceRoot}/Projects/FreeRTOS/UCI/DWM3001CDK/CreateTarget.
→py -build Debug",
    },
    "linux": {
        "command": "python3 ${workspaceRoot}/Projects/FreeRTOS/UCI/DWM3001CDK/CreateTarget.
→py -build Debug",
    },
    "hide": true,
    "presentation": {
        "group": "build",
        "clear": true
    }
},
{
    "label": "Flash UCI | DWM3001CDK | Debug",
    "type": "shell",
    "windows": {
        "command": "python flash_target.py ${workspaceRoot}/BuildOutput/UCI/FreeRTOS/
→DWM3001CDK/Debug/DWM3001CDK-UCI-FreeRTOS.elf nrf52833",
    },
    "linux": {
        "command": "python3 flash_target.py ${workspaceRoot}/BuildOutput/UCI/FreeRTOS/
→DWM3001CDK/Debug/DWM3001CDK-UCI-FreeRTOS.elf nrf52833",
    },
    "group": {
        "kind": "build",
        "isDefault": true
    },
    "options": {
        "cwd": "${workspaceRoot}/.vscode/scripts"
    },
    "presentation": {

```

(continues on next page)

(continued from previous page)

```

    "group": "flash",
    "clear": true
}
]
}
}

```

The following example demonstrates how to set up the `.vscode/launch.json` and `.vscode/tasks.json` files to build, flash, and debug the *Debug build of the UCI example on the DWM3001CDK board*. You can extend and modify these files to support multiple examples, boards, and build types.

You can customize these files to suit your needs by adding support for multiple examples, boards, and build types. This flexibility allows you to tailor the development environment to your specific requirements.

5.4 Standalone building and flashing

5.4.1 Building SDK Firmware

5.4.1.1 Create a Target

`CreateTarget.py` automates the process of generating a target for a project using common settings. It calls CMake with the appropriate options to configure the build environment and generate the target.

Warning: Make sure your Python virtual environment is activated before running this script.

5.4.1.1.1 Usage

Navigate to the project folder `SDK/Firmware/Projects/FreeRTOS/<example>/<board_name>` and execute:

- On Linux:

```
./CreateTarget.py [-no-force]
```

By default, the execution permission may not be set to the file. If it is the case, run the following command:

```
chmod +x ./CreateTarget.py
```

- On Windows:

```
python ./CreateTarget.py [-no-force]
```

5.4.1.1.2 Options

- `-no-force`: Optional flag to prevent the removal of the old build folder.

5.4.1.1.3 Example

Navigate to the project folder `SDK/Firmware/Projects/FreeRTOS/<example>/<board_name>` and execute:

- On Linux:

```
./CreateTarget.py
```

- On Windows:

```
python ./CreateTarget.py
```

5.4.1.1.4 Help output

```
./CreateTarget.py -h
usage: CreateTarget.py [-h] [-no-force] [-build {Debug,Release,RelWithDebInfo,MinSizeRel,Custom}
                      [-custom-flags CUSTOM_FLAGS]

Build specified target

options:
-h, --help            show this help message and exit
-no-force             Do not remove old build folder
-build {Debug,Release,RelWithDebInfo,MinSizeRel,Custom}
                      Specify the build type. Choices are 'Debug', 'Release', 'RelWithDebInfo',
                      'MinSizeRel', 'Custom' (default: Debug). If 'Custom' is selected, no value will be
                      assigned to
                      CMAKE_BUILD_TYPE, allowing customization with -custom-flags option.
-custom-flags CUSTOM_FLAGS
                      Custom flags for the build, only available with -build 'Custom'.
Example: '-O2 -g -DDEBUG'
```

5.4.1.2 Build the Firmware

This step compiles the firmware, producing output files in three formats: `.hex`, `.bin`, and `.elf`.

5.4.1.2.1 Usage

Navigate to the project folder `SDK/Firmware/BuildOutput/<example>/FreeRTOS/<board_name>/<build>` and execute:

```
make -j
```

5.4.1.2.2 Examples

- compile CLI build for QM33120WDK1:

```
cd BuildOutput/CLI/FreeRTOS/nRF52840DK
make -j
```

- compile UCI build for DWM30001CDK:

```
cd BuildOutput/UCI/FreeRTOS/DWM3001CDK
make -j
```

Note: To speed up the building process, we use the `-j` flag so all CPU cores will be utilized to build the firmware. You can specify to use e.g. only 4 cores with `make -j4`.

5.4.2 Flashing the development kit

There is a wide variety of tools which can be used to flash the target. Below is an example demonstrating how to use the JFLashLite tool.

Warning: Type 2AB EVB boards are not shipped preprogrammed. UCI firmware needs to be flashed to continue.

- To flash the board, it must be powered on by connecting a micro-USB cable to the board (or via external power depending on jumpers configuration). Refer to [muRata® UWB Module\(2AB\) EVB User Manual¹⁵](#) if needed.

For flashing or debugging, connect the JTAG connector to a J-Link Debug Probe by SEGGER® as follow:

VTref	1 ● ● 2	NC
Not used	3 ● ● 4	GND
Not used	5 ● ● 6	GND
SWDIO	7 ● ● 8	GND
SWCLK	9 ● ● 10	GND
Not used	11 ● ● 12	GND
SWO	13 ● ● 14	GND*
RESET	15 ● ● 16	GND*
Not used	17 ● ● 18	GND*
5V-Supply	19 ● ● 20	GND*

Figure 5.28: J-Link SWD Connector Pinout

¹⁵ <https://www.murata.com/en-us/products/connectivitymodule/ultra-wide-band/qorvo#datasheet-and-documentations>

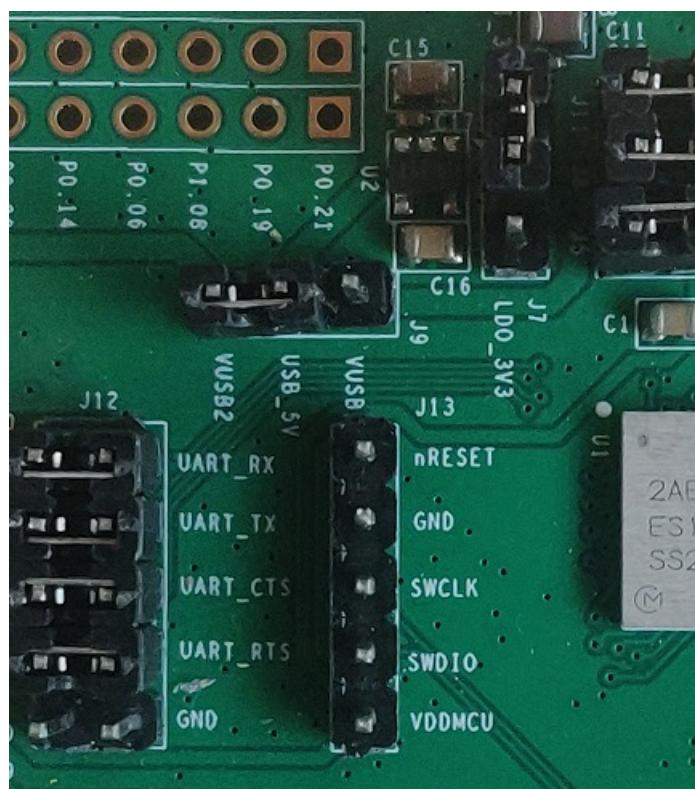


Figure 5.29: Type 2AB EVB SWD Connector

Following signal names from previous pictures, here is the connections to do:

J-Link	Type 2AB EVB
RESET	nRESET
GND	GND
SWCLK	SWCLK
SWDIO	SWDIO
VTref	VDDMCU

2. Locate the JFLashLite tool in your installation directory (e.g. C:\Program_Files\SEGGER\FlashLite.exe in Windows or /usr/bin/JFlashLite in Ubuntu) and launch the application.

Note: When you connect the development kit for the first time, you may be prompted to update firmware of the on-board programmer. Please approve the update by clicking **OK** and wait for the update to complete.

3. Upon starting JFlashLite, the Device and Interface selection dialog will appear. Select **NRF52840_xxAA** device by clicking on “...” button. On the right side of dialog window, select **SWD** Interface with clock speed of **4000 kHz**. Click **OK** to proceed.



Figure 5.30: Select CPU settings.

4. Next, select the firmware file you wish to flash onto the board. Click the “...” button to browse for the firmware (e.g. SDK/Binaries/Type2AB_EVB/Type2AB_EVB-UCI-FreeRTOS.hex).

After selecting the file, click **Program Device** to start the flashing process.

Note: If the board was previously flashed with a different firmware version, it is recommended to erase the flash memory before programming the new firmware. To perform this, please click **Erase Chip** button and then proceed with flashing the new firmware.

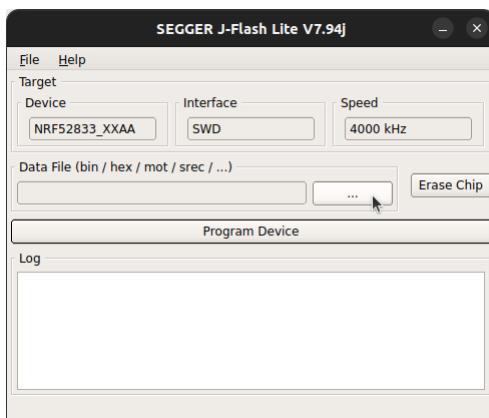


Figure 5.31: Select .hex file to flash.

5. Once the progress bar is completed, the device has been successfully flashed. Perform a power cycle by disconnecting and reconnecting the power supply to reset the board. Your device should be now ready to use.

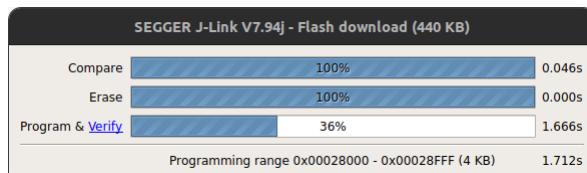


Figure 5.32: Programming progress window.

6 Projects

This SDK provides a set of projects to help you get started with development. Each project includes comprehensive details covering its overview, requirements, source code structure, expected outputs, and step-by-step instructions. For complete information on each project, please refer to the associated README file or chapter in this document:

- **Hello World:**

- Demonstrates a basic interaction with the UWB transceiver chip using the Qorvo UWB stack.
- Key functionalities:
 - * Initializes the UWB stack.
 - * Retrieves and displays device information.
 - * Transmits information over RTT.
- For more details and setup instructions, refer to [SDK/Firmware/Projects/FreeRTOS/HelloWorld/README.md](#).

- **CLI:**

- Demonstrates a basic functionality of the uwb-stack

- Key functionalities:

- Run basic FiRa TWR applications.
- Run UWB sniffer.
- Get basic information about the device.
- Perform configuration and calibration of the device.

- For more details and setup instructions, refer to [CLI - Command Line Interface](#).

- **UCI:**

- Provides a way to fully control FiRa uwb-stack over UCI interface.

- Key functionalities:

- Run FiRa TWR sessions.
- Perform configuration and calibration of the device.

- For more details and setup instructions, refer to [UCI - UWB Command Interface](#).

- **QANI:**

- Implements the **QANI** (Qorvo Nearby Interaction) to enable proximity interaction using Apple Nearby Interaction protocol. This serves as an example of third-party accessory device that interacts with compatible Apple devices through **BLE** and **UWB** technologies.

- Key functionalities:

- Initializes the Nearby Interaction and BLE stack.
- Configures FiRa session parameters out-of-band and initiates ranging.

- * Pairs with an iPhone to perform Two-Way Ranging (TWR).
- * Displays real-time distance, azimuth, and elevation data.
- For more details and setup instructions, refer to [SDK/Firmware/Projects/FreeRTOS/QANI/README.md](#).

Referencing these README files will guide you through the setup and compilation for each example, as well as provide insights into expected behaviors.

7 CLI - Command Line Interface

7.1 Overview

CLI stands for Command Line Interface.

CLI build provides an easy way to send and receive commands to test the different features offered by the software. This section describes how to use the different CLI commands.

7.2 Setup

Note: See for details *Board Connections*

Open a Serial Terminal (example using [Tera Term Open Source Project¹⁶](#)) and set “Serial Port” as follow:

- Baud rate: 115200
- Number of data bits: 8
- Parity: None
- Number of stop bits: 1
- Flow control: None

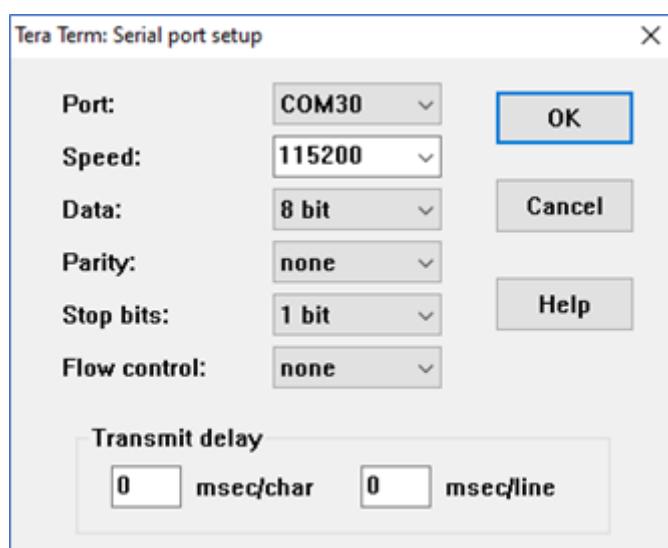


Figure 7.1: Com serial port settings

¹⁶ <https://teratermproject.github.io/index-en.html>

To use the numpad operation keys in Tera Term, go to Setup -> Load Key Map... and select FUNCTION.CNF.

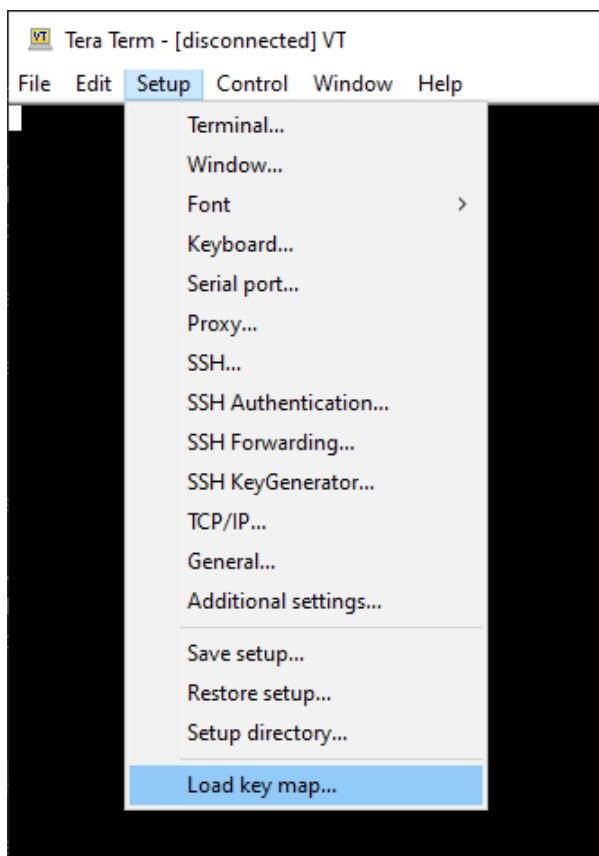


Figure 7.2: Load Key Map option

Warning: Before starting FiRa applications, please check [Calibration important notes](#).

7.3 Anytime Commands

7.3.1 Overview

Commands listed in table below can be executed at any time.

Table 7.1: Anytime Commands

Command	Description
HELP	Outputs a list of all known commands available in the current mode of operation or shows the help of the command <CMD>. Equivalent shortcut is “?”.
STAT	Reports the status. Gives a dump of software version info, configuration values and the current operation mode.
STOP	Stops running any of top-level applications and places the device in the NONE mode, where only core tasks are running.
THREAD	Reports information about running threads and memory consumption.

Application or setting commands will return “ok” if the command was properly executed.

7.3.2 HELP

This command displays the help information.

Command:

```
HELP
```

or

```
?
```

Response:

```
nRF52840DK - DW3_QM33_SDK - FreeRTOS

---      Anytime commands      ---
HELP      ?      STOP      THREAD
STAT

---      Application selection    ---
LISTENER  RESPF      INITF

---      IDLE time commands      ---
UART      CALKEY      LISTCAL

---      Service commands      ---
RESTORE   DIAG      LCFG      DECAID
SAVE      SETAPP      GETOTP

---      LISTENER Options      ---
LSTAT
```

Note: This command may also take any command as a parameter. It allows you to display its description. Examples will follow.

Command:

```
HELP INITF
```

Response:

```
INITF:
INITF [Option1] [Option2] ...
Options: (only default values are shown, check the SDK Manual to know more about the available
→configurations)
-CHAN=9      --> Channel number
-PRFSET=BPRF4 --> PRF set
-PCODE=10     --> Preamble code index
-SLOT=2400    --> Slot duration [RSTU]
-BLOCK=200    --> Block duration [ms]
-ROUND=25    --> Round duration [slots]
-RRU=DSTWR    --> Ranging round usage
-ID=42      --> Session ID
-VUPPER=01:02:03:04:05:06:07:08  --> vUpper64
-MULTI       --> Activate one-to-many mode
```

(continues on next page)

(continued from previous page)

```

-HOP          --> Activate round hopping
-ADDR=0      --> Device own address (Initiator address)
-PADDR=1      --> 1st Responder address
               or to set multiple responders:
-PADDR=[1,2,...,n] --> to set n Responder addresses (for one-to-many)

```

Command:

HELP RESPF

Response:

```

RESPF:
RESPF [Option1] [Option2] ...
Options: (only default values are shown, check the SDK Manual to know more about the available
         configurations)
-CHAN=9        --> Channel number
-PRFSET=BPRF4 --> PRF set
-PCODE=10      --> Preamble code index
-SLOT=2400     --> Slot duration [RSTU]
-BLOCK=200     --> Block duration [ms]
-ROUND=25      --> Round duration [slots]
-RRU=DSTWR    --> Ranging round usage
-ID=42         --> Session ID
-VUPPER=01:02:03:04:05:06:07:08 --> vUpper64
-MULTI         --> Activate one-to-many mode
-HOP           --> Activate round hopping
-ADDR=1        --> Device own address (Responder address)
-PADDR=0        --> Peer address (Initiator address)

```

7.3.3 STAT

This command reports the status. It gives a dump of software version information, supported applications and the current operation mode.

Command:

STAT

Response:

```

MODE: NONE
JS0108{"Info": {
"Device": "nRF52840DK - DW3_QM33_SDK - FreeRTOS",
"Current App": "NONE",
"Version": "1.0.0",
"Build": "May 30 2024 08:38:01",
"Apps": ["LISTENER", "RESPF", "INITF"],
"Driver": "DW3XXX Device Driver Version 08.00.26",
"UWB stack": "R12.7.0-227-gf8c9f0f27"}}

```

7.3.4 STOP

It stops running any of top-level applications and places the device in the NONE mode, where only core tasks are running.

Command:

STOP

7.3.5 THREAD

This command reports information about running threads and their stack usage.

For each Thread, there is a ratio displayed in the Stack usage column.

- The 1st value is the maximum stack depth reached during runtime.
- The 2nd value is the maximum stack size allocated to the thread.

Command:

THREAD

Response:

THREAD NAME	Stack usage
Control	960/2048
IDLE	164/512
Flush	176/512
Logger	100/768
Default	172/4304
Tmr Svc	156/512
Total HEAP	51200
Current HEAP used	8544
Max HEAP used	8544

7.4 Service Commands

Service commands can only be executed in the NONE mode when no applications are running.

Table 7.2: Service Commands

Command	Description
RESTORE	Restores the default configuration of CLI applications and L1 calibration keys.
LCFG	Sets/Gets a list of configuration parameters and values for LISTENER application.
DIAG	Enables diagnostic mode.
DECAID	Reports information about the chip.
SAVE	Saves default running application and all the configuration parameters to NVM.
SETAPP	Sets the default application that starts automatically after reboot or power cycle.
GETOTP	Reads the content of OTP memory.

7.4.1 RESTORE

This command restores the default configuration of CLI applications and L1 calibration keys.

The restored parameters are automatically saved and written to NVM.

Command:

```
RESTORE
```

Warning: Please see *Calibration important notes*.

7.4.2 LCFG

Note: This command only concerns LISTENER application

Sets/Gets a list of configuration parameters and values for LISTENER application.

To see LISTENER configuration parameters, use “LCFG”.

Command:

```
LCFG
```

Response:

```
JS00C1{"LCFG PARAM":{  
    "CHAN":9,  
    "PAC":8,  
    "PCODE":10,  
    "SFDTYPE":3,  
    "DRATE":6810,  
    "PHRMODE":0,  
    "PHRRATE":0,  
    "STSMODE":0,  
    "STSLEN":64,  
    "PDOAMODE":1,  
    "XTALTRIM":46}}
```

To set LISTENER configuration parameters, use “LCFG <List of parameters>” with parameters listed in Table *Parameters list and default values..*

Command:

```
LCFG -PCODE=9
```

Response:

```
JS00C2{"LCFG PARAM":{  
    "CHAN":9,  
    "PAC":8,  
    "PCODE":9,  
    "SFDTYPE":3,  
    "DRATE":6810,  
    "PHRMODE":0,  
    "PHRRATE":0,  
    "STSMODE":0,  
    "STSLEN":64,  
    "PDOAMODE":1,  
    "XTALTRIM":46}}
```

(continues on next page)

(continued from previous page)

```
"PHRRATE":0,
"STSMODE":0,
"STSLEN":64,
"PDOAMODE":1,
"XTALTRIM":46}}
```

Table 7.3: Parameters list and default values

Order	Parameter	Default value	Description
1	CHAN	9	Channel selection [5, 9]
2	PAC	8	Preamble Acquisition Chunk size [4 or 8 when RX Preamble Length is <= 128, 16 for 256, 32 when higher]
3	PCODE	10	Preamble Code [9, 10, 11, 12]
4	SFDTYPE	3	Start Frame Delimiter [0 = 4A 8-bit, 1 = DW 8-bit, 2 = DW 16-bit, 3 = 4z 8-bit]
5	DRATE	6810	PSDU Data Rate in kbps [850, 6810]
6	PHRMODE	0	PHR Mode [0 - Standard, 1 - DW Extended]
7	PHRRATE	0	PHR Data Rate [0 - Standard, 1 - PHR at data rate]
8	STSMODE	0	Scrambled Time Sequence Mode [0 - off, 1 - STS mode 1, 2 - STS mode 2, 3 - STS with no data, 8 - STS mode SDC]
9	STSLEN	64	STS Length (power of 2) [32, . . , 2048]
10	PDOAMODE	1	PDoA Mode [0 - off, 1 - PDOA mode 1, 3 - PDOA mode 3]
11	XTALTRIM	46	Crystal oscillator trimming value [0, . . , 63]

7.4.3 DIAG

Reports diagnostics status or changes its value.

DIAG provides a way to display additional information about the performance while ranging. By default, it is set to 0.

To see diagnostics status, use “DIAG”.

Command:

```
DIAG
```

Response:

```
DIAG: 0
```

To set diagnostics status, use “DIAG <status>”,

- where <status> value:
 - 0: disables diagnostics,
 - 1: enables diagnostics.

Command:

```
DIAG 1
```

With DIAG set to 1, it allows to display RSSI for the whole ranging round and many more information about every frame sent/received.

RSSI calculations make use of IPATOV, STS1, STS2, and CIR (Channel Impulse Response) parameters. In particular, three points of the first path amplitude magnitude, the number of symbols accumulated, and the CIR power value. Details on RSSI can be found in the DW3000 User Manual (Diagnostic APIs sections).

Output when diagnostics enabled:

```
SESSION_INFO_NTF: {session_handle=1, sequence_number=263, block_index=263, n_measurements=1
[mac_address=0x0001, status="SUCCESS", distance[cm]=224, rmt_az=4.78, RSSI[dBm]=-73.5]}
RANGE_DIAGNOSTICS_NTF: {n_reports=6
[msg_id=CONTROL, action=TX, antenna_set=0, frame_status={SUCCESS: 1, WIFI_COEX: 0, GRANT_
↪DURATION_EXCEEDED: 0}, cfo_present=0, nb_aoa=0];
[msg_id=RANGING_INITIATION, action=TX, antenna_set=0, frame_status={SUCCESS: 1, WIFI_COEX: 0,_
↪GRANT_DURATION_EXCEEDED: 0}, cfo_present=0, nb_aoa=0];
[msg_id=RANGING_RESPONSE, action=RX, antenna_set=0, frame_status={SUCCESS: 1, WIFI_COEX: 0,_
↪GRANT_DURATION_EXCEEDED: 0}, cfo_present=1, cfo_ppm=0.48, nb_aoa=0];
[msg_id=RANGING_FINAL, action=TX, antenna_set=0, frame_status={SUCCESS: 1, WIFI_COEX: 0, GRANT_
↪DURATION_EXCEEDED: 0}, cfo_present=0, nb_aoa=0];
[msg_id=MEASUREMENT_REPORT, action=TX, antenna_set=0, frame_status={SUCCESS: 1, WIFI_COEX: 0,_
↪GRANT_DURATION_EXCEEDED: 0}, cfo_present=0, nb_aoa=0];
[msg_id=RESULT_REPORT, action=RX, antenna_set=0, frame_status={SUCCESS: 1, WIFI_COEX: 0, GRANT_
↪DURATION_EXCEEDED: 0}, cfo_present=1, cfo_ppm=0.36, nb_aoa=0]}
```

7.4.4 DECAID

This command reports information about the chip.

Note: Qorvo SoC ID is defined as a concatenation of Qorvo Lot ID and Qorvo Part ID (in that order).

Command:

```
DECAID
```

Response:

```
Qorvo Device ID = 0xdeca0304
Qorvo Lot ID = 0x0000503639463438
Qorvo Part ID = 0x8124d5b7
Qorvo SoC ID = 00005036394634388124d5b7
```

7.4.5 SAVE

This command saves configuration parameters of CLI applications to NVM (Non-Volatile Memory). The saved parameters will stay after reboot or power cycle therefore making this ideal for battery powered configuration options.

Note: The SAVE command cannot be used during a ranging session.

Note: The default FiRa parameters could be changed manually in the common_fira.c or driver_app_config.c, if desired.

Command:

SAVE

This command can be used to run FiRa applications (INITF/RESPF) without providing parameters again.

Commands:

```
INITF -BLOCK=400
STOP
SAVE

INITF
<runs with BLOCK=400>
```

7.4.6 SETAPP

This command allows configuration of the default application that starts automatically after reboot or power cycle.

Note: Use SAVE command after SETAPP to store default application in NVM.

To check the default application, use “SETAPP”.

Command:

```
SETAPP
```

Response:

```
SETAPP: NONE
```

To set the default application, use “SETAPP <app>”,

- where <app> value:
 - LISTENER: UWB Sniffer,
 - INITF: Initiator in a FiRa two-way ranging session,
 - RESPF: Responder in a FiRa two-way ranging session,
 - NONE: No default application (IDLE state as default).

Command:

```
SETAPP RESPF
```

7.4.7 GETOTP

Reads the content of OTP memory.

Command:

```
GETOTP
```

Response:

```
OTP CONTENT: {
    "0x000": "0x00000000",
    "0x001": "0x00000000",
    "0x002": "0x00000000",
    "0x003": "0x00000000",
    "0x004": "0x00000001",
    "0x005": "0x06867777",
    "0x006": "0x8124d5b7",
    "0x007": "0x00000000",
    "0x008": "0x01758d40",
    "0x009": "0x0000008d",
    ...
    "0x07d": "0x00000000",
    "0x07e": "0x00000000",
    "0x07f": "0x00000000",
}
```

7.5 IDLE time Commands

7.5.1 Overview

IDLE time commands listed in table below can be executed only when the mode is NONE, corresponding to IDLE state.

Table 7.4: IDLE time Commands

Command	Description
UART	Enables selected UART.
CALKEY	Sets or gets a calibration key.
LISTCAL	Lists all available calibration keys and their respective values.

7.5.2 UART

Note: The “UART” command is not available when the compile flag “USB_ENABLE” was removed from the project.

Reports UART communication status or changes its value.

To see UART communication status, use “UART”.

Command:

UART

Response:

UART: 0

To set UART communication status, use “UART <status>”,

- where <status> value:
 - 0: disables communication using UART,
 - 1: enables communication using UART.

Command:

UART 1

7.5.3 CALKEY

Sets or gets a calibration key.

To get a value of a calibration key, use “CALKEY <KEY>”:

Command:

CALKEY xtal_trim

Response:

xtal_trim: 0x32 (len: 1)

See *Calibration and Configuration* chapter for more details on calibration possibilities.

To set a value of a calibration key, use “CALKEY <KEY> <VALUE>”:

Command:

CALKEY xtal_trim 1

Response:

xtal_trim: 0x1 (len: 1)

7.5.4 LISTCAL

Lists all available calibration keys and their respective values.

Command:

LISTCAL

Response:

```
restricted_channels: 0x0000 (len: 2)
wifi_coex_mode: 0x00 (len: 1)
wifi_coex_time_gap: 0x00 (len: 1)
ch5.wifi_coex_enabled: 0x01 (len: 1)
ch5 pll_locking_code: 0x00 (len: 1)
ch9.wifi_coex_enabled: 0x01 (len: 1)
...
ant_set0.rx_ants: 0x000001 (len: 3)
ant_set0.tx_ant_path: 0x00 (len: 1)
ant_set0.nb_rx_ants: 0x01 (len: 1)
ant_set0.rx_ants_are_pairs: 0x00 (len: 1)
ant_set0.tx_power_control: 0x01 (len: 1)
experimental.mac.session_scheduler.id: 0x00 (len: 1)
```

7.6 Application Commands

7.6.1 SDK Applications

Top-level applications cannot run concurrently since they use the same resources.

Table 7.5: List of applications

Command	Description
INITF	Sets the board as INITIATOR in a FiRa two-way ranging session.
RESPF	Sets the board as RESPONDER in a FiRa two-way ranging session.
LISTENER	Puts the board into receive mode and reports any received packets.

7.6.2 FiRa applications: INITF/RESPF

7.6.2.1 INITF/RESPF command

Warning: If you are not using a QM33120WDK1, DWM3001CDK or Type2AB EVB (e.g. DWM3000EVB), you must flash the calibration before starting a ranging session.

Warning: Using the default calibration leads to decreased ranging accuracy (see section [Calibration important notes](#) for more details).

INITF sets the board as INITIATOR in a FiRa two-way ranging session. See table below for configuration parameters.
Usage:

```
INITF [Option1] [Option2] ...
Options: (only default values are shown, check the SDK Manual to know more about the available configurations)
-CHAN=9      --> Channel number
-PRFSET=BPRF4 --> PRF set
-PCODE=10    --> Preamble code index
-SLOT=2400   --> Slot duration [RSTU]
-BLOCK=200   --> Block duration [ms]
-ROUND=25    --> Round duration [slots]
-RRU=DSTWR   --> Ranging round usage
-ID=42       --> Session ID
-VUPPER=01:02:03:04:05:06:07:08 --> vUpper64
-MULTI        --> Activate one-to-many mode
-HOP          --> Activate round hopping
-ADDR=0       --> Device own address (Initiator address)
-PADDR=1      --> 1st Responder address
               or to set multiple responders:
-PADDR=[1,2,...,n] --> to set n Responder addresses (for one-to-many)
```

RESPF sets the board as RESPONDER in a FiRa two-way ranging session. See table below for configuration parameters.
Usage:

```
RESPF [Option1] [Option2] ...
Options: (only default values are shown, check the SDK Manual to know more about the available
```

(continues on next page)

(continued from previous page)

```

→configurations)
-CHAN=9      --> Channel number
-PRFSET=BPRF4 --> PRF set
-PCODE=10     --> Preamble code index
-SLOT=2400    --> Slot duration [RSTU]
-BLOCK=200    --> Block duration [ms]
-ROUND=25     --> Round duration [slots]
-RRU=DSTWR    --> Ranging round usage
-ID=42       --> Session ID
-VUPPER=01:02:03:04:05:06:07:08   --> vUpper64
-MULTI        --> Activate one-to-many mode
-HOP          --> Activate round hopping
-ADDR=1       --> Device own address (Responder address)
-PADDR=0      --> Peer address (Initiator address)

```

Table 7.6: INITF/RESPF parameters

Option	Default Value	Range	Description
CHAN	9	5 or 9	Channel number
PRFSET	BPRF4	BPRF3 BPRF4 BPRF5 BPRF6	Pulse Repetition Frequency parameter sets See BPRF mode operating parameter sets
PCODE	10	9 to 12	Preamble code index
SLOT	2400	2400 to 65535	Duration of the slot in RSTU. 1 ms = 1200 RSTU
BLOCK	200	1 to 65535	Duration of the ranging block in ms
ROUND	25	1 to 255	Number of slots inside a ranging round
RRU	DSTWR	SSTWR DSTWR SSTWRNDEF DSTWRNDEF	Ranging round usage SSTWR: Single-Sided Two-Way Ranging with Deferred mode DSTWR: Double-Sided Two-Way Ranging with Deferred mode SSTWRNDEF: Single-Sided Two-Way Ranging with Non-Deferred mode DSTWRNDEF: Double-Sided Two-Way Ranging with Non-Deferred mode
ID	42	1 to 65535	Session ID
VUPPER	01:02:03:04:05:06:07:08	00:00:00:00:00:00:00:00 to FF:FF:FF:FF:FF:FF:FF:FF	vUpper64 Eight hexadecimal numbers, representing static part of the STS in FiRa standard
MULTI	Not activated		Activate one-to-many mode (instead of unicast mode)
HOP	Not activated		Activate round hopping
ADDR	Initiator: 0, Responder: 1	0 to 65535	Device own address
PADDR	Initiator: 1, Responder: 0	0 to 65535	Address(es) of peer devices For Responder, it is the Initiator address (only one) For Initiator, it is the Responder address or the list of multiple Responder addresses

Note: The options are all optional and can be set in any order and capital letters are not mandatory.

Note: Even if the full range for some parameters is shown, not all combinations are valid. Example:

- **ROUND** parameter has to be adjusted to the given **RRU** and a number of controlees.
- **SLOT** time multiplied by the number of slots in a round cannot exceed time of the **BLOCK**.
- It's not recommended to exceed 17 s for the **BLOCK** time.

Table 7.7: BPRF mode operating parameter sets

BPRF Set#	SYNC PSR	SFD# (802.15.4-2020)	SFD Length	STS nr of Segments	STS Segment Length	PHR + data	Data Rate (Mbps)	Description
BPRF3	64	2	8	1	64	Yes	6.8	SP1 (Mandatory)
BPRF4	64	2	8	1	64	No	n/a	SP3 (Mandatory)
BPRF5	64	0	8	1	64	Yes	6.8	SP1 (Optional)
BPRF6	64	0	8	1	64	No	n/a	SP1 (Optional)

Listing 7.1: Example of default INITF command

```
INITF
```

Listing 7.2: Equivalent of default INITF command

```
INITF -CHAN=9 -PRFSET=BPRF4 -SLOT=2400 -BLOCK=200 -ROUND=25 -RRU=DSTWR -ID=42 -
↪VUPPER=01:02:03:04:05:06:07:08 -ADDR=0 -PADDR=1
```

Listing 7.3: Example of default RESPF command

```
RESPF
```

Listing 7.4: Equivalent of default RESPF command

```
RESPF -CHAN=9 -PRFSET=BPRF4 -SLOT=2400 -BLOCK=200 -ROUND=25 -RRU=DSTWR -ID=42 -
↪VUPPER=01:02:03:04:05:06:07:08 -ADDR=1 -PADDR=0
```

Note: SAVE can be used to run the next INITF/RESPF (without specifying parameters) with previously saved parameters.

Listing 7.5: Example of INITF command with multiple Responders

```
INITF -MULTI -ADDR=0 -PADDR=[1, 2, 3]
```

Listing 7.6: Example of RESPF commands for responder devices to work with multi node Initiator example above

```
RESPF -MULTI -PADDR=0 -ADDR=1
RESPF -MULTI -PADDR=0 -ADDR=2
RESPF -MULTI -PADDR=0 -ADDR=3
```

Listing 7.7: Run application with saved parameters

```
RESPF -CHAN=5 -BLOCK=400
```

(continues on next page)

(continued from previous page)

```
STOP
SAVE
RESPF
<application running with CHAN=5 and BLOCK=400>
```

Next application command with provided any parameter will reset parameters and apply only provided one.

Listing 7.8: Change one of the parameters

```
RESPF -CHAN=5 -BLOCK=400
STOP
SAVE
RESPF -PRFSET=BPRF3
<application running with PRFSET=BPRF3 but CHAN=9 (default), BLOCK=200 (default)>
```

To restore all default parameters, run INITF/RESPF with at least one given default parameter or use RESTORE command.

Listing 7.9: Reset parameters to default

```
RESPF -CHAN=5 -BLOCK=400
STOP
SAVE
RESPF -CHAN=9
<application running with all default parameters>
STOP
SAVE
RESPF
<application running with all default parameters>
```

Listing 7.10: Reset parameters to default with RESTORE command

```
RESPF -CHAN=5 -BLOCK=400
STOP
SAVE
RESTORE
RESPF
<application running with all default parameters>
```

7.6.2.2 INITF/RESPF messages

INITF and RESPF commands display notifications during the ranging session.

Listing 7.11: Example log from INITF execution

```
SESSION_INFO_NTF: {session_handle=1, sequence_number=0, block_index=0, n_measurements=1
 [mac_address=0x0001, status="SUCCESS", distance[cm]=91, loc_az_pdoa=65.35, loc_az=24.90, loc_el_pdoa=32.12, loc_el=12.01, rmt_az=22.84, rmt_el=13.59, RSSI[dBm]=-66.5]}
SESSION_INFO_NTF: {session_handle=1, sequence_number=1, block_index=1, n_measurements=1
 [mac_address=0x0001, status="SUCCESS", distance[cm]=101, loc_az_pdoa=57.79, loc_az=21.87, loc_el_pdoa=35.54, loc_el=14.62, rmt_az=20.92, rmt_el=15.72, RSSI[dBm]=-66.5]}
SESSION_INFO_NTF: {session_handle=1, sequence_number=2, block_index=2, n_measurements=1
 [mac_address=0x0001, status="SUCCESS", distance[cm]=105, loc_az_pdoa=57.71, loc_az=21.85, loc_el_pdoa=35.54, loc_el=14.62, rmt_az=20.92, rmt_el=15.72, RSSI[dBm]=-66.5]}
```

(continues on next page)

(continued from previous page)

```

→el_pdoa=33.34, loc_el=12.89, rmt_az=21.37, rmt_el=14.02, RSSI[dBm]=-67.5]}
SESSION_INFO_NTF: {session_handle=1, sequence_number=3, block_index=3, n_measurements=1
[mac_address=0x0001, status="SUCCESS", distance[cm]=110, loc_az_pdoa=56.48, loc_az=21.37, loc_
→el_pdoa=34.47, loc_el=13.16, rmt_az=21.12, rmt_el=15.47, RSSI[dBm]=-65.5]}

```

Table 7.8: INITF/RESPF log legend

session_handle	Unique number generated by the device used to identify the UWB session.
sequence_number	Counter which starts from 0 and is incremented after every notification sent.
block_index	Current block index.
n_measurements	Number of measurements in this notification (one measurement = one pair of initiator/responder devices).
mac_address	MAC address of the peer device (the one that the current device is ranging with).
status	Status of the current ranging round.
distance[cm]	Distance between devices in centimeters.
loc_az_pdoa	Local raw PDoA measurements for Azimuth in deg [-180..+180].
loc_az	Local AoA Azimuth in deg [-90..+90].
loc_el_pdoa	Local raw PDoA measurements for Elevation in deg [-180..+180].
loc_el	Local AoA Elevation in deg [-90..+90].
rmt_az	Remote AoA Azimuth in deg [-90..+90].
rmt_el	Remote AoA Elevation in deg [-90..+90].
RSSI	RSSI (displayed only when DIAG is enabled, see DIAG command).

Set AoA/PDoA Enhancement Parameters

Some parameters can be used to improve AoA/PDoA performance. These settings need to be set on the device where the local AoA will be collected. This is where the AoA antenna is used and generally (but not always) it is the initiator.

- The calibration parameters can be used to enhance accuracy (see [Calibration and Configuration](#) chapter for more details).

7.6.3 UWB sniffer: LISTENER

PHY-level LISTENER application. This command will start a sniffer application based on the **LCFG** configuration.

Usage:

```

LISTENER:
Listen for the UWB packets using the UWB configuration and displays.
1/ Maximum 127 bytes of payload.
2/ "TS4ns": Timestamp of each valid frame reception.
3/ "0": Crystal frequency offset in ppm units.
4/ RSL, FSL: RX Level and First Path Power level (dBm).

```

LSTAT command displays the statistics inside the LISTENER application. It needs to be called while LISTENER application is running and it shows the following information:

Table 7.9: LSTAT details

Info	Description
CRCG	12-bit number of good CRC received frame events.
CRCB	12-bit number of bad CRC (CRC error) received frame events.
ARFE	8-bit number of address filter error events.
PHE	12-bit number of received header error events.

continues on next page

Table 7.9 – continued from previous page

Info	Description
RSL	12-bit number of received frame sync loss event events.
SFDTO	12-bit number of SFD timeout events.
PTO	12-bit number of Preamble timeout events.
FTO	8-bit number of RX frame wait timeout events.
SFDD	Counts the number of SFD detections (RXFR shall be set also).

Example:*Responder side: start LISTENER with default LCFG settings***Command:**

LISTENER

Response:

```
Found AOA chip. PDoA is available.
Listener Top Application: Started
```

*Initiator side: start INITF with default settings***Command:**

INITF

Response:

```
FiRa Session Parameters: {
SESSION_ID: 42,
CHANNEL_NUMBER: 9,
DEVICE_ROLE: INITIATOR,
RANGING_ROUND_USAGE: DS_TWR_DEFERRED,
SLOT_DURATION [rstu]: 2400,
RANGING_DURATION [ms]: 200,
SLOTS_PER_RR: 25,
MULTI_NODE_MODE: UNICAST,
HOPPING_MODE: Disabled,
RFRAIME_CONFIG: SP3,
SFD_ID: 2,
PREAMBLE_CODE_INDEX: 10,
STATIC_STS_IV: "01:02:03:04:05:06",
VENDOR_ID: "07:08",
DEVICE_MAC_ADDRESS: 0x0000,
DST_MAC_ADDRESS[0]: 0x0001
}
```

Listener logs

Type “LISTENER” to dump all the data captured. This cannot run faster than every 1ms. Please note, if the buffer is full, some data packets may be skipped. Type “STOP” to halt.

*LISTENER application output of SP0 packets (STSMODE parameter set to 0):***Command:**

LISTENER

Response:

Command:

STOP

8 Qorvo One TWR GUI

8.1 Overview

The Qorvo One TWR application provides a graphical user interface to demonstrate features of the QM SDK like Two-Way Ranging, configuration and calibration.

8.2 User Manual

8.2.1 Installation

The Qorvo One TWR GUI supports desktop platforms running Windows, Linux or MacOS.

The installation executables for each supported operating system are located within the SDK package, specifically under the SDK/Tools/GUI directory.

8.2.1.1 Windows

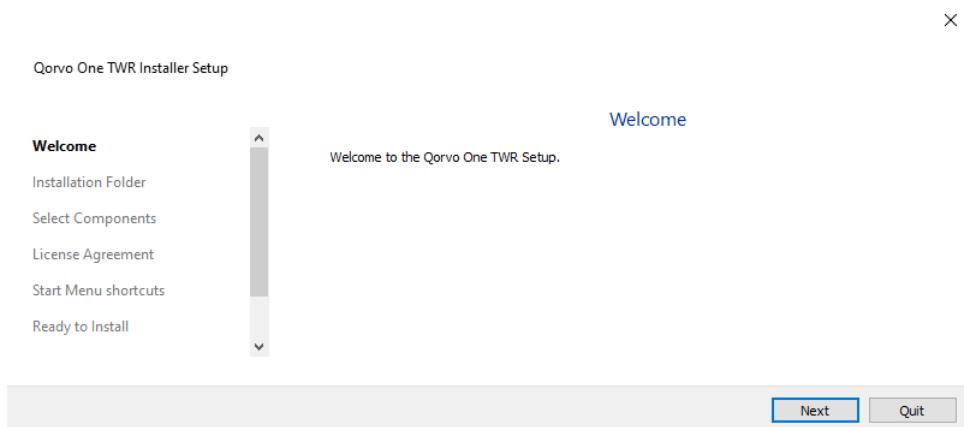


Figure 8.1: Qorvo One TWR application installer.

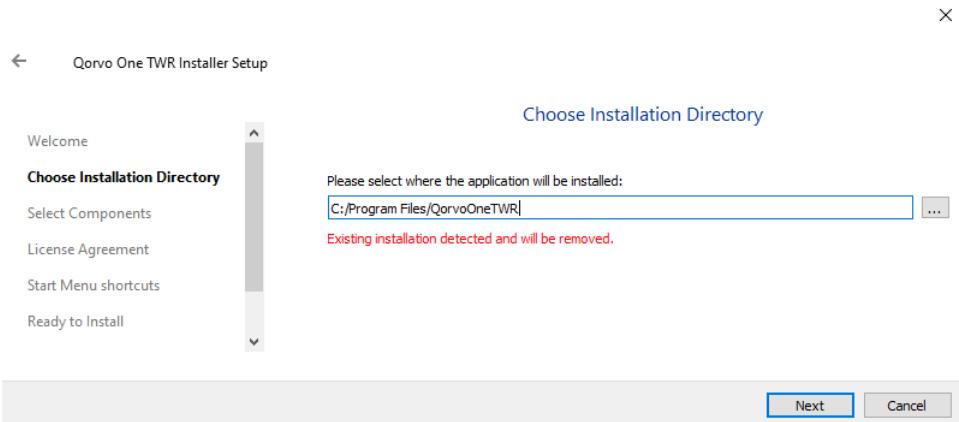


Figure 8.2: Removing existing installation.

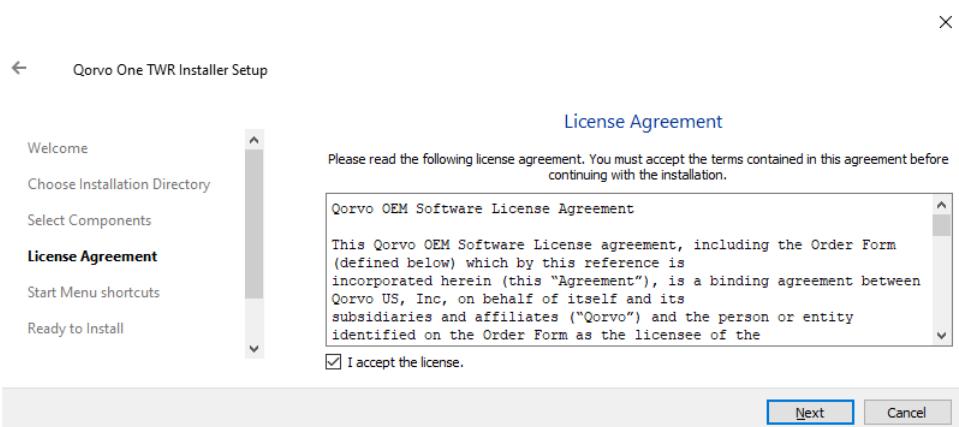


Figure 8.3: License agreement.

The application can be installed on Windows using **QorvoOneTwr-2.1.0-setup.exe** installer. Successful installation requires the license agreement to be accepted. And the previous installation is uninstalled.

The application can be started by clicking on the shortcut automatically created by the installer.

8.2.1.2 Linux

Make Qorvo One TWR installer script executable and run it from terminal. The installer will extract the application into the same folder by default.

```
$ ./QorvoOneTWR-2.1.0-x86_64-install.sh
```

The Qorvo One TWR application is packaged for the Linux distribution as [AppImage](#)¹⁷. To start the application simply run the installed AppImage.

Warning:

- If you got the error:

¹⁷ <https://appimage.org/>

```
dlopen(): error loading libfuse.so.2
```

Install libfuse2 using the following commands:

```
sudo apt update
```

```
sudo apt install libfuse2
```

- On Ubuntu 20.04, you may also need to install qt5-default:

```
sudo apt update
```

```
sudo apt install qt5-default
```

- On Ubuntu 22.04 & Ubuntu 24.04, you may also need to install qtbase5-dev:

```
sudo apt update
```

```
sudo apt install qtbase5-dev
```

8.2.1.3 Mac OS

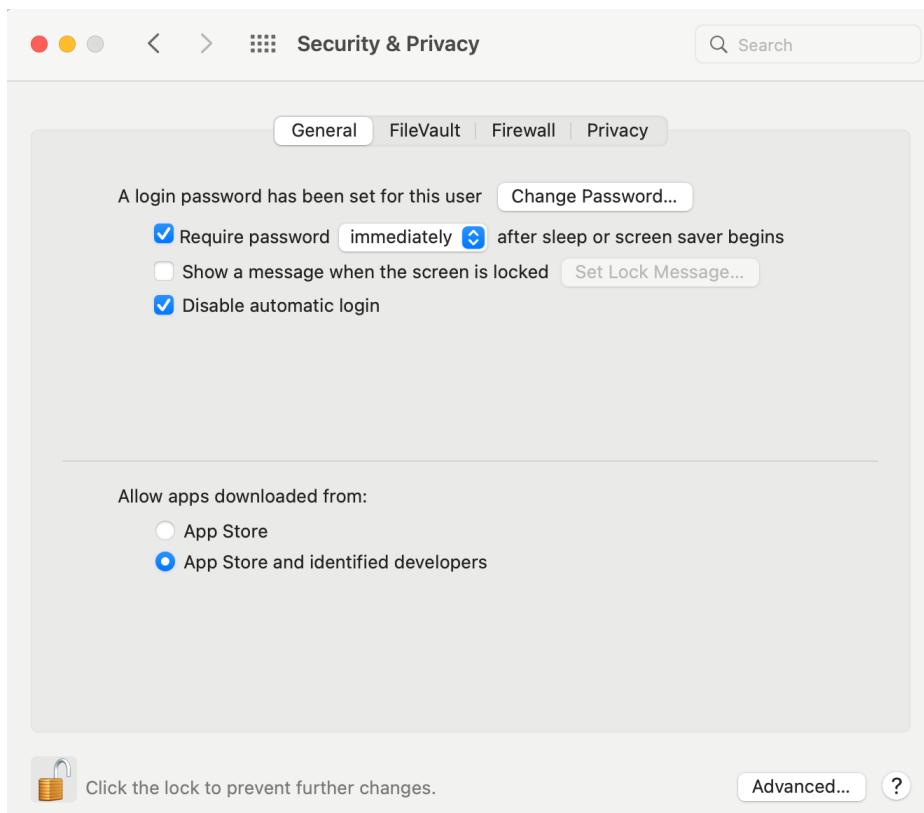


Figure 8.4: Allow applications from identified developers.

Configure security settings in **System Preferences** click the **Security & Privacy**, then click the **General**. Select the **App Store and identified developers** in the **Allow apps downloaded from**.

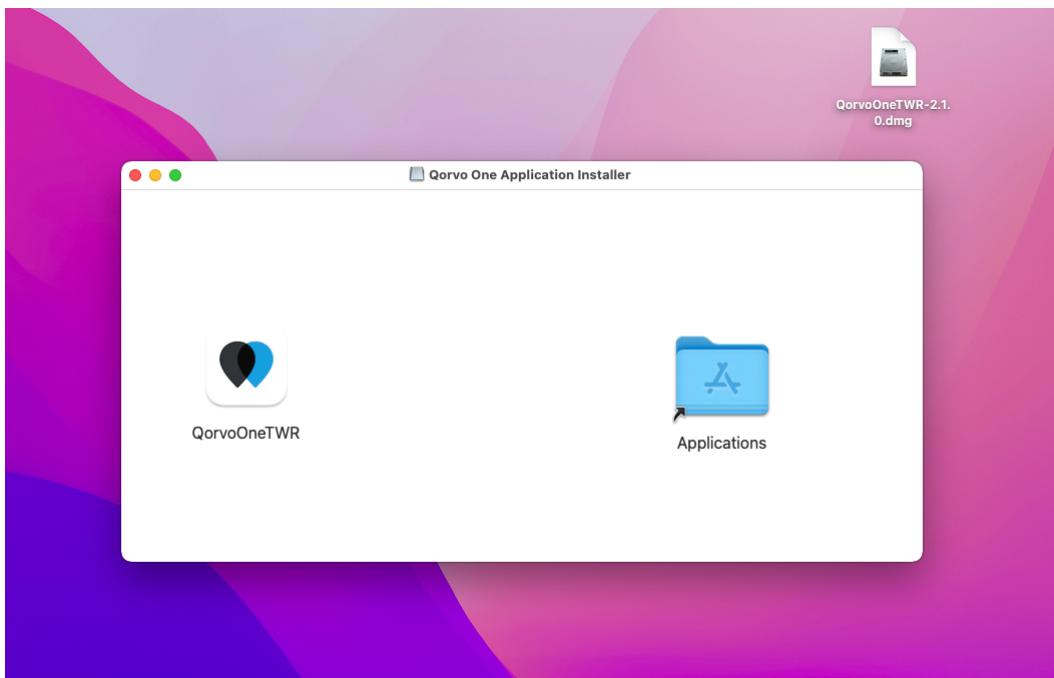


Figure 8.5: Installing into the Applications folder.

To install the **QorvoOneTwr-2.1.0.dmg**:

1. Double-click the DMG file.
2. Drag the application from the DMG window into the Applications directory to install (you may need the administrator password).
3. Eject the DMG installer.
4. Optionally delete the DMG from the Downloads directory.

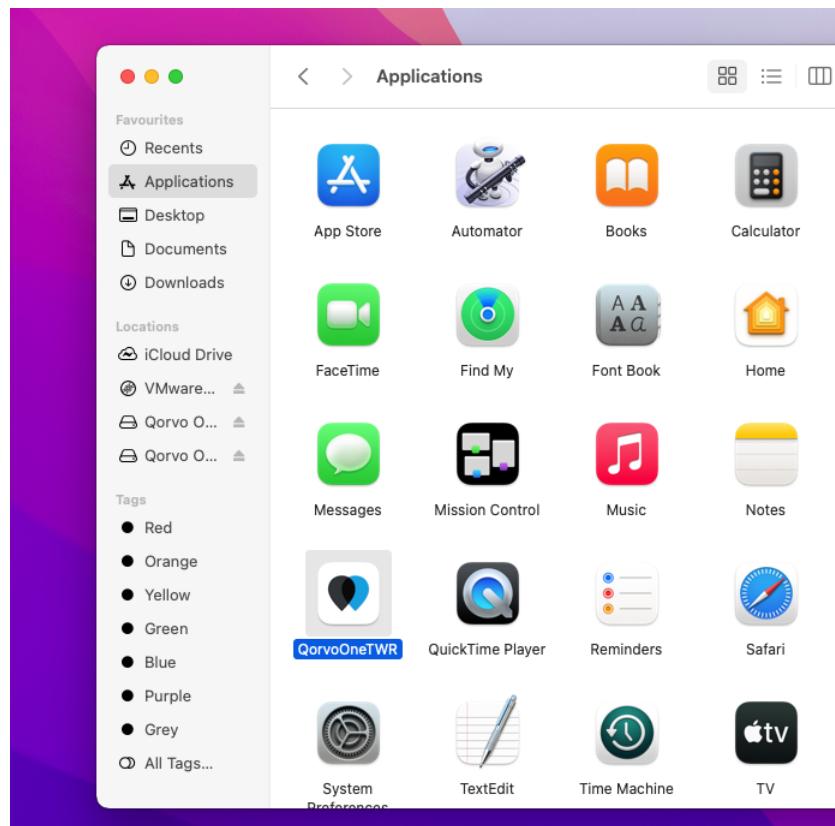


Figure 8.6: Installed Qorvo One TWR application.

8.2.2 Welcome screen

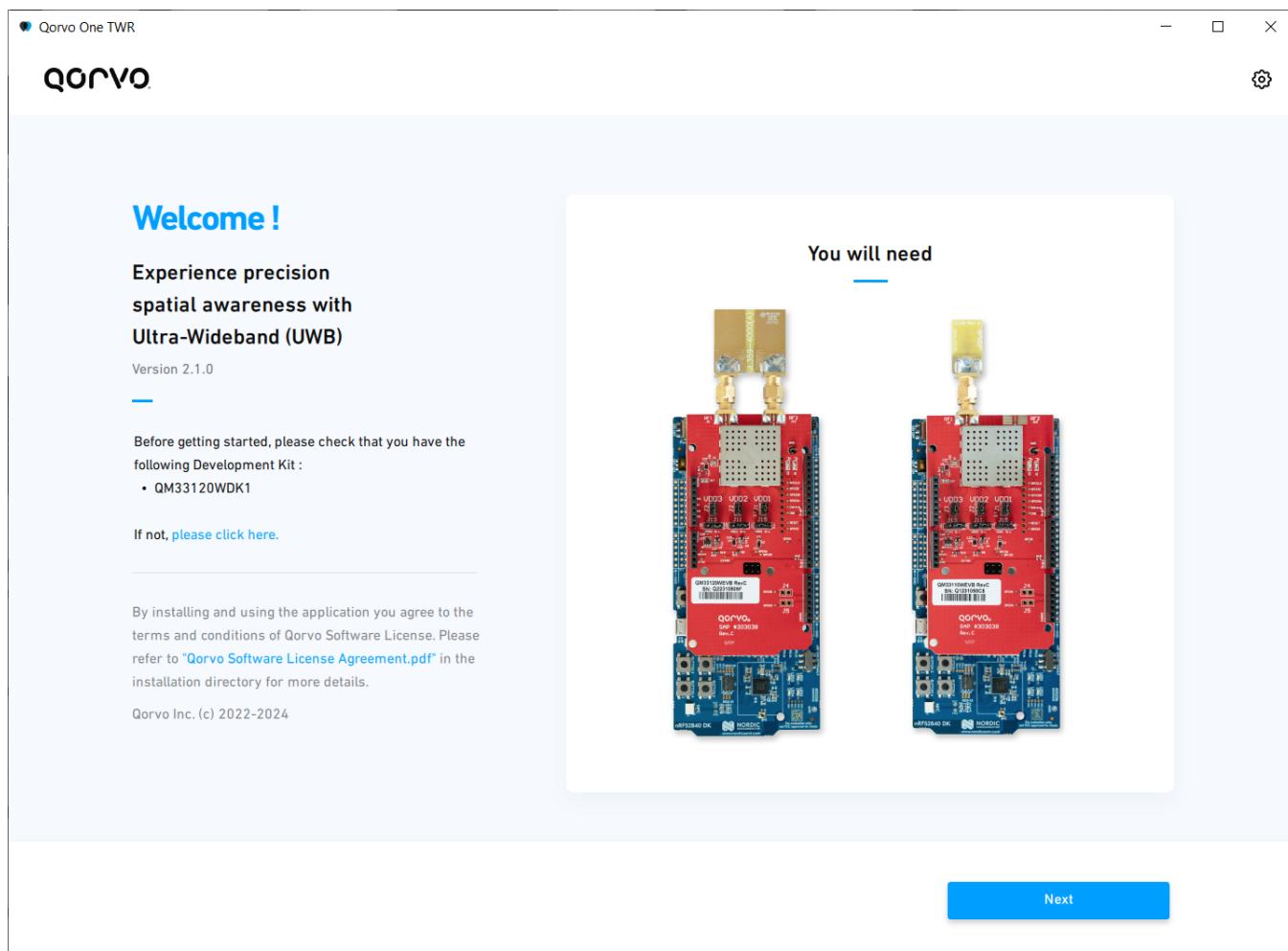


Figure 8.7: Welcome screen.

The initial screen provides information about required tools and the actual version of the application. The license agreement can be viewed by clicking on the link.

8.2.3 Setup

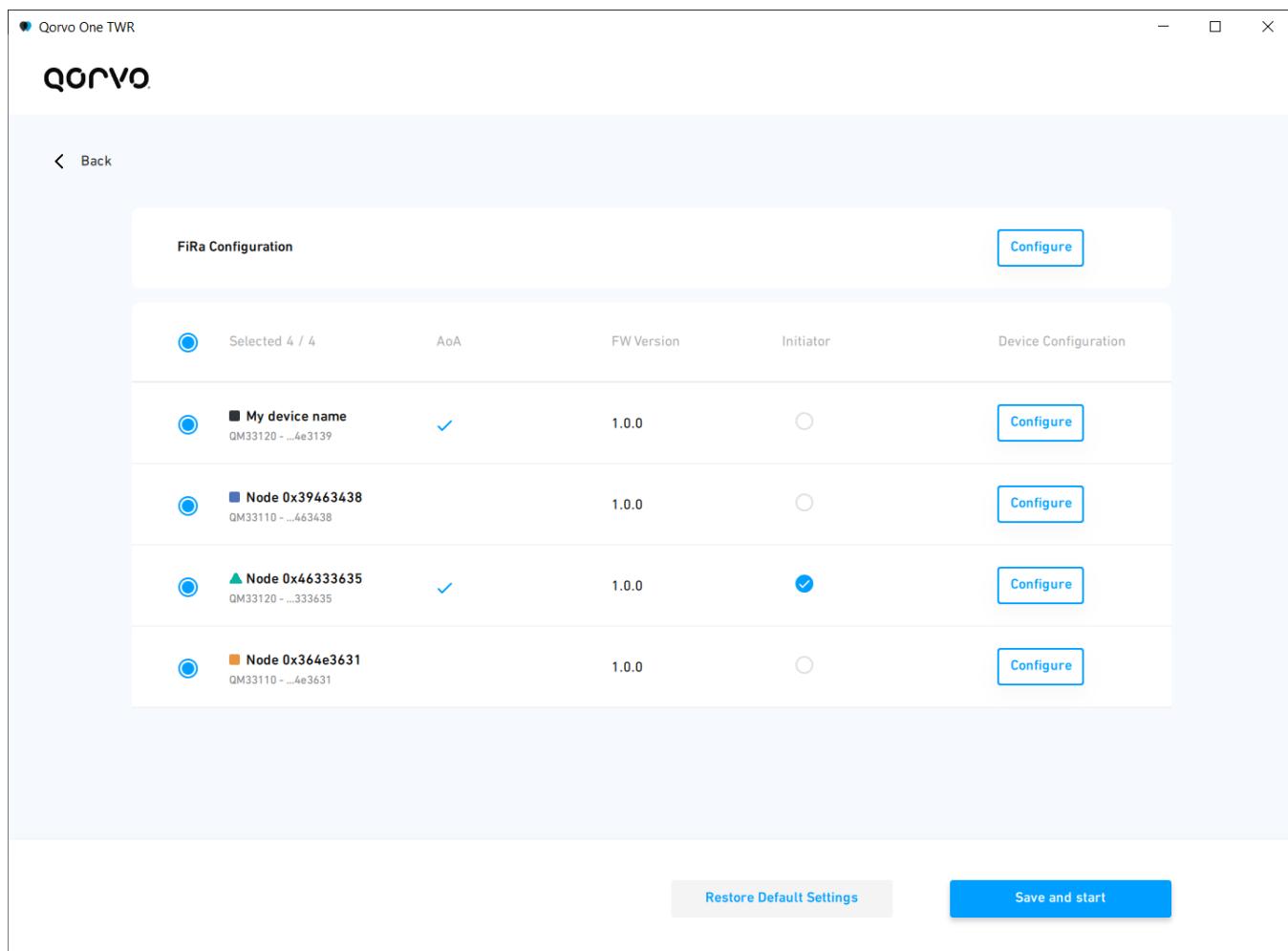


Figure 8.8: Setup screen with the list of detected devices.

When the setup screen is displayed the application continuously scans all serial ports for new devices. Detected devices appear in the device list on the screen. Device that is disconnected from the host is removed from the list. Device can be deselected which will exclude it from the TWR session.

Double clicking the device name enables editing of the name. Default name is “Node” + unique ID as hexadecimal number. The part ID read from the device is used as the unique ID. If no part ID is provided, the application uses the serial number of the serial port or a hash of the port name respectively as the part ID.

Below the device name is the chip name and the unique ID as hexadecimal number where the chip ID takes the upper 4 bytes and the unique ID takes lower 4 bytes.

There can be at most one initiator (the controller) and rest of the devices are configured automatically as responders (the controleses).

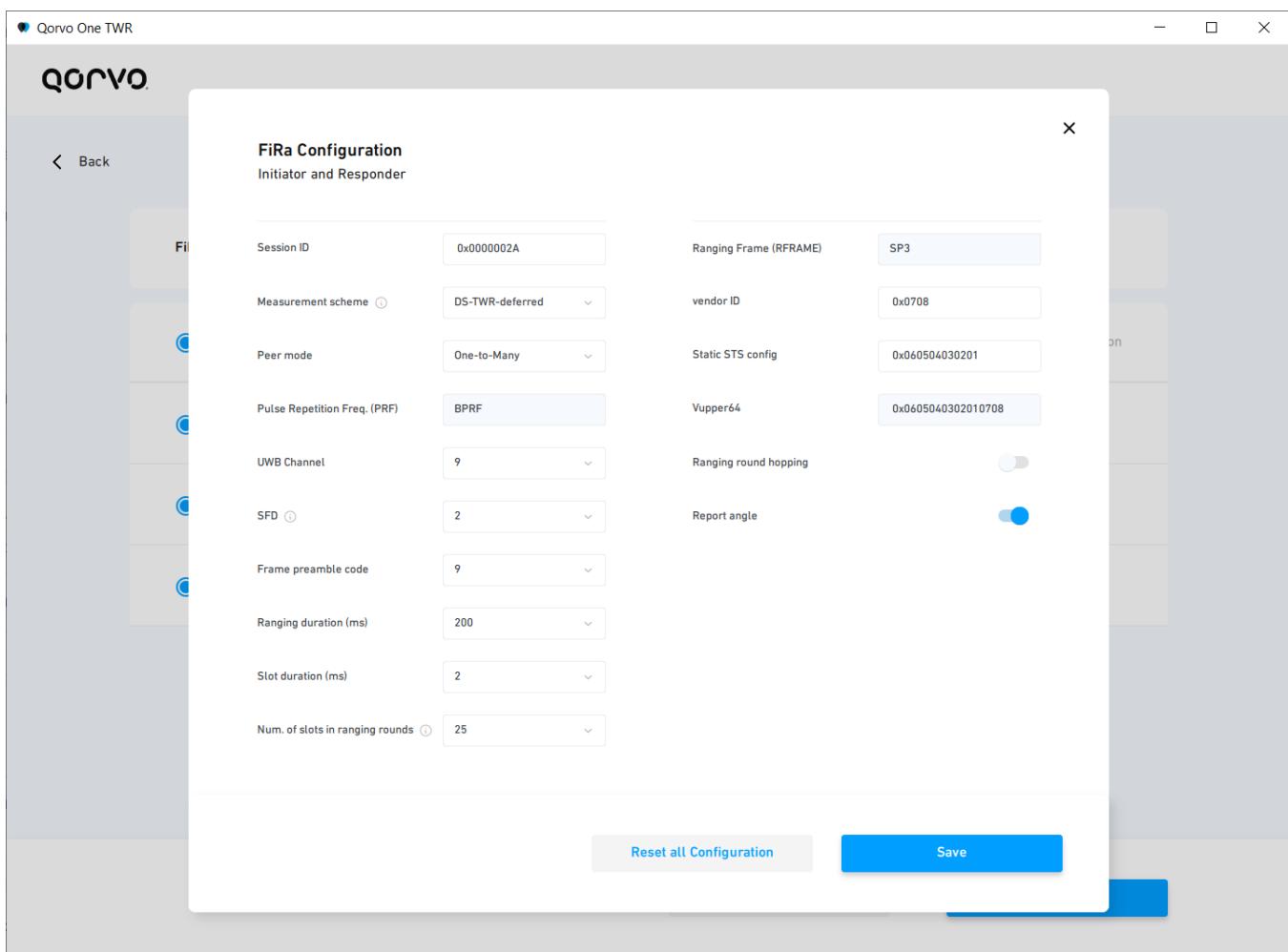


Figure 8.9: FiRa configuration parameters.

The FiRa Configuration popup provides parameters defined in the [FiRa¹⁸](#) specification. The popup can be activated by the **Configure** button in the FiRa Configuration tab and the parameters are used for all the devices in the TWR session.

¹⁸ <https://www.firaconsortium.org/>

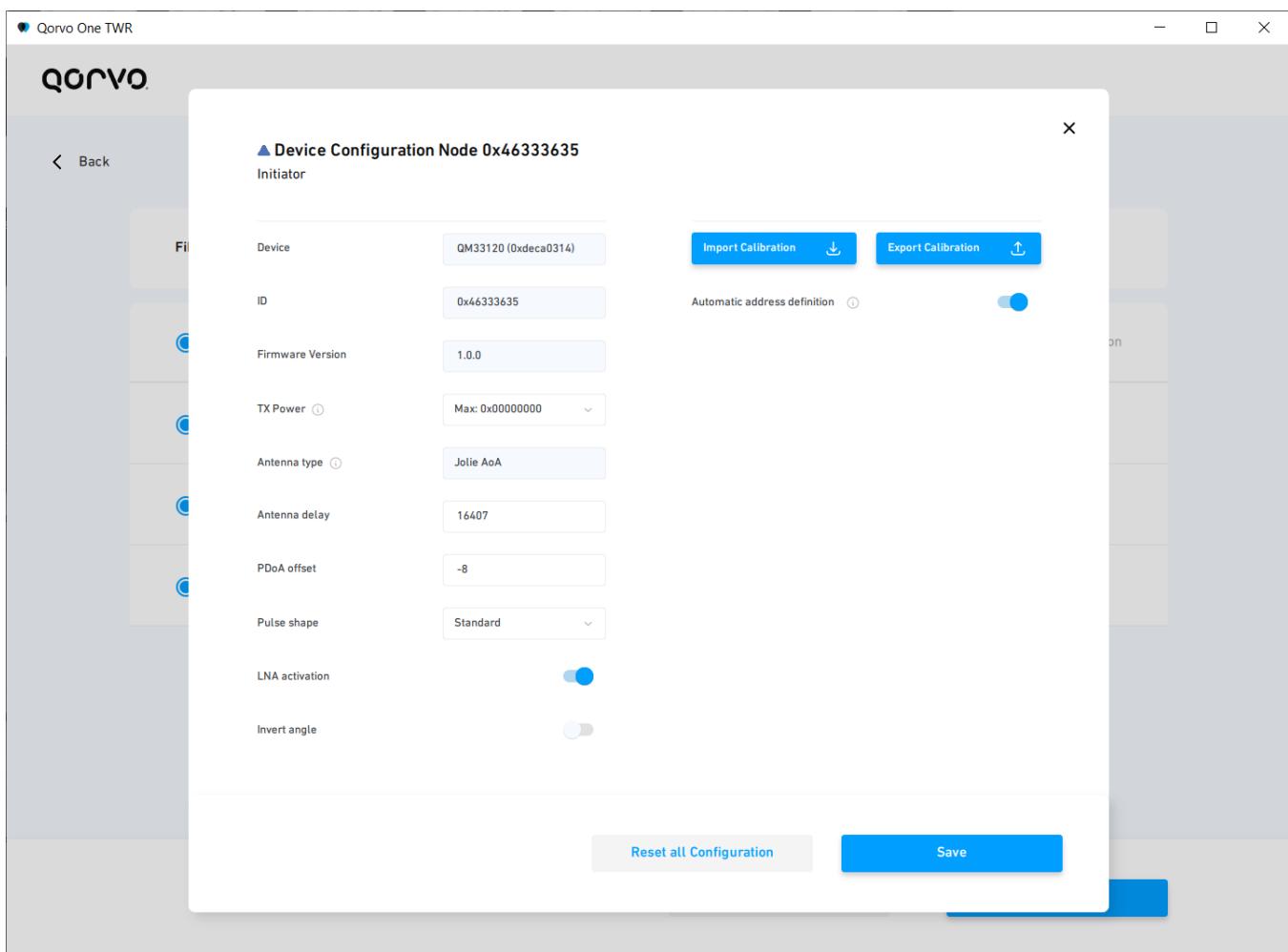


Figure 8.10: Device-specific configuration.

Device-specific parameters can be configured in the **Device Configuration** popup. The popup is activated by the **Configure** button on the device tab. Some of the configuration widgets might not be visible for the device that does not support all of the API.

The content of the **Device** text box is the chip/device name + (chip/device ID as hexadecimal number).

The content of the **ID** text box represents the unique ID of the device.

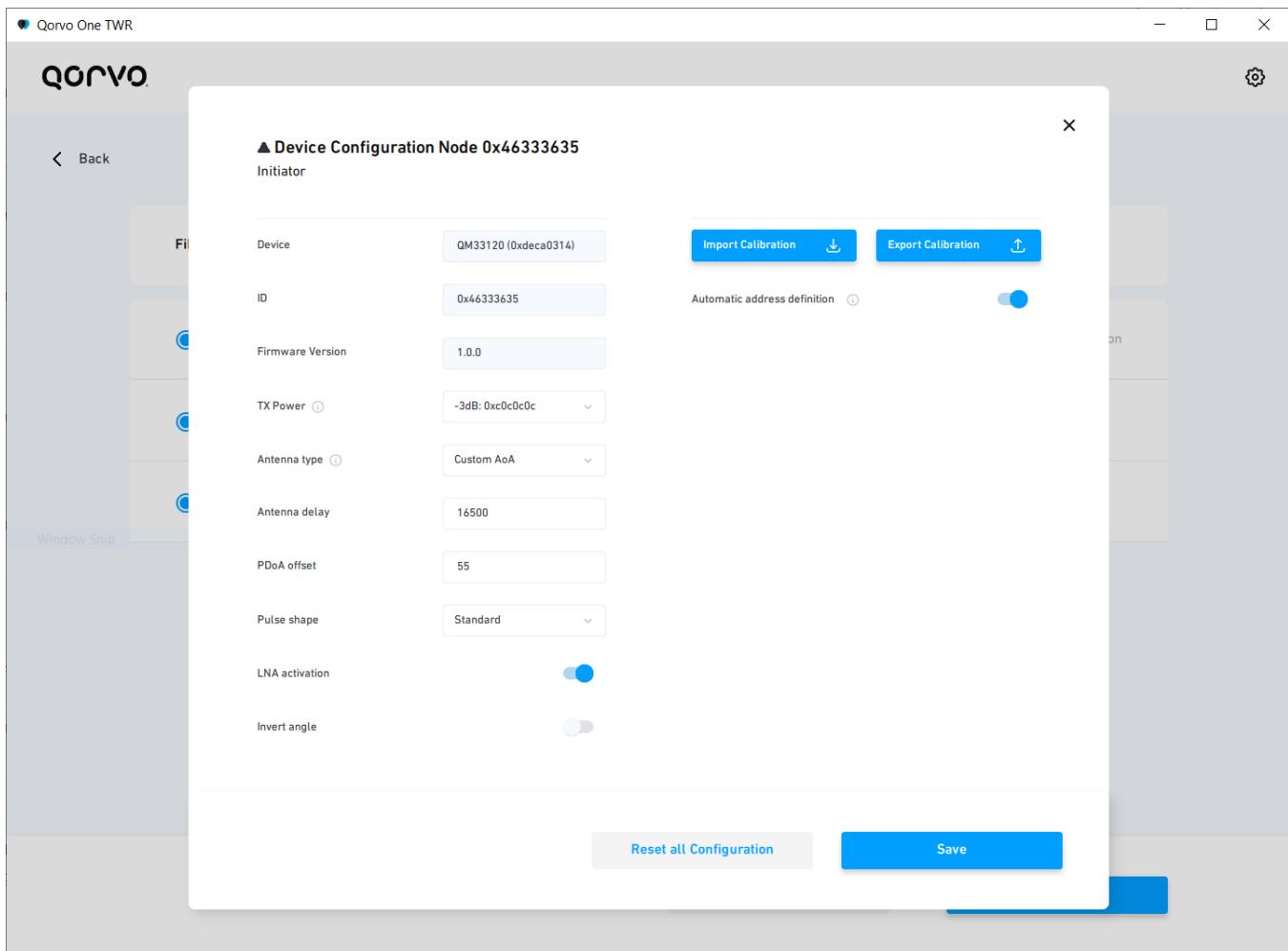


Figure 8.11: Custom antenna configuration.

Antenna type can be **Custom** or **Jolie** for each of the UWB channels and depends on the actual PDoA look-up table (LUT) of the channel. The **Custom** represents configuration of the antenna imported by the user which LUT is different from the default **Jolie** antenna. Antenna delay and PDoA offset are stored along with the LUT to allow switching between the **Custom** and **Jolie** antenna configuration.

Suffix **AoA** or **non AoA** is added based on the AoA capability. Device not capable of the AoA has no LUT neither PDoA configuration options in the device configuration page.

Inverting the angle of the initiator will cause showing opposite angles of all the responders in the *Real-Time Location* screen since the AoA is measured on the initiator side. Inverting the angle of the responder has effect only in case when there is no local initiator and the AoA is read from the responders instead.

Warning: Using the default calibration leads to decreased ranging accuracy (see section [Calibration important notes](#) for more details).

The calibration parameters can be exported or imported in the form of a JSON file. The imported file must respect calibration schema of the device to be successfully saved otherwise the import fails and user is notified by the state of the **Save** button. The button indicates busy state while the parameters are being saved into the device via the serial interface.

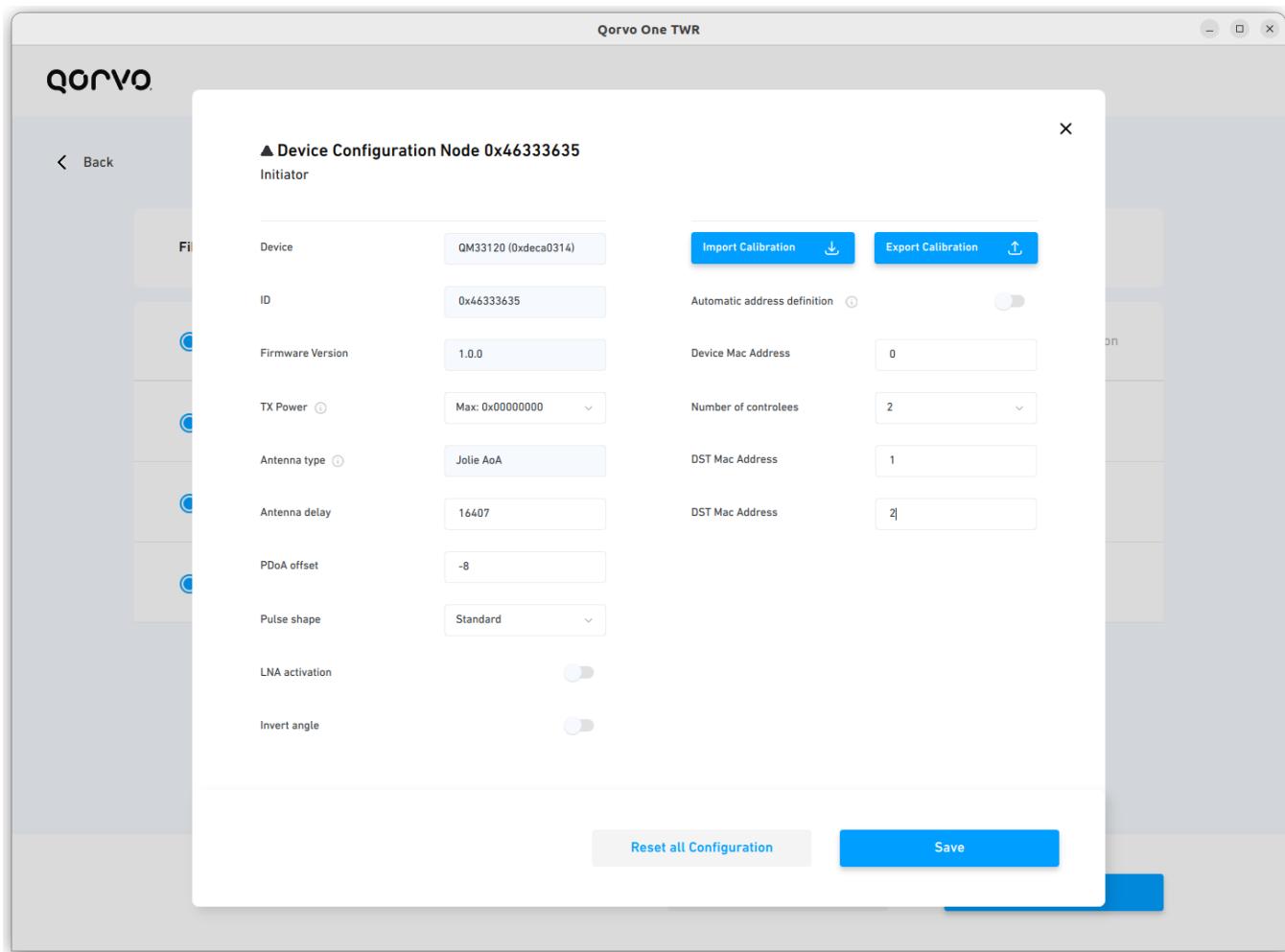


Figure 8.12: Manual configuration of the MAC addresses.

The MAC address and the destination MAC address list as defined in the FiRa specification can be configured manually which is necessary when ranging with the devices that are controlled by the different host or application.

The configuration is applied by clicking the **Save** button or it can be reset to default by clicking the **Reset all Configuration**. Resetting all the configuration on the setup screen will reset the FiRa configuration, the configuration of all the devices, the floor plan, the grid and the geofencing configuration. The FiRa configuration can be reset separately on the *[FiRa popup](#)*. The configuration can be reset per device using *[Device Configuration popup](#)*.

The TWR session starts by clicking the **Save and Start** button.

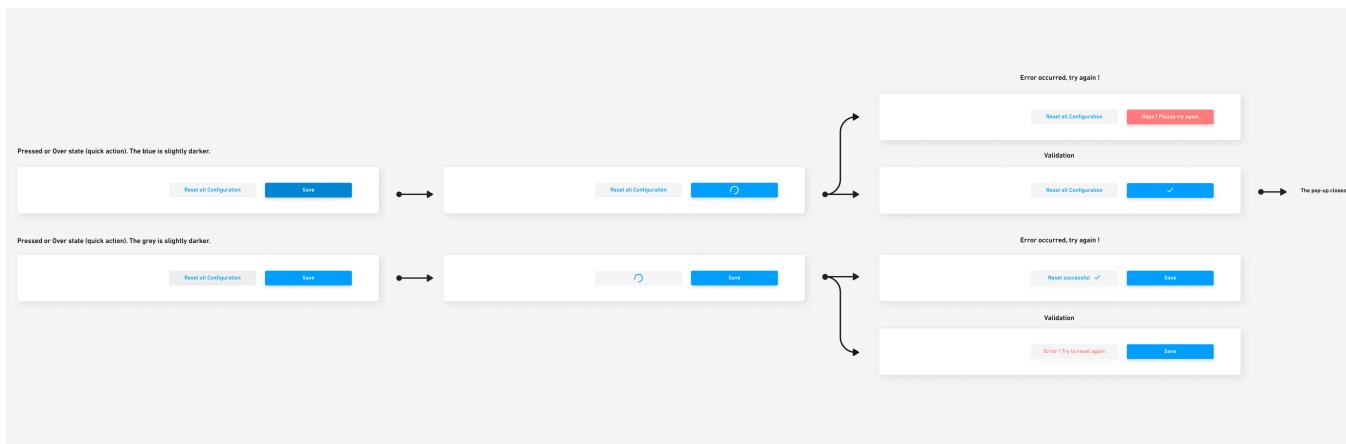


Figure 8.13: Possible states of the buttons.

8.2.4 Real-Time Location

The screen demonstrates the TWR between the devices where the initiator is at the top of the screen at fixed position and the position of the responders is being updated. The card with information details can be enabled or disabled by clicking on the responder. Clicking on the card can bring it forward. If the initiator is one of the local devices the position of the responders is shown as measured by the initiator.

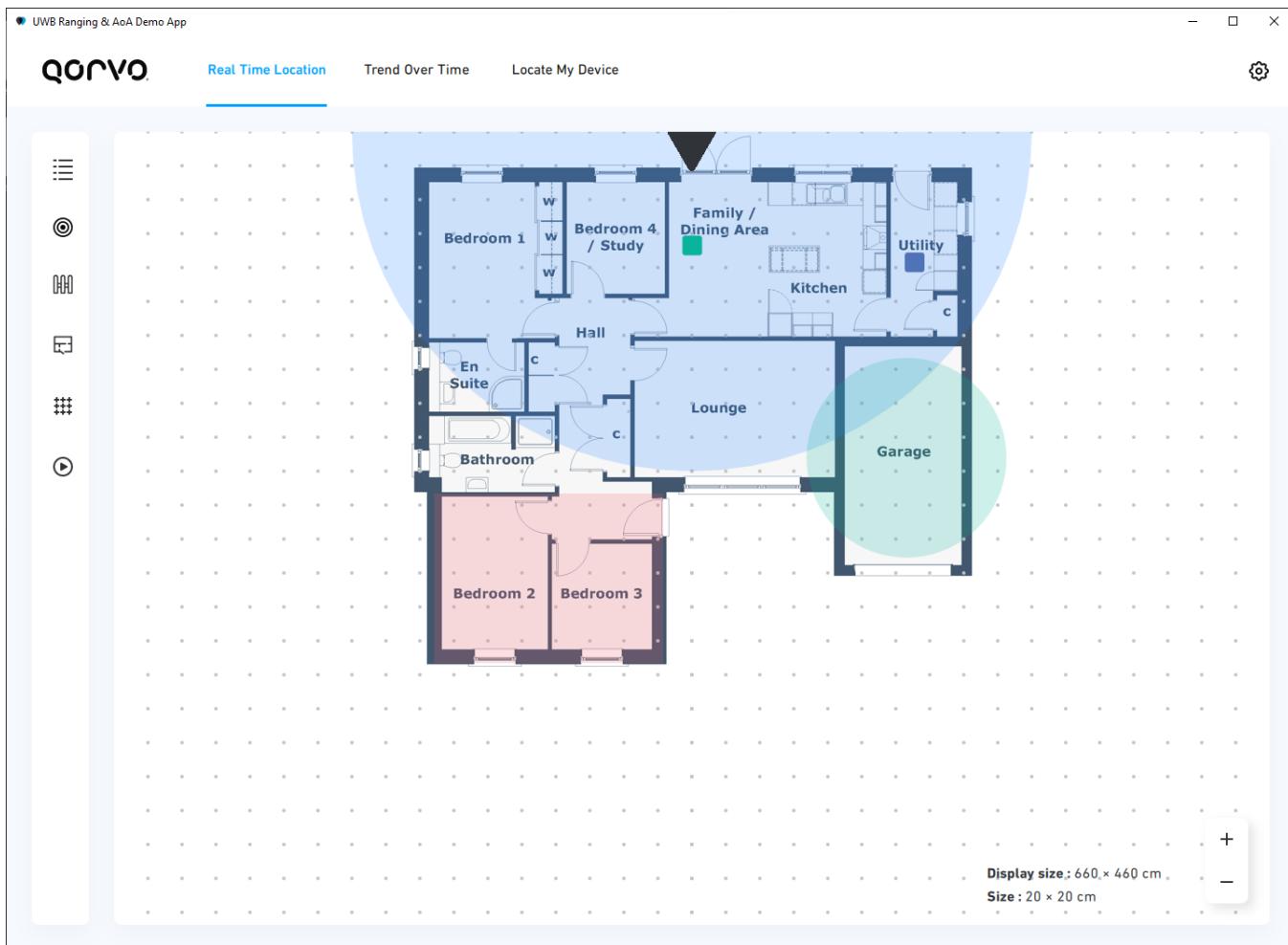


Figure 8.14: TWR between 1 initiator and 2 responders.

If none of the local devices is the initiator, the position to the unknown initiator is the distance and angle measured by each of the responders. The unknown initiator is shown as grey polygon at fixed position at the top of the screen.

The left bar allows to configure the *Grid*, *Floor Plan* and *Geofencing* zones shown on the screen. TWR can be stopped by clicking the icon on the right side of the top bar.

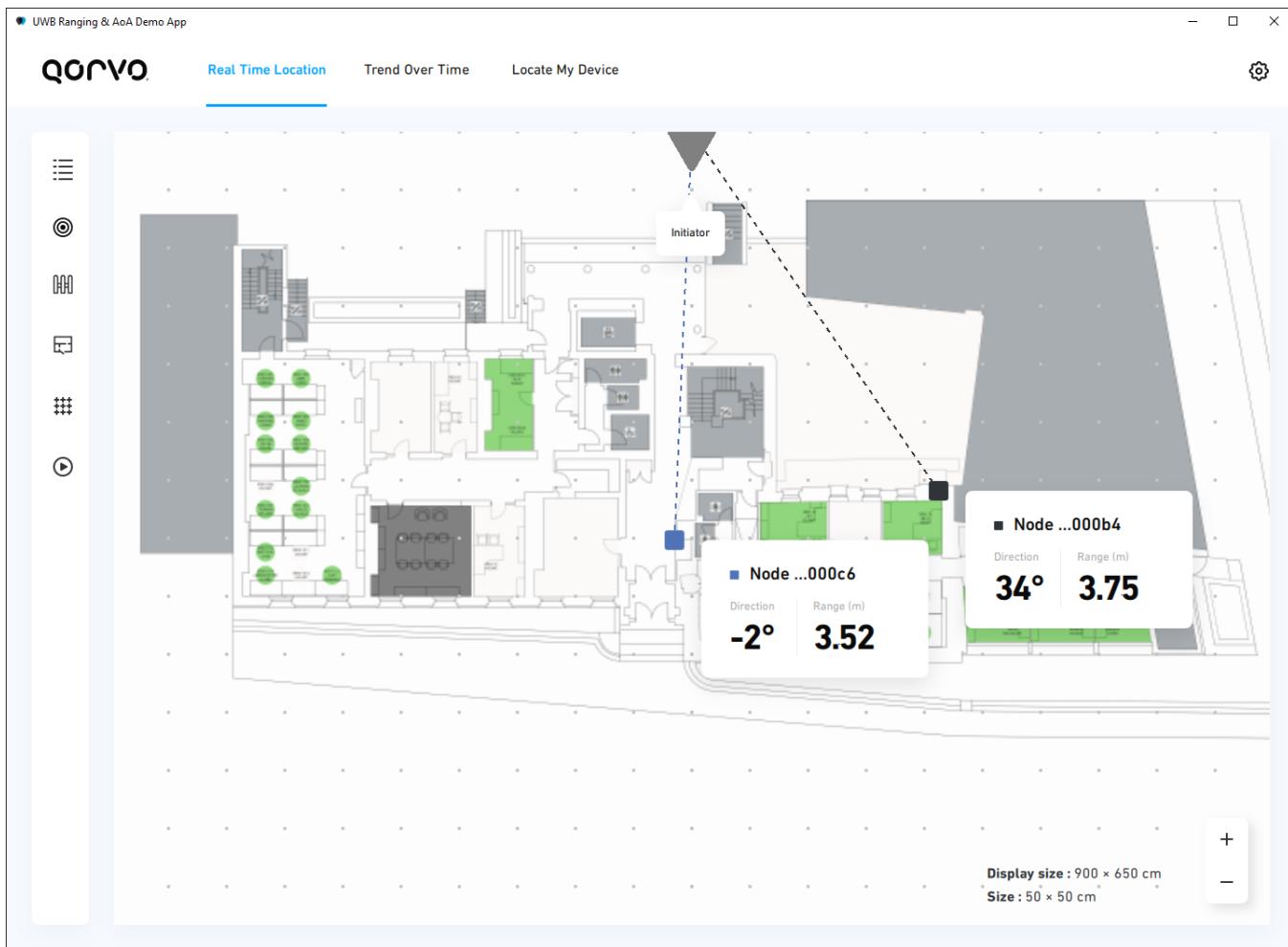


Figure 8.15: Ranging with the initiator that is on different host.

8.2.5 Trend Over Time

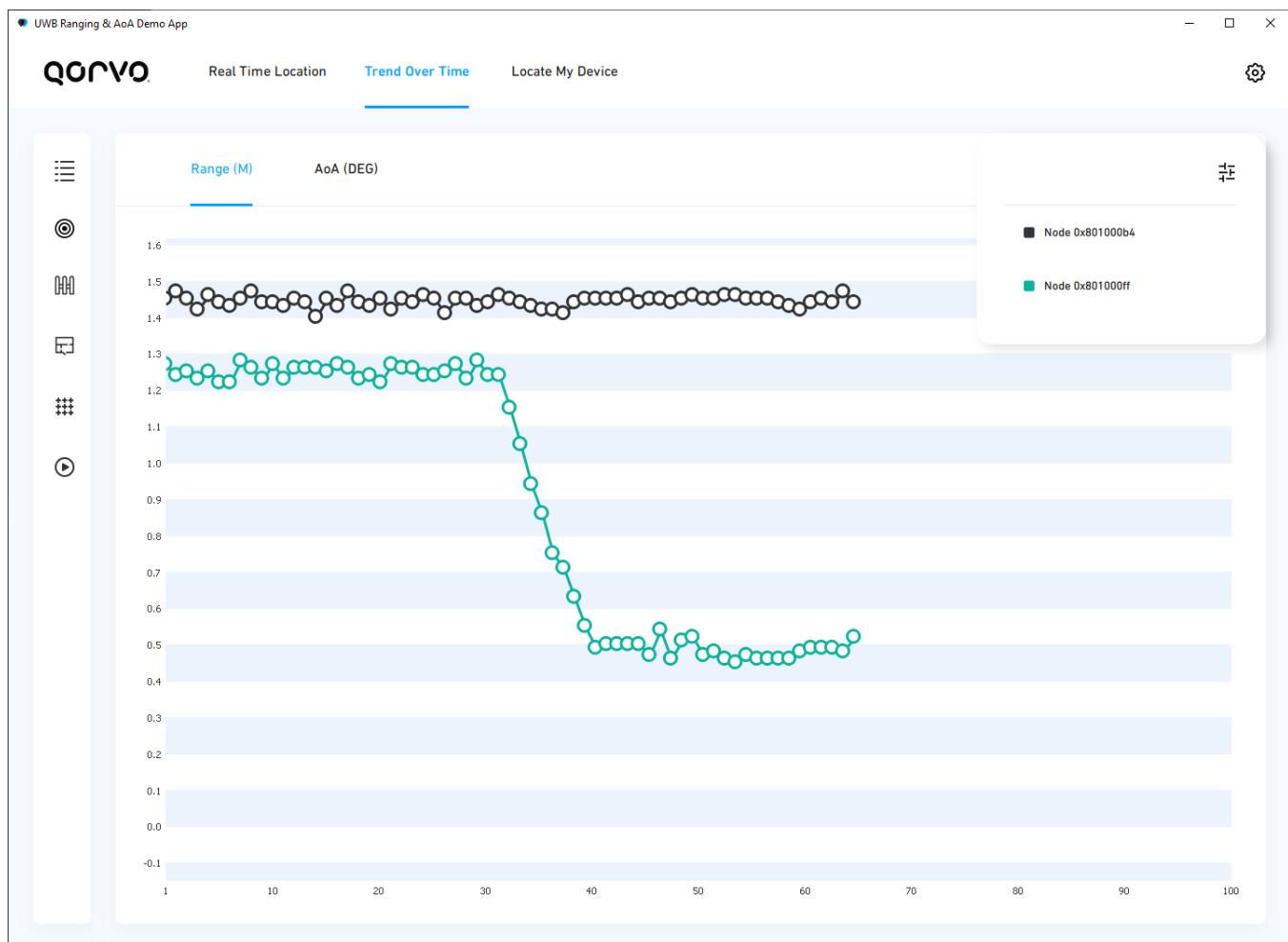


Figure 8.16: Distance samples of the responders.

The graph shows samples of either distance (Range) in meters or angle (AoA) in degrees for each responder. The responder data can be hidden on the graph by clicking on the card in the top right corner.

8.2.6 Locate My Device

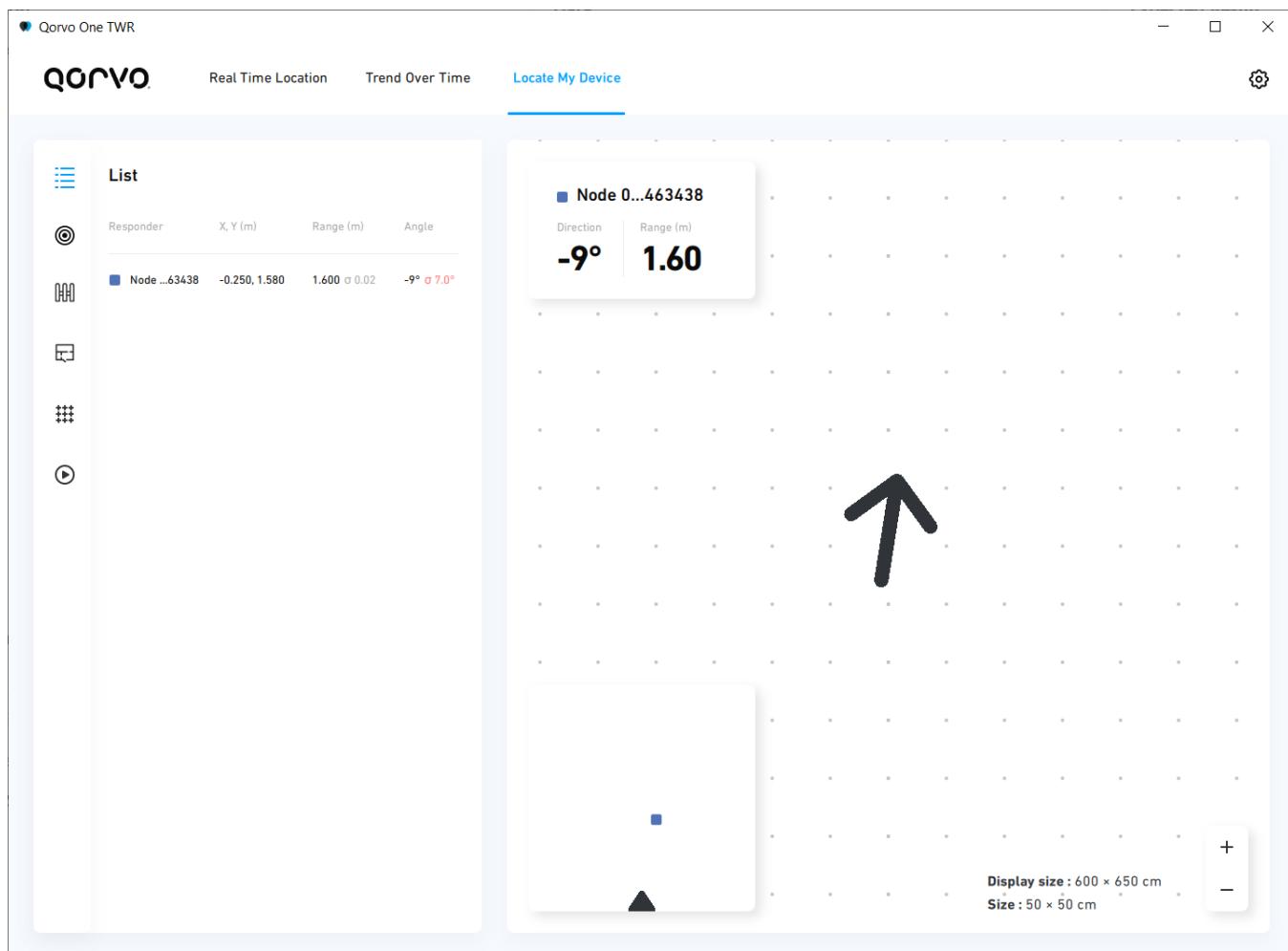


Figure 8.17: The “Locate My Device” screen.

The angle of the arrow in the middle of the screen points in the direction of responder that can be selected by the combo box in the top left corner. The minimap in the bottom left corner shows position of all the responders.

8.2.7 Device List

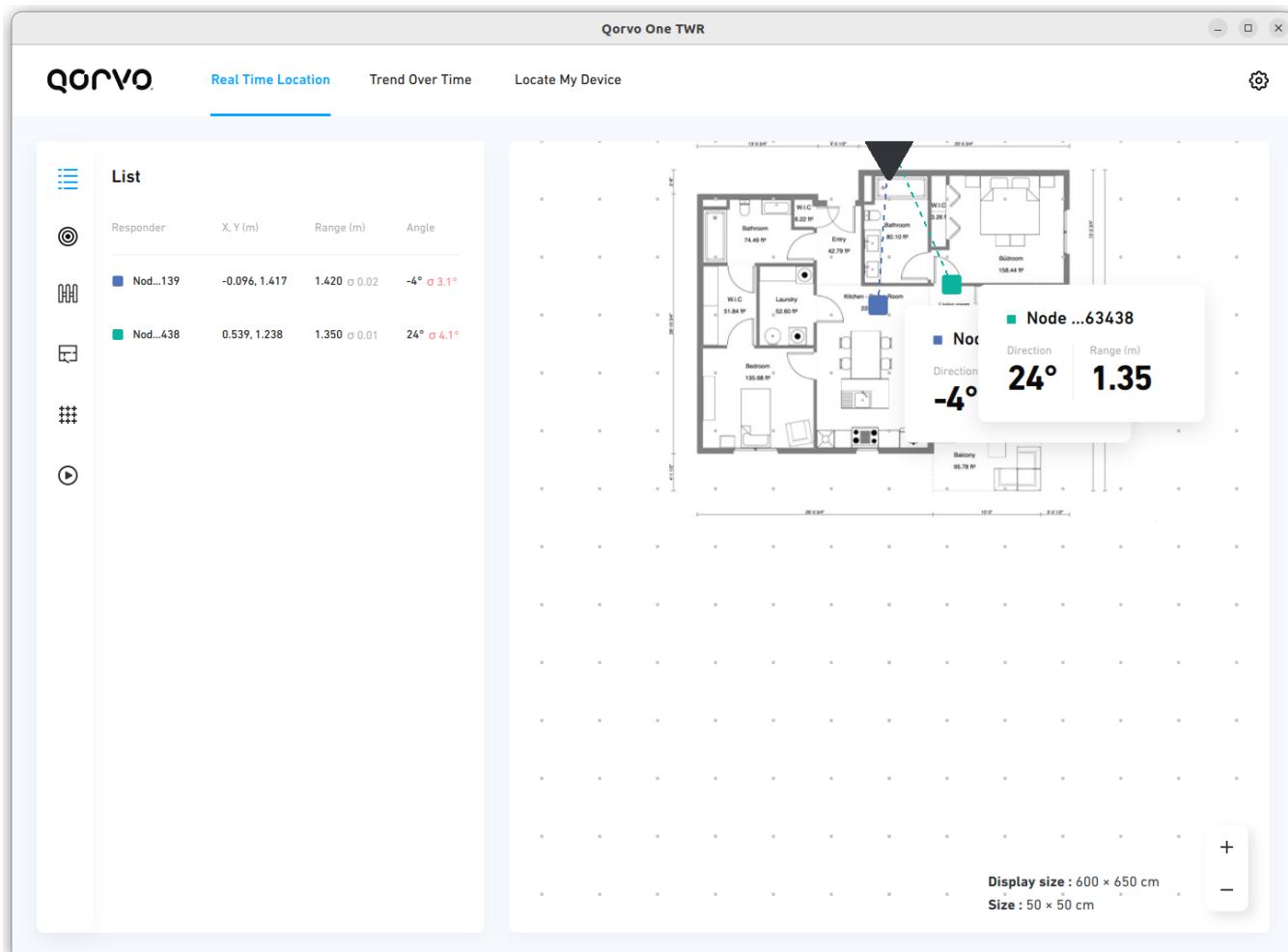


Figure 8.18: List of the responders.

The list of the responders can be displayed by activating the **List** card on the left bar. The x, y coordinates and the polar coordinates are shown for each responder as well the standard deviation of the distance and angle. Responder can be identified by the color and by the name which can be edited by double clicking.

The standard deviation σ changes color to red when the corresponding value exceeds limit. The limit for the standard deviation for the distance is 0.03 meters and 3 degrees for the angle.

8.2.8 Calibration

Devices can be calibrated using the **Calibration** card on the left bar. The calibrated device has to be placed at 2 meters distance from the device that will be used as the peer (the calibrator) during the procedure. The responder always uses the initiator as the calibrator.

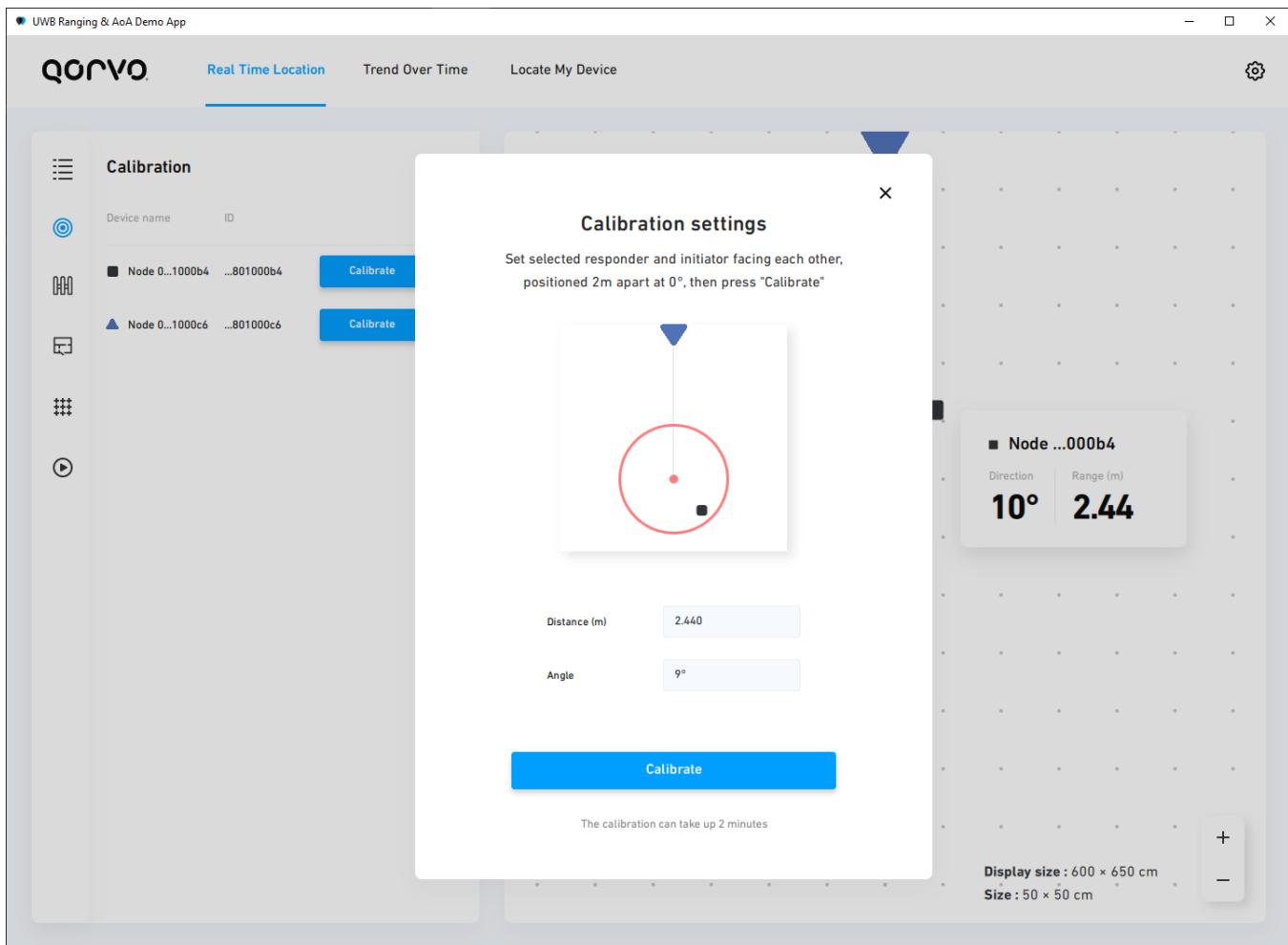


Figure 8.19: Calibration popup.

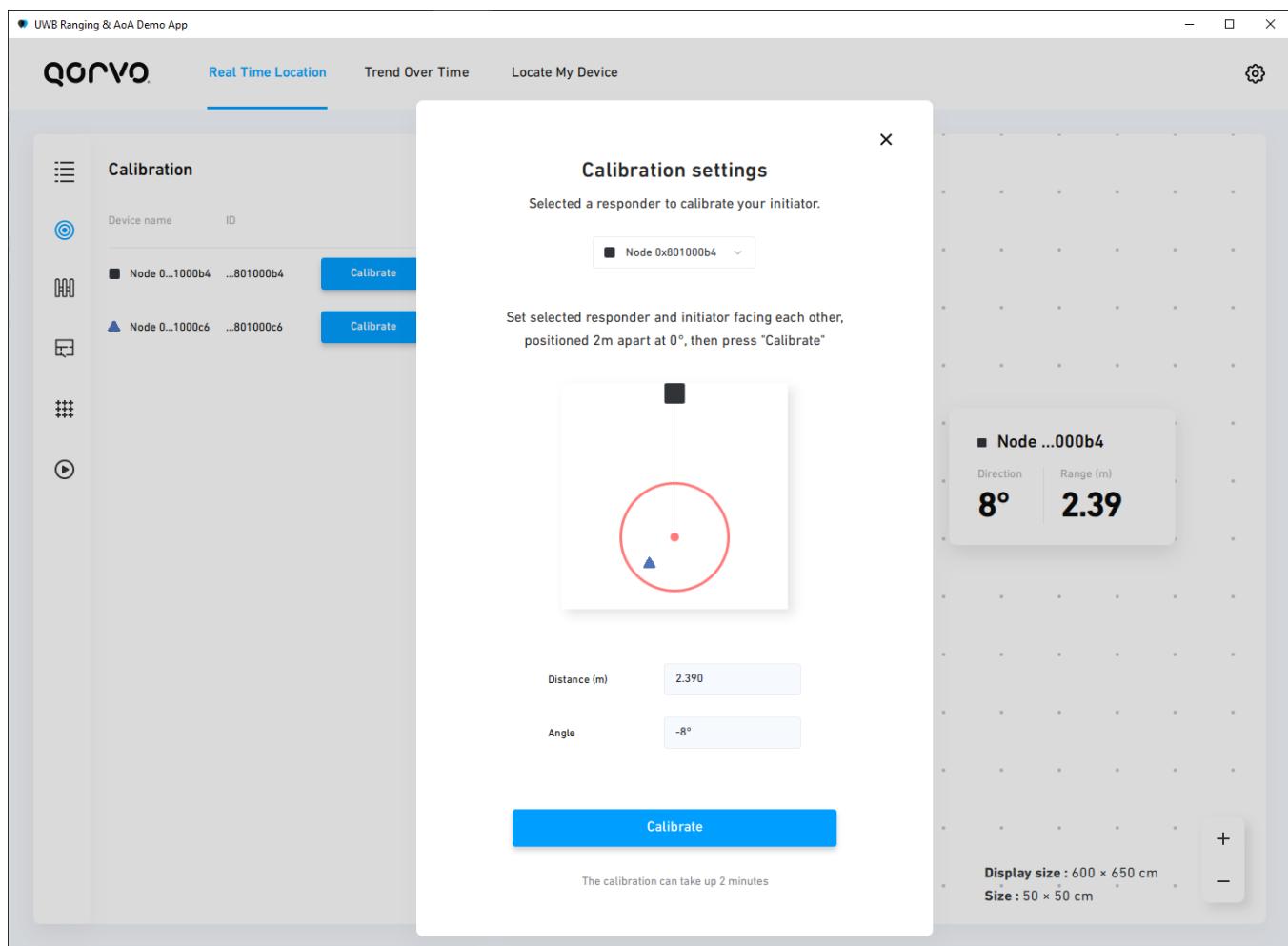


Figure 8.20: Calibration of the initiator.

Initiator can choose the calibrator from one of the responders that are connected locally to the application. A device that is connected to the distant host cannot be used for calibration.

The time the calibration procedure can take depends on the ranging duration of the [FiRa configuration](#).

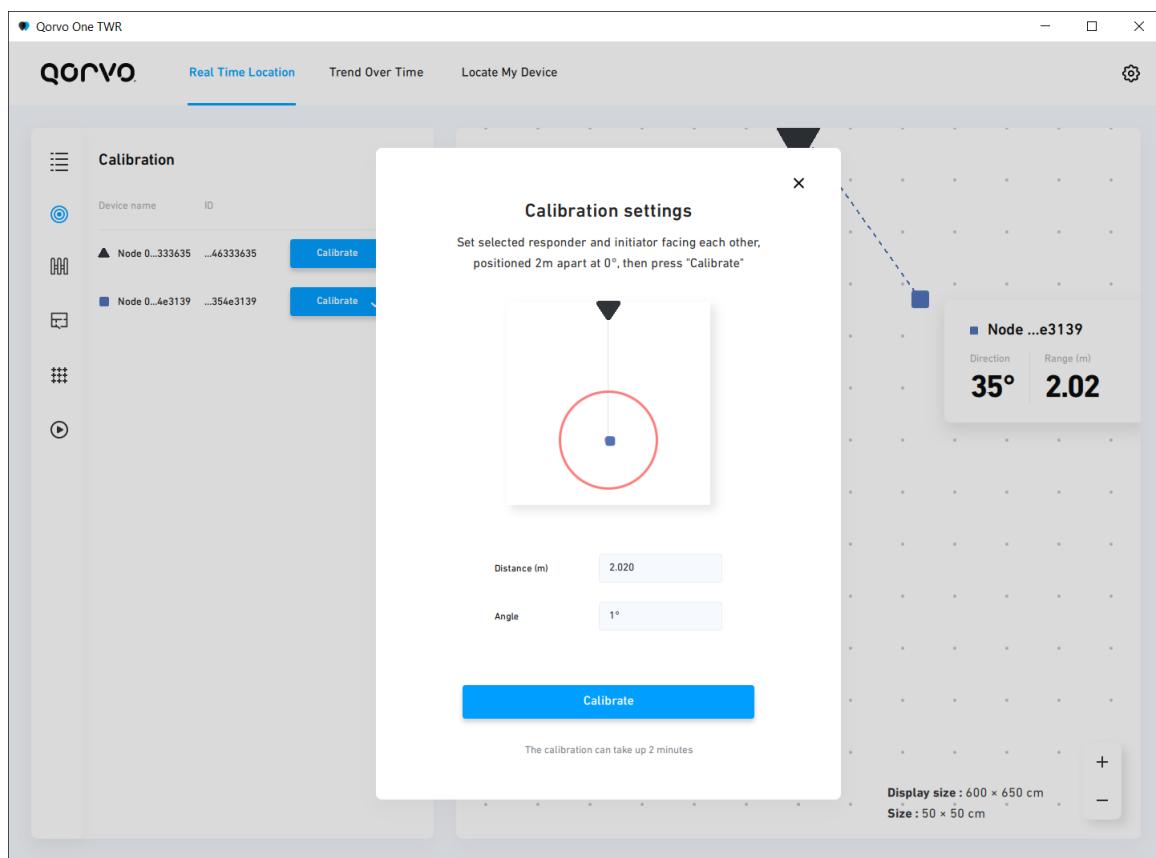


Figure 8.21: Successful calibration.

The state of the calibration is displayed by the button. Closing the popup will stop the ongoing calibration. The calibration can fail either due to timeout or due to communication error with the devices.

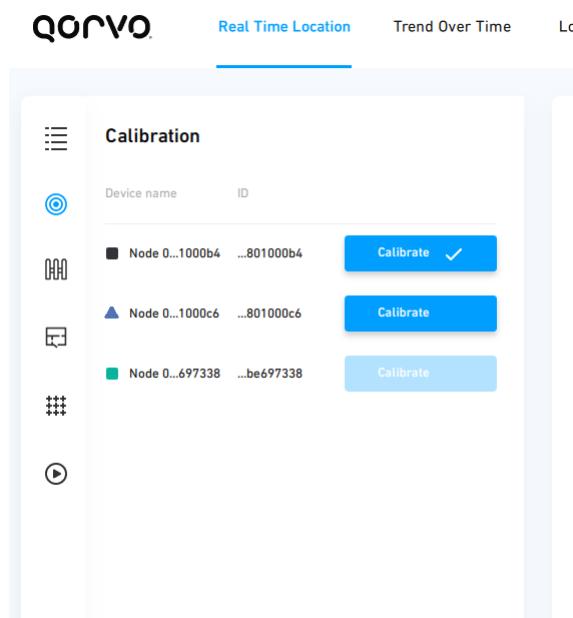


Figure 8.22: List of devices for the calibration.

Device that has been already calibrated is marked by the check icon in the calibration button. Device that does not support calibration have the button disabled.

8.2.9 Geofencing

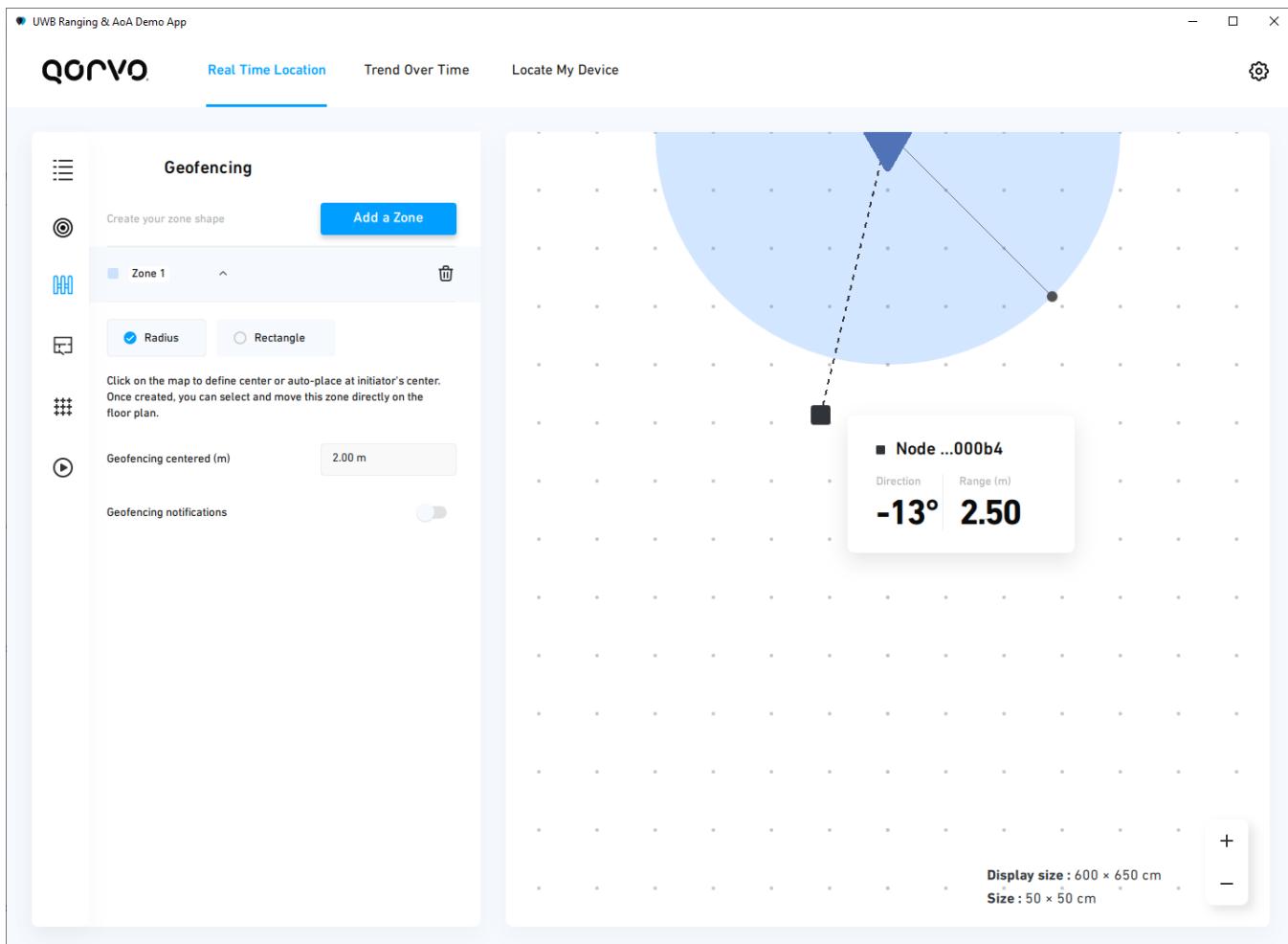


Figure 8.23: Radius zone.

Up to 5 geofencing zones can be added to the *Real-Time Location* screen. To add or to remove the zone, click the **Geofencing** icon on the left bar. The radius zone is created either by clicking twice on the screen to place the center and the radius or by writing the radius value in meters into the dialog in which case the center of the zone is placed automatically at the initiator.

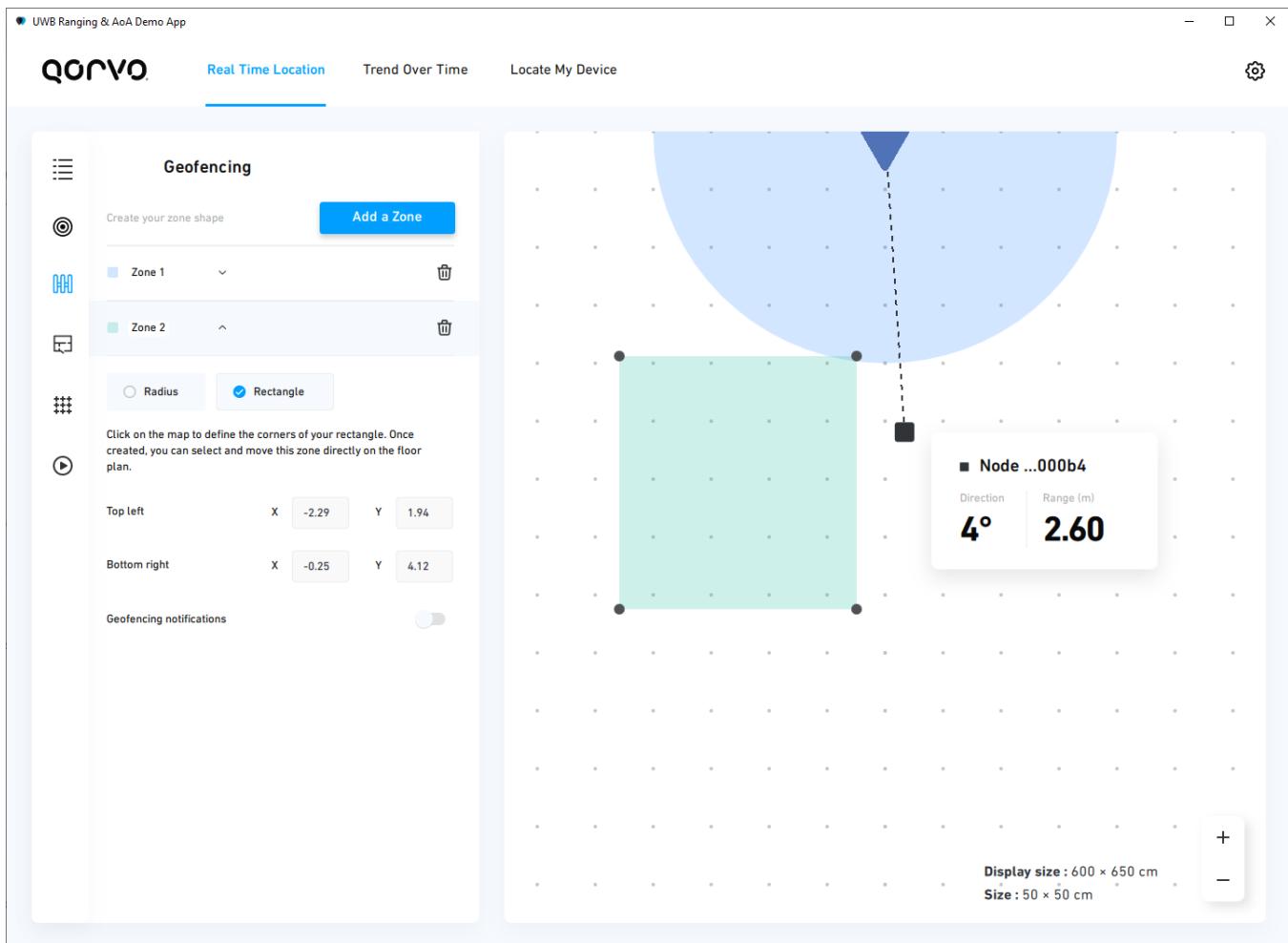


Figure 8.24: Rectangular zone.

Rectangular zone can be created by clicking on the screen to define top left and bottom right corners or by specifying the coordinates in the text edit. Each zone can be resized, moved or configured when activated by clicking on the icon next to the zone name. The name of the zone can be edited by double clicking on the name.

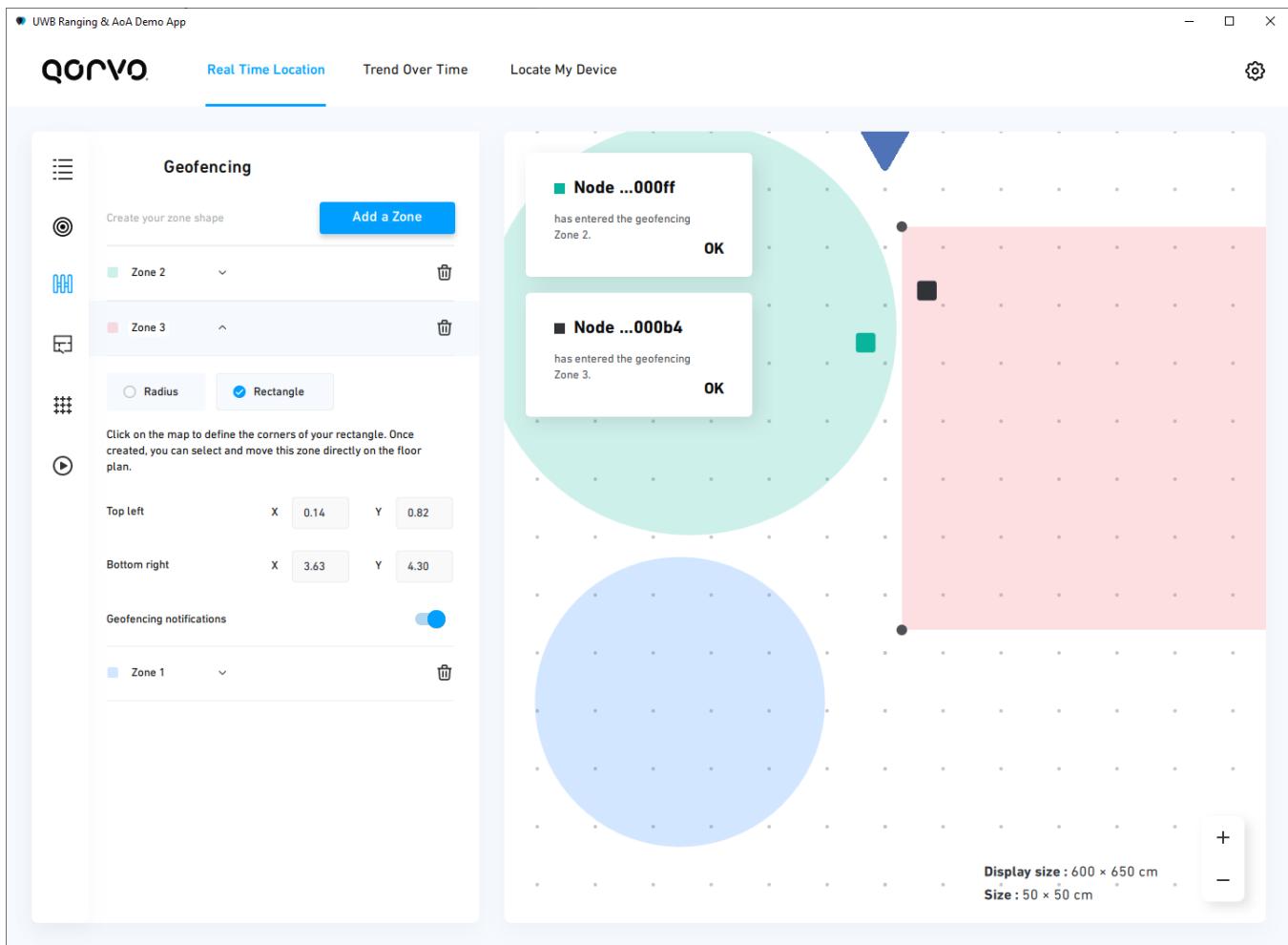


Figure 8.25: Geofencing notifications.

The notifications about the responders entering the zone can be enabled by clicking the button in the zone card. The notification disappears automatically after 3 seconds.

8.2.10 Floor Plan

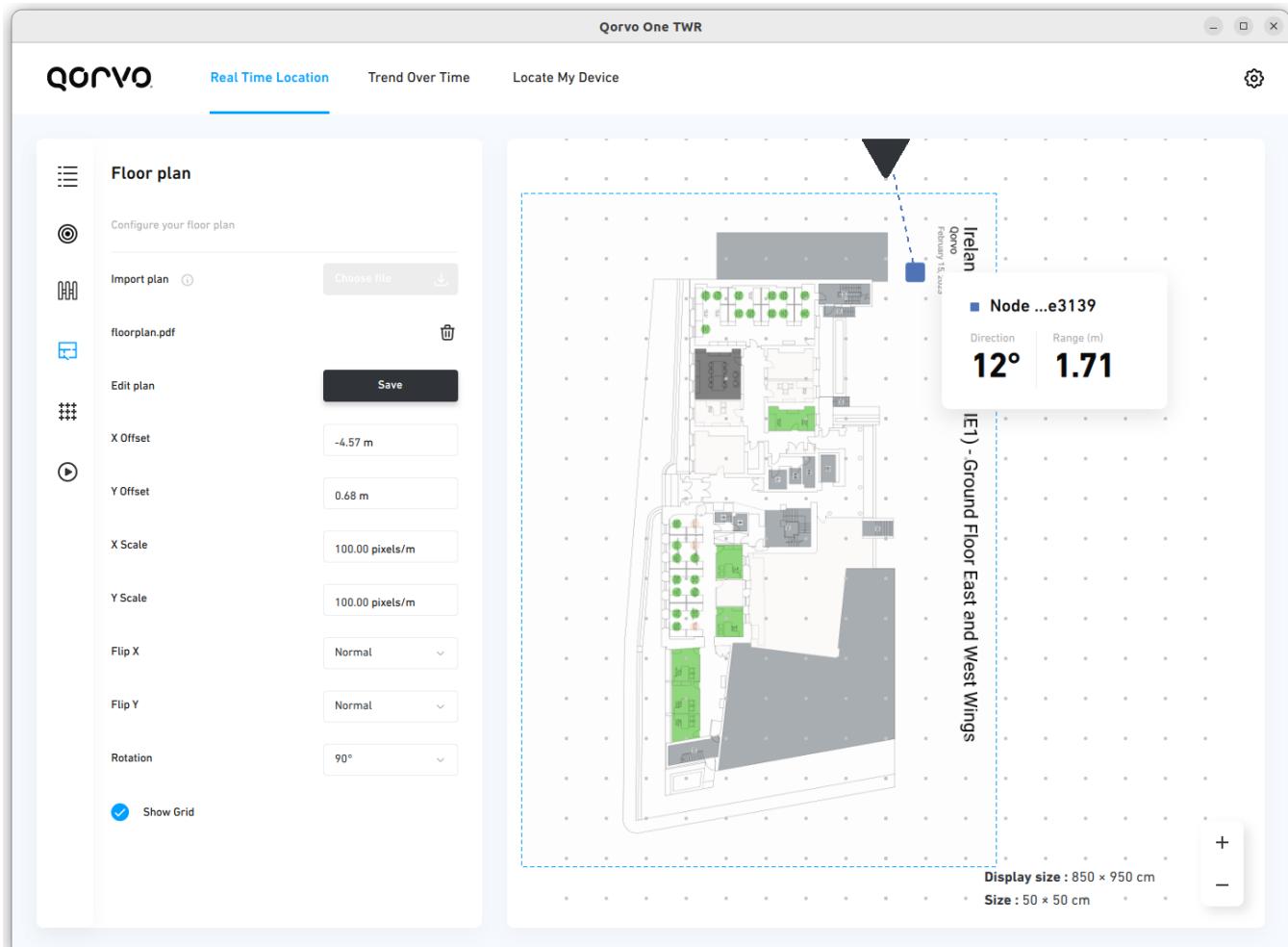


Figure 8.26: Floor plan configuration.

The application supports following file types to be imported as floor plan: png, jpeg, jfif, jpg, jpe, tiff, tif, pdf, gif, bmp. The maximum file size is limited to 50 MB. The plan can be resized, flipped and rotated. The imported image is displayed on the *Real-Time Location* screen. The plan can be deleted by clicking on the trash icon.

8.2.11 Grid

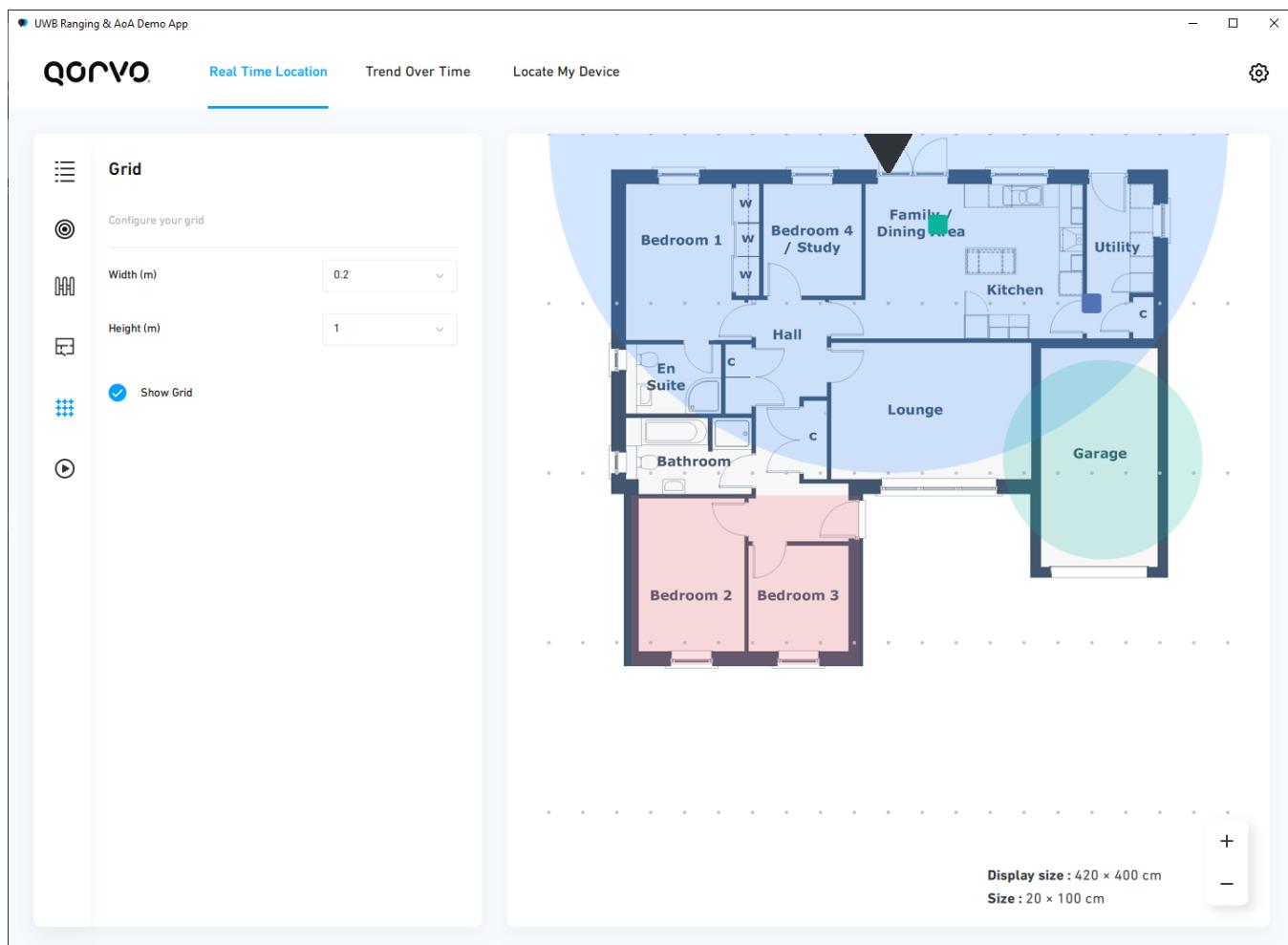


Figure 8.27: The grid configuration.

The **Grid** card on the left bar allows the grid density and visibility to be configured. The grid is displayed on the *Real-Time Location* screen and on the *Locate My Device* screen. Total size of the grid and the grid resolution is shown in the bottom right corner.

The screen can be zoomed in and out either by clicking on the +/- button in the bottom right corner or using the mouse wheel together with the Ctrl key.

8.2.12 Logging

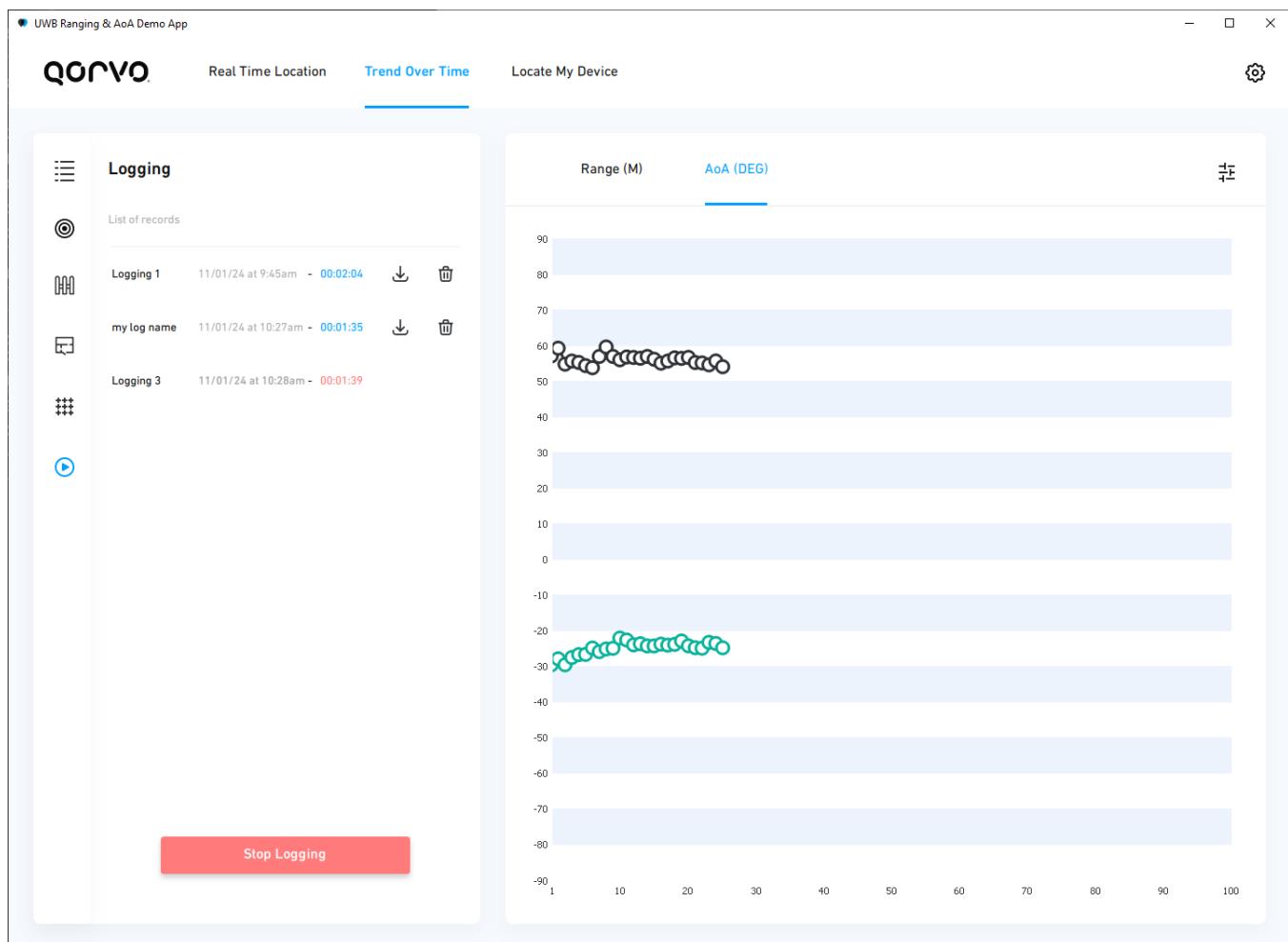


Figure 8.28: List of logging records.

Logging is started automatically by the application when opened. New logging record can be appended to the list by clicking on the **Start Logging** button. The logging record that is not currently active can be deleted or exported by clicking on the icons in the tab.

The logging is exported as two files. One file contains the UCI communication frames. The second file contains the main logging messages. Name of the logging record can be edited by double clicking on the name.

9 UWB Qorvo Tools

9.1 Overview

The UWB Qorvo Tools gathers various tools provided by Qorvo for operating Ultra-Wideband (UWB) devices through the UCI (UWB Communication Interface). These tools are located in the `SDK/Tools/uwb-qorvo-tools` directory.

Warning: The UWB Qorvo Tools are delivered by Qorvo as UCI examples on an as-is basis and are not optimized nor meant to be used as production software.

Note: A separate documentation is provided specifically for the UWB Qorvo Tools. For information on how to use the UWB Qorvo Tools, please refer to this documentation located in `SDK/Tools/uwb-qorvo-tools`.

Warning: If you are not using a QM33120WDK1, DWM3001CDK or Type2AB EVB (e.g. DWM3000EVB), you must flash the calibration before starting a ranging session.

Warning: Using the default calibration leads to decreased ranging accuracy (see section [Calibration important notes](#) for more details).

10 UCI - UWB Command Interface

10.1 Overview

In UCI project, UCI will be the only available application and active by default.

For details about UCI protocol, please refer to:

- [UCI¹⁹](#) generic specifications can be found in [FiRa²⁰](#) website (membership is needed to download the documents)
- [uwb-uci-messages-api](#) document

¹⁹ <https://groups.firaconsortium.org/wg/members/document/folder/96>

²⁰ <https://www.firaconsortium.org/>

11 SDK Runtime

11.1 Always running Threads

Once RTOS kernel has been started, the following threads are always running:

- Default Task
- Control Task
- Flush Task
- Logger Task
- Tmr Svc Task
- Idle Task

11.1.1 Default Task

The Default Task is responsible for starting one of the top-level applications. It receives events from the Control task to switch to a particular mode of operation and starts corresponding top-level application.

The Default task waits for a global event from EventManager, with a MESSAGE_QUEUE_TIMEOUT_MS (200ms) timeout.

- If an event is received within the timeout, Default task stops all running top-level tasks and it starts the requested top-level application.
- If no event is received within the timeout, Default task executes the USB_VBUS driver, which checks whether the VBUS pin of the MCU is attached to a power source or not.

This task is executed endlessly.

11.1.2 Control Task

The Control Task is responsible for reception and execution of commands from external IO interfaces USB and/or UART.

On reception of a valid command by the Control Task, the command is processed, and the corresponding reply is sent for output by triggering Flush task.

11.1.3 Flush Task

The Flush Task is responsible for transmitting any output data to the external I/O interfaces USB and/or UART. Any functions (including ISR functions) or tasks can request to send data to the non-blocking output (USB and/or UART).

When `port_tx_msg` function is called, the data is copied to an intermediate Report Buffer, which is a statically allocated circular buffer. The size of Report buffer is sufficient that any task/function can send a chunk of data for background output without delaying its throughput, even during an ISR.

The Flush task is then emptying the Report buffer onto the USB and/or the UART.

11.1.4 Logger Task

The Logger Task is a low-priority task which periodically calls `NRF_LOG_PROCESS()` function. It is mainly responsible for processing and transmitting log entries from a queue of deferred logs and flushing processed logs into configured backends (RTT by default). The task is running when `CONFIG_LOG` and `NRF_LOG_ENABLED` are defined.

11.1.5 Tmr Svc Task

The Timer Service Task manages software timers in FreeRTOS. It is created on scheduler start and used by FreeRTOS kernel.

11.1.6 Idle Task

The Idle Task is a low-priority task that runs when no other tasks are ready to execute in FreeRTOS. It handles system housekeeping functions like freeing memory from deleted tasks and can enter low-power modes to save energy. Users can also add custom background operations via an idle hook function.

11.2 Application specific Threads

11.2.1 Listener Task

The Listener Task is running only when Listener application is started in CLI build. It runs Listener application to manage exchange of data between DWT UWB drivers and the host machine. For more detailed information about application, please refer to [UWB sniffer: LISTENER](#).

11.2.2 LLHW MCPS Task

LLHW MCPS Task is used to handle signals and callbacks between LLHW and MAC layer. Task is created on `llhw_mcps_init()` call and used internally by LLHW and UWB stack. Typically, this task is running when INITF/RESPF application is started in CLI build or from the startup of the UCI build.

11.2.3 Qworkqueue Task

The Qworkqueue is a task holding a queue which manages work that needs to be performed outside the context of the interrupt service routines. Qworkqueue API can be found in **QOSAL** library. During the SDK runtime, multiple qworkqueues are created when UWB stack is used (when INITF/RESPF application is started in CLI build or from the startup of UCI build). They are used internally by UWB stack e.g. to handle UCI data to be reported from stack to the host.

11.2.4 UCI task

The UCI Task is responsible for opening UCI backends and UCI server inside the uwb-stack at the startup. During the runtime, its primary role is to transfer UCI messages from UART/USB interface to uwb-stack. This task is running from the startup of the firmware and it operates only in the UCI build.

11.3 Threads stack usage

11.3.1 FreeRTOS

The table below shows the threads stack usage of the tasks used in FreeRTOS in two different cases:

- Idle state (no applications running)

THREAD NAME	Stack usage
Control	968/2048
IDLE	164/512
Flush	184/512
Logger	100/768
Default	172/4304
Tmr Svc	156/512
Total HEAP	51200
Current HEAP used	8824
Max HEAP used	8824

- INITF application is running

THREAD NAME	Stack usage
Control	968/2048
IDLE	164/512
FLush	220/512
Logger	112/768
qworkqueue	212/3072
llhw_mcps	1200/2052
qworkqueue	2064/3072
Tmr Svc	156/512
Default	3300/4304
Total HEAP	51200
Current HEAP used	20000
Max HEAP used	21440

12 SDK SW Architecture

12.1 Overview

The SDK platform supports FreeRTOS on ARM Cortex M4. The sections below discuss the architecture, structure and workflow of the software. This should enable the developer to understand the philosophy and be able to add functionality or port the project to other platforms if needed.

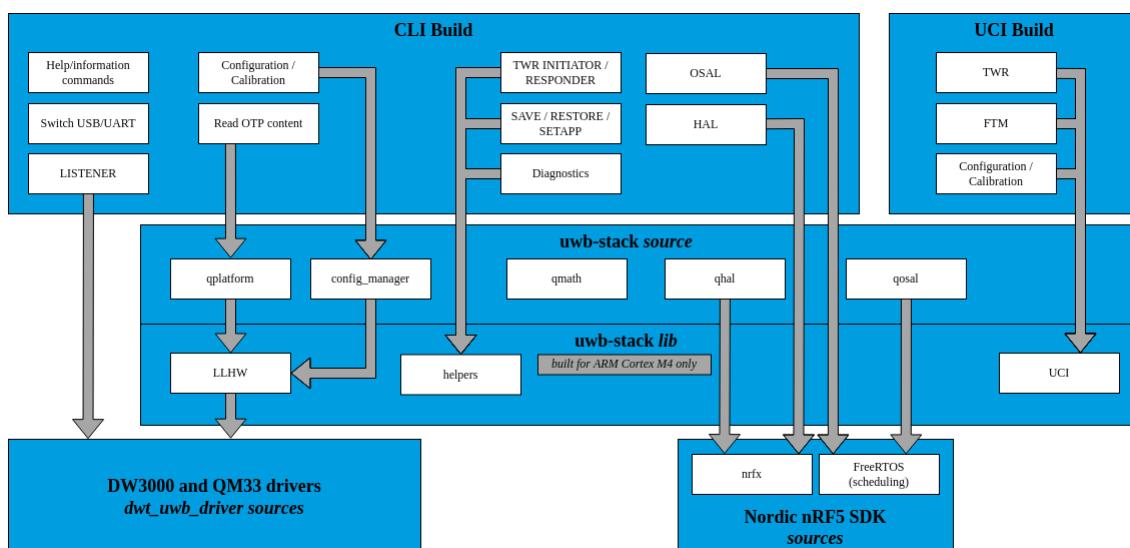


Figure 12.1: Overview of FreeRTOS based architecture.

The figure above shows the layered structure: “Top-level applications”, “FreeRTOS”, “UWB Stack”, “HAL” and “SDK BSP”.

12.2 Linker sections

The SDK relies on linker sections to automatically register applications, drivers, NVM configurations, etc... This section defines how to modify your linker section to handle this. The linker description file (ld file) is located in your project folder. See section [Projects folder](#) for details.

12.2.1 Flash sections

12.2.1.1 ISR Vectors section

This section contains the traditional ISR vector table. It starts with a vector array defined with symbol .isr_vector. It is usually located at the Flash entry address.

12.2.1.2 NVM section

This section is a placeholder for storing configuration which will not be lost between a power cycle.

It starts with symbol __fconfig_start and ends with symbol __fconfig_end.

It is mandatory to ensure that the __fconfig_start is aligned at the start of a flash page and that a full page is reserved for this purpose.

Only one variable will be explicitly placed in this section, defined like this:

```
__attribute__((section(".fconfig"))) const uint32_t DummyConfig[1] = {0xDEADBEEF};
```

Its role is to guarantee that the checksum will be incorrect at first powerup after download, to force a factory default to take place.

See section [NVM configuration](#) for details about NVM save and restore mechanism.

12.2.1.3 Applications and Commands section

This series of sections will determine which applications are available for the current build.

They are placed in a way that the FW browses them and displays them in the correct order when executing HELP command.

See section [HELP](#) for details.

12.2.1.4 Applications section

This series of sections will determine which applications are available for the current build.

12.2.1.5 Applications NVM Configuration section

This series of sections will determine which applications are available for the current build.

The first sub-section will handle all the commands which are allowed to execute at any time.

It starts with symbol __known_commands_start.

It contains the default function defined with symbol .known_commands_anytime.

The second sub-section will handle all the commands which are considered as Applications.

It starts with symbol __known_commands_app_start.

It contains the default function defined with symbol .known_commands_app.

The third sub-section will handle all the sub-commands of individual applications.

It starts with symbol __known_app_subcommands_start.

It contains the default function defined with symbol .known_app_subcommands.

The fourth sub-section will handle all the commands that can only be ran if we are in idle state (meaning, no application running).

It starts with symbol __known_commands_ilde_start.

It contains the default function defined with symbol .known_commands_ilde.

The fifth sub-section will handle all the commands considered service commands.

It starts with symbol __known_commands_service_start.

It contains the default function defined with symbol .known_commands_service.

Lastly, the symbol __known_commands_end will mark the end of this section.

12.2.1.6 Restoring default Configuration section

This section is a placeholder for storing configuration default functions.

It starts with symbol __config_entry_start and ends with symbol __config_entry_end.

It contains the default function defined with symbol .config_entry.

When the user will call RESTORE command (see [RESTORE](#)) or if the version is invalid, then a factory default will be restored, including the calibration values.

To do so, each function listed in this section will be executed.

For example, for FiRa application, the function below will be executed.

```
static void restore_fira_default_config(void)
{
    memcpy(&fira_config_ram, &fira_config_flash_default, sizeof(fira_config_ram));
}

__attribute__((section(".config_entry"))) const void (*p_restore_fira_default_config)(void) =_
→(const void *)&restore_fira_default_config;
```

12.2.2 RAM sections

12.2.2.1 Applications NVM Configuration section

This section is a placeholder for every parameter an application wants to store into NVM.

It starts with symbol __rconfig_start and ends with symbol __rconfig_end.

One structure uses .rconfig, which contains all parameter and attributes to run the application:

```
static dwt_app_config_t dwt_app_config_ram __attribute__((section(".rconfig"))) = {0};
```

See section [NVM configuration](#) for details about NVM save and restore mechanism.

12.3 NVM configuration

12.3.1 Overview

- The way the configuration is handled inside the SDK is done through linker sections.
- All the variables that have to be stored in NVM are stored in a consecutive RAM section.
- This RAM section is copied as a unique block inside NVM.
- At Power-up, the configuration is loaded from NVM and copied inside this RAM section.

- Sending the SAVE command will write the RAM configuration block into Flash.
- Sending RESTORE command will restore the default configuration along with the default calibration values, please check [Calibration important notes](#).

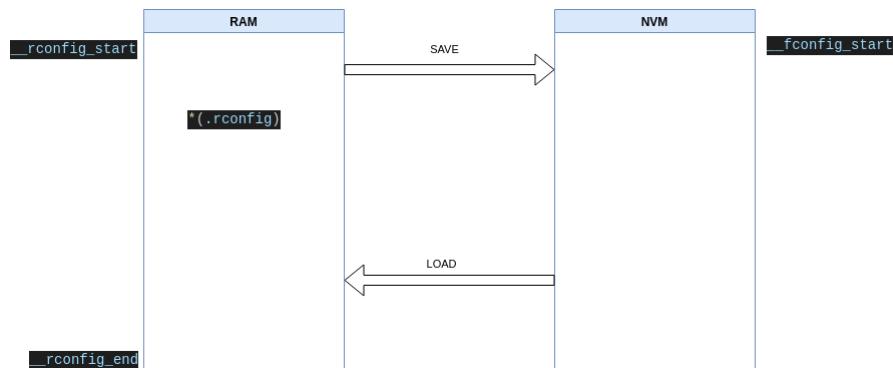


Figure 12.2: NVM configuration

12.3.2 Initialization sequence during powerup

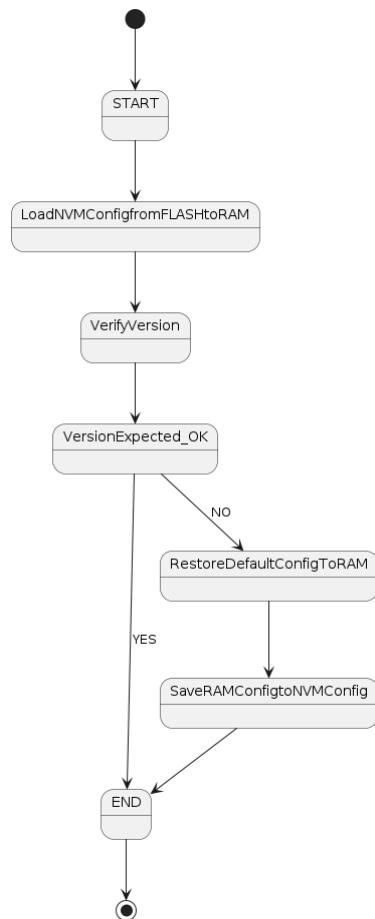


Figure 12.3: NVM boot sequence

12.4 Folders structure

The table below shows how the source code is distributed among the different folders.

root	
Libs	Components provided as libraries
dwt_uwb_driver	Libraries, UWB chip driver layer (sources)
niq	Libraries, Nearby interaction qorvo module layer (static)
libraries	
uwbstack	Libraries, MAC layer (sources)
uwbstack_libs	Libraries, MAC layer (static libraries)
LICENSES	Software Licenses
Projects	Contains different projects grouped by OS and target
Common	Common project files for all projects
FreeRTOS	Contains FreeRTOS based projects
SDK_BSP	Board support package provided by MCU manufacturer
Nordic	Nordic BSP folder
Src	
AppConfig	Load/Save/Restore apps configuration
Apps	Implementation of the apps
Boards	API for the board's specific implementation
Bootloader	Bootloader resources
Comm	Projects Communication resources
EventManager	Event manager functionalities implementation
HAL	Hardware Abstraction Layer
Inc	HAL API header files
Src	HAL API source files
Helpers	Generic Helpers (JSON, CRC, DBG, ...)
Logger	Logger implementation
OS	OS
OSAL	OS Abstraction Layer
UWB	UWB project's utilities implementation

12.5 Layers

12.5.1 AppConfig API

This folder contains what is needed to store non-volatile configuration into Non-Volatile Memory (NVM).

The way to save data to NVM is Board dependent (what flash memory is used).

For this reason, we implemented this as standalone folder to offer the required flexibility.

A user can then adapt the code to match their requirements.

The key file is located inside SDK/Firmware/Src/AppConfig/Src/config.c.

Please also see the [NVM configuration](#) section on NVM configuration for further details.

12.5.1.1 API

The API header file is located inside SDK/Firmware/Src/AppConfig/Inc/appConfig.h

It contains three functions:

```
void load_bssConfig(void)
```

This function will load config from FLASH to RAM.

```
void restore_bssConfig(void)
```

This function will restore default config to RAM and save it back to FLASH.

```
error_e save_bssConfig(void)
```

This function will save the RAM config to FLASH.

Please also see [NVM configuration](#).

12.5.1.2 Linker symbols used

```
extern uint8_t __config_entry_start[];  
extern uint8_t __config_entry_end[];  
extern uint8_t __fconfig_start[];  
extern uint8_t __rconfig_start[];  
extern uint8_t __rconfig_end[];
```

Please also see [Linker sections](#).

12.5.2 Applications layer (Apps)

12.5.2.1 common folder

See also [Always running Threads](#)

12.5.2.2 FiRa demo application: INITF / RESPF

INITF (Initiator FiRa) is an application that sets the unit as initiator in a FiRa two-way ranging session.

RESPF (Responder FiRa) is an application that sets the unit as responder in a FiRa two-way ranging session.

12.5.2.3 UWB sniffer: LISTENER

LISTENER is an application that sets the unit into receive mode and it reports any received data.

12.5.3 BLE Folder

BLE folder contains the source code to all components of the BLE layer:

- Initialise/handle the BLE Peripheral
- Define the Qorvo NI BLE service
- Define the Apple NI BLE service

The BLE Communication layer is designed to run all the steps of a BLE connection based on Qorvo NI Service (QNIS). The iOS app can filter the BLE devices advertising by their Unique ID (UUID) and connect only with those running QNIS.

12.5.3.1 Nordic BLE Stack

The Nordic BLE stack is part of Nordic SDK, and it includes all functions to configure and initialise the Nordic BLE peripheral. To make the Qorvo NI device act as a peripheral, BLE advertising and connecting to the iPhone, the following functions are called in `ble_init()`.

```
static void ble_stack_init(void)
```

To be called during the initialisation. It calls the following functions from the BLE stack:

- `nrf_sdh_enable_request()`: Function for requesting to enable the SoftDevice;
- `nrf_sdh_ble_default_cfg_set()`: Configures the BLE stack with the settings specified in the SoftDevice handler BLE configuration;
- `nrf_sdh_ble_enable()`: Configure and enable the BLE stack;
- `NRF_SDH_BLE_OBSERVER()`: Register the `ble_evt_handler()` as a handler for BLE events.

```
static void gap_params_init(char *gap_name)
```

This function sets up all the necessary GAP (Generic Access Profile) parameters of the device including the device name, appearance, and the preferred connection parameters:

- `sd_ble_gap_device_name_set()`: Set the device name from the string given as parameter;
- `sd_ble_gap_appearance_set()`: Set in GAP a predefined value for appearance, to inform the Central the nature of the device;
- `sd_ble_gap_ppcp_set()`: Set GAP Peripheral Preferred Connection Parameters.

```
void gatt_init(void)
```

Initialises the Generic Attribute Profile (GATT) module. The GATT defines how two BLE devices exchange data using Services (QNIS and ANIS for this project) and Characteristics. This function calls:

- `nrf_ble_gatt_init()`: Initialise the GATT and defines the `gatt_evt_handler()` as its event handler;
- `nrf_ble_gatt_att_mtu_central_set()`: Set the ATT_MTU size for the next connection that is established as central.

```
static void services_init(void)
```

Function to initialise the Services, it set the `qnis_init` structure and use it as an input parameter for `ble_qnis_init()` and calls `nrf_ble_qwr_init()` to initialise the Queued Writes module.

```
static void advertising_init(void)
```

Working as Peripheral, the Qorvo NI device implements advertising to be detected by the Central. For this, this function calls `ble_advertising_init()` with the `ble_advertising_init_t` init as input parameter, and `ble_advertising_conn_cfg_tag_set()` to configure the connections settings tag.

12.5.3.2 BLE Services

Initialised by `services_init()`, Qorvo NI device uses two services to communicate with the iOS app, QNIS (Qorvo NI Service) and ANIS (Apple NI Service). QNIS is used to exchange the out of band messages and it is the only service advertised.

ANIS is defined by Apple in their “Nearby Interaction Accessory Protocol Specification Release R2” and it is required only to ranging in Background Mode.

Note: It is possible to implement the OOB communication using a channel other than BLE (Wi-Fi, Internet, etc.) but to use Background mode BLE is mandatory, as the iOS device will periodically check the ANIS service.

QNIS

The QNIS service has three characteristics: RX, TX and a Secure characteristic:

- RX Characteristic has the properties “Write Request” and “Write Command” permitted, and security level “Open” for Read and Write;
- TX Characteristic has notification enabled and the same security level as RX (“Open” for Read and Write);
- The Security Characteristic is only used to trigger the pairing process on the iOS side. The security level is set to “Just Works”.

The 128-bit QNIS Service UUID (Universally Unique Identifier) is:

“2E938FD0-6A61-11ED-A1EB-0242AC120002”

The QNIS Characteristic UUIDs are:

RX Characteristic:

“2E93998A-6A61-11ED-A1EB-0242AC120002”

TX Characteristic:

“2E939AF2-6A61-11ED-A1EB-0242AC120002”

Secure Characteristic:

“2E93941C-6A61-11ED-A1EB-0242AC120002”

ANIS

The ANIS service has only one characteristic, Accessory Configuration Data (ACD). The characteristic has the property “Read Request” permitted, and its value is set to the “Accessory Configuration Data” payload when populated and sent to iOS before start ranging (function `set_accessory_uwb_config_data()` in `ble_anis.c`). The security level is set to “Just Works”.

The 128-bit ANIS Service UUID is:

“48FE3E40-0817-4BB2-8633-3073689C2DBA”

And the ACD Characteristic UUID is:

"95E8D9D5-D8EF-4721-9A4E-807375F53328"

Note: For details on the Apple Nearby Interaction Service implementation, please refer to the Apple® Nearby Interaction Accessory Protocol Specification Release R2 (referenced_documents).

Warning: If using an auxiliary BLE sniffer during implementation, be aware that ANIS service cannot be seen by iOS devices running iOS 16.

12.5.4 Board Abstraction Layer (Boards)

Board Abstraction Layer (BAL) allows to customize the middleware initialization and logic for the target. Each board supported by SDK has its dedicated folder that contains modules which implement the BAL API.

Currently, the BAL layer is implemented and defined in two locations:

- **Board Components** SDK/Firmware/Src/Boards - implementations and customization of generic part of BAL for specific targets,
- **Qplatform Component** SDK/Firmware/Libs/uwb-stack/libs/qplatform - implementations of the uwb-stack specific part of BAL for specific targets.

The **Board Component** implements the generic platform initialization logic and provides the means for customization of the generic BAL related part (as well as customization of the HAL).

The **Qplatform Component** implements the uwb-stack initialization related to platform, so it provides the means for customization of the uwb-stack related part of BAL.

If the implementation for the BAL for the given platform does not already exist, the implementation is required to be created. In this case, please refer to [Porting BAL](#).

12.5.4.1 Board Component

12.5.4.1.1 Files

The structure of the Board Component, located in the SDK/Firmware/Src/Boards folder, is as follows:

root	
└ Src	
└ Boards	Boards Abstraction Layer
└ Inc	HAL API header files
└ int_priority.h	Interrupt priorities definition
└ thisBoard.h	Board API header file
└ Src	Boards source files
└ Common	Common code for all board
└ helper_platform_l1_config.c	Common helpers to develop platform L1 Config
└ helper_platform_l1_config.h	
└ Common OTP	Common OTP code
└ OTP_config.h	
└ <target_1>	<target_1> specific code
└ CMakeLists.txt	
└ Common	
└ custom_board.h	<target_1> BAL customization header

(continues on next page)

(continued from previous page)

<pre> └── FreeRTOS └── <target_1>.c platform_l1_config.c └── <target_2> ├── CMakeLists.txt ├── Common │ └── custom_board.h ├── FreeRTOS │ └── <target_2>.c └── platform_l1_config.c ... </pre>	<pre> <target_1> specific BAL implementation <target_1> specific l1 config ops <target_2> specific code <target_2> BAL customization header <target_2> specific BAL implementation <target_2> specific l1 config ops </pre>
--	---

12.5.4.1.2 API definition

thisboard.h

The main() function will call two key functions during the initialization phase:

```

void BoardInit(void);
void board_interface_init(void);

```

custom_board.h

This file is a key file in target SDK mechanism.

It is used to configure the target SDK the way we need for our project, such as pinout assignment and other features.

int_priority.h

This file contains the definitions of thread priorities.

Other important files

Files used to customize a particular target:

12.5.4.2 Qplatform Component

For the detailed description of the **Qplatform component** please refer to *Documentation/uwb-qplatform-api.pdf*

The customization of the **Qplatform components** is done by means of: `SDK/Firmware/Projects/<OS>/<example>/<target>/ProjectDefinition/uwb_stack_l1hw.cmake` (ProjectDefinition target).

Below is an example table of the components customized by ProjectDefinition target in `uwb_stack_l1hw.cmake` file for nRF52840DK board.

LLHW drv definitions
Nordic SDK definitions SPI
GPIO
CLOCK
ARM SYSTICK
TIMER
DTU Base timer
UWB stack Idle timer
HAL timer
HAL FS timer
LOG timestamp Timer
RTC
RTC conf.
Watchdog
NMV Flash Controller
USB Controller

12.5.5 Hardware Abstraction Layer (HAL)

HAL (Hardware Abstraction Layer) provides all the services needed by the application layer so the access to hardware resources is agnostic.

12.5.5.1 Overview

The Hardware Abstraction Layer is designed to make the application adaptable to any hardware it may be running on. All the possible hardware peripherals have their own HAL files.

In general, project defines its own *HAL Layer* implementation, but a part of the HAL Layer implementation is delegated to QHAL (Qorvo HAL), provided by uwbstack sublibrary Firmware/Libs/uwb-stack/libs/qhal (refer to Documentation/uwb-qhal-api.pdf). In other words, the *HAL Layer* in SDK is covered by two implementations of HAL: HAL and QHAL. The generic part of functionalities of the *HAL Layer* for the platform are covered by the HAL, the specific part of the *HAL Layer*, for components provided by uwb-stack, is covered by the QHAL.

Note: We are going to use following terminology:

HAL - part of *HAL Layer* defined in Firmware/Src/HAL/Src

QHAL - part of *HAL Layer* defined in Firmware/Libs/uwb-stack/libs/qhal

HAL Layer = HAL + QHAL

- Files location

root	
└─ Src	
└─ HAL	Hardware Abstraction Layer
└─ Inc	HAL API header files used
└─ Src	HAL API source files to adapt

root	
└─ Libs	
└─ uwb-stack	└─ libs

(continues on next page)

(continued from previous page)

└─ qhal	Qorvo Hardware Abstraction Layer
└─ Inc	QHAL API header files used
└─ Src	QHAL API source files to adapt

We provide *HAL Layer* implementation for nrf52 boards family.

If a use case is not covered in the supported list, then refer to [Porting HAL](#) for guidance on how to do the porting.

12.5.5.2 HAL UART

Use this API when your design is using real UART to interface to the host.

When your design is USB Virtual Com based, use HAL USB.

Note: This API is using a legacy style and will be refactored in a future release

12.5.5.2.1 Files

root	
└─ Src	
└─ HAL	Hardware Abstraction Layer
└─ Inc	HAL API header files
└─ Src	HAL UART API header file HAL API source files
└─ nrfx	HAL Nordic implementation
└─ HAL_uart.c	HAL UART implementation for Nordic

12.5.5.2.2 API definition

bool IsUartDown(void);
--

Get UART power state

Parameters: none

Return: TRUE if the UART is in low power mode

void SetUartDown(bool val);
--

Change UART low-power state

Parameters: power state

Return: none

void deca_uart_close(void);

Flush UART buffers and close it

Parameters: none

Return: none

```
void deca_uart_init(void);
```

Initialize UART as the main interface (chosen setup is 115200, 8, N, 1, no Hardware flow control)

Parameters: none

Return: none

```
void deca_uart_receive(void);
```

Receive UART data and put it in a buffer

Parameters: none

Return: none

```
int deca_uart_transmit(uint8_t *ptr, uint16_t sz);
```

Send data over UART

Parameters:

uint8_t *ptr: pointer to the buffer to send

uint16_t sz: size of the data to send

Return: error status

```
int deca_uart_putc(uint8_t car);
```

Send one character over UART

Parameters: character to send

Return: error status

```
void deca_uart_flush(void);
```

Flush UART buffer

Parameters: none

Return: none

12.5.5.3 HAL USB

12.5.5.3.1 Files

```

root
└── Src
    └── HAL
        ├── Inc
        │   └── HAL_usb.h
        └── Src
            └── target
                └── HAL_usb.c

```

Hardware Abstraction Layer
HAL API header files
HAL USB API header file
HAL API source files
HAL target implementation
HAL USB implementation for target

12.5.5.3.2 API definition

```
struct hal_usb_s
{
    void (*init)(void(*rx_callback)(uint8_t *ptr, size_t len));
    void (*deinit)(void);
    bool (*transmit)(uint8_t *tx_buffer, int size);
    void (*receive)(void);
    void (*update)(void);
    bool (*isTxBufferEmpty)(void);
};

extern const struct hal_usb_s Usb;
```

- **void** (*init)(**void**(*rx_callback)(**uint8_t** *ptr, **size_t** len));

USB initialization

Parameters:

void(*rx_callback)(**uint8_t** *ptr, **size_t** len): Optional USB reception callback

Return: none

- **void** (*deinit)(**void**);

USB deinitialization

Parameters: none

Return: none

- **bool** (*transmit)(**uint8_t** *tx_buffer, **int** size);

USB send data

Parameters:

uint8_t *tx_buffer: buffer containing data to send

int size: size of the data to send

Return: none

- **void** (*receive)(**void**);

USB receive data

Parameters: none

Return: none

- **void** (*update)(**void**);

USB update state machine

Parameters: none

Return: none

- **bool** (*isTxBufferEmpty)(**void**);

Check if there is nothing to send/sending

Parameters: none

Return: TRUE if tx buffer empty

12.5.5.4 HAL CPU

The HAL CPU module, located in SDK/Firmware/Src/HAL folder, provides common functionalities for families of CPUs that are used in the SDK. It includes the CPU protection function that enables or disables the Access Port Protection (APPROTECT) based on the compilation flag CONFIG_SOC_PROTECT.

12.5.5.4.1 Files

root		
└ Src		
└ HAL	Hardware Abstraction Layer	
└ Inc	HAL API header files	
└ HAL_cpu.h	HAL CPU API header file	
└ Src	HAL API source files	
└ nrfx	HAL Nordic implementation	
└ HAL_cpu.c	HAL CPU implementation for Nordic	

12.5.5.4.2 API definition

```
void handle_cpu_protect(void);
```

Configures Access Port Protection (APPROTECT) based on build settings.

Parameters: none

Return: none

Description:

Enables or disables the Access Port Protection (APPROTECT) feature, which secures the CPU by restricting access to the debug interface. The configuration is determined by the presence of the *CONFIG_SOC_PROTECT* compilation flag.

- If *CONFIG_SOC_PROTECT* is defined, APPROTECT will be enabled.
- If *CONFIG_SOC_PROTECT* is not defined, APPROTECT will be disabled.

The function ensures the appropriate configuration is applied and triggers a system reset if necessary to enforce the changes.

Note:

Ensure the *CONFIG_SOC_PROTECT* flag is correctly set during compilation to achieve the desired level of protection.

Warning:

Enabling APPROTECT will restrict debug access. Use with caution during development to avoid locking out debugging capabilities.

12.5.5.4.2.1 CPU Protection

```
/* Enable or disable APPROTECT (Access Port Protection) depending on a compilation flag. */
void handle_cpu_protect(void);
```

- **void handle_cpu_protect(void);**

This function enables or disables the Access Port Protection (APPROTECT) based on the compilation flag CONFIG_SOC_PROTECT.

When CONFIG_SOC_PROTECT is defined, the function enables the APPROTECT if it is not already enabled. Otherwise, it disables the APPROTECT.

12.5.5 HAL WATCHDOG

12.5.5.1 Files

root		
└─ Src		
└─ HAL	Hardware Abstraction Layer	
└─ Inc	HAL API header files	
└─ HAL_watchdog.h	HAL watchdog API header file	
└─ Src	HAL API source files	
└─ nrfx	HAL Nordic implementation	
└─ HAL_watchdog.c	HAL watchdog implementation for Nordic	

12.5.5.2 API definition

```
struct hal_watchdog_s
{
    void (*init)(int ms);
    void (*refresh)(void);
};

extern const struct hal_watchdog_s Watchdog;
```

- **void (*init)(int ms);**

Initialize watchdog

Parameters: Reload time in ms

Return: none

- **void (*refresh)(void);**

Refresh/reload watchdog

Parameters: none

Return: none

12.5.5.6 HAL ERROR

12.5.5.6.1 Files

root		
└ Src		
└ HAL	Hardware Abstraction Layer	
└ Inc	HAL API header files	
└ HAL_error.h	HAL ERROR API header file	
└ Src	HAL API source files	
└ nrfx	HAL Nordic implementation	
└ HAL_error.c	HAL ERROR implementation for Nordic	

12.5.5.6.2 API definition

```
void error_handler(int block, error_e err);
```

Register and indicate errors

Parameters:

int block: error handler will be blocking forever if TRUE

error_e err: error type

Return: none

```
error_e get_lastErrorCode(void);
```

Get last indicated error

Parameters: none

Return: last raised error code

12.5.5.7 HAL LOCK

All the necessary APIs to lock or unlock handle.

12.5.5.7.1 Files

root		
└ Src		
└ HAL	Hardware Abstraction Layer	
└ Inc	HAL API header files	
└ HAL_lock.h	HAL LEDs API header file	

12.5.5.7.2 API definition

`QHAL_LOCK(__HANDLE__)`

Macro to lock handle

Parameters: handle

`QHAL_UNLOCK(__HANDLE__)`

Macro to unlock handle

Parameters: handle

12.5.6 OSAL folder

OSAL stands for OS Abstraction Layer.

OSAL folder contains the source code to abstract the OS used in a project:

Project uses QOSAL (Qorvo OSAL) as OSAL Layer.

QOSAL is provided by Qorvo UWB Stack's sublibrary SDK/Firmware/Libs/uwb-stack/libs/qosal (refer to SDK/Documentation/uwb-stack/uwb-qosal-api-R<x.x.x-y>.pdf).

12.5.7 Projects folder

Projects folder contains the SDK project files.

12.5.7.1 Files

Projects	
Common	
cmakefiles	
arm-none-eabi-gcc.cmake	Build script for dwt driver
dwt_uwbdriver.cmake	Build script for uwbstack
scripts	
CreateTargetCommon.py	Generic create target script
pyproject.toml	
FreeRTOS	
CLI	
Common	
cmakefiles	
DW3_QM33_SDK-FreeRTOS.cmake	Main project cmake script
hooks.c	Interrupt function hooks
main.c	CLI main entry
nRF52840DK	
CMakeLists.txt	Generic Project Cmake script
CreateTarget.py	Specific create target script
ProjectDefinition	
CMakeLists.txt	ProjectDefinition module build script
FreeRTOSConfig.h	Project FreeRTOS customization header
ProjectName.c	Name of project
sdk_config.h	Project SDK customization script

(continues on next page)

(continued from previous page)

uwb_stack_llhw.cmake	uwbstack build customization script
ci-project.yml	
isr_hs.c	
nRF52840.ld	Project linker script
project_CLI.cmake	Specific CLI application build script
DWM3001CDK	
...	
Type2AB_EVB	
...	
HelloWorld	
Common	
DWM3001CDK	
nRF52840DK	
README.md	
UCI	
Common	
cmakefiles	
DW3_QM33_SDK-FreeRTOS.cmake	Main project cmake script
hooks.c	Interrupt function hooks
main.c	UCI main entry
nRF52840DK	
CMakeLists.txt	Generic Project Cmake script
CreateTarget.py	Specific create target script
ProjectDefinition	
CMakeLists.txt	ProjectDefinition module build script
FreeRTOSConfig.h	Project FreeRTOS customization header
ProjectName.c	Name of project
sdk_config.h	Project SDK customization script
uwb_stack_llhw.cmake	uwbstack build customization script
ci-project.yml	
isr_hs.c	
nRF52840.ld	
project_UCI.cmake	
DWM3001CDK	
...	
Type2AB_EVB	
...	

The Project's customizing details, due to supported platform, are defined in `SDK/Firmware/Projects /<OS>/<EXAMPLE>/<BOARD>/project_<EXAMPLE>.cmake`.

Below is the table of most significant parameters controlled by `project_common.cmake`:

MY_BOARD	board type
MY_OS	chosen OS
MY_PROJECT	project chosen by name
MY_HAL	chosen HAL implementation
MY_BSP	chosen BSP SDK
MY_CPU	platform MCU type
MY_LD_FILE	path to linker script
PROJECT_ARCH	architecture of MCU
PROJECT_FP	type of chosen floating point ABI
PROJECT_FPU	type of chosen floating point architecture
USE_CRYPTO	control whether the encryption engine is on or off
USE_DRV_DW3xxx	control type of supported transceiver

See also *Build common definitions* section in [CMake flags customization](#).

For details of the porting process of Project folder refer to [Porting Project](#).

12.5.8 SDK BSP Folder

SDK BSP folder, located in SDK/Firmware/SDK_BSP, includes the Board Support Package (drivers) needed for the different targets supported.

Currently Project natively support SDK_17_1_0.

12.5.8.1 Files

```
SDK_BSP
└── Nordic
    └── SDK_17_1_0
```

12.5.9 Libs Folder

12.5.9.1 UWB driver library

Chip driver is provided as a library in **dwt_uwb_driver** folder:

- Inc: driver APIs header files.
- lib: driver APIs library files.

12.5.9.2 UWB stack library

UWB MAC layer is provided as library in **uwbstack** folder:

- cmsis_v1
- cmsis_v2
- uwbstack_lib: MAC library header files and library files.

12.5.9.3 Libraries for a different MCU core

We build a set of common Cortex MCU libraries with different ABIs.

- **Cortex MCU core**
 - Cortex-M4
 - Cortex-M33
- **ABI**
 - Soft floating point (SFP)
 - Hard floating point (HFP)
 - No floating point (NOFP)

Only Cortex-M4 libraries are provided in the SDK. If you need a different build, please contact your Qorvo FAE representative.

13 Customizing the firmware

13.1 CMake flags customization

13.1.1 Project CMake Configuration

Each target within the project is defined by a set of CMake files:

1. **<example>-FreeRTOS.cmake**: this file contains configurations that apply to all board types.
 - `SDK/Firmware/Projects/FreeRTOS/<example>/Common/cmakefiles/<example>-FreeRTOS.cmake`
2. **project_<example>.cmake**: these files contain configurations that are specific to a particular board.
 - `SDK/Firmware/Projects/FreeRTOS/<example>/<board_name>/project_<example>.cmake`

You can configure various aspects of the project by setting CMake flags in the mentioned configuration files. In sections below, you can find an overview of some key flags that you might adjust.

13.1.1.1 Example common definitions

Inside **<example>-FreeRTOS.cmake** file you can set following configurations which are specific for example but common for all boards:

- The flag below turns on generation of the `compile_commands.json` file, which can be useful for many IDEs to enhance their IntelliSense engines. If you do not use this file, you can disable the flag below.

```
set(CMAKE_EXPORT_COMPILE_COMMANDS ON)
```

- The following flag enables the usage of crypto library, which is used e.g. for STS calculation or UWB frame encryption. It is required when uwb-stack is used.

```
set(USE_CRYPTO 1)
```

- The flag below is essential for the correct operation of the UWB-STACK, it shall not be changed.

```
set(CONFIG_QM33 1)
```

- The flag below manages HW protection of CPU core and memory. During the development phase, we advise not to alter this flag to maintain an effective debugging process. Modifying this flag could result in the nRF52 CPU locking the programming and debugging ports, until the full erase of the chip is performed.

The HW protection management is implemented in `SDK/Firmware/Src/HAL/nrfx/HAL_cpu.c` explicitly for the nRF52 SoC because some of this SoC revisions may have the HW protection enabled by default. This implementation needs to be ported if you want to keep the HW protection management when using a different SoC/CPU.

```
set(CONFIG_SOC_PROTECT 0)
```

- The flag below enables the logging system. When it is not needed, disabling this flag will result in saving some CPU resources (OS task, RTC timer, RTT block, memory).

```
set(CONFIG_LOG ON)
```

- When logs are turned on, you can adjust filtering to print more or less logs:

```
# 0=NONE 1=ERROR 2=WARNING 3=INFO 4=DEBUG
set(CONFIG_QLOG_LEVEL 3)
```

- The flag below includes QOSAL implementation for FreeRTOS. If you port SDK to a different OS, it is crucial to change this flag (refer to [Porting to other OS](#)).

```
set(CONFIG_QOSAL_IMPL_FREERTOS 1)
```

- Configs below are specific to L1 and calibration, we recommend leaving them unchanged.

```
# set l1 config needs (used on Src/UWB/CMakelists.txt)
set(CONFIG_L1_CONFIG_CUSTOM_DEFAULT_USE_DEDICATED_SECTION ON)
set(CONFIG_SECURE_PARTITIONS_UWB_L1_CONFIG_SHA256_SIZE 32)
set(CONFIG_SECURE_PARTITIONS_UWB_L1_CONFIG_SIZE 4096)

# Linker calib flags
set(CMAKE_EXE_LINKER_FLAGS
    "${CMAKE_EXE_LINKER_FLAGS} \
-Wl,--defsym=CONFIG_SECURE_PARTITIONS_UWB_L1_CONFIG_SHA256_SIZE=${CONFIG_SECURE_PARTITIONS_ \
-UWB_L1_CONFIG_SHA256_SIZE}"
)

set(CMAKE_EXE_LINKER_FLAGS
    "${CMAKE_EXE_LINKER_FLAGS} \
-Wl,--defsym=CONFIG_SECURE_PARTITIONS_UWB_L1_CONFIG_SIZE=${CONFIG_SECURE_PARTITIONS_UWB_L1_ \
-CONFIG_SIZE}"
)
```

- Below you can find a set of flags for C compiler, assembler and linker. You can change here e.g. floating point unit architecture, standard of compiler, level of debug information and many more. This is also the best place to add your own compiler/assembler/linker flags.

```
set(CMAKE_C_FLAGS
    "-mcpu=cortex-${PROJECT_ARCH}\
     -mfpu=${PROJECT_FPU}\
     -mfloat-abi=${PROJECT_FP}\
     -mthumb\
     -specs=nano.specs\
     -specs=nosys.specs\
     -fdata-sections\
     -ffunction-sections\
     -std=c11\
     -Wall\
     -g3\
     -D${MY_CPU}\
     ${CMAKE_CUSTOM_C_FLAGS}"
```

(continues on next page)

(continued from previous page)

```
)  
  
set(CMAKE_ASM_FLAGS  
    "-mcpu=cortex-${PROJECT_ARCH}\  
     -mfpu=${PROJECT_FPU}\  
     -mfloat-abi=${PROJECT_FP}\  
     -mthumb\  
     -fdata-sections\  
     -ffunction-sections\  
     -Wall\  
     -g3\  
     -D${MY_CPU}\  
     -D__STACK_SIZE=2048\  
     -D__HEAP_SIZE=0\  
     ${CMAKE_CUSTOM_C_FLAGS}"  
)  
  
set(CMAKE_EXE_LINKER_FLAGS  
    "${CMAKE_EXE_LINKER_FLAGS} -u _printf_float -Wl,--gc-sections,-T${CMAKE_SOURCE_DIR}/$  
    ↵${MY_LD_FILE},-Map=${PROJECT_NAME}.map -Wl,--print-memory-usage"  
)
```

- Below you can find HAL specific flags, for more information please refer to [Porting Platform](#).

```
# Set qhal needs  
set(CONFIG_QHAL_IMPL_NRFX ON)  
set(CONFIG_QPLATFORM_IMPL_QM33_QHAL_NON_ZEPHYR ON)  
set(CONFIG_QHAL_MAX_GPIO_CALLBACKS 6)
```

13.1.1.2 Board specific definitions

Inside `project_<example>.cmake` file you can set following configurations:

- The project is built with optimization flag for size `-Os` to reduce the size of the generated firmware. For debugging purposes, you can change the optimization flag to `-Og`. However, be aware that the debugged application might not behave as expected and may not meet certain performance requirements.

```
add_definitions(-Os)
```

- The following flag enables workaround for nRF SDK issue. This issue is related to USB enumeration, it can hang for up to 14 seconds so that access to device via debug interface or UART is delayed after device reset. For smooth experience of development and debugging process, we advise to leave this flag enabled.

Warning: When the device is battery supplied (e.g. by a coin battery or by a power bank), it is required to turn off the workaround by removing this flag. Otherwise, UWB ranging may not work properly.

```
add_definitions(-DUSE_USB_ENUM_WORKAROUND)
```

- The following flag modifies the optimization level of the crypto module from the project-wide default (`-Os`) to `-O3`, which optimizes for execution speed. By default, this flag is disabled to save the flash memory. However, enabling it may be necessary for performance-critical applications.

```
set(USE_CRYPTO_SPEED_OPTIMIZATION 0)
```

- The following flags are used to select backend of the crypto module.
 - Utilizing the CC310 backend can reduce memory consumption and improve performance. However, it has a limitation: only 128-bit keys can be used for FiRa session keys. Additionally, the CC310 backend is not available on all platforms and is therefore disabled by default.

```
set(USE_CRYPTO_BACKEND_CC310 0)
```

- By default, the Mbed-TLS backend is turned on. It does not use hardware acceleration but it supports 256-bit keys.

```
set(USE_CRYPTO_BACKENDMBEDTLS 1)
```

Note: Only one backend can be used at the same time.

- Flags below are used for proper driver choice (also used in LLHW and UWB-STACK libraries). At least one flag needs to be enabled. If you set both, then both drivers will be built and linked to your executable. During runtime, LLHW will choose the proper driver to use, based on the UWB device detected. If your solution design does not require such dynamic driver selection, we encourage you to enable only the required driver to save FLASH memory.

```
set(USE_DRV_DW3000 0)
set(USE_DRV_DW3720 1)
```

13.1.1.3 CLI specific definitions

The key difference between the UCI and CLI builds is their method of communication and control. The UCI build leverages the UCI protocol, requiring an external host to configure and initiate operations on the device. In contrast, the CLI build interacts with the UWB-STACK directly via the helpers API, which allows for autonomous operation without the need for an external interface, making it ideal for standalone devices such as tags.

Inside **CLI-FreeRTOS.cmake** file you can set following configurations:

- If you wish to control device via UCI, use enable the *USE_UCI* flag. Otherwise, use the *USE_CLI* flag.

```
set(USE_CLI 1)
set(USE_UCI 0)
```

- The Listener is an example of non-FiRa application. It uses DWT UWB Drivers directly, without the UWB-STACK. If you want to disable it from the CLI build, turn off the flag below.

```
set(USE_LISTENER 1)
```

- If you do not want to use FiRa specific functionalities in CLI build, you can disable the flag below. However, it will not affect memory footprint of UWB-STACK, this library is always built with FiRa region.

```
set(USE_FIRA 1)
```

13.1.1.4 UCI specific definitions

Inside **UCI-FreeRTOS.cmake** file you can set following configurations:

- If you do not need PCTT functionalities (UWB tests), you can disable flags below:

```
set(USE_PCTT 1)
add_definitions(-DUCI_FTM_BACKEND -DCONFIG_MCPS802154_TESTMODE)
```

- Definitions below are crucial for the system when UCI is used, we advise not to disable them.

```
add_definitions(-DUCI_CONF_MANAGER)
add_definitions(-DUCI_MAC_CALIB_BACKEND)
add_definitions(-DUCI_MAC_BACKEND)
```

13.1.2 UWB-STACK CMake Configuration

UWB-STACK is configured inside CMake file:

- `SDK/Firmware/Projects/Common/cmakefiles/uwbstack.cmake`.

Use mentioned file to set following configurations:

- The UWB-STACK library is delivered in two versions - FiRa only and full (FiRa + PCTT). If you do not require the PCTT functionalities in the UCI build, you can change condition below to include only `SDK/Firmware/Libs/uwb-stack_libs/delivery/full/Release`. This will decrease the memory footprint of your design.

```
if(USE_UCI OR USE_PCTT)
    message("Using full uwbstack")
    set(UWBSTACK_LIBS_PATH
        "${CMAKE_CURRENT_LIST_DIR}/../../../../${LIBS_PATH}/uwbstack_libs/delivery/full/Release"
    )
else()
    message("Using Fira only uwbstack")
    set(UWBSTACK_LIBS_PATH
        "${CMAKE_CURRENT_LIST_DIR}/../../../../${LIBS_PATH}/uwbstack_libs/delivery/fira/
    Release"
    )
endif()
```

13.1.3 DWT UWB Drivers CMake Configuration

DWT UWB Drivers library is configured inside CMake file:

- `SDK/Firmware/Projects/Common/cmakefiles/dwt_uwbdriver.cmake`.

We recommend retaining the default settings.

13.2 Adding a new CLI command

13.2.1 CLI Overview

13.2.1.1 Command parser

Core functionality of CLI is placed inside the command parser module, which is implemented in `command_parser()` function of `SDK/Firmware/Src/Apps/Src/common/cmd/cmd.c` file. It is a first component which receives command from the host terminal and performs initial parsing of the command.

After the command is parsed and successfully recognized, the parser calls the function associated with this command when mode of operation condition is met (see [Mode of operation](#)).

Command can be extended with parameters that can be categorized into three distinct groups:

- **single parameter** - only the first number after command string is recognized as parameter and it is passed via the `val` parameter to the command's function (e.g. DIAG 1).
- **multiple parameters** - multiple parameters can be added after the command string, whole string is passed to the command's function, and the function is responsible for further parsing (e.g. INITF -CHAN=5 -BLOCK=200).
- **JSON parameter** - whole command along with parameters are passed to the command's function as a cJSON object, command's function is responsible for further parsing (currently no example in the SDK).

Parsing of a JSON commands is performed by the cJSON library, for more details please refer to [cJSON GitHub²¹](#).

13.2.1.2 API description

13.2.1.2.1 Command function

To provide a unified interface for declaring a CLI command function, the `REG_FN` macro is used:

```
#define REG_FN(x) const char *x(char *text, int val, cJSON *params)
```

Parameters:

- **char *text**: The text provided by the CLI user.
- **int val**: A numeric value provided after the command (simple parameter).
- **cJSON *params**: Parameters in cJSON format (JSON parameter).
- **return**: String to be printed after command is executed. Usually macro `CMD_FN_RET_OK` or `CMD_FN_RET_NOK` is used.

13.2.1.2.2 Command API

The command is specified by the `command_s` structure, refer to the description below.

```
struct command_s
{
    const char *name;
    const mode_e mode;
    REG_FN((*fn));
    const char *cmnt;
};
```

²¹ <https://github.com/DaveGamble/cJSON>

- **char *text:** Name of command which will be used in CLI. It is limited to 9 characters, can be modified with macro **MAX_CMD_LEN** in SDK/Firmware/Src/Apps/Src/common/cmd/cmd.h.
- **const mode_e mode:** see [Mode of operation](#).
- **REG_FN(*fn):** Function pointer which will be called for given command.
- **const char *cmnt:** Help string for your command which will be printed in the CLI.

13.2.1.2.3 Subcommand API

The command can be also used as a subcommand of an application. Then, it can be executed only when a correct application is running. The subcommand structure needs to be added to application's definition as described in [Implementation of the new subcommand](#).

Subcommands of one application are grouped in a *subcommand_group_s* structure, refer to the description below.

```
struct subcommand_group_s
{
    const char *name;
    const struct command_s *subcommands;
    const uint8_t cnt;
};
```

- **const char *name:** Text to be printed in help command to delimit command group.
- **const struct command_s *subcommands:** Pointer to the array of subcommands.
- **const uint8_t cnt:** Number of subcommands in the array.

13.2.1.2.4 Mode of operation

To distinguish between command types within the CLI engine, each command includes a field indicating its mode of operation (**enum mode_e**). During command parsing, the parser checks this parameter to determine if the command can be executed in the current state of the system.

```
typedef enum
{
    mANY = 0,
    mIDLE = 1,
    mAPP = 2,
} mode_e;
```

There are three types of operation defined:

- **mANY** - a general-purpose command which can be executed always.
- **mIDLE** - a general-purpose command which can be executed only in the IDLE mode (when no application is running).
- **mAPP** - a command that starts an application and can only be executed in the IDLE mode. This mode of operation is also used for additional functionalities, such as subcommand resolution.

13.2.2 Implementation of the new command

Note: Please note the following substitution is used throughout this guide:

- Replace <command_name> with the name of your new command.

All general-purpose commands are placed inside `SDK/Firmware/Src/Apps/Src/common/cmd/cmd_fn.c`. We encourage you to define your new CLI commands in this file.

1. To declare your new command function, please follow this template:

```
REG_FN(f_<command_name>)
{
    // command main function
}
```

If you want to provide more parameters than one into the command, the command function will be responsible for parsing. Whole command along with parameters is passed to the function as a C string via the `text` variable.

2. Add a help string for your command:

```
const char COMMENT_<command_name>[] = {"Text of your command help."};
```

3. Add your command definition in the proper command's array, e.g. if you define an anytime command, place it inside `known_commands_anytime_all`:

```
const struct command_s known_commands_anytime_all[] __attribute__((section(".known_
˓→commands_anytime"))) = {
    // ... (existing command definitions)
    {"<command_name>", mANY, f_<command_name>, COMMENT_<command_name>} ,
};
```

4. After building the firmware, the new command should be ready to use and visible when using the `HELP` command.

13.2.3 Implementation of the new subcommand

Note: Please note the following substitution is used throughout this guide:

- Replace <command_name> with the name of your new command.
- Replace <application_name> with the name of your application.

Note: For reference, you can take as an example the `LSTAT` subcommand of `LISTENER` application, which is implemented in `SDK/Firmware/Src/Apps/Src/listener/listener_fn.c` and `SDK/Firmware/Src/Apps/Src/listener/task_listener.c`.

1. Repeat steps 1. and 2. from [Implementation of the new command](#).
2. Define your command in a new array and place it inside `.known_app_subcommands` section.

```
const struct command_s subknown_commands_<application_name>[] __attribute__((section(
    ".known_app_subcommands")))
= {
    {"<command_name>", mAPP, f_<command_name>, COMMENT_<command_name>},
};
```

3. Add subcommand to the proper application:

```
const struct subcommand_group_s subcommands = {"<application_name> Options", &known_
    subcommands_<application_name>, 1};
```

4. After building the firmware, the new command should be ready to use and visible when using the **HELP** command both in IDLE mode and during runtime of the corresponding application.

13.3 Porting Platform

13.3.1 Memory requirements

13.3.1.1 Overview

When porting the solution to a new platform, it is crucial to assess the application's memory requirements. Below is a detailed report of the memory usage for various components built for a specific application type.

Some libraries are hardware / application specific, and their sizes vary depending on the implementation. Libraries marked with "X" are provided as precompiled and cannot be further optimized. However, note that the linker may exclude unused symbols, meaning the actual size of these libraries in the final executable will depend on the specific functionalities utilized.

Note: The following memory reports for both FLASH and RAM were generated using the DWM3001CDK target and Release build configuration. While the values may vary slightly for other targets, particularly for components such as the nRF SDK, HAL, and UWB Driver, the overall memory consumption should remain comparable. Additionally, the memory usage may differ based on the selected compiler optimization level.

13.3.1.2 Flash

13.3.1.2.1 CLI

Table 13.1: Flash report

Component	Size	Precompiled
Core	64.9 KiB	
Standard libs	+25.6 KiB	
nRF SDK	+32.8 KiB	
FreeRTOS	+6.4 KiB	
Middleware	4.8 KiB	
HAL	+1.8 KiB	
Other	+2.9 KiB	
Apps	21.1 KiB	
Common	+11.1 KiB	
Fira	+7.4 KiB	
Listener	+2.6 KiB	
UWB Stack	122.9 KiB	
MAC	+78.3 KiB	X
UWB Driver	+39.6 KiB	
QLIBS	+5.0 KiB	
Unclassified	0.0 KiB	
All symbols	218.2 KiB	
Unnamed symbols	17.9 KiB	
Total	236.1 KiB	

13.3.1.2.2 UCI

Table 13.2: Flash report

Component	Size	Precompiled
Core	53.8 KiB	
Standard libs	+14.5 KiB	
nRF SDK	+32.8 KiB	
FreeRTOS	+6.4 KiB	
Middleware	2.3 KiB	
HAL	+1.8 KiB	
Other	+0.5 KiB	
Apps	2.7 KiB	
Common	+1.2 KiB	
UCI	+1.4 KiB	
UWB Stack	181.6 KiB	
UCI	+40.0 KiB	X
MAC	+97.1 KiB	X
UWB Driver	+39.5 KiB	
QLIBS	+5.0 KiB	
Unclassified	0.0 KiB	
All symbols	240.4 KiB	
Unnamed symbols	12.4 KiB	
Total	252.8 KiB	

13.3.1.2.3 HelloWorld

Table 13.3: Flash report

Component	Size	Precompiled
Core	37.6 KiB	
Standard libs	+14.5 KiB	
nRF SDK	+17.1 KiB	
FreeRTOS	+6.0 KiB	
Middleware	6.1 KiB	
Log	+4.3 KiB	
HAL	+0.9 KiB	
Other	+0.9 KiB	
Apps	0.4 KiB	
HelloWorld	+0.4 KiB	
UWB Stack	116.1 KiB	
MAC	+71.3 KiB	X
UWB Driver	+39.5 KiB	
QLIBS	+5.4 KiB	
Unclassified	0.1 KiB	
All symbols	160.4 KiB	
Unnamed symbols	6.0 KiB	
Total	166.3 KiB	

13.3.1.2.4 QANI

Table 13.4: Flash report

Component	Size	Precompiled
Core	47.5 KiB	
Standard libs	+14.7 KiB	
nRF SDK	+26.2 KiB	
FreeRTOS	+6.6 KiB	
Middleware	137.0 KiB	
Log	+4.5 KiB	
HAL	+1.7 KiB	
BLE	+130.3 KiB	
Other	+0.4 KiB	
Apps	3.1 KiB	
Fira	+0.6 KiB	
QANI	+2.5 KiB	
UWB Stack	122.1 KiB	
MAC	+78.1 KiB	X
UWB Driver	+38.5 KiB	
QLIBS	+5.5 KiB	
Unclassified	0.3 KiB	
All symbols	309.9 KiB	
Unnamed symbols	16.2 KiB	
Total	326.1 KiB	

13.3.1.3 RAM

13.3.1.3.1 CLI

Table 13.5: RAM report

Component	Size	Precompiled
Core	61.0 KiB	
Standard libs	+0.5 KiB	
nRF SDK	+9.9 KiB	
FreeRTOS	+50.6 KiB	
Middleware	3.5 KiB	
HAL	+3.0 KiB	
Other	+0.5 KiB	
Apps	33.4 KiB	
Common	+33.3 KiB	
Fira	+0.1 KiB	
Listener	+0.0 KiB	
UWB Stack	7.1 KiB	
MAC	+4.6 KiB	X
UWB Driver	+0.5 KiB	
QLIBS	+2.1 KiB	
Unclassified	0.0 KiB	
All symbols	105.3 KiB	
Unnamed symbols	0.1 KiB	
Total	105.4 KiB	

13.3.1.3.2 UCI

Table 13.6: RAM report

Component	Size	Precompiled
Core	61.0 KiB	
Standard libs	+0.5 KiB	
nRF SDK	+10.0 KiB	
FreeRTOS	+50.6 KiB	
Middleware	3.0 KiB	
HAL	+3.0 KiB	
Other	+0.0 KiB	
Apps	32.4 KiB	
Common	+32.3 KiB	
UCI	+0.1 KiB	
UWB Stack	8.4 KiB	
UCI	+0.3 KiB	X
MAC	+5.6 KiB	X
UWB Driver	+0.5 KiB	
QLIBS	+2.1 KiB	
Unclassified	0.0 KiB	
All symbols	104.8 KiB	
Unnamed symbols	0.1 KiB	
Total	104.9 KiB	

13.3.1.3.3 HelloWorld

Table 13.7: RAM report

Component	Size	Precompiled
Core	60.0 KiB	
Standard libs	+0.5 KiB	
nRF SDK	+8.9 KiB	
FreeRTOS	+50.6 KiB	
Middleware	5.8 KiB	
Log	+5.5 KiB	
HAL	+0.3 KiB	
Other	+0.0 KiB	
Apps	0.5 KiB	
HelloWorld	+0.5 KiB	
UWB Stack	7.1 KiB	
MAC	+4.6 KiB	X
UWB Driver	+0.5 KiB	
QLIBS	+2.1 KiB	
Unclassified	0.0 KiB	
All symbols	73.4 KiB	
Unnamed symbols	0.1 KiB	
Total	73.4 KiB	

13.3.1.3.4 QANI

Table 13.8: RAM report

Component	Size	Precompiled
Core	60.5 KiB	
Standard libs	+0.5 KiB	
nRF SDK	+9.4 KiB	
FreeRTOS	+50.6 KiB	
Middleware	9.3 KiB	
Log	+7.5 KiB	
HAL	+0.3 KiB	
BLE	+1.5 KiB	
Apps	0.8 KiB	
QANI	+0.8 KiB	
UWB Stack	7.1 KiB	
MAC	+4.6 KiB	X
UWB Driver	+0.5 KiB	
QLIBS	+2.1 KiB	
Unclassified	0.0 KiB	
All symbols	77.7 KiB	
Unnamed symbols	0.1 KiB	
Total	77.8 KiB	

Note: FreeRTOS library includes 50 KiB of HEAP, which should be adjusted according to the application requirements. E.g. single TWR active session requires about 16 KiB of HEAP.

13.3.2 Porting HAL

13.3.2.1 Introduction

As *HAL Layer* is the link with the hardware level, its implementation allows to support various boards and adapt to any new board.

For porting the entire project to a new platform, you have to adapt the entire *HAL Layer* (HAL and QHAL), which means to create an implementation for `Firmware/Src/HAL/Src` and `Firmware/Libs/uwb-stack/libs/qhal` for the platform, see [Architecture/Layers/HAL_Overview](#) for more details.

Note: Only Nordic MCUs using nRF SDK for the nRF52 Series SoCs with FreeRTOS are supported for now. HAL sources for Nordic can be found in `Firmware/Src/HAL/Src/nrfx`, QHAL sources for Nordic can be found in `Firmware/Libs/uwb-stack/libs/qhal/src/nrfx`, nRF5 SDK path is `Firmware/SDK_BSP/Nordic/17_1_0`.

If an MCU brand other than Nordic is used, the corresponding SDK will be needed. SDKs can be stored in the `SDK_BSP` folder.

13.3.2.2 Porting process

Note: For all steps it is recommended to copy the skeletal structure from existing implementations for already supported platforms.

To support any other platform, on HAL level, following steps have to be accomplished:

1. (If needed) create a new folder under `Firmware/SDK_BSP/<MCU_manufacturer>` and add framework for chosen MCU.
2. Create a new folder `<board>` under `Firmware/Src/HAL/Src/<board>` and `Firmware/Libs/uwb-stack/libs/qhal/src/<board>`, implement there all the functions defined by API of the *HAL Layer* (HAL and QHAL respectively):

Note: For HAL api refer to [Architecture/Layers/HAL](#)

For QHAL api refer to [Documentation/uwb-qhal-api.pdf](#)

3. Adapt CMake files for integrating the new platform:

- `Firmware/Src/HAL/Src/<MCU>/CMakeLists`, HAL component is integrated with the project by means of CMake's library target *HAL*.
- `Firmware/Libs/uwb-stack/libs/qhal/CMakeLists`, QHAL component is integrated with the project by means of CMake's library target *qhal*.

Note:

Customization of the Project due to *HAL Layer* is done at the level of Project definition [Architecture/Layers/Project](#).

- Some of customization may be required to be done in `SDK/Firmware/Projects/FreeRTOS/cmakefiles/<example>-FreeRTOS.cmake` the generic common project CMake file, e.g. new parameter `CONFIG_QHAL_IMPL_XXX` for new *HAL Layer* implementation. Below is a table of QHAL configuration parameters that may require to be customized within the file.

<code>CONFIG_QHAL_IMPL_NRFX</code>	Enabling support of NRFX based platform in the QHAL.
<code>CONFIG_QHAL_MAX_GPIO_CALLBACKS</code>	Define max number of callbacks in the QHAL.

Note that a new parameter may need to be defined if a new SDK is added.

Note: There are two flavors of applications in the SDK:

Note: This section describes two types of applications supported by the HAL layer:

- **uwb-stack based FiRa applications:** These applications utilize the UWB stack for FiRa compliance. Examples include: - CLI: INITF, RESPF - UCI: uci
- **Direct dwt_driver based applications:** These applications directly interact with the dwt_driver for lower-level operations. Example: - CLI: LISTENER

Depending on which flavor of application you want to port to a new platform, some part of the HAL layer may not need to be ported.

13.3.2.3 Low power considerations

When porting the HAL to a new platform, managing power consumption is an important aspect to address. Proper handling of both CPU and peripheral power modes is key when the firmware enters the IDLE state (i.e., no computations are ongoing, and the system is waiting for UWB transmission or reception). For optimal results in IDLE mode, the CPU should enter the deepest possible sleep state, and all unused peripherals—especially those driven by high-frequency clocks—should be turned off.

Power management of the UWB module is handled by the UWB stack, which is responsible for putting the QM33 transceiver into deep sleep whenever there is sufficient time before the next transmission or reception. Additionally, the SPI peripheral is powered down (by the **QHAL**) between transactions with the UWB transceiver to further reduce power consumption.

From the UWB stack's perspective, two timers drive its core functionalities and play a crucial role in power management. When porting the solution to hardware platforms other than the nRF52x family, careful consideration must be given to designing an efficient implementation.

1. Idle timer

The idle timer is a one shot timer used to wake up the UWB stack from sleep for a next Rx/Tx operation. This timer is implemented and used in UWB stack as a one of instances of the **QTIMER** interface.

Note: The **QTIMER** API is designed to manage a pool of timers that can be dynamically allocated and used by different components. However, the UWB stack currently utilizes only a single timer instance, which simplifies implementation when porting to other hardware.

In the current version of the SDK, the **QTIMER** implementation is based exclusively on a single RTC instance of nRF to optimize power consumption. Depending on the protocol's use case, timeout of the idle timer can be set to a very long time. While using RTC is not strictly required, be aware that choosing a different timer source may impact power efficiency.

QTIMER API allows to configure idle timer on init. However, this timer is created in precompiled UWB stack library. To allow the user to have influence on the configuration, **QPLATFORM** module provides API dedicated to idle timer. Please implement those two functions to return preferred timer configuration to UWB stack:

- `qplatform_get_idle_timer_config()`
- `qplatform_get_idle_timer_instance()`

These functions are used inside uwb stack implementation to get the configuration of idle timer. Please refer to *Documentation/uwb-qplatform-api.pdf* for their detailed description.

2. Persistent timer

The persistent timer is a continuously running timer that provides a common time reference for the entire system (e.g., for logs). It is also synchronized with the UWB transceiver's clock, ensuring precise scheduling of Tx/Rx operations.

The persistent timer is implemented in the **QHAL** component using the nRF RTC. For porting reference, we encourage to check current implementation in `Firmware/Libs/uwb-stack/libs/qhal/src/nrfx/persistent_time.c`. For API description, please refer to *Documentation/uwb-qhal-api.pdf*.

The timer is accessed and controlled through the **QRTC** API, both in user code as well as in the UWB stack. This timer needs to provide continuous time reference even when entering sleep mode of the MCU, so it is important to use power efficient timer implementation.

Currently one nRF RTC instance is shared between both **QRTC** and **QTIMER** timers. This allow to provide a power efficient systime and up to 4 channels for generic timers. For this purpose, additional module `qrtc_share.c` is provided in `Firmware/Libs/uwb-stack/libs/qhal/src/nrfx/qrtc_share.c`.

Note: Both timers mentioned above should meet the following requirements:

- Precision: best under 1 μ s, tolerable 31 μ s.
 - Clock source: with a low drift and a tolerance within ± 100 ppm. An external crystal oscillator is recommended.
-

In conclusion, the UWB stack requires two low-power timers. The SDK utilizes a single nRF RTC instance for this purpose. Additionally, FreeRTOS requires one more RTC instance, as does the nRF SoftDevice (if used in a given example). Consequently, at least three RTC instances may be required in the most demanding scenario. Some HW platforms may not have sufficient RTC peripherals to meet this requirement. If the available number of RTCs is limited, there are several possible alternative solutions. The choice depends on the user, who must balance timer's precision and power consumption.

Example solutions for limited RTC availability:

- Configure FreeRTOS to use SysTick with tickless idle mode and use **QRTC** to resynchronize SysTick upon wake-up.
- Modify the **QTIMER** API to use alternative timers instead of RTC (though this may increase power consumption depending on the hardware implementation).

13.3.3 Porting BAL

13.3.3.1 Introduction

To have the software fully working on a new board, adapting it to the board is necessary. Adaptation of the project's middleware to the new board is done by Board Abstraction Layer (BAL). For details see [Board Abstraction Layer \(Boards\)](#).

13.3.3.2 Porting process

To port entire BAL layer, you have to:

1. Create the implementation of the **Board Component** for your target board.

The implementation should be located at `SDK/Firmware/Src/Boards/Src/<target>`:

Note: It is recommended to create a copy of the Board Component of an already existing target and adjust it for the new target board.

- Create the implementation of the `SDK/Firmware/Src/Boards/Inc/thisBoard.h` interface at location `SDK/Firmware/Src/Boards/Src/<target>/<OS>/<target>.c`,
 - `peripherals_init(void)` must contain the initialization of the clock and watchdog.
 - `BoardInit(void)` must contain the pin initialization, call peripheral initialization and can be customized to flash LEDs.
- Create the customization file `SDK/Firmware/Src/Boards/Src/<target>/Common/custom_board.h`,

Below is an example table of the components customized by `custom_board.h` for nRF52840DK board.

UART
LED

- Create the implementation of `ll_config_reset_to_default` procedure in `SDK/Firmware/Src/Boards/Src/<target>/platform_ll_config.c` (refer to [Documentation/uwb-l1-configuration.pdf](#)).
- Adjust the `SDK/Firmware/Src/Boards/Src/<target>/CMakeLists.txt` by customizing it for the current target board. The Board Component is integrated with project as CMake's library target Board.

2. Customize the **Qplatform component** for your target board.

Since the Qplatform component implementation is generic, it only requires customization for the new target. The customization of Qplatform component is done by means of `ProjectDefinition` in `SDK/Firmware/Projects/<EXAMPLE>/<target>/ProjectDefinition/uwb_stack_llhw.cmake`. Especially section LLHW drv definitions. For more details see description of **Qplatform Component** section in chapter [Board Abstraction Layer \(Boards\)](#).

13.3.4 Porting to other OS

13.3.4.1 Introduction

Before adding a new OS to the SDK, we encourage you to familiarize yourself with how FreeRTOS is integrated into the SDK's structure along with **OSAL** layer by referring to [Projects folder](#) and [OSAL folder](#).

The OSAL layer of the Project is delegated to the QOSAL (Qorvo Operating System Abstraction Layer). In other words, project uses/implements QOSAL as OSAL Layer.

13.3.4.2 QOSAL

13.3.4.2.1 Introduction

The key to adapting the SDK for your chosen OS is to port the QOSAL.

QOSAL API is defined in `SDK/Firmware/Libs/uwb-stack/libs/qosal/include`. Start by consulting the **UWB QOSAL API** document (`SDK/Documentation/uwb-stack/uwb-qosal-api-R<x.x.x-y>.pdf`), which provides a comprehensive API overview and porting instructions.

13.3.4.2.2 QOSAL implementation

We recommend using the FreeRTOS QOSAL implementation as a reference when porting to a different OS. Implementation of the QOSAL for FreeRTOS is placed inside following directories:

- `SDK/Firmware/Libs/uwb-stack/libs/qosal/src/freertos`
- `SDK/Firmware/Libs/uwb-stack/libs/qosal/include/freertos`

The QOSAL library directory includes a Zephyr implementation that you may use as a reference. However, be aware that this implementation has not been tested with the current SDK.

The QOSAL API and its implementation are designed to support the full functionality of the uwb-stack and code within this SDK. Bear in mind that you may need to extend the API to accommodate the particular needs of your OS and product design.

13.3.4.2.3 Functionalities description

API	FUNCTIONALITY
QASSERT	Assertion handling (no specific implementation for FreeRTOS)
QATOMIC	Wrapper for GCC atomic operations (no specific implementation for FreeRTOS)
QERR	Qorvo specific error codes and enum conversion
QIRQ	Enabling/disabling interrupts in the system
QLOG	Logging system (depends on nrfx library, implementation placed inside SDK/Firmware/Libs/uwb-stack/libs/qhal/src/nrfx/qlog_impl.h)
QMALLOC	Dynamic memory management
QMSQ_QUEUE	Message queues management
QMUTEX	Mutex management
QOS	High level OS functions, e.g. to start the scheduler
QPM	ACPI standard power management of the system (not supported for FreeRTOS in this SDK, please follow Zephyr implementation for design guides)
QPRIVATE	Private API implementation specific for FreeRTOS (used only inside QOSAL library)
QMPROFILING	Task and stack usage info
QRAND	Random number generation (currently not used in uwb-stack and this SDK)
QSEMAPHORE	Semaphores management
QSIGNAL	Management of signals to communicate between tasks
QTHREAD	Threads/tasks management
QTIME	System time and task sleep management
QMTOOLCHAIN	Toolchain specific definitions (no specific implementation for this SDK)
QTRACING	Printing MAC traces (not supported in this SDK)
QWORKQUEUE	Managing work queue - tasks to delegate low-priority work from ISRs or high-priority tasks

13.3.4.3 Porting

Note: Please note the following substitutions are used throughout this guide:

- Replace **<os_name>** with the name of the operating system you are using.
- Replace **<board_name>** with the name of your chosen development kit.

13.3.4.3.1 Add a new OS

Start the porting process by adding your OS as a CMake library named **OS** in the directory `SDK/Firmware/Src/OS/<os_name>/`. Refer to the FreeRTOS configuration in `SDK/Firmware/Src/OS/FreeRTOS/CMakeLists.txt` for an example.

13.3.4.3.2 Port the QOSAL

1. Provide your implementation of **QOSAL** in `SDK/Firmware/Libs/uwb-stack/libs/qosal/src/<os_name>/`, and `SDK/Firmware/Libs/uwb-stack/libs/qosal/include/<os_name>/`. Then add sources to `SDK/Firmware/Libs/uwb-stack/libs/qosal/CMakeLists.txt`.
2. Hide your QOSAL implementation under CMake flag `CONFIG_QOSAL_IMPL_<os_name>` inside `SDK/Firmware/Libs/uwb-stack/libs/qosal/CMakeLists.txt`.
3. Set up the **OSAL** library:

- Copy `SDK/Firmware/Src/OSAL/Src/FreeRTOS/CMakeLists.txt` to `SDK/Firmware/Src/OSAL/Src/<os_name>/CMakeLists.txt`.
- Modify **OSAL** library in `SDK/Firmware/Src/OSAL/Src/<os_name>/CMakeLists.txt` to suit your OS requirements.

Warning: Due to system constraints, uwb-stack library is precompiled with `SDK/Firmware/Libs/uwb-stack/libs/qosal/include/dummy`. This enforces the values defined by `QOSAL_IMPL_THREAD_MAX_NAME_LEN` and implementation of `QOSAL_IMPL_THREAD_STACK_DEFINE` from “dummy” implementation are used within the uwb-stack. Consequently, any changes made to these macros in the SDK will not affect the precompiled uwb-stack, which continues to rely on the “dummy” definitions.

Additionally, the uwb-stack uses “dummy” version of QLOG, so that logs produced by uwb-stack are not accessible. Despite this, the OS-specific implementation within the SDK will function correctly.

Warning: QLOG implementation is based on nrfx libraries and depends on QHAL, the implementation is placed inside `SDK/Firmware/Libs/uwb-stack/libs/qhal/src/nrfx/qlog_impl.h`.

4. Check if your OS design is aligned to macros `QOSAL_IMPL_THREAD_MAX_NAME_LEN` and `QOSAL_IMPL_THREAD_STACK_DEFINE` from `SDK/Firmware/Libs/uwb-stack/libs/qosal/include/dummy/qosal_impl.h`.
5. Check if QLOG implementation in `SDK/Firmware/Libs/uwb-stack/libs/qhal/src/nrfx/qlog_impl.h` is aligned to needs of your OS and system design.

13.3.4.3.3 Port the Project

Copy the `SDK/Firmware/Projects/FreeRTOS` directory to `SDK/Firmware/Projects/<os_name>/`, substituting instances of **FreeRTOS** with your `<os_name>` in all subdirectories. Then, adjust the contents to fit your OS-specific configurations:

1. **To compile the SDK with your OS, replace `CONFIG_QOSAL_IMPL_FREERTOS` with `CONFIG_QOSAL_IMPL_<os_name>`** in each example-specific file located at `SDK/Firmware/Projects/<os_name>/<example>/Common/cmakefiles/<example>-<os_name>.cmake`.
2. Update **ProjectName** string in each example-specific file located at `SDK/Firmware/Projects/<os_name>/<example>/<board_name>/ProjectDefinition/ProjectName.c`.
3. **There are also two FreeRTOS specific definitions in each example-specific file located at `SDK/Firmware/Projects/<os_name>/<example>/<board_name>/ProjectDefinition/uwb_stack_llhw.cmake` that you may need to update for your OS needs:**
 - `NRF_LOG_TIMESTAMP_DEFAULT_FREQUENCY`,
 - `RTC1_ENABLED`.
1. Update FreeRTOS with `os_name` in each example-specific file located at `SDK/Firmware/Projects/<os_name>/<example>/CreateTarget.py`.
2. **Set the proper name of your OS in `MY_OS` flag in each example-specific file located at `SDK/Firmware/Projects/<os_name>/<example>/<board_name>/project_<example>.cmake`.**
3. **Align `CONFIG_QPLATFORM_IMPL_QM33_QHAL_NON_ZEPHYR` flag in each example-specific file located at `SDK/Firmware/Projects/<os_name>/<example>/Common/cmakefiles/<example>-<os_name>.cmake`.** Set it to **OFF** when Zephyr OS is used.

13.3.4.3.4 Other alignments

1. Replace **FreeRTOS** in `SDK/Firmware/Projects/Common/scripts/CreateTargetCommon.py` with name defined by **MY_OS** cmake flag.
2. Check if priorities definitions in `SDK/Firmware/Src/Boards/Inc/Common/int_priority.h` are aligned with needs of your OS.
1. You can also remove FreeRTOS specific parts:
 - `SDK/Firmware/Projects/<os_name>/<example>/<board_name>/ProjectDefinition/FreeRTOSConfig.h`,
 - `SDK/Firmware/Projects/<os_name>/<example>/Common/hooks.c`
1. To keep VS Code integration, replace occurrences of **FreeRTOS** string in `SDK/Firmware/.vscode/tasks.json` and `SDK/Firmware/.vscode/launch.json` with name defined by **MY_OS** CMake flag.

13.3.5 Porting Project

13.3.5.1 Introduction

Adding support for the new platform requires porting the Project folder to new platform. For architectural details of Project folder refer to [Projects folder](#).

Note: We describe the porting process with the assumption that the supported OS is FreeRTOS. For the other OS see [Porting to other OS](#).

13.3.5.2 Porting process

For adding support for a new board to the Project folder, the following steps must be done:

1. Create a new instance of Project folder `SDK/Firmware/Projects/FreeRTOS/<example>/<board>` for the board you are porting the project to.

Note: It is recommended to create a copy of the already existing target (e.g. `SDK/Firmware/Projects/FreeRTOS/CLI/nRF52840DK`) and adjust it for the new one being created.

2. Adjust files inside the folder:

- `CreateTarget.py` - update the board variable (`board=<board_name>`),
- `<board>.ld` - rename and adjust linker script for your platform,
- `project-<example>.cmake` - update board-specific fields. For detailed description of the fields, see [Projects folder](#).
- `ProjectDefinition/ProjectName.c` - update the ProjectName description with the proper board name.
- `ProjectDefinition/uwb_stack_llhw.cmake` - customize the GPIO board pinouts for the board.

Note: `ProjectDefinition/uwb_stack_llhw.cmake` is a part of BAL layer which is responsible for customization of board specific details. See [Porting BAL](#).

13.3.6 Porting to other Board

13.3.6.1 Introduction

For porting the project to the target board, these steps are to be done:

Note: Depending on the board family you want to port, some steps may not need to be done because the particular layers may already be implemented in the SDK.

1. Porting HAL Layer [*Porting HAL*](#) (if required)
2. Porting BAL Layer [*Porting BAL*](#) (if required)
3. Porting OS Layer [*Porting to other OS*](#) (if required)
4. Porting Project [*Porting Project*](#)

14 Calibration and Configuration

14.1 Overview

14.1.1 Introduction

The calibration and configuration system allows to push some settings to the device.

The calibration and configuration settings are used to specify:

- The antenna set used to transmit and receive frames
- The antenna set and the frame segment used to compute the Angle of Arrivals (AoAs)
- The lookup table matching the mounted antenna to translate measured Phase Difference of Arrival (PDoA) to AoA
- The TX power for different transmission phases, the antenna delay, ...

The calibration and configuration are based on a Key/Value mechanism. It is possible to set the calibration and configuration through UCI or through CLI build.

The calibration and configuration are contained inside a JSON file, which is parsed and sent to the device through UCI, using the UCI version of the binary. The CLI version allows to set or get calibration values manually.

14.1.1.1 Calibration important notes

Note: Configuration and calibration settings are non-volatile, they are retained after power cycle or firmware update (without Non-Volatile Memory erase).

Warning: If you are not using a QM33120WDK1, DWM3001CDK or Type2AB EVB (e.g. DWM3000EVB), you must flash the calibration before starting a ranging session.

Warning: Using the default calibration leads to decreased ranging accuracy. The device uses the default settings under one of the following circumstances:

- when you receive a new development kit board,
- when whole FLASH / NVM is erased,
- when you use the RESTORE CLI command,
- when you use `reset_calibration` python script.

In any of these situations, calibrate your board manually before starting the ranging session. A step-by-step guide can be found in the chapter [Distance and Angle Calibration](#).

The compatibility of the calibration data from one SDK version to another is not guaranteed and ranging may be not possible at all.

For instructions, please refer to the [Pushing calibration to a device](#) section.

14.1.2 Understanding the Calibration and Configuration

14.1.2.1 TxPower

The Calibration allows to define up to 8 reference frames. Each of these reference frames is configured with a set of properties (for instance, payload length, frame type, power index...).

When a frame is transmitted by the chip, the firmware will select the nearest corresponding reference frame and use the transmit power of this reference frame as a working base. This transmit power is then adjusted according to the characteristics of the frame to be sent in order to optimize the transmit power while staying within the regulation requirements.

The corresponding power index calibration key allows to change the transmit power of the frame and to stay compliant with the FCC requirements.

14.1.2.1.1 Low Noise Amplifier

Note: Please note that the LNA is not part of the provided development kit.

A dedicated GPIO on the chip is activated (goes high) when the chip is in receive mode.

QM33100 and DW3000 have an internal LNA but it is possible to enhance reception performance furthermore by integrating a LNA to the front end of your design. To enable the LNA, adjust the relevant configuration key ant<K>.lna.

Once the LNA key is enabled, a dedicated GPIO on the UWB chip is activated (goes high) when the chip is in a receive mode to activate the LNA.

This GPIO depends on the UWB chip, following this table:

Table 14.1: LNA GPIO

Chip	Signal name
DW3xx0	GPIO6 / EXTRXE ²²
QM331x0	GPIO5 / EXTRXE

We recommend the [UWB LNA QM14068](#)²³

For more information, check the UWB chip datasheet and the [Application Note APS304: Increasing Range Using an External LNA DW3000 / QM33100](#)²⁴

²² EXTRXE - External Receiver Enable

²³ <https://store.qorvo.com/products/detail/qm14068-qorvo/695711/>

²⁴ <https://www.qorvo.com/products/d/da009232>

14.1.2.1.2 Power Amplifier

Note: Please note that the PA is not part of the provided development kit.

Enhancing transmission performance is possible by integrating a PA into your design. To enable the PA, adjust the relevant configuration key `ant<K>.pa`.

Once the PA key is enabled, a dedicated GPIO on the UWB chip is activated (goes high) when the chip is in a transmit mode to activate the PA.

This GPIO depends on the UWB chip, following this table:

Table 14.2: PA GPIO

Chip	Signal name
DW3xx0	GPIO5 / EXTTXE ²⁵
QM331x0	GPIO4 / EXTTXE

We recommend the [UWB PA QM14070²⁶](#)

For more information, check the UWB chip datasheet.

14.1.2.2 Calibration

Depending on the board and the antenna design, it is possible to set calibration data which are applied to the chip, like the TxPower, the antenna delay, the PDoA offset, ...

14.2 Pushing calibration to a device

14.2.1 Pushing the calibration over UCI

The `uwb-qorvo-tools` package, located in `SDK/Tools/uwb-qorvo-tools`, provides the `load_cal` script. This script can be utilized to load calibration values from a JSON file into the device through UCI.

Note: For more detailed information about `load_cal`, including command options and descriptions, as well as some default calibration files for supported targets, refer to the [SDK/Tools/uwb-qorvo-tools/UWB-Qorvo-Tools-guide.pdf](#) documentation.

14.2.2 Pushing the calibration over GUI

The Qorvo One TWR GUI allows for loading calibration into the device using the [Device Configuration popup](#).

Default Calibration and Configuration

The GUI uses default calibration and configuration data encapsulated within JSON files. To load default configuration:

1. Click *Reset all Configuration* button and wait for the finish.
2. Click the *Save* button.

To extract and analyze the current calibration:

²⁵ EXTTXE - External Transmit Enable

²⁶ <https://store.qorvo.com/products/detail/qm14070-qorvo/705311/>

1. Click the *Export Calibration* button and choose your desired path for the JSON file.
2. Click the Save button.

Customizing Calibration Files

The exported default configuration JSON files can be customized to meet specific requirements. To use a customized calibration file:

1. Modify the exported JSON file as needed.
2. Click the *Import Calibration* button and import the customized JSON calibration file.
3. Click the Save button.

By following these steps, you can tailor the calibration settings to fit your specific requirements.

14.3 Calibration and Configuration keys dictionary

For details about L1 Configuration and Calibration keys, please refer to the uwb-11-configuration document.

14.4 Erase calibration

To apply a new calibration, it is recommended to reset the already existing calibration of the UWB device.

It is also possible to fully erase it if needed.

14.4.1 How to reset calibration

After following Prerequisites chapter from `SDK/Tools/uwb-qorvo-tools/UWB-Qorvo-Tools-guide.pdf` documentation, use `reset_calibration.py` script (under `SDK/Tools/uwb-qorvo-tools/scripts/generic/device/reset_calibration`) to reset the calibration to default one.

More details about this script can be found in `SDK/Tools/uwb-qorvo-tools/UWB-Qorvo-Tools-guide.pdf` documentation.

Warning: If you are not using a QM33120WDK1, DWM3001CDK or Type2AB EVB (e.g. DWM3000EVB), you must flash the calibration before starting a ranging session.

Warning: Using the default calibration leads to decreased ranging accuracy (see section [Calibration important notes](#) for more details).

14.4.2 How to erase calibration

Erasing the calibration is supported using both embedded J-link (J-Link OB) as well as external J-Link, depending on the board that is being used.

Steps to erase calibration are identical to those described in section flashing. Instead of flashing, erase the device using the “Erase chip” button, then reflash the firmware.

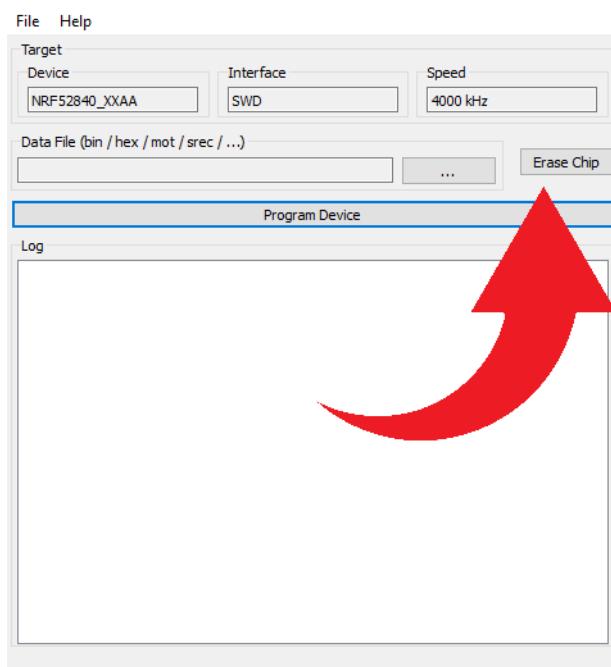


Figure 14.1: Erase chip button

Warning: If you are not using a QM33120WDK1, DWM3001CDK or Type2AB EVB (e.g. DWM3000EVB), you must flash the calibration before starting a ranging session.

14.5 Distance and Angle Calibration

Warning: For any production tests and end product calibration, please refer to application note [APS312 Production Tests for DW3000-Based Products²⁷](#).

For evaluation, measurements over the air can already provide good performances by following simple steps.

Note: It is possible to automatically calibrate the distance by using the GUI, please check [Calibration](#) in the GUI chapter for more details.

²⁷ <https://www.qorvo.com/products/p/QM33120W#documents>

14.5.1 Distance

You can calibrate the distance manually by tweaking the antenna delay calibration key `ant<x>.ch<y>.ant_delay` (with x corresponding to the antenna path used and y the channel used). See [Calibration and Configuration keys dictionary](#) for more details.

This calibration is stored in NVM so you can perform it only once.

Here are the steps to follow in order to calibrate the antenna delay using CLI and UCI builds:

14.5.1.1 CLI

- Put a **known distance** between your two devices. It is recommended to place them approximately two meters apart.
- Start a Two-Way Ranging session, using `INITF` and `RESPF` commands (see [INITF/RESPF command](#) for more details).
- Stop the ranging on the device you want to calibrate.
- Adjust its antenna delay, using `CALKEY` command (see [IDLE time Commands](#) for more details).
- Restart the Two-Way Ranging session.
- Repeat the last three steps until reported distance corresponds to real measured distance.

14.5.1.2 UCI

- Put a **known distance** between your two devices. It is recommended to place them approximately two meters apart.
- Start a Two-Way Ranging session, using `run_fira_twr` script on both devices. It is recommended to use `-stat` parameter to get the average distance at the end of the script (see [SDK/Tools/uwb-qorvo-tools/UWB-Qorvo-Tools-guide.pdf](#) for more information).
- Wait for the script to finish.
- On the device you want to calibrate, adjust the antenna delay, using `set_cal` script (see [SDK/Tools/uwb-qorvo-tools/UWB-Qorvo-Tools-guide.pdf](#) for more information).
- Restart the Two-Way Ranging session.
- Repeat the last three steps until reported distance corresponds to real measured distance.

14.5.2 Angle

You can calibrate the angle manually by tweaking the PDoA (Phase Difference of Arrival) offset calibration key `ant_pair<x>.ch<y>.pdoa.offset` (with x corresponding to the antenna pair used and y the channel used). See [Calibration and Configuration keys dictionary](#) for more details.

This calibration is stored in NVM so you can perform it only once.

Here are the steps to follow in order to calibrate the PDoA offset using CLI and UCI builds:

14.5.2.1 CLI

- Put a known angle of **0 degree** between your two devices.
- Start a Two-Way Ranging session, using INITF and RESPF commands (see [INITF/RESPF command](#) for more details). If only one device is AoA-capable, use this device to run INITF.
- Stop the ranging on the device you want to calibrate.
- Adjust its PDoA offset, using CALKEY command (see [IDLE time Commands](#) for more details).
- Restart the Two-Way Ranging session.
- Repeat the last three steps until reported angle corresponds to known angle (0 degree).

14.5.2.2 UCI

- Put a known angle of **0 degree** between your two devices.
- Start a Two-Way Ranging session, using `run_fira_twr` script on both devices. It is recommended to use `-stat` parameter to get the average angle at the end of the script (see [SDK/Tools/uwb-qorvo-tools/UWB-Qorvo-Tools-guide.pdf](#) for more information). If only one device is AoA-capable, remember to set this device as an initiator.
- Wait for the script to finish.
- On the device you want to calibrate, adjust the PDoA offset, using `set_cal` script (see [SDK/Tools/uwb-qorvo-tools/UWB-Qorvo-Tools-guide.pdf](#) for more information).
- Restart the Two-Way Ranging session.
- Repeat the last three steps until reported angle corresponds to known angle (0 degree).

15 OTP Memory Map

The DW3000 and QM33 UWB chips series contain an on-chip One-Time Programmable (OTP) memory.

This memory can be used to store calibration data such as TX power level and crystal initial frequency error adjustment.

The presented OTP memory map contains values calibrated after production of development kits they do not apply to individual chips.

OTP memory locations are defined in the following tables.

- OTP memory locations are each 32-bits wide.
- OTP addresses are word addresses, so each increment of address specifies a different 32-bit word.
- Memory allocation depends on the OTP Revision written at the address 0x1F.

15.1 Murata Type 2AB EVB specific OTP

		Written to OTP				
		Calibrated and written to OTP				
OTP Address	Size (Used Bytes)	Byte_3	Byte_2	Byte_1	Byte_0	Programmed by
0x000	4					
0x001	4					
0x002	4			64 bit EUID		Customer
0x003	4			Alternative 64 bit EUID (Selected via reg/SR register)		Customer
0x004	4					
0x005	4					
0x006	4	{ "0001, 0000, 0001", "CHIP ID 5 nibbles (20 bits)" }				IC Prod. Test
0x007	4	["0001", "LOT ID - 7 nibbles (28 bits)"]				IC Prod. Test
0x008	3	VBAT @ 3.0 V	VBAT @ 3.62 V	VBAT @ 1.62 V		IC Prod. Test
0x009	1			Temp @ 22±2° C		IC Prod. Test
0x00A	4			Bias Tune		IC Prod. Test
0x00B	4	Antenna Delay - Rx RF Loop		Antenna Delay - Tx RF Loop		IC Prod. Test
0x00C	4	PDoA Iso. Ch9 RF2 → RF1	PDoA Iso. Ch9 RF1 → RF2	PDoA Iso. Ch5 RF2 → RF1	PDoA Iso. Ch5 RF1 → RF2	IC Prod. Test
0x00D	4	W.S. Lot ID [3]	W.S. Lot ID [2]	W.S. Lot ID [1]	W.S. Lot ID [0]	IC Prod. Test
0x00E	2			W.S. Lot ID [5]	W.S. Lot ID [4]	IC Prod. Test
0x00F	3		W.S. Wafer no.	W.S. Y Loc	W.S. X Loc	IC Prod. Test
0x010	4					Customer
0x011	4	Ch5 Tx Power Level - PRF64				Customer
0x012	4					Customer
0x013	4	Ch9 Tx Power Level - PRF64				Customer
0x014	4					Customer
0x015	4					Customer
0x016	4					Customer
0x017	4					Customer
0x018	4					Customer
0x019	4			Temperature	Voltage	Customer
0x01A	4	Ch5 PDoA Code (for offset)		Ch9 PDoA Code (for offset)		Customer
0x01B	4	Ch5 PG Count - PRF64		Ch9 PG Count - PRF64		Customer
0x01C	4	Ch5 Internal antenna delay - PRF64		Ch9 Internal antenna delay - PRF64		Customer
0x01D	4					Customer
0x01E	3	Ch5 PG Delay - PRF64	Ch9 PG Delay - PRF64		XTAL Trim	Customer
0x01F	1				OTP Rev.	Customer
0x020	4		Rx_Tune_Cal: DGS_CFG0			IC Prod. Test
0x021	4		Rx_Tune_Cal: DGS_CFG1			IC Prod. Test
0x022	4		Rx_Tune_Cal: DGS_CFG2			IC Prod. Test
0x023	4		Rx_Tune_Cal: DGS_CFG3			IC Prod. Test
0x024	4		Rx_Tune_Cal: DGS_CFG4			IC Prod. Test
0x025	4		Rx_Tune_Cal: DGS_CFG5			IC Prod. Test
0x026	4		Rx_Tune_Cal: DGS_CFG6			IC Prod. Test
0x027	4		Rx_Tune_Cal: DGC_LUT_0 - CH5			IC Prod. Test
0x028	4		Rx_Tune_Cal: DGC_LUT_1 - CH5			IC Prod. Test
0x029	4		Rx_Tune_Cal: DGC_LUT_2 - CH5			IC Prod. Test
0x02A	4		Rx_Tune_Cal: DGC_LUT_3 - CH5			IC Prod. Test
0x02B	4		Rx_Tune_Cal: DGC_LUT_4 - CH5			IC Prod. Test
0x02C	4		Rx_Tune_Cal: DGC_LUT_5 - CH5			IC Prod. Test
0x02D	4		Rx_Tune_Cal: DGC_LUT_6 - CH5			IC Prod. Test
0x02E	4		Rx_Tune_Cal: DGC_LUT_0 - CH9			IC Prod. Test
0x02F	4		Rx_Tune_Cal: DGC_LUT_1 - CH9			IC Prod. Test
0x030	4		Rx_Tune_Cal: DGC_LUT_2 - CH9			IC Prod. Test
0x031	4		Rx_Tune_Cal: DGC_LUT_3 - CH9			IC Prod. Test
0x032	4		Rx_Tune_Cal: DGC_LUT_4 - CH9			IC Prod. Test
0x033	4		Rx_Tune_Cal: DGC_LUT_5 - CH9			IC Prod. Test
0x034	4		Rx_Tune_Cal: DGC_LUT_6 - CH9			IC Prod. Test
0x035	4	PLL_Lock_Code				IC Prod. Test
0x036 - 0x05A	4	Unallocated				Customer
0x05B	1				Platform ID	Customer
0x05C - 0x05F	4	Unallocated				Customer
0x060	4	OSR Register (Special function register)				Reserved
0x061	1			Q_RR Register		Reserved
0x062 - 0x077	4	Unallocated				Customer
0x078	4	AES_Key [127:96] (Big endian order)				Customer
0x079	4	AES_Key [95:64] (Big endian order)				Customer
0x07A	4	AES_Key [63:32] (Big endian order)				Customer
0x07B	4	AES_Key [31:0] (Big endian order)				Customer
0x07C	4	AES_Key [255:224] (Big endian order)				Customer
0x07D	4	AES_Key [223:192] (Big endian order)				Customer
0x07E	4	AES_Key [191:160] (Big endian order)				Customer
0x07F	4	AES_Key [159:128] (Big endian order)				Customer

Figure 15.1: OTP for Type 2AB EVB

© 2025 Qorvo US, Inc.
Qorvo® Proprietary Information

15.2 OTP Customer part description

- 0x010: Not filled after production
- 0x011: FCC compliant Tx Power value using channel 5
- 0x012: Not filled after production
- 0x013: FCC compliant Tx Power value using channel 9
- 0x018: PGCOUNT (pulse generator count) value used during production tests
- 0x019: Temperature and Voltage during the production tests
- 0x01A: PDoA offsets for channel 5 and channel 9, to get the best PDoA correction
- 0x01B: PGCOUNT (pulse generator count) value to optimize bandwidth compensation over temperature stored at 0x019
- 0x01C: Antenna delays for channel 9 with STS segment length = 64
- 0x01E: PG Delay (Pulse generator delay) to optimize spectrum bandwidth | Crystal trimming to fine tune UWB central frequency
- 0x01F: OTP Revision: organization of OTP memory
- 0x036 - 0x05A: Free to use
- 0x05B: Platform ID: unique identifier to distinguish hardware platform
- 0x05C - 0x05F: Free to use
- 0x062 - 0x077: Free to use
- 0x078 - 0x07F is recommended for AES key, if needed

Note: Preprogrammed antenna delay values in the OTP are calibrated for the specific antennas provided with the development kit. If a different antenna is used in your solution, the antenna delay values must be recalibrated individually. For instructions, please refer to the [Distance and Angle Calibration](#) section.

16 Licenses in this SDK

Read the header of each file to know more about the license concerning this file.

Following licenses are used directly in the SDK:

- **Qorvo licenses** with identifiers **LicenseRef-Qorvo-1**, **LicenseRef-Qorvo-2**: majority of the source code (SDK/LICENSES directory).
- **Nordic Semiconductor ASA license**: sdk_config.h files (header of respective file).
- **CodeSourcery Inc license**: nRF linker scripts (header of respective file).
- **sha256 license**: sha256 library (SDK/Firmware/Libs/uwb-stackconfig_manager/src/sha256.c)

Following licenses are included from nRF5 SDK:

- **nRF5 SDK license**: The majority of the source code included in the nRF5 SDK (SDK/Firmware/SDK_BSP/Nordic/SDK_17_1_0/documentation/nRF5_Nordic_license.txt).
- **nRF5 Garmin Canada license**: Files released by Garmin Canada included in the nRF5 SDK (nRF5_Garmin_Canada_license.txt).
- **SoftDevice license**: The SoftDevice and its headers (SDK/Firmware/SDK_BSP/Nordic/SDK_17_1_0/components/softdevice/sxxx/doc/<license file>).
- **ARM 3-clause BSD license**: CMSIS and system files (SDK/Firmware/SDK_BSP/Nordic/SDK_17_1_0/components/toolchain).
- **FreeRTOS license**: FreeRTOS configuration files (FreeRTOSConfig.h in the examples that show how to use FreeRTOS).
- **Third-party licenses**: All third-party code contained in SDK/Firmware/SDK_BSP/Nordic/SDK_17_1_0external/ (respective licenses included in each of the imported projects).

The provided HEX files were compiled using the projects located in the folders. For license and copyright information, see the individual .c and .h files that are included in the projects.

17 Getting support and contacting Qorvo®

For the latest specifications, additional product information, worldwide sales and distribution locations:

Web: www.qorvo.com

Tel: 1-844-890-8163

Email: customer.support@qorvo.com

For additional technical assistance, visit and participate in our community forum at <https://forum.qorvo.com/>.

18 Important Notice

The information contained in this Document and any associated documents are believed to be reliable; however, Qorvo makes no warranties regarding Documents Information and assumes no responsibility or liability whatsoever for the use of said information. All Documents Information are subject to change without notice. Customers should obtain and verify the latest relevant Documents Information before placing orders for Qorvo® products. Documents Information or the use thereof do not grant, explicitly, implicitly or otherwise any rights or licenses to any third party with respect to patents or any other intellectual property whether with regard to such Documents Information themselves or anything described by such information.

DOCUMENTS INFORMATION DO NOT CONSTITUTE A WARRANTY WITH RESPECT TO THE PRODUCTS DESCRIBED HEREIN, AND QORVO HEREBY DISCLAIMS ANY AND ALL WARRANTIES WITH RESPECT TO SUCH PRODUCTS WHETHER EXPRESS OR IMPLIED BY LAW, COURSE OF DEALING, COURSE OF PERFORMANCE, USAGE OF TRADE OR OTHERWISE, INCLUDING THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Without limiting the generality of the foregoing, Qorvo® products are not warranted or authorized for use as critical components in medical, life-saving, or life-sustaining applications, or other applications where a failure would reasonably be expected to cause severe personal injury or death. Applications described in the Documents Information are for illustrative purposes only. Customers are responsible for validating that a particular product described in the Documents Information are suitable for use in a particular application.

FiRa, FiRa Consortium, the FiRa logo, the FiRa Certified logo, and FiRa tagline are trademarks or registered trademarks of FiRa Consortium or its licensor(s)/ supplier(s) in the US and other countries and may not be used without permission. All other trademarks, service marks, and product or service names are trademarks or registered trademarks of their respective owners.

© 2025 Qorvo US, Inc. All rights reserved. This document is subject to copyright laws in various jurisdictions worldwide and may not be reproduced or distributed, in whole or in part, without the express written consent of Qorvo US, Inc. | QORVO® is a registered trademark of Qorvo US, Inc.

Index

A

ACM, **3**
 AES, **3**
 ANIS, **3**
 AoA, **3**
 API, **3**
 ARM, **3**

B

BAL, **3**
 BLE, **3**
 BPRF, **3**

C

CDC, **3**
 CFO, **3**
 CIR, **3**
 CLI, **3**
 COM, **3**
 CRC, **3**

D

DEVKIT, **3**
 DS-TWR, **3**

E

EVB, **3**

F

FCC, **4**
 FiRa, **4**
 FoM, **4**
 FP, **4**
 FPU, **4**
 FSL, **4**
 FW, **4**

G

GCC, **4**
 GDB, **4**
 GPIO, **4**
 GUI, **4**

H

HAL, **4**

I

IDE, **4**
 IPATOV, **4**
 ISR, **4**
 IV, **4**

J

JSON, **4**

L

L1, **4**
 LL, **4**
 LLHW, **4**
 LNA, **4**
 LUT, **4**

M

MAC, **5**
 MCU, **5**

N

NA, **5**
 NI, **5**
 NVM, **5**

O

OOB, **5**
 OS, **5**
 OSAL, **5**
 OTP, **5**

P

PA, **5**
 PAC, **5**
 PCTT, **5**
 PDoA, **5**
 PHR, **5**
 PHY, **5**
 PLL, **5**
 PRF, **5**
 PSDU, **5**
 PSR, **5**

Q

QANI, **5**
 QHAL, **5**

QNIS, **5**
QOSAL, **6**

R

RAM, **6**
RFRAME, **6**
RFU, **6**
RR, **6**
RRU, **6**
RSL, **6**
RSP, **6**
RSSI, **6**
RSTU, **6**
RTOS, **6**
RTT, **6**
RX, **6**

S

SFD, **6**
SOC, **6**
SPI, **6**
SS-TWR, **6**
STS, **6**
SW, **6**
SWD, **6**

T

TWR, **6**
TX, **6**

U

UART, **7**
UCI, **7**
USB, **7**
UWB, **7**
UWBMAC, **7**
UWBS, **7**

X

XTAL, **7**