

# DW3000 and QM33100

## Qorvo Nearby Interaction

### Accessory Developer Guide

Qorvo

Release C

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Qorvo NI Communication Flow</b>	<b>4</b>
2.1	Out of Band Messages Sequence	4
2.2	UWB messages sequence	6
<b>3</b>	<b>SDK Architecture</b>	<b>8</b>
3.1	Overview	8
3.2	Folders structure	9
3.3	Layers	9
3.3.1	Board Abstraction Layer (Boards)	9
3.3.2	Hardware Abstraction Layer (HAL)	11
3.3.3	Libs Folder	25
3.3.4	OSAL folder	26
3.3.5	Projects folder	26
3.3.6	SDK BSP Folder	28
3.3.7	UWB Folder	28
3.3.8	BLE Folder	28
3.4	Linker sections	31
3.4.1	Flash sections	31
3.4.2	RAM sections	32
3.5	NVM configuration	32
3.5.1	Overview	32
3.5.2	Initialization sequence during powerup	33
<b>4</b>	<b>SDK Runtime</b>	<b>34</b>
4.1	Always running Threads	34
4.1.1	Default Task	34
4.2	Application specific Threads	34
4.2.1	UwbTask	34
4.2.2	ReportTask	34
<b>5</b>	<b>Building/Debugging the QNI Project</b>	<b>35</b>
5.1	Required Tools	35
5.1.1	Software	35
5.1.2	Hardware	35
5.2	Building	37
5.3	Debugging	39
<b>6</b>	<b>Customizing the firmware for your Project</b>	<b>43</b>
6.1	Step 1: Create a new project	43
6.2	Step 2: Adapt HAL to the SDK for your board	43
6.3	Step 3: Describing your board	43
6.4	Step 4: Select which Apps you need	43
6.5	OTP Memory Map	43
6.5.1	OTP Revision 1	44



6.5.2	OTP Revision 2 . . . . .	45
7	Referenced Documents	46
8	Revision History	47

## List of Figures

2.1	Qorvo NI Communication Flow . . . . .	4
2.2	Qorvo NI OOB messages sequence . . . . .	5
2.3	Qorvo NI UWB messages sequence . . . . .	7
3.1	Overview of the QNI Project, FreeRTOS based architecture. . . . .	8
3.2	Segger Embedded Studio project options . . . . .	23
3.3	Segger Embedded Studio include directories . . . . .	24
3.4	NVM configuration . . . . .	33
3.5	Initialization sequence . . . . .	33
5.1	Choose the correct SES solution for the Qorvo device . . . . .	37
5.2	Select the project file and click on “Open” . . . . .	37
5.3	Right click on the project name and select “Build” . . . . .	38
5.4	Build information when successfully built . . . . .	38
5.5	Segger Debug Mode. Disassembly on the left, running code on the right. . . . .	39
5.6	Debug controls, Run, Stop, Reset and execution in steps. . . . .	40
5.7	Click on the left of a code line to set a breakpoint (or press F9). . . . .	40
5.8	View breakpoints and the breakpoint window on the right. . . . .	41
5.9	Set PRIMASK to 1. . . . .	41
5.10	Debug Terminal showing RTT output. . . . .	42
6.1	OTP Revision 1 . . . . .	44
6.2	OTP Revision 2 . . . . .	45



# List of Tables

2.1	Accessory Configuration Data . . . . .	5
2.2	Preferred Update Rate Options . . . . .	6
3.1	List of targets/OS supported in the SDK . . . . .	27

# 1 Introduction

This document describes Qorvo Nearby Interaction accessory Application version C.

Nearby Interaction with Third Party Accessories was introduced starting from iOS 15 for iPhones equipped with a U1 chip:

- iPhone 11 (11, Pro and Pro Max)
- iPhone 12 (12, Mini, Pro and Pro Max)
- iPhone 13 (13, Mini, Pro and Pro Max)
- iPhone 14 (14, Pro and Pro Max)

With iOS 16 release, new features were added to Apple NI framework, like background mode, providing developers new ways to interact with iOS devices.

The project consists of a Qorvo NI ready board communicating with a compatible iPhone equipped with U1 chip and running the “Qorvo Nearby Interaction” iOS application.

The Qorvo Nearby Interaction accessory application is a “Segger Embedded Studio” project written in C language. The Qorvo NI ready boards are powered by Nordic nRF52XXX MCU, which provide the BLE communication layer. DW3000 and QM33100 Qorvo transceivers enable the UWB FiRa Double Sided Two-Way Ranging with iOS.

## 2 Qorvo NI Communication Flow

Qorvo NI uses BLE for discovery and exchange parameters between a Qorvo accessory and an iOS device, and then uses UWB for FiRa compliant (version 1.3) Double Sided Two-Way Ranging (DS-TWR), using the parameters exchanged in the BLE phase. The *NI communication flow* is shown in the following picture.

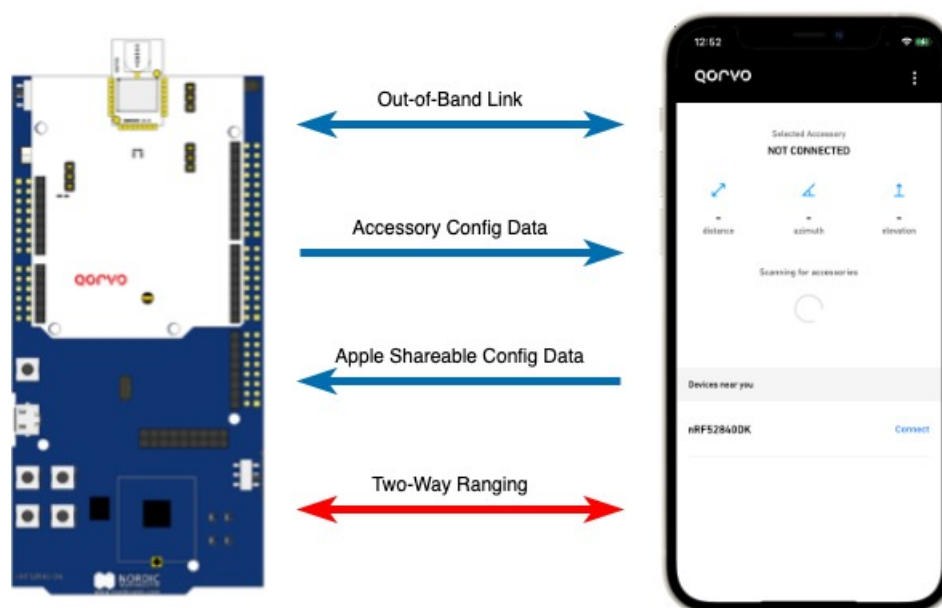


Fig. 2.1: Qorvo NI Communication Flow

### 2.1 Out of Band Messages Sequence

The project uses BLE as Out-of Band communication link (OOB), the BLE layer is based on the functions available by Nordic BLE SoftDevice which is “a feature-rich Bluetooth® Low Energy protocol stack for the nRF52832, nRF52833 and nRF52840 System-on-Chips (SoCs)”.

The iPhone will act as a Central (Scanner) connecting with the Qorvo Device as Peripheral (Advertiser) using the Qorvo NI Service (QNIS).

Once the OOB link is established, both NI applications (iOS and Device) can exchange messages, always starting with the **SampleAppMessageId**.

The *Qorvo NI OOB messages sequence* is shown in the following picture.

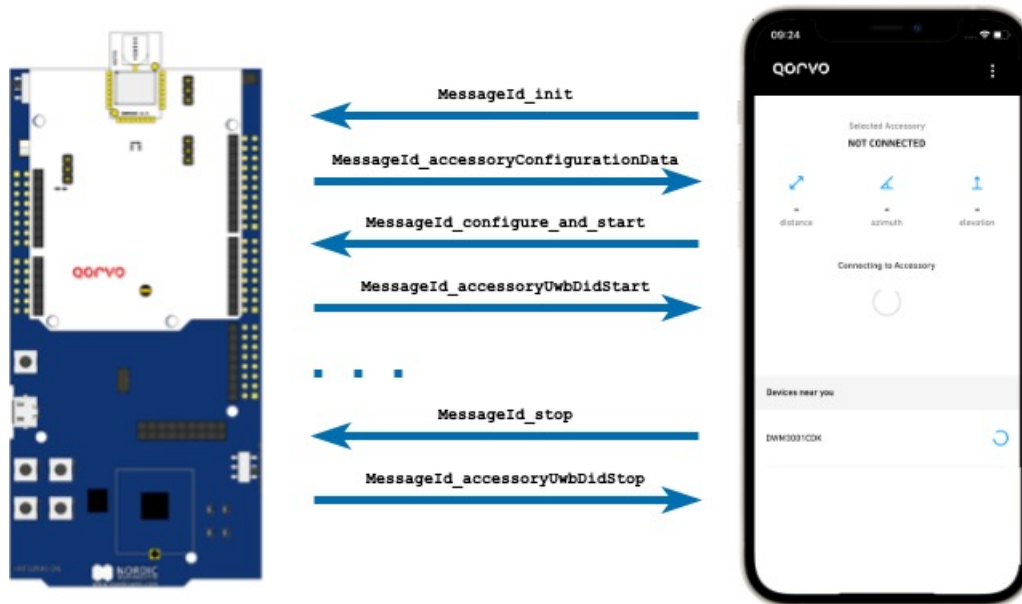


Fig. 2.2: Qorvo NI OOB messages sequence

**MessageId\_init:** The first message is sent by the iOS device. It is a single byte message with the MessageId\_init. After reception by the Qorvo device, it will use the NIQ library to generate the AccessoryConfigurationData and then, send it back via BLE.

**MessageId\_accessoryConfigurationData:** From the Qorvo device to the iPhone, the message starts with the MessageId\_accessoryConfigurationData and has the AccessoryConfigurationData as its payload, the AccessoryConfigurationData structure is detailed in the [Accessory Configuration Data](#) table.

**MessageId\_configure\_and\_start:** Next in the sequence, the iPhone generates and send the shareableConfigurationData structure, starting with MessageId\_configure\_and\_start. The Qorvo device identify the packet and parse the payload using the NIQ Library (niq\_configure\_and\_start\_uwb()), after that it sends the MessageId\_accessoryUwbDidStart command before switch to UWB and start Two-Way Ranging.

**MessageId\_accessoryUwbDidStart:** Last message from the Qorvo device to the iPhone before start Ranging. Single byte MessageId\_accessoryUwbDidStart.

**MessageId\_stop:** From iPhone to the Qorvo device, single byte message requesting the accessory to stop ranging.

**MessageId\_accessoryUwbDidStop:** Response to MessageId\_stop from the Qorvo device to the iPhone, single byte message confirming that the accessory has stopped.

Table 2.1: Accessory Configuration Data

Parameter	Data type	Size (bytes)	Description
MajorVersion	UInt16	2	The major version of the Nearby Interaction Accessory Protocol Specification
MinorVersion	UInt16	2	The minor version of the Nearby Interaction Accessory Protocol Specification
PreferredUpdateRate	UInt8	1	A selection of one of the options from <a href="#">Preferred Update Rate Options</a>
RFU	Bytes	10	Reserved for future use
UWBConfigDataLength	UInt8	1	The length of the UWB config data as provided by Qorvo UWB middleware
UWBConfigData	Bytes	Variable	the UWB config data as provided by Qorvo UWB middleware



Table 2.2: Preferred Update Rate Options

Update Rate Option	Value	Description
Automatic	0	The Apple device will select an update rate
Infrequent	10	Periodic updates on a scale of approximately once per second (~1.3Hz)
User Interactive	20	Update rate that is suitable for interactive experiences, On a scale of 5 updates per second (~5.5Hz)

Besides the OOB message sequence the BLE has not much use in this project, and more Sample Message IDs can be created for device purpose.

In this embedded app version, three new Message IDs were added. They were named as User ID to identify them as not part of NI Protocol:

**Userld\_getDeviceStruct:** Request from iPhone to the Qorvo device to send the current Device Struct, which will have device parameters and status. For future implementation.

**Userld\_setDeviceStruct:** Message from iPhone to the Qorvo device to set the Device Struct with the message payload data. For future implementation.

**Userld\_iOSNotify:** Message followed by a string, which will trigger a notification on the iOS device. Notifications only work when the app is in background.

## 2.2 UWB messages sequence

Once OOB connection is established and accessory and iOS exchanges necessary parameters, they will start the UWB Two-Way Ranging session.

It is provided by Qorvo UWB stack library, which supports FiRa standard (version 1.3).

The Qorvo accessory supply to iOS its “preferred parameters”, and it can act as “Initiator” or “Responder”.

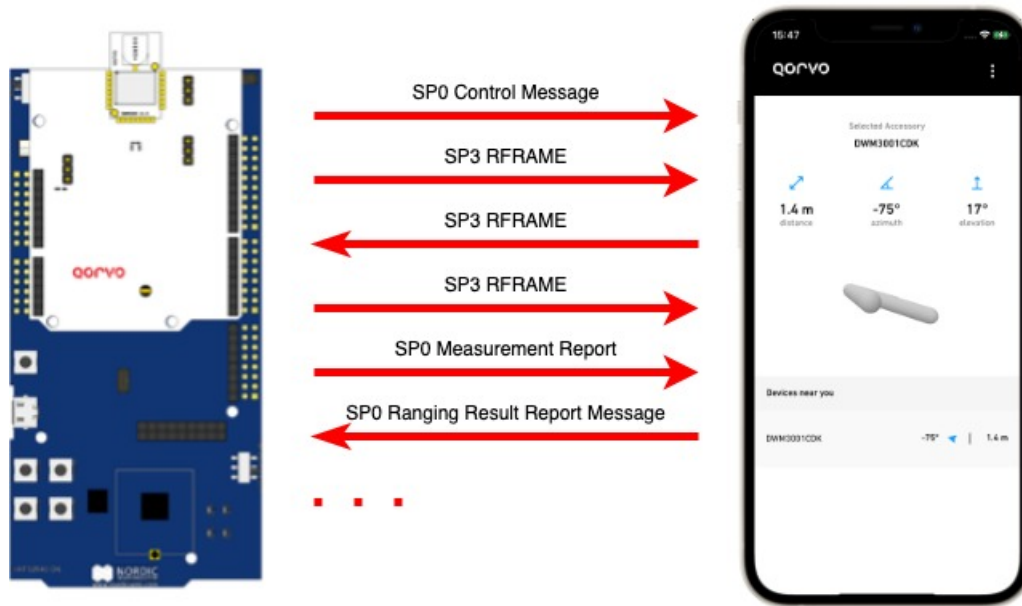


Fig. 2.3: Qorvo NI UWB messages sequence

The [Qorvo NI UWB messages sequence](#) shows a complete Unicast Double-Side Two-Way Ranging (DS-TWR) round between a Qorvo accessory and an iOS Device.

Until the NI Session is stopped, TWR rounds will continue at a rate defined by the iOS device, in the current version around 6 rounds per second in foreground mode, and 2 rounds per second in background mode.

iOS will always decide whether to start/stop UWB TWR and the session parameters.

For further information, please refer to the configuration messages/structures in the code.

## 3 SDK Architecture

### 3.1 Overview

The Qorvo NI accessory project is a CMSIS OS (CMSIS-RTOS) based application which has the Qorvo UWB driver, Qorvo UWB Stack, and the Nearby Interaction Library as the main components to implement the Nearby Interaction functionality.

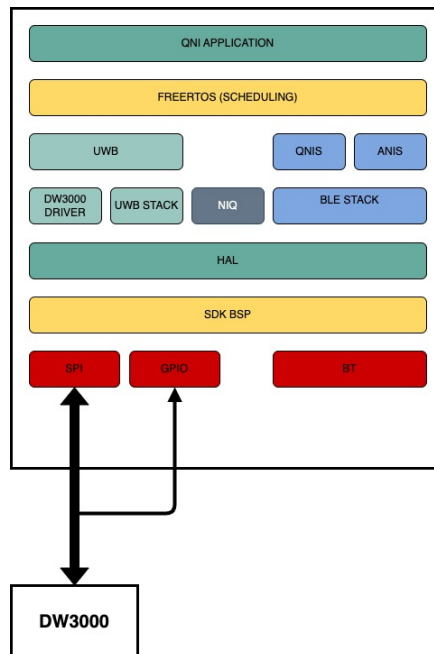


Fig. 3.1: Overview of the QNI Project, FreeRTOS based architecture.

The figure above shows the layered structure: “QNI Application”, “FreeRTOS”, “UWB Stack” the “BLE Stack” and “NIQ” library, “HAL” and “SDK BSP”.

## 3.2 Folders structure

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

root	
├── Libs	Components provided as libraries
│   ├── dwt_uwb_drivers	Libraries, UWB chip driver layer
│   ├── uwbstack	Libraries, MAC layer
│   └── niq	Nearby Interaction Library
├── Projects	Projects list grouped by OS and target
│   └── QANI	In this release the only project is QANI
│       ├── FreeRTOS	And the only OS is FreeRTOS
│           ├── <target_board>	This folder is named after the target Developer Kit used
│           └── QANI-FreeRTOS-Common	Common project files for all Developer Kits
├── SDK_BSP	Board support package provided by MCU manufacturer
│   └── Nordic	Nordic BSP folder
├── Src	
│   ├── AppConfig	Load/Save/Restore apps configuration
│   ├── Apps	Implementation of the three apps used by QNI
│   ├── Boards	API for the board's specific implementation
│   ├── Comm	
│       ├── Src	Projects Common source files
│           ├── BLE	BLE specific files
│               ├── anis	Apple NI Service
│               └── qnis	Qorvo NI Service
│   ├── 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, ...)
│   ├── OSAL	OS Abstraction Layer
│   ├── UWB	UWB implementation
│   └── UWBUtils	UWB supporting functions

## 3.3 Layers

### 3.3.1 Board Abstraction Layer (Boards)

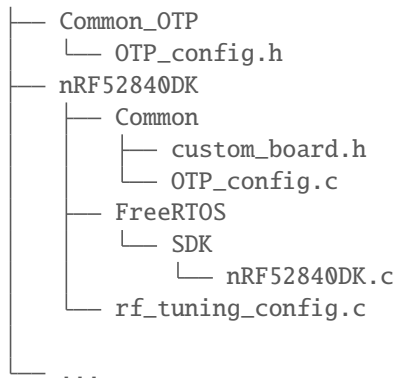
Boards folder contains a sub-folder for each target supported in the SDK.

#### 3.3.1.1 Files

root	
├── Src	
│   └── Boards	Boards Abstraction Layer
│       ├── Inc	HAL API header files
│           ├── int_priority.h	interrupt priorities definition
│           ├── rf_tuning_config.h	RF tuning parameters
│           └── thisBoard.h	Board API header file
│       └── Src	Boards source files
│           ├── Common_NRF	
│               └── board.c	

(continues on next page)

(continued from previous page)



### 3.3.1.2 API definition

#### thisboard.h

The main() function will call 3 key functions during the initialization phase

```
void BoardInit(void);
void board_interface_init(void);
int uwb_init(void);
```

#### custom\_board.h

This file is a key file in nRF SDK mechanism.

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

#### int\_priority.h

**Warning:** To be Documented.

We need to describe here what is the priority order the SDK and the MAC expect and why it is important to respect the correct priority levels.

#### Other important files

Two files are used to customize a particular target:

**<target\_name>.c** - Example nRF52840DK.c

Code used during startup for target's initialization (board, peripherals, interfaces)

**rf\_tuning\_config.c** - code to change the pulse shape between standard and alternate pulse shape and default values (Antenna delay, Offsets, Antenna, Tx Power, Crystal Trim, averaging, PG Delay)

### 3.3.1.3 Customizing to your board

To have a software fully working on a non-supported board, adapting it to the board is necessary.

This part is done in Src/Boards/Src. Recommended practice is to create a new folder and change the included headers to the project. (See [Porting to a new MCU](#) chapter):

../../../../../../../../Src/Boards/Src/nRF52840DK/Common and ../../../../../../Src/Boards/Src/nRF52840DK/FreeRTOS/QANI must be replaced.

In the new folder, three files are needed:

- rf\_tuning\_config.c
- custom\_board.h
- <board\_name>.c

rf\_tuning\_config.c contains all the radiofrequency related parameters. Tweaking them can be done later.

custom\_board.h contains the pin definition. This need to correspond to the board definitions such as LEDs, BUTTONs, UART, SPI.

<board\_name>.c contains the initialization functions of your board:

- 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
- uwb\_init(void) can be left as is as it is initializing UWB part and SPI

Any board specific function should be put in this folder.

## 3.3.2 Hardware Abstraction Layer (HAL)

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

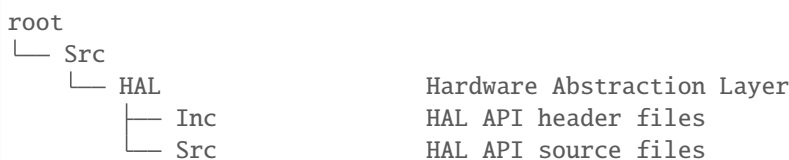
### 3.3.2.1 Overview

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.

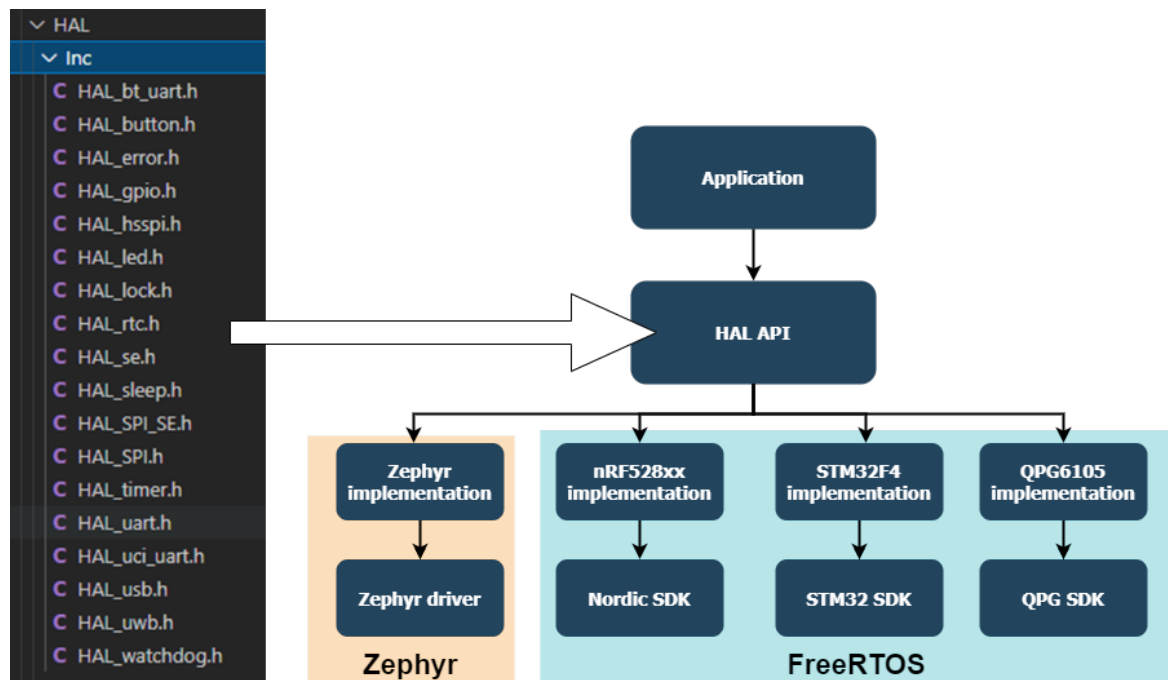
We provide HAL implementation for various boards.

If a use case is not covered in the supported list then refer to customization section for guidance on how to do the porting.

- Files location



- Layer overview



### 3.3.2.2 HAL SPI

This SPI Hardware Abstraction layer will abstract SPI accesses to the UWB transceiver.

#### 3.3.2.2.1 Files

```

root
├── Src
│   ├── HAL
│   │   ├── Inc
│   │   │   ├── HAL_SPI.h
│   │   │   └── HAL_SPI.c
│   │   └── Src
│   │       ├── nrfx
│   │       └── HAL_SPI.c
  
```

Hardware Abstraction Layer  
HAL API header files  
HAL SPI API header file  
HAL API source files  
HAL Nordic implementation  
HAL SPI implementation for Nordic

#### 3.3.2.2.2 API definition

##### 3.3.2.2.2.1 SPI functions API

```

struct spi_s
{
    void (*cs_low)(void *handler);
    void (*cs_high)(void *handler);
    void (*slow_rate)(void *handler);
    void (*fast_rate)(void *handler);
    int (*read)(void *handler, uint16_t headerLength, const uint8_t *headerBuffer, uint16_t_
    ↪ readlength, uint8_t *readBuffer);
    int (*write)(void *handler, uint16_t headerLength, const uint8_t *headerBuffer, uint16_t_
    ↪ readlength, const uint8_t *readBuffer);
  
```

(continues on next page)

(continued from previous page)

```
int (*read_write)(void *handler, uint8_t *readBuffer, uint16_t readlength, uint8_t *  
↪ *writebuffer, uint16_t writeLength);  
int (*write_with_crc)(void *handler, uint16_t headerLength, const uint8_t *headerBuffer, ↪  
↪ uint16_t bodyLength, const uint8_t *bodyBuffer, uint8_t crc8);  
void *handler;  
};  
typedef struct spi_s spi_t;
```

- **void** (\*cs\_low)(void \*handler);  
Unset Chip Select pin  
Parameters: SPI handler  
Return: none
- **void** (\*cs\_high)(void \*handler);  
Set Chip Select pin  
Parameters: SPI handler  
Return: none
- **void** (\*slow\_rate)(void \*handler);  
Deinit SPI instance Set SPI handler frequency to slow (frequency should be defined in init\_spi) Init SPI instance  
Parameters: SPI handler  
Return: none
- **void** (\*fast\_rate)(void \*handler);  
Deinit SPI instance Set SPI handler frequency to slow (frequency should be defined in init\_spi) Init SPI instance GPIO configuration may need to be changed to support rised frequency  
Parameters: SPI handler  
Return: none
- **int** (\*read)(void \*handler, uint16\_t headerLength, const uint8\_t \*headerBuffer, uint16\_t readlength, uint8\_t \*readBuffer);  
Unset Chip Select pin Read SPI frame from instance buffer Set Chip Select pin  
Parameters:  
  - void** \*handler: SPI handler
  - uint16\_t** headerLength: length of the header
  - const uint8\_t** \*headerBuffer: reception buffer of the header
  - uint16\_t** readlength: length of the frame without header
  - uint8\_t** \*readBuffer: reception buffer of the frame  
Return: error code
- **int** (\*write)(void \*handler, uint16\_t headerLength, const uint8\_t \*headerBuffer, uint16\_t readlength, const uint8\_t \*readBuffer);  
Unset Chip Select pin Write frame ti SPI instant buffer Set Chip Select pin  
Parameters:



**void** \*handler: SPI handler  
**uint16\_t** headerLength: length of the header  
**const uint8\_t** \*headerBuffer: dispatch buffer of the header to send  
**uint16\_t** readlength: length of the frame without header  
**const uint8\_t** \*readBuffer: dispatch buffer of the frame to send

Return: error code

- **int** (\*read\_write)(**void** \*handler, **uint8\_t** \*readBuffer, **uint16\_t** readlength, **uint8\_t** \*writebuffer, **uint16\_t** writeLength);

Unset Chip Select pin Read and write SPI frame Set Chip Select pin

Parameters:

**void** \*handler: SPI handler  
**uint8\_t** \*readBuffer: reception buffer  
**uint16\_t** readlength: length of the frame to receive  
**uint8\_t** \*writebuffer: dispatch buffer  
**uint16\_t** writeLength: length of the frame to send

Return: error code

- **int** (\*write\_with\_crc)(**void** \*handler, **uint16\_t** headerLength, **const uint8\_t** \*headerBuffer, **uint16\_t** bodyLength, **const uint8\_t** \*bodyBuffer, **uint8\_t** crc8);

---

**Note:** This function is unused

---

Write SPI frame with CRC

Parameters:

**void** \*handler: SPI handler  
**uint16\_t** headerLength: length of the header  
**const uint8\_t** \*headerBuffer: dispatch buffer of the header to send  
**uint16\_t** bodyLength: length of the frame without header  
**const uint8\_t** \*bodyBuffer: dispatch buffer of the frame to send  
**uint8\_t** crc8: calculated CRC

Return: error code

- **void** \*handler;

SPI handler

### 3.3.2.2.2 SPI GPIOs definition

```
struct spi_port_config_s
{
    uint32_t idx;
    uint32_t cs;
    uint32_t clk;
    uint32_t mosi;
    uint32_t miso;
    uint32_t min_freq;
    uint32_t max_freq;
};
typedef struct spi_port_config_s spi_port_config_t;
```

- `idx`  
Index of the SPI peripheral (example SPI0, SPI1, SPI2, ...)
- `cs, clk, mosi, miso`  
Index of the GPIO used for the CS, CLK, MOSI and MISO (example `cs=12` means CS pin is GPIO12)
- `min_freq`  
Low SPI clock speed frequency (kHz)
- `max_freq`  
High SPI clock speed frequency (kHz)

### 3.3.2.2.3 SPI initialization

```
const struct spi_s *init_spi(const spi_port_config_t *spi_port_cfg);
```

Initialisation of SPI connected to UWB chip

Parameters: spi config (see above)

Return: spi structure configured

### 3.3.2.3 HAL POWER

This file implements weak FreeRTOS hooks, applying optimized power settings.

#### 3.3.2.3.1 Files

```
root
├── Src
│   ├── HAL                                Hardware Abstraction Layer
│   │   ├── Src                            HAL API source files
│   │   │   ├── nrfx                       HAL Nordic implementation
│   │   │   └── HAL_power.c                HAL POWER hooks implementation for Nordic
```

### 3.3.2.3.2 API definition

- `__STATIC_INLINE void nrf_gpio_cfg_input_sense_none(uint32_t, nrf_gpio_pin_pull_t);`  
Used by `vPortSuppressTicksAndSleep()` to set a pin to a low power state  
Parameters:  
`uint32_t`: Pin number  
`nrf_gpio_pin_pull_t`: Pull configuration  
Return: none
- `void xPortSysTickHandler2(void);`  
Wrapper to not break the Nordic SDK while supporting (`configUSE_TICKLESS_IDLE == 2`)  
Parameters: none  
Return: none
- `void vPortSuppressTicksAndSleep(TickType_t);`  
Implementation of `vPortSuppressTicksAndSleep()` (weak and empty) from “hooks.c”, this function is called by `portSUPPRESS_TICKS_AND_SLEEP()`  
Parameters:  
`TickType_t`: Time to sleep, in FreeRTOS ticks  
Return: none

### 3.3.2.4 HAL ERROR

#### 3.3.2.4.1 Files

```

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

```

#### 3.3.2.4.2 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

### 3.3.2.5 HAL RTC

#### 3.3.2.5.1 Files

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

#### 3.3.2.5.2 API definition

```
struct hal_rtc_s
{
    void (*init)(void);
    void (*deinit)(void);
    void (*enableIRQ)(void);
    void (*disableIRQ)(void);
    void (*setPriorityIRQ)(void);
    uint32_t (*getTimestamp)(void);
    uint32_t (*getTimeElapsed)(uint32_t start, uint32_t stop);
    void (*reload)(uint32_t time);
    void (*configureWakeup_ms)(uint32_t period_ms);
    void (*configureWakeup_ns)(uint32_t period_ns);
    float (*getWakeupResolution_ns)(void);
    void (*setCallback)(void (*cb)(void));
};
extern const struct hal_rtc_s Rtc;
```

- **void (\*init)(void);**  
Initialization and enabling of the RTC driver instance  
Parameters: none  
Return: none
- **void (\*deinit)(void);**  
Disable and deinitialize RTC timer  
Parameters: none  
Return: none
- **void (\*enableIRQ)(void);**  
Enable RTC interrupt  
Parameters: none

Return: none

- **void** (\*disableIRQ)(**void**);

Disable RTC interrupt

Parameters: none

Return: none

- **void** (\*setPriorityIRQ)(**void**);

Set RTC Wakeup timer with a high priority

Parameters: none

Return: none

- **uint32\_t** (\*getTimestamp)(**void**);

Get RTC current counter value

Parameters: none

Return: RTC counter value

- **uint32\_t** (\*getTimeElapsed)(**uint32\_t** start, **uint32\_t** stop);

Returns elapsed time

Parameters:

start: Start date

stop: Stop date

Return: Elapsed time

- **void** (\*reload)(**uint32\_t** time);

Reload RTC registers after interrupt to prepare next compare match event

Parameters: Absolute value to be set in the compare register

Return: none

- **void** (\*configureWakeup\_ms)(**uint32\_t** period\_ms);

Setup RTC Wakeup timer period\_ms is awaiting time in ms

Parameters: Period in ms

Return: none

- **void** (\*configureWakeup\_ns)(**uint32\_t** period\_ns);

Setup RTC Wakeup timer period\_ns is awaiting time in ns

Parameters: Period in ns

Return: none

- **float** (\*getWakeupResolution\_ns)(**void**);

Get WKUP timer resolution

Parameters: none

Return: Resolution in ns

- **void** (\*setCallback)(**void** (\*cb)(**void**));

Set RTC callback

Parameters: callback

Return: none

### 3.3.2.6 HAL SLEEP

This API provides sleep functionalities.

---

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

---

#### 3.3.2.6.1 Files

```
root
├── Src
│   └── HAL
│       ├── Inc
│       │   ├── HAL_sleep.h    HAL API header files
│       │   └── HAL_SLEEP_API.h HAL SLEEP API header file
│       └── Src
│           ├── nrfx            HAL API source files
│           └── HAL_sleep.c     HAL Nordic implementation
│                               HAL SLEEP implementation for Nordic
```

#### 3.3.2.6.2 API definition

```
void deca_sleep(unsigned int time_ms);
```

Delay execution for specified time in ms

Parameters: time in ms of the delay

Return: none

```
void deca_usleep(unsigned long time_us);
void usleep(unsigned long time_us);
```

---

**Note:** deca\_usleep() and usleep() are identical to support legacy application

---

Delay execution for specified time in  $\mu$ s

Parameters: time in  $\mu$ s of the delay

Return: none

### 3.3.2.7 HAL TIMER

#### 3.3.2.7.1 Files

```
root
├── Src
│   ├── HAL
│   │   └── Hardware Abstraction Layer
│   ├── Inc
│   │   ├── HAL API header files
│   │   └── HAL_timer.h
│   │       └── HAL TIMER API header file
│   ├── Src
│   │   ├── HAL API source files
│   │   ├── nrfx
│   │   │   ├── HAL Nordic implementation
│   │   │   └── HAL_timer.c
│   │       └── HAL TIMER implementation for Nordic
```

#### 3.3.2.7.2 API definition

##### 3.3.2.7.2.1 Fast sleep timer

```
struct hal_fs_timer_s
{
    /* timer instance */
    void *timer;

    /* timer external ISR */
    void (*timerIsr)(void *dwchip);

    /* @brief Initialization/uninit of the Fast Sleep Timer
     *      cb - is the "timerIsr"
     */
    void (*init)(void *self, void *dwchip, void (*cb)(void *));

    /* @brief De-Initialization/uninit of the Fast Sleep Timer */
    void (*deinit)(void *self);

    /* @brief start the timer
     *      int_en - True/False to generate an interrupt on counter compare event
     *      time_us - is time to wait in microseconds
     *      corr_tick - the number of timer's ticks it needs to start the counting from
     */
    void (*start)(void *self, bool int_en, uint32_t next_cc_us, int corr_tick);

    /* @brief stop the timer
     */
    void (*stop)(void *self);

    /* @brief returns the fast sleep timer current tick value
     * this will be the number of ticks since its last reset (since last CC event)
     */
    uint32_t const (*get_tick)(void *self);

    /* timer tick frequency */
    int freq;
};
```

(continues on next page)

(continued from previous page)

```
typedef struct hal_fs_timer_s hal_fs_timer_t;
```

The purpose of this timer is to fast wake the ultra wide band chip from the sleep.

- **void \*timer;**  
Instance of the sleep timer. This timer is used to enable low-power sleep of UWB subsystem.

For Nordic target, the timer 4 instance is used as below:

```
static nrf_drv_timer_t fs_timer = NRF_DRV_TIMER_INSTANCE(4);
```

- **void (\*timerIsr)(void \*dwchip);**  
This is the callback which will be called by the hal\_fs\_timer. The function lp\_timer\_fira is used as a callback for the timer 4.

The role of this callback is to handle the state of the uwb chip and set the next operational state.

- **void (\*init)(void \*self, void \*dwchip, void (\*cb)(void \*));**  
Initialization of the fast Sleep Timer driver instance. This timer is: - Running on 1 MHz. - in continuous mode with counter compare events.

```
htimer->init(htimer, dw, lp_timer_fira);
```

- **void (\*deinit)(void \*self);**  
De-Initialization of the fast Sleep Timer driver instance.
- **void (\*start)(void \*self, bool int\_en, uint32\_t next\_cc\_us, int corr\_tick);**  
Start the timer.

Parameters:

**bool** int\_en: True/False to generate an interrupt on counter compare event.

**uint32\_t** next\_cc\_us: Time to wait in microseconds.

**int** corr\_tick: The number of timer's ticks it needs to start the counting from

Return: none

- **void (\*stop)(void \*self);**  
Stop the timer.
- **uint32\_t const (\*get\_tick)(void \*self);**  
Returns the fast sleep timer current tick value. This will be the number of ticks since its last reset (since last Counter Compare event).
- **int freq;**  
Timer tick frequency (1 MHz).

### 3.3.2.7.2.2 General purpose timer

```
struct hal_timer_s
{
    uint32_t (*init)(void);
    void (*start)(volatile uint32_t *p_timestamp);
    bool (*check)(uint32_t timestamp, uint32_t time);
};

extern const struct hal_timer_s Timer;
```

- **uint32\_t (\*init)(void);**



Initialize timer using interrupt to create a tick each one millisecond

Parameters: none

Return: error code

- **void** (\*start)(**volatile uint32\_t** \*p\_timestamp);

Save system timestamp

Parameters:

**volatile uint32\_t** \*p\_timestamp: pointer on current system timestamp

Return: none

- **bool** (\*check)(**uint32\_t** timestamp, **uint32\_t** time);

Check if time from current timestamp over expectation

Parameters:

**uint32\_t** timestamp: Current timestamp

**uint32\_t** time: Time expectation

**Return:**

True if time is over False if time is not over yet

### 3.3.2.8 HAL WATCHDOG

#### 3.3.2.8.1 Files

```
root
├── Src
│   ├── HAL
│   │   ├── Inc
│   │   │   ├── HAL_watchdog.h
│   │   │   └── HAL_watchdog.c
│   │   └── Src
│   │       ├── nrfx
│   │       └── HAL_watchdog.c
│   └── HAL_watchdog.h
│   └── HAL_watchdog.c
└── HAL_watchdog.h
    └── HAL_watchdog.c
```

#### 3.3.2.8.2 API definition

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

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

Initialiaze watchdog

Parameters: Reload time in ms

Return: none

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

Refresh/reload watchdog

Parameters: none

Return: none

### 3.3.2.9 Porting to a new MCU

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

Headers from Src/HAL/Inc are used in the rest of the software. So, porting a new MCU means adapting Src/HAL/Src content.

```

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

```

**Note:** Only Nordic MCUs using nRF SDK for the nRF52 Series SoCs with FreeRTOS are supported for now.

Other targets are under developement.

If a MCU brand other than Nordic is used, the corresponding SDK will be needed. SDKs can be stored in the SDK\_BSP folder. (nRF5 SDK path is: SDK\_BSP/Nordic/NORDIC\_SDK\_17\_1\_0)

HAL sources for Nordic can be found in Src/HAL/Src/nrfx.

To support any other MCU or OS, the best pratice is to create a new folder under SDK\_BSP/<MCU\_manufacturer> first.

Adding any header in the Segger Embedded Studio project, which is needed to use a new SDK, is done in the project option.

To access project option, click right on the project name on the left as below:

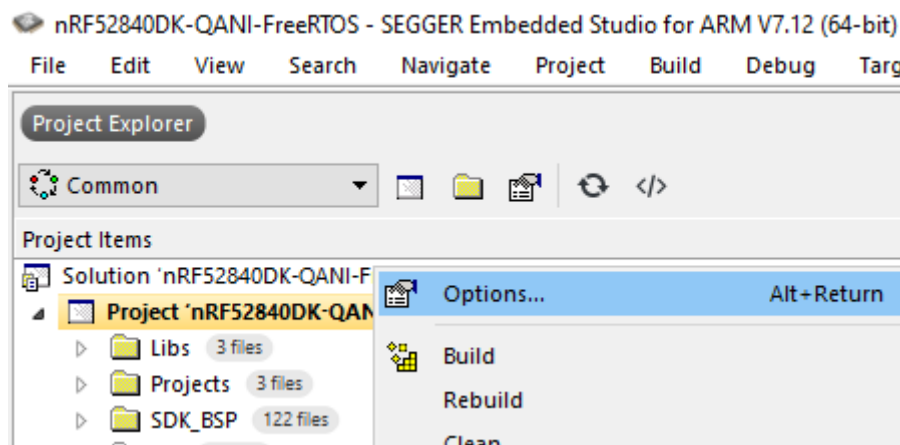


Fig. 3.2: Segger Embedded Studio project options

Then in Preprocessor, you can add a folder of header in User Include Directories.

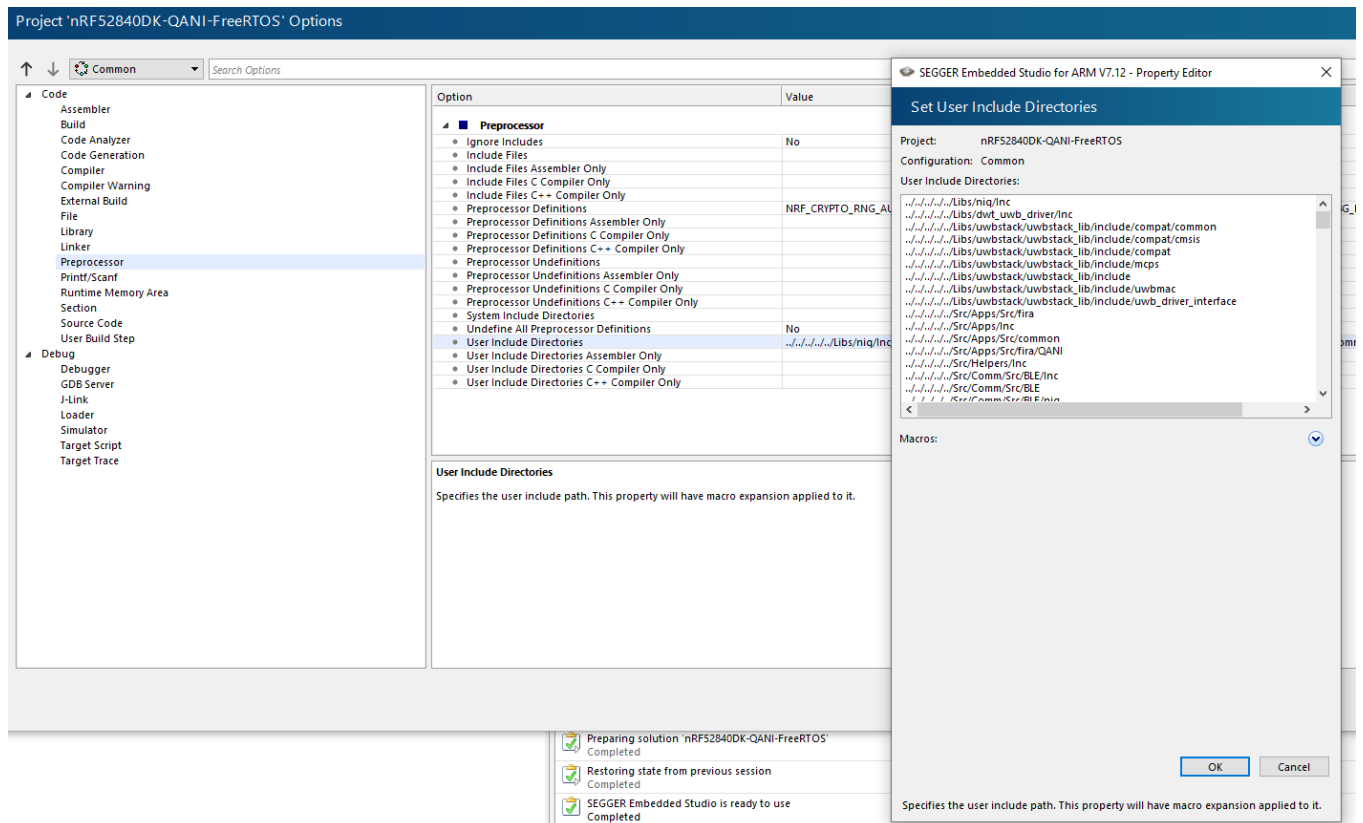


Fig. 3.3: Segger Embedded Studio include directories

Now, create a new folder under Src/HAL/Src, adapt all the functions and remove the default HAL files.

Here is an example for `bool isButtonPressed(void)` from `HAL_button.c`:

For Nordic, using nRF SDK for the nRF52 Series:

```

bool isButtonPressed(void)
{
    return (nrf_gpio_pin_read(BUTTON_1) == GPIO_PIN_RESET);
}

```

For ST MCU, using ST SDK it should be:

```

bool isButtonPressed(void)
{
    return HAL_GPIO_ReadPin(USER_BUTTON_GPIO_PORT, USER_BUTTON_PIN) == GPIO_PIN_SET;
}

```

For other implementation, using the associated SDK:

```

bool isButtonPressed(void)
{
    return <Use the SDK function to read the GPIO corresponding to the button>;
}

```

This task needs to be done on the HAL functions to have a fully working porting. Some function might not be necessary depending on your hardware/OS (example `void configure_button(void)`).

### 3.3.3 Libs Folder

#### 3.3.3.1 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.

#### 3.3.3.2 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.

#### 3.3.3.3 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.

#### 3.3.3.4 NIQ Library

The NIQ library defines the OOB messages and provides the code structures and functions to initialize and run the FiRa Session. The main functions are:

```
int niq_init(void (*start_uwb)(void),
             void (*stop_uwb)(void),
             void (*crypto_init)(void),
             void (*crypto_deinit)(void),
             void (*crypto_range_vector_generator)(uint8_t * const data, size_t data_len)):
```

To be called during the app initialization. It sets callback functions needed to handle the Nearby Interaction Session:

**start\_uwb()**: Implementation to start the UWB TWR session. If implementing NI for another platform/architecture a function to start/resume FiRa TWR must be the parameter here;

**stop\_uwb()**: Implementation to stop an ongoing UWB TWR session. If implementing NI for another platform/architecture a function to stop/pause FiRa TWR must be the parameter here;

**crypto\_init()**: Initialization of the platform crypto module, it can be an empty function if the chosen crypto module does not require to be initialized.

`crypto_deinit()`: Deinitialization of the platform crypto module, it can be an empty function if the chosen crypto module does not require to be deinitialized.

`crypto_range_vector_geneator()`: Function (or wrapper) required to generate random numbers used by each NI Session.

```
void niq_set_ranging_role(uint8_t role)
```

To set the accessory role, 0 for Controlee/Responder, 1 for Controller/Initiator. Preferable to call during the app initialization. The iOS device will assume the opposite role when starting FiRa TWR.

```
void niq_populate_accessory_uwb_config_data(void * out_buffer, uint8_t * out_buffer_len)
```

After connecting to an iOS device (BLE) the first step after receiving “Messageld\_init” from iOS is to send the Accessory Configuration Data. The embedded application initializes an `AccessoryConfigurationData` structure and use this function to populate it before sending to the iOS device.

---

**Note:** Right after calling `niq_populate_accessory_uwb_config_data()` the embedded app calls `set_accessory_uwb_config_data()` from `ble_anis.c`, using the BLE packet payload as parameter. This is to save the message content to the “Apple NI Service” characteristic, “Accessory Configuration Data”, to be read by the iOS app when requested.

---

```
int niq_configure_and_start_uwb(uint8_t *payload, uint8_t length, void *arg)
```

Last step before start ranging, the embedded device receives “Messageld\_configure\_and\_start” from the iOS app and call this function, using as parameter the message payload, payload size (message length - 1, due to the one byte Message ID) and the main FiRa Configuration structure. This function will set the configuration structure and start ranging with the iOS device, calling the `start_uwb()` callback set in `niq_init()`.

### 3.3.4 OSAL folder

OSAL stands for OS Abstraction Layer

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

- Dynamic memory (calloc, malloc, free)
- Critical section access (enter/leave)
- OS task / signal management

---

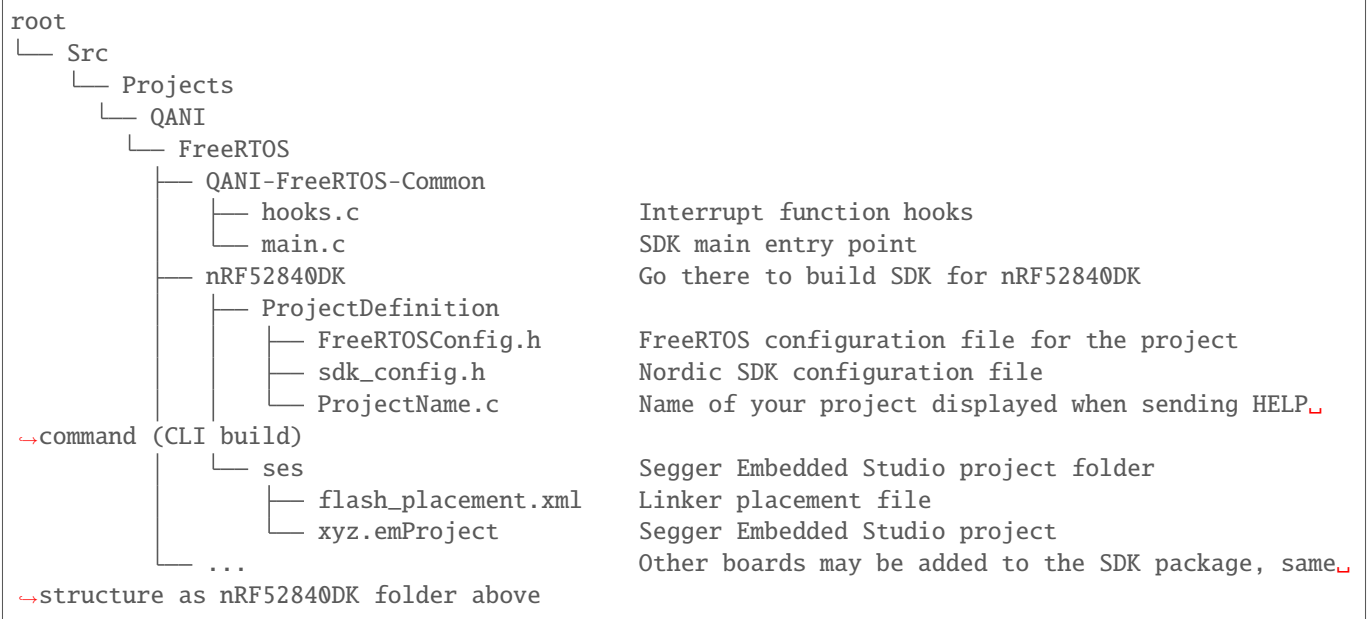
**Note:** OSAL API will be deprecated and replaced with a new QOSAL (Qorvo OS Abstraction layer in a future release)

---

### 3.3.5 Projects folder

Projects folder contains the SDK project files, split according to the OS used:

### 3.3.5.1 Files



### 3.3.5.2 SDK support list

Table 3.1: List of targets/OS supported in the SDK

Target	FreeRTOS	Zephyr
DWM3001CDK	Yes	No
nRF52832-DK	Yes	No
nRF52833-DK	Yes	No
nRF52840-DK	Yes	No
QTag	Yes	No
Murata Type 2AB EVB	Yes	No

### 3.3.5.3 Customizing for my project

Check [Libraries for a different MCU core](#) chapter for the list of supported devices.

#### 3.3.5.3.1 FreeRTOS

FreeRTOS is the OS provided in the SDK, so using it will ease the creation of a new project.

In this case, duplicating an already existing project is the easiest way to proceed.

Go to Projects/<SDK>/FreeRTOS, and duplicate the folder nRF52840DK.

If the new project is based on an already supported MCU, choose the corresponding project. The already supported MCUs are:

- nRF52832
- nRF52833
- nRF52840

Open this new folder. The Segger Embedded Studio project is located in the “ses” folder and is called nRF52xxxDK-QANI-FreeRTOS.emProject.

Feel free to rename the file first then the solution and the project after opening it the Segger Embedded Studio project.

### 3.3.5.3.2 Other OS

---

**Note:** This chapter will be filled out in a future release.

---

Provided Segger Embedded Studio projects are built around FreeRTOS. Using a different OS will require much more changes.

Check [OSAL folder](#) for further information.

## 3.3.6 SDK BSP Folder

SDK BSP folder includes the Board Support Package (drivers) needed for the different targets supported.

## 3.3.7 UWB Folder

UWB folder contains the source code to:

- Interact with the MAC
- Manage chip states
- Control some chip registers (Crystal trimming)
- Determine RF parameters (Lambda, distance between antennas for AoA, LUT)
- Convert PDoA in AoA

## 3.3.8 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.

### 3.3.8.1 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 (QNUS 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.

### 3.3.8.2 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 Sequence](#) 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.

## 3.4 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.

### 3.4.1 Flash sections

#### 3.4.1.1 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.

#### 3.4.1.2 NVM section

This section is a placeholder for storing configuration which will not be lost between 2 power cycles.

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 guaranty that the checksum will be incorrect at 1st powerup after download to force a factory default to take place.

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

#### 3.4.1.3 UWB Drivers section

This section is only valid for DW3 QM33 based projects.

It is a placeholder for storing the list of possible DW3 QM33 drivers.

It starts with symbol `__dw_drivers_start` and ends with symbol `__dw_drivers_end`.

Drivers from the lib `dwt_uwb_driver` are defining sections `.dw_drivers`.

During the driver probing phase, the function will browse this section through all the register drivers.

It will determine which one is appropriate for the version of DW3 QM33 connected to the main CPU.

#### 3.4.1.4 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 default function defined with symbol `.config_entry`.

If the Configuration CRC is invalid or if the user inserts a condition to RESTORE Configuration, then a factory default will be called.

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;
```

## 3.4.2 RAM sections

### 3.4.2.1 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`.

It is immediately followed by a CRC.

The end symbol for RAM Config + CRC block is `__rconfig_crc_end`

Variables and structures are defined using sections `.rconfig`.

For example, the FiRa Application will store its NVM parameter like this:

```
static fira_param_t fira_config_ram __attribute__((section(".rconfig"))) = {0};
```

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

## 3.5 NVM configuration

### 3.5.1 Overview

- The way the configuration is handled inside the SDK is done through linker sections.
- All the variable that have to be stored in NVM and kept between power cycles are stored in a consecutive RAM section.
- This RAM section is ended with a CRC16.
- 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.
- Any command will update the RAM version of the configuration. The RAM version is considered as volatile.
- Sending the SAVE command will save by copy the RAM configuration block into Flash.
- Sending RESTORE command will restore default configuration.

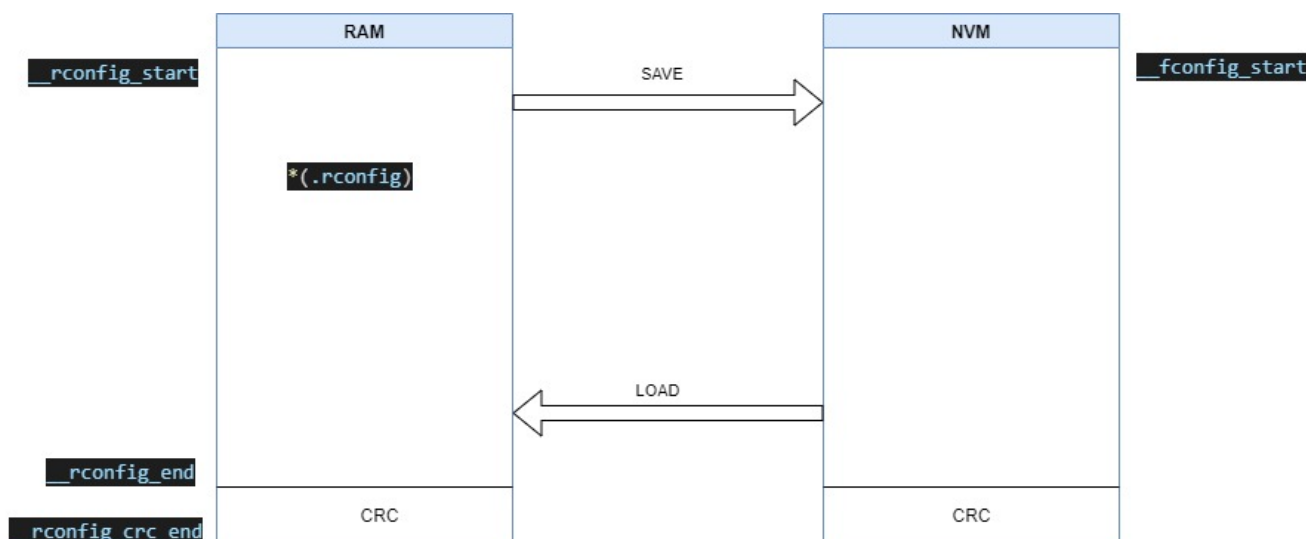


Fig. 3.4: NVM configuration

### 3.5.2 Initialization sequence during powerup

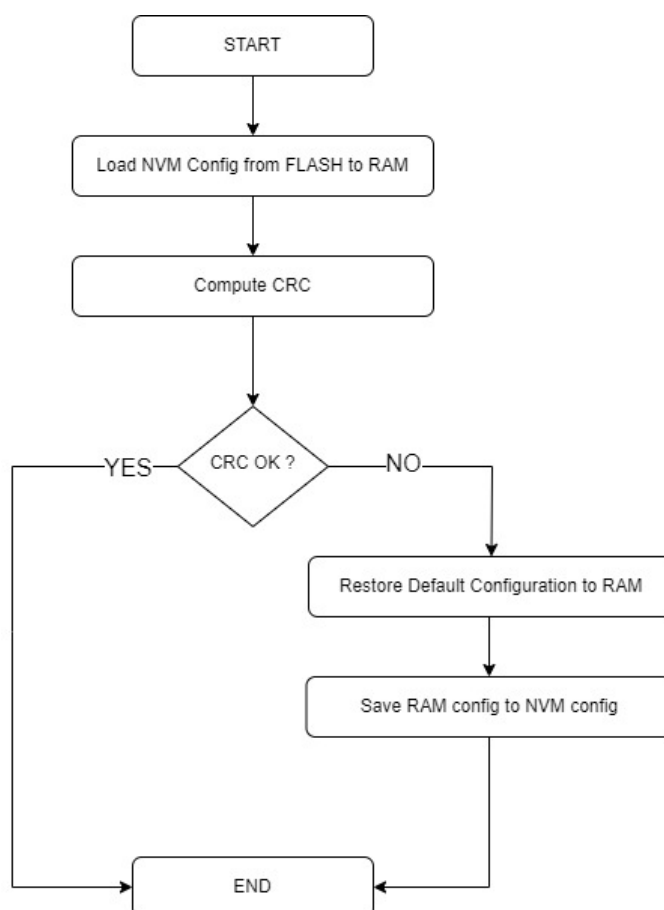


Fig. 3.5: Initialization sequence

## 4 SDK Runtime

### 4.1 Always running Threads

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

- Default Task

#### 4.1.1 Default Task

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.

- If an event is received within the timeout, Default task stops all running top-level tasks and it starts the requested top-level application.

This task is executed endlessly.

### 4.2 Application specific Threads

#### 4.2.1 UwbTask

Supports the FiRa part of the application. NIQ Lib can stop/resume it when a NI Session is started/stopped, using StopUwbTask() and ResumeUwbTasks().

#### 4.2.2 ReportTask

Report Task is used to handle the data reporting from the MAC layer to the application layer.

## 5 Building/Debugging the QNI Project

The QNI firmware is complementary to the Qorvo Nearby Interaction for iOS, so is useful being able to follow the device status when learning or debugging your own code.

### 5.1 Required Tools

The QNI sources include Segger Embedded Studio projects to each supported target. The sources are common to all “.emProject” projects.

#### 5.1.1 Software

To debug or implement your own changes to the QNI application it will be required Segger Embedded Studio [version 5.x]:

[Segger Embedded Studio for ARM<sup>1</sup>](#)

Segger Embedded Studio is a paid tool, but freely available for development kits evaluation (No commercial use), such as the DWM3001-CDK or nRF52xxx-DKs.

---

**Note:** A free SES license to use with Nordic products can be obtained [here<sup>2</sup>](#)

---

**Warning:** Segger Embedded Studio version 6.x presented issues when building projects using “\_\_vprintf.h”, so the recommended version is 5.x.

**Warning:** The project was tested with Segger Embedded Studio version 5.60a and 5.68.

#### 5.1.2 Hardware

It is required to have one of the Qorvo development kit, ready for Nearby Interaction such as:

- DWM3000EVB or QM33100EVB connected to a nRF52840-DK
- DWM3000EVB or QM33100EVB connected to a nRF52833-DK
- DWM3000EVB or QM33100EVB connected to a nRF52832-DK
- DWM3001CDK
- QTag (Qorvo Reference Design)

---

<sup>1</sup> <https://www.segger.com/downloads/embedded-studio/>

<sup>2</sup> [https://wiki.segger.com/Get\\_a\\_License\\_for\\_Nordic\\_Semiconductor\\_Devices](https://wiki.segger.com/Get_a_License_for_Nordic_Semiconductor_Devices)

- MurataEVB Type 2AB

More info at: [UWB SOLUTIONS COMPATIBLE WITH APPLE'S U1 CHIP](https://www.qorvo.com/innovation/ultra-wideband/products/ulwb-solutions-compatible-with-apple-u1)<sup>3</sup>

---

<sup>3</sup> <https://www.qorvo.com/innovation/ultra-wideband/products/ulwb-solutions-compatible-with-apple-u1>

## 5.2 Building

Open Segger Embedded Studio, click on File→Open Solution (Ctrl+Shift+O), navigate to the QNI Release folder and open the FreeRTOS folder, it has a list of sub-folders, one for each supported device.

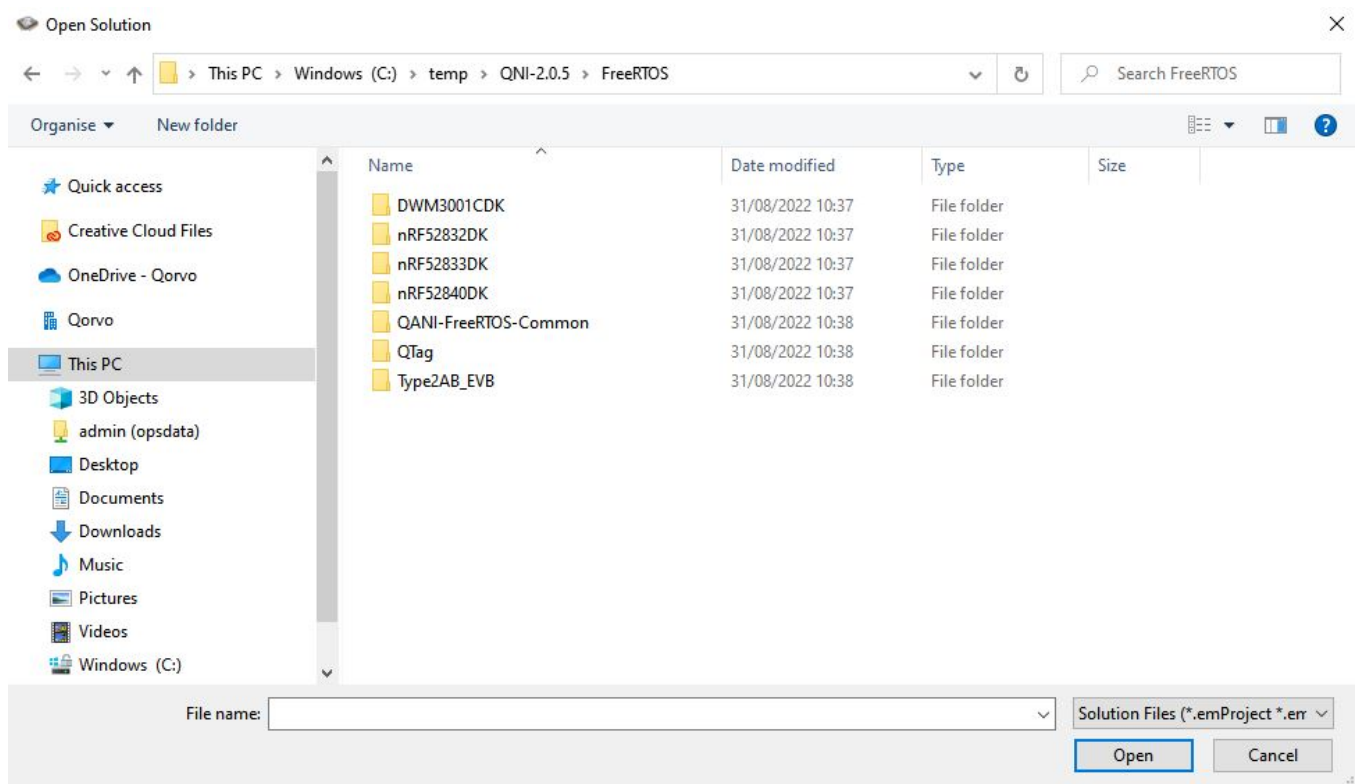


Fig. 5.1: Choose the correct SES solution for the Qorvo device

Open the folder that matches the development kit and open the “/ses/” folder inside, it contains the project file “.emProject”. Select it and click on “Open”

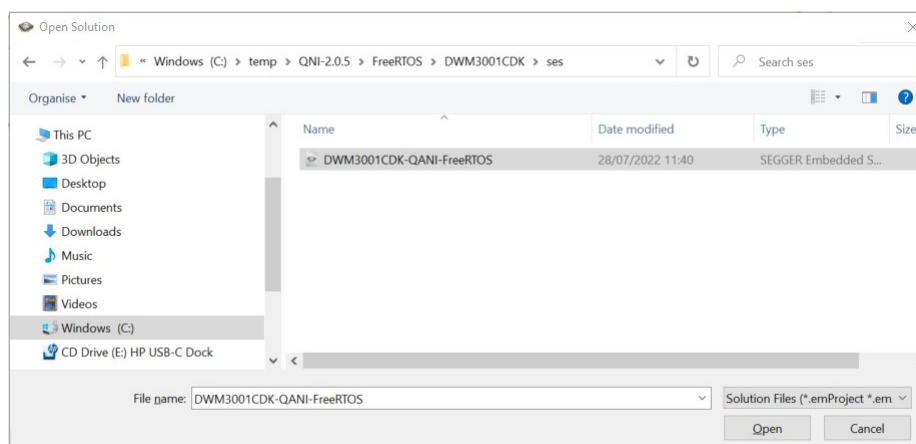


Fig. 5.2: Select the project file and click on “Open”



In the Solution structure on the left, right click on the project name and select “Build” (or select the project and press “F7”).

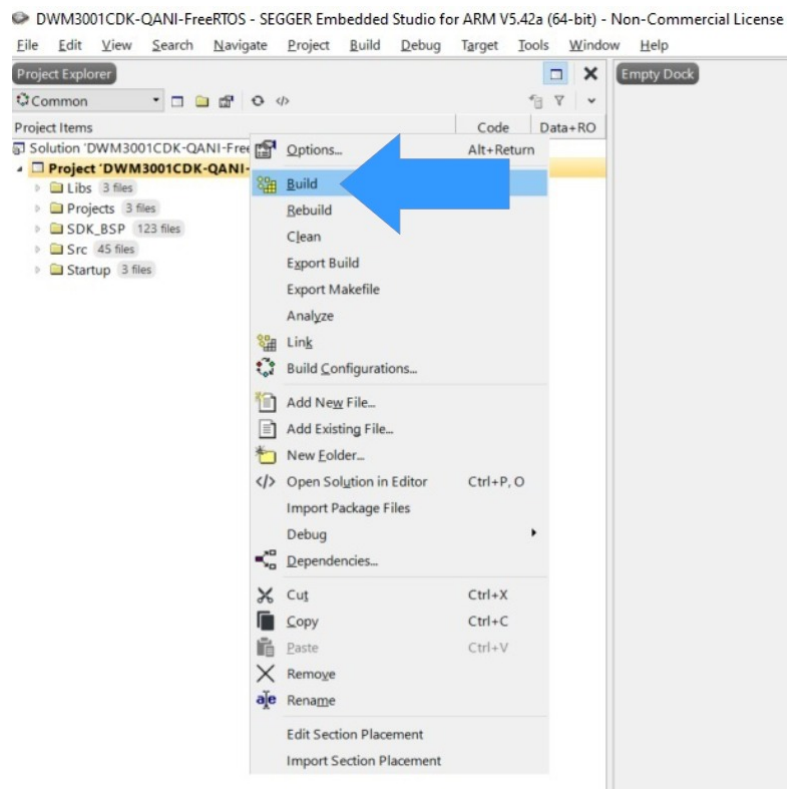


Fig. 5.3: Right click on the project name and select “Build”

When finished, the Output window (bottom in the middle) will show the build information.

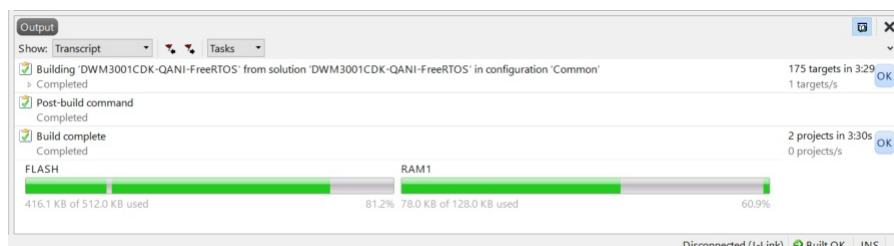


Fig. 5.4: Build information when successfully built

## 5.3 Debugging

With the project building debug can be started. Connect the development kit into a free USB port of the PC running Segger Embedded Studio, it must be connected via the J-Link USB. Then click on Debug→Go, the project will be flashed to the development kit and Segger will switch to Debug Mode:

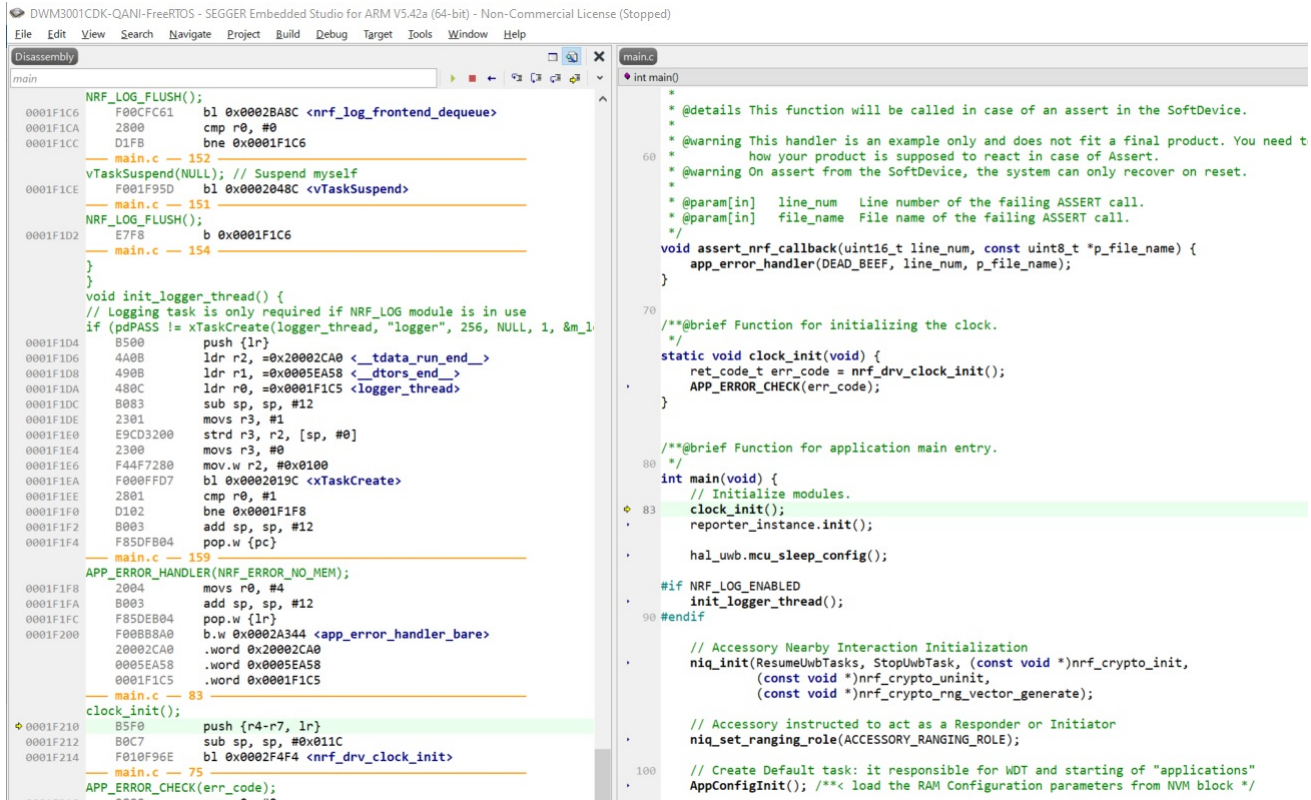


Fig. 5.5: Segger Debug Mode. Disassembly on the left, running code on the right.

The window in the middle will show the code in execution, on the left the functions broken in assembly lines. The execution can be controlled by the controls shown in [Debug controls](#), [Run](#), [Stop](#), [Reset](#) and [execution in steps](#).

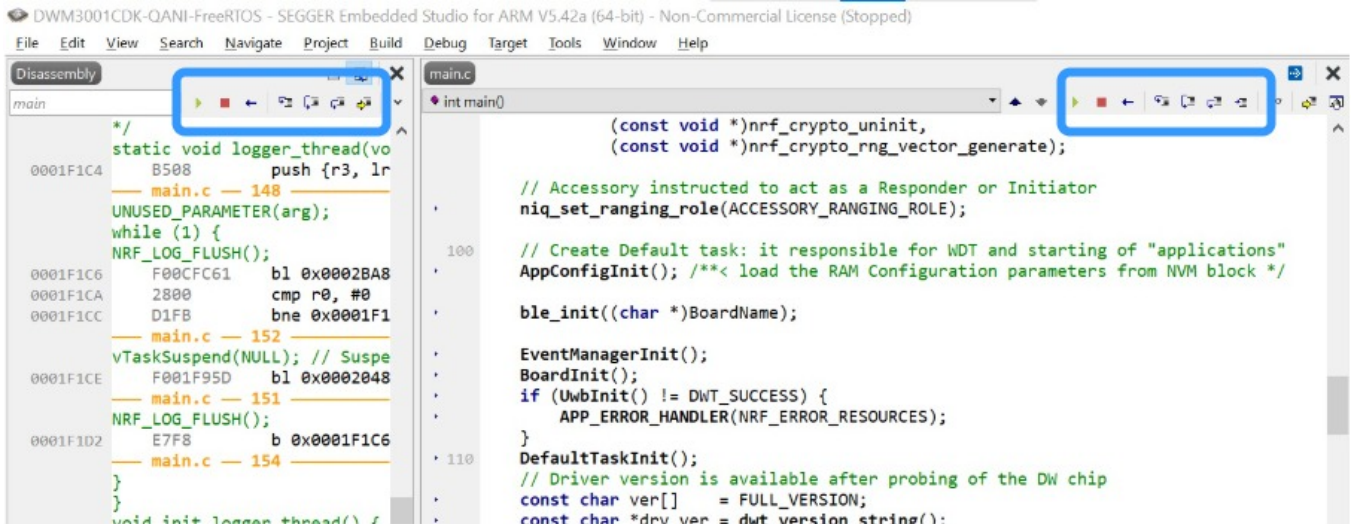


Fig. 5.6: Debug controls, Run, Stop, Reset and execution in steps.

Breakpoints can be added to the code, and when the execution reaches the line with a breakpoint it will be paused.

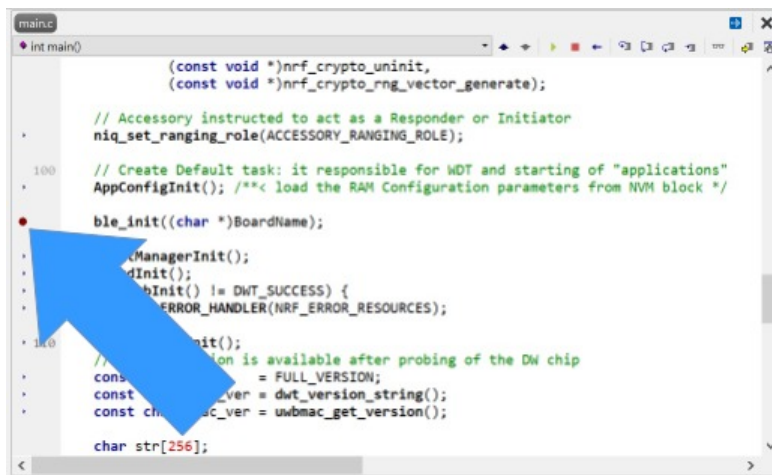


Fig. 5.7: Click on the left of a code line to set a breakpoint (or press F9).

To view all breakpoints click on Debug→Breakpoints→Breakpoints (Ctrl+Alt+B), a window on the right will list all breakpoint.

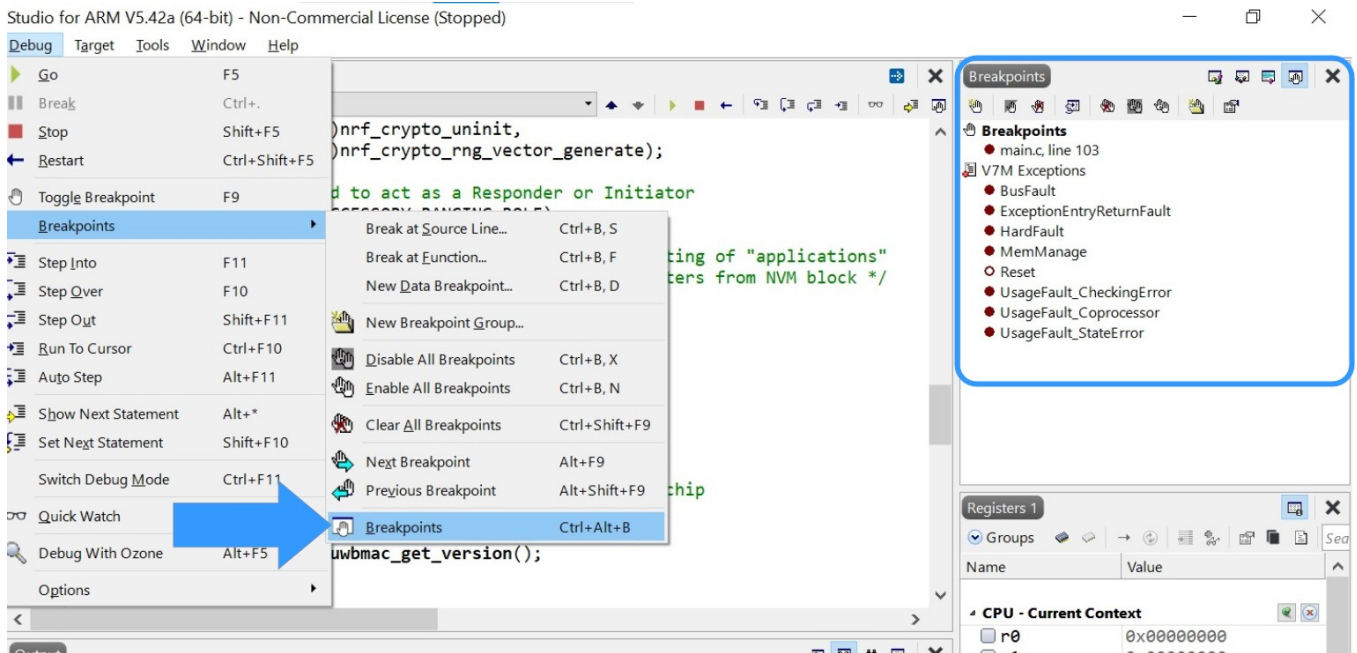


Fig. 5.8: View breakpoints and the breakpoint window on the right.

**Note:** When debugging using SoftDevice, it's important to set the PRIMASK register to 1 as shown in [Set PRIMASK to 1](#). below in order to prevent the activation of all exceptions with configurable priority.

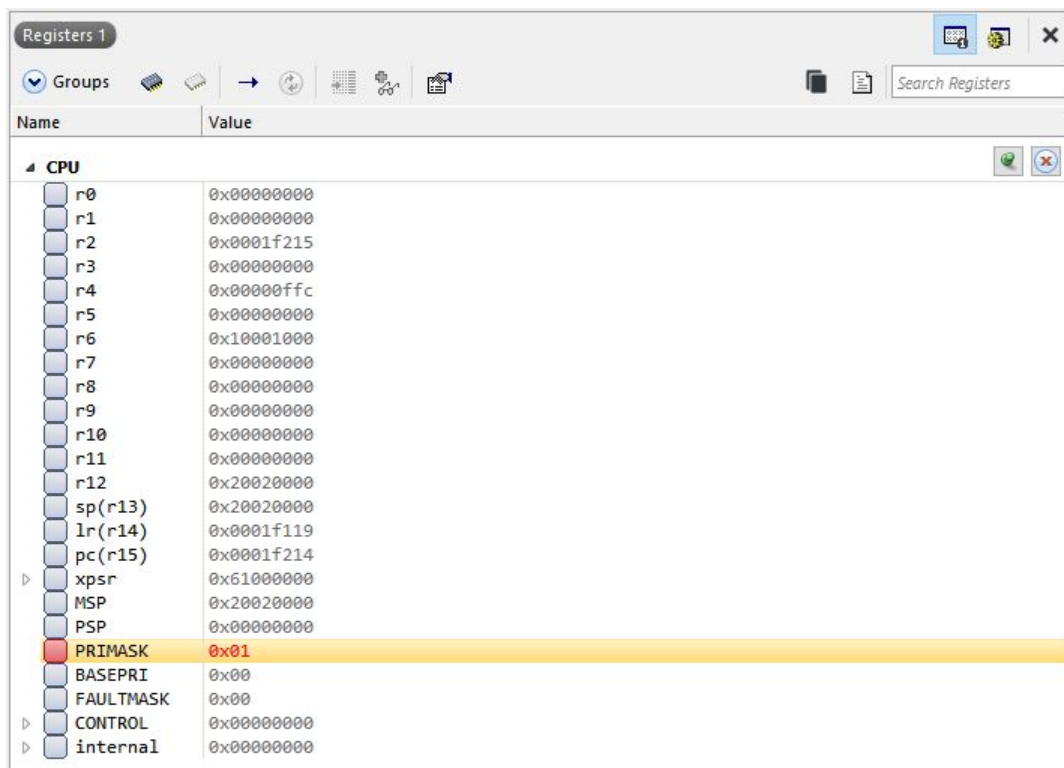
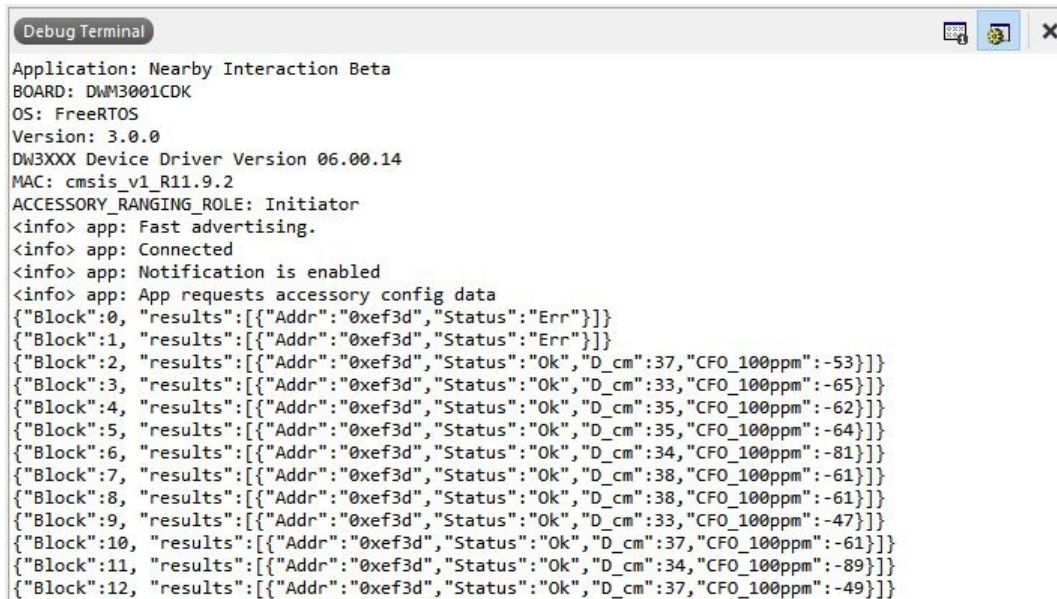


Fig. 5.9: Set PRIMASK to 1.



The QNI Projects use RTT to output important info and debug lines, when in debugging mode you can see the RTT output in the Debug Terminal (window on the bottom middle)



```
Application: Nearby Interaction Beta
BOARD: DWM3001CDK
OS: FreeRTOS
Version: 3.0.0
DW3XXX Device Driver Version 06.00.14
MAC: cmsis_v1_R11.9.2
ACCESSORY_RANGING_ROLE: Initiator
<info> app: Fast advertising.
<info> app: Connected
<info> app: Notification is enabled
<info> app: App requests accessory config data
{"Block":0, "results":[{"Addr":"0xef3d", "Status":"Err"}]}
{"Block":1, "results":[{"Addr":"0xef3d", "Status":"Err"}]}
{"Block":2, "results":[{"Addr":"0xef3d", "Status":"Ok", "D_cm":37, "CFO_100ppm":-53}]}
{"Block":3, "results":[{"Addr":"0xef3d", "Status":"Ok", "D_cm":33, "CFO_100ppm":-65}]}
{"Block":4, "results":[{"Addr":"0xef3d", "Status":"Ok", "D_cm":35, "CFO_100ppm":-62}]}
{"Block":5, "results":[{"Addr":"0xef3d", "Status":"Ok", "D_cm":35, "CFO_100ppm":-64}]}
{"Block":6, "results":[{"Addr":"0xef3d", "Status":"Ok", "D_cm":34, "CFO_100ppm":-81}]}
{"Block":7, "results":[{"Addr":"0xef3d", "Status":"Ok", "D_cm":38, "CFO_100ppm":-61}]}
{"Block":8, "results":[{"Addr":"0xef3d", "Status":"Ok", "D_cm":38, "CFO_100ppm":-61}]}
{"Block":9, "results":[{"Addr":"0xef3d", "Status":"Ok", "D_cm":33, "CFO_100ppm":-47}]}
{"Block":10, "results":[{"Addr":"0xef3d", "Status":"Ok", "D_cm":37, "CFO_100ppm":-61}]}
{"Block":11, "results":[{"Addr":"0xef3d", "Status":"Ok", "D_cm":34, "CFO_100ppm":-89}]}
{"Block":12, "results":[{"Addr":"0xef3d", "Status":"Ok", "D_cm":37, "CFO_100ppm":-49}]}
```

Fig. 5.10: Debug Terminal showing RTT output.

## 6 Customizing the firmware for your Project

### 6.1 Step 1: Create a new project

Please refer to [Customizing for my project](#) section for details.

### 6.2 Step 2: Adapt HAL to the SDK for your board

If you are using one of the MCU supported by Qorvo as a standard UWB SDK project, you can skip this section and directly go to step 3.

Else, you will need to adapt the HAL to the MCU SDK provided by your MCU manufacturer.

Please refer to HAL [Overview](#) section for understanding what needs to be done.

Please refer to the HAL [Porting to a new MCU](#) section for more details

### 6.3 Step 3: Describing your board

Please refer to [Customizing to your board](#) section for details.

### 6.4 Step 4: Select which Apps you need

This chapter will be filled out in a future release.

### 6.5 OTP Memory Map

OTP memory map contains values calibrated after production.

The OTP memory locations are defined in the following tables. The 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.

## 6.5.1 OTP Revision 1

	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	64 bit EUID				Customer	
0x001	4						
0x002	4						
0x003	4						
0x004	4	Alternative 64 bit EUID (Selected via reg/SR register)				Customer	
0x005	4						
0x006	4	LDO Tune				IC Prod. Test	
0x007	4	{ "0001,0000,0001" , "CHIP ID 5 nibbles (20 bits)" }					
0x008	3	{ "0001" , "LOT ID - 7 nibbles (28 bits)" }				IC Prod. Test	
0x009	1	VBAT @ 3.0 V		VBAT @ 3.62 V	VBAT @ 1.62 V		
0x00A	4	Bias Tune				Temp @ 22±2° C	IC Prod. Test
0x00B	4						
0x00C	4	Antenna Delay - Rx RF Loop		Antenna Delay - Tx RF Loop		IC Prod. Test	
0x00D	4	PDoA Iso. Ch9 RF2 → RF1	PDoA Iso. Ch9 RF1 → RF2	PDoA Iso. Ch5 RF2 → RF1	PDoA Iso. Ch5 RF1 → RF2		
0x00E	4	W.S. Lot ID [3]	W.S. Lot ID [2]	W.S. Lot ID [1]	W.S. Lot ID [0]	IC Prod. Test	
0x00F	2	W.S. Lot ID [5]		W.S. Lot ID [4]			
0x010	3	W.S. Wafer no.		W.S. Y Loc	W.S. X Loc	IC Prod. Test	
0x011	4	Ch5 Tx Power Level - PRF16					
0x012	4	Ch5 Tx Power Level - PRF64				Customer	
0x013	4	Ch9 Tx Power Level - PRF16					
0x014	4	Ch9 Tx Power Level - PRF64				Customer	
0x015	4						
0x016	4					Customer	
0x017	4						
0x018	4	Ch5_PGCNT		Ch9_PGCNT		Customer	
0x019	4						
0x01A	4	CH5 Rx Antenna Delay - PRF64		CH5 Tx Antenna Delay - PRF64		Customer	
0x01B	4	CH5 Rx Antenna Delay - PRF16		CH5 Tx Antenna Delay - PRF16			
0x01C	4	CH9 Rx Antenna Delay - PRF64		CH9 Tx Antenna Delay - PRF64		Customer	
0x01D	4	CH9 Rx Antenna Delay - PRF16		CH9 Tx Antenna Delay - PRF16			
0x01E	1	Frame Duration - us			Xtal Trim [6:0]	Customer	
0x01F	1	Platform ID			Cal Rev		
0x020	4	Rx_Tune_Cal: DGS_CFG0			OTP Rev.	IC Prod. Test	
0x021	4	Rx_Tune_Cal: DGS_CFG1					
0x022	4	Rx_Tune_Cal: DGS_CFG2				IC Prod. Test	
0x023	4	Rx_Tune_Cal: DGS_CFG3					
0x024	4	Rx_Tune_Cal: DGS_CFG4				IC Prod. Test	
0x025	4	Rx_Tune_Cal: DGS_CFG5					
0x026	4	Rx_Tune_Cal: DGS_CFG6				IC Prod. Test	
0x027	4	Rx_Tune_Cal: DGC_LUT_0 - CH5					
0x028	4	Rx_Tune_Cal: DGC_LUT_1 - CH5				IC Prod. Test	
0x029	4	Rx_Tune_Cal: DGC_LUT_2 - CH5					
0x02A	4	Rx_Tune_Cal: DGC_LUT_3 - CH5				IC Prod. Test	
0x02B	4	Rx_Tune_Cal: DGC_LUT_4 - CH5					
0x02C	4	Rx_Tune_Cal: DGC_LUT_5 - CH5				IC Prod. Test	
0x02D	4	Rx_Tune_Cal: DGC_LUT_6 - CH5					
0x02E	4	Rx_Tune_Cal: DGC_LUT_0 - CH9				IC Prod. Test	
0x02F	4	Rx_Tune_Cal: DGC_LUT_1 - CH9					
0x030	4	Rx_Tune_Cal: DGC_LUT_2 - CH9				IC Prod. Test	
0x031	4	Rx_Tune_Cal: DGC_LUT_3 - CH9					
0x032	4	Rx_Tune_Cal: DGC_LUT_4 - CH9				IC Prod. Test	
0x033	4	Rx_Tune_Cal: DGC_LUT_5 - CH9					
0x034	4	Rx_Tune_Cal: DGC_LUT_6 - CH9				IC Prod. Test	
0x035	4	PLL_Lock_Code					
0x036 - 0x05F	4	Unallocated				Customer	
0x060	4	QSR Register (Special function register)					
0x061	1				Q_RR Register	Reserved	
0x062 - 0x07F	4	Unallocated					
0x078	4	AES_Key [127:96] (Big endian order)				Customer	
0x079	4	AES_Key [95:64] (Big endian order)					
0x07A	4	AES_Key [63:32] (Big endian order)				Customer	
0x07B	4	AES_Key [31:0] (Big endian order)					
0x07C	4	AES_Key [255:224] (Big endian order)				Customer	
0x07D	4	AES_Key [223:192] (Big endian order)					
0x07E	4	AES_Key [191:160] (Big endian order)				Customer	
0x07F	4	AES_Key [159:128] (Big endian order)					

Fig. 6.1: OTP Revision 1

## 6.5.2 OTP Revision 2

	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	64 bit EUID				Customer
0x001	4					
0x002	4					
0x003	4					
0x004	4	Alternative 64 bit EUID (Selected via reg/SR register)				Customer
0x005	4					
0x006	4	LDO Tune				IC Prod. Test
0x007	4	{ "0001,0000,0001" , "CHIP ID 5 nibbles (20 bits)" }				
0x008	3	{ "0001" , "LOT ID - 7 nibbles (28 bits)" }				IC Prod. Test
0x009	1	VBAT @ 3.0 V		VBAT @ 3.62 V	VBAT @ 1.62 V	
0x00A	4	Temp @ 22±2° C				IC Prod. Test
0x00B	4	Bias Tune				
0x00C	4	Antenna Delay - Rx RF Loop		Antenna Delay - Tx RF Loop		IC Prod. Test
0x00D	4	PDoA Iso. Ch9 RF2 → RF1	PDoA Iso. Ch9 RF1 → RF2	PDoA Iso. Ch5 RF2 → RF1	PDoA Iso. Ch5 RF1 → RF2	
0x00E	4	W.S. Lot ID [3]	W.S. Lot ID [2]	W.S. Lot ID [1]	W.S. Lot ID [0]	IC Prod. Test
0x00F	2	W.S. Lot ID [5]		W.S. Lot ID [4]		
0x010	3	W.S. Wafer no.		W.S. Y Loc	W.S. X Loc	IC Prod. Test
0x011	4	Ch5 Tx Power Level - PRF16				
0x012	4	Ch5 Tx Power Level - PRF64				Customer
0x013	4	Ch9 Tx Power Level - PRF16				
0x014	4	Ch9 Tx Power Level - PRF64				Customer
0x015	4	Ch5 PDoA Offset		Ch9 PDoA Offset		
0x016	4					Customer
0x017	4					
0x018	4	Ch5_PGCNT		Ch9_PGCNT		Customer
0x019	4					
0x01A	4	Ch5 Rx Antenna Delay - PRF64		Ch5 Tx Antenna Delay - PRF64		Customer
0x01B	4	Ch5 Rx Antenna Delay - PRF16		Ch5 Tx Antenna Delay - PRF16		
0x01C	4	Ch9 Rx Antenna Delay - PRF64		Ch9 Tx Antenna Delay - PRF64		Customer
0x01D	4	Ch9 Rx Antenna Delay - PRF16		Ch9 Tx Antenna Delay - PRF16		
0x01E	1	Frame Duration - us			Xtal Trim [6:0]	Customer
0x01F	1	Platform ID		Cal Rev	OTP Rev.	
0x020	4	Rx_Tune_Cal: DGS_CFG0				IC Prod. Test
0x021	4	Rx_Tune_Cal: DGS_CFG1				
0x022	4	Rx_Tune_Cal: DGS_CFG2				IC Prod. Test
0x023	4	Rx_Tune_Cal: DGS_CFG3				
0x024	4	Rx_Tune_Cal: DGS_CFG4				IC Prod. Test
0x025	4	Rx_Tune_Cal: DGS_CFG5				
0x026	4	Rx_Tune_Cal: DGS_CFG6				IC Prod. Test
0x027	4	Rx_Tune_Cal: DGC_LUT_0 - CH5				
0x028	4	Rx_Tune_Cal: DGC_LUT_1 - CH5				IC Prod. Test
0x029	4	Rx_Tune_Cal: DGC_LUT_2 - CH5				
0x02A	4	Rx_Tune_Cal: DGC_LUT_3 - CH5				IC Prod. Test
0x02B	4	Rx_Tune_Cal: DGC_LUT_4 - CH5				
0x02C	4	Rx_Tune_Cal: DGC_LUT_5 - CH5				IC Prod. Test
0x02D	4	Rx_Tune_Cal: DGC_LUT_6 - CH5				
0x02E	4	Rx_Tune_Cal: DGC_LUT_0 - CH9				IC Prod. Test
0x02F	4	Rx_Tune_Cal: DGC_LUT_1 - CH9				
0x030	4	Rx_Tune_Cal: DGC_LUT_2 - CH9				IC Prod. Test
0x031	4	Rx_Tune_Cal: DGC_LUT_3 - CH9				
0x032	4	Rx_Tune_Cal: DGC_LUT_4 - CH9				IC Prod. Test
0x033	4	Rx_Tune_Cal: DGC_LUT_5 - CH9				
0x034	4	Rx_Tune_Cal: DGC_LUT_6 - CH9				IC Prod. Test
0x035	4	PLL_Lock_Code				
0x036 - 0x05F	4	Unallocated				Customer
0x060	4	QSR Register (Special function register)				
0x061	1				Q_RR Register	Reserved
0x062 - 0x077	4	Unallocated				
0x078	4	AES_Key [127:96] (Big endian order)				Customer
0x079	4	AES_Key [95:64] (Big endian order)				
0x07A	4	AES_Key [63:32] (Big endian order)				Customer
0x07B	4	AES_Key [31:0] (Big endian order)				
0x07C	4	AES_Key [255:224] (Big endian order)				Customer
0x07D	4	AES_Key [223:192] (Big endian order)				
0x07E	4	AES_Key [191:160] (Big endian order)				Customer
0x07F	4	AES_Key [159:128] (Big endian order)				

Fig. 6.2: OTP Revision 2



## 7 Referenced Documents

1. Qorvo Nearby Interaction resources  
<https://www.qorvo.com/feature/uwb-solutions-compatible-with-apple-u1>
2. Apple® Nearby Interaction Source code example  
[https://developer.apple.com/documentation/nearbyinteraction/implementing\\_spatial\\_interactions\\_with\\_third-party\\_accessories](https://developer.apple.com/documentation/nearbyinteraction/implementing_spatial_interactions_with_third-party_accessories)
3. Apple® Nearby Interaction Accessory Protocol Specification Release R2 (login required)  
<https://developer.apple.com/nearby-interaction/specification/>
4. Nordic Semiconductors developer's site  
<https://infocenter.nordicsemi.com>

## 8 Revision History

Version	Date	Comment
C	2023-04-06	Expanded the Accessory Developer Guide
B	2022-12-21	iOS 16 features update
A	2022-09-01	Initial version