# UM11516

## Component Library – Sensor Drivers Component

Rev. 1 — 2 November 2020 User manual

**Document information**

| Information | Content |
|---|---|
| Keywords | Component Library, Sensor Drivers |
| Abstract | Getting started with sensor drivers component |

## Revision history

| Rev | Date | Description |
|---|---|---|
| 1 | 20201102 | Initial release |

# 1 Prerequisites

This document assumes following prerequisites prior to attempting use of this platform agnostic component library – sensor drivers:

- Only basic software development knowledge is needed when using the example application (provided in this component library package) with the same hardware used in the example application.
- The user is familiar with the chosen microcontroller unit (MCU), corresponding software development kit (SDK) and cross-compilation tool chain to integrate sensor driver component.
- The user is familiar with the MCU SDK implementation for underlying microcontroller peripherals such as $I^2C$, SPI, etc., in order to integrate with the sensor driver component.

# 2 Overview

The sensor drivers component is a development model that provides sensor driver development rules for specific sensors manufactured by NXP with platform independent interfaces. The platform interface provides abstraction to the underlying communication driver in SDK, tool chains and MCUs.

## 2.1 Sensor driver component design

The sensor driver is an abstraction of the sensor driver interface, which has specific communication interfaces defined. It provides users the flexibility of drop the files in from sources and includes or from the library into the user application space. It could run as a standalone application in the application space or run in a multi-threaded environment. In the multi-threaded environment, the user application is responsible for handling the multi-threading synchronization and resource handling. It is designed to work seamlessly in any SDK environment and application resource handlers.

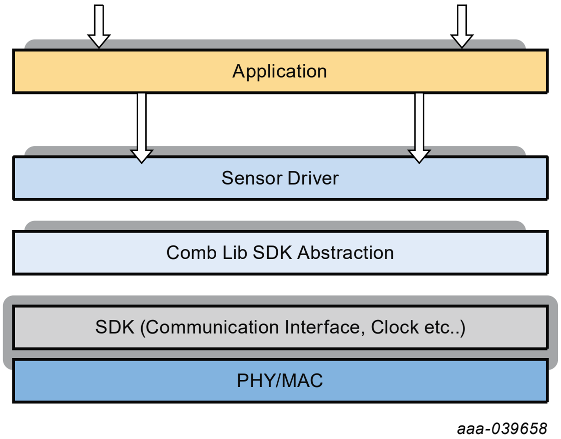


Figure 1. Sensor driver component design

# 3 Sensor driver component integration for non-example SDKs

The sensor driver models are microcontroller agnostic. This section describes development steps to integrate sensor drivers into any microcontroller software development kit (SDK). The sensor driver interfaces are generic for all sensor operations by using the common sensor communication virtual interface. The virtual interfaces (sensor_comm.c and sensor_comm.h) need to be implemented by the user of sensor driver component for underlying microcontroller SDK communication interfaces similar to $I^2C$, and SPI, for example. The implementation could be just the integration of underlying SDK drivers ($I^2C$, SPI) to the common interface. The example application included in this sensor component package provides the reference for the integration.

## 3.1 Sensor driver component directory structure

This section provides a snapshot of the sensor driver component directory structure. The below provided snapshot shows directory structure for the sensor drivers component.

```
sensor_driver/
|-- <src>common
|    |-- sensor_comm.c
|    |-- sensor_comm.h
|    |-- sensor_common.c
|    `-- sensor_common.h
|-- <src><sensor>
|    |-- <sensor>_config.c
|    |-- <sensor>_config.h
|    |-- <sensor>_driver.c
|    |-- <sensor>_driver.h
|    `-- <sensor>_regdef.h
|-- examples
|    |---|MCUXpresso
|    |   |    └── <project_name>
`-- docs
|    |-- Sensor_Drivers_API_Reference_Manual.zip
```

Where <sensor> is used as a common nomenclature of supported sensors, for example, fxos8700, fxls8471q, mma865x and fxls8962, etc.

The sensor driver implementations and interface definitions for a <sensor> (e.g. fxls8471, fxos8700, MMA865x and FXLS8962) are available under their respective sensor folder (for more details on sensor driver file descriptions, please refer to Section 3.2 ). The sensor driver interface definitions use common sensor communication interfaces provided as template functions under the common folder. End users should update the common sensor communication function template using the SDK implementation for underlying microcontroller peripherals such as $I^2C$, SPI. The sensor drivers have undergone limit testing for NXP microcontrollers FRDM-K64F, FRDM-K22F (Arm Cortex M4F core) and FRDM-KL25Z (Arm Cortex M0+ core) using MCUXpresso SDK implementation for underlying NXP microcontroller peripherals. The reference example project for testing sensor driver integration with MCUXpresso SDK is available under the “examples” folder.

## 3.2 Sensor driver component content overview

This section provides a brief overview of the sensor driver source file contents and file descriptions:

```
sensor_drivers/
|--<src> common     <Folder containing common sensor
 communication interfaces>
|    |-- sensor_comm.c       <Files containing common sensor
 communication interfaces template>
|    |-- sensor_comm.h
|    |-- sensor_common.c     <Files containing common sensor
 communication definitions>
|    `-- sensor_common.h
|
|-- <src>   <Folder containing driver implementation for
 <sensor>>
|    |-- <sensor>_config.c   <File containing register config
 definitions for <sensor>>
|    |-- <sensor>_config.h
|    |-- <sensor>_driver.c   <File containing driver interface
 definitions for <sensor>>
|    |-- <sensor>_driver.h
|    `-- <sensor>_regdef.h   <File containing register
 definitions & bit-map for <sensor>>
|-- example   <Folder containing <sensor> driver integration
 example with MCUXpresso SDK>
|    |    |-- MCUXpresso[1]
|    |    |    └── <project_name>

`-- doc   <Folder containing release documentation for sensor
 driver component>
|    |-- Sensor_Drivers_API_Reference_Manual.zip <Sensor Drivers
 Component API RM >
```

***Note:*** *The standalone MCUXpresso IDE supported example applications demonstrating integration of component libraries for underlying microcontroller MCUXpresso SDK requires corresponding microcontroller SDK package to be downloaded and installed on MCUXpresso IDE before importing the component library example project.*

## 3.3 Sensor driver component integration steps for any MCU and SDK

This section provides steps to be followed to start development/porting of <sensor> drivers on chosen microcontroller platform and integrating with corresponding SDK implementation for microcontroller peripherals.

1. Identify sensor use case and choose appropriate <sensor> driver component and source files from the component folder to start development on any host microcontroller unit (MCU) using corresponding MCU SDK and required cross-compilation tool chain. The source files provided for each component are platform agnostic and can be directly used in any MCU and SDK. There might be some tool chain specific adjustments that need to be done in the source files according to chosen tool chain.

[1] Components libraries are provided with NXP’s MCUXpresso SDK integration example application. The integration test example applications demonstrate how to integrate platform agnostic component libraries with underlying microcontroller SDK communication interfaces using virtual interface abstraction provided by component libraries.

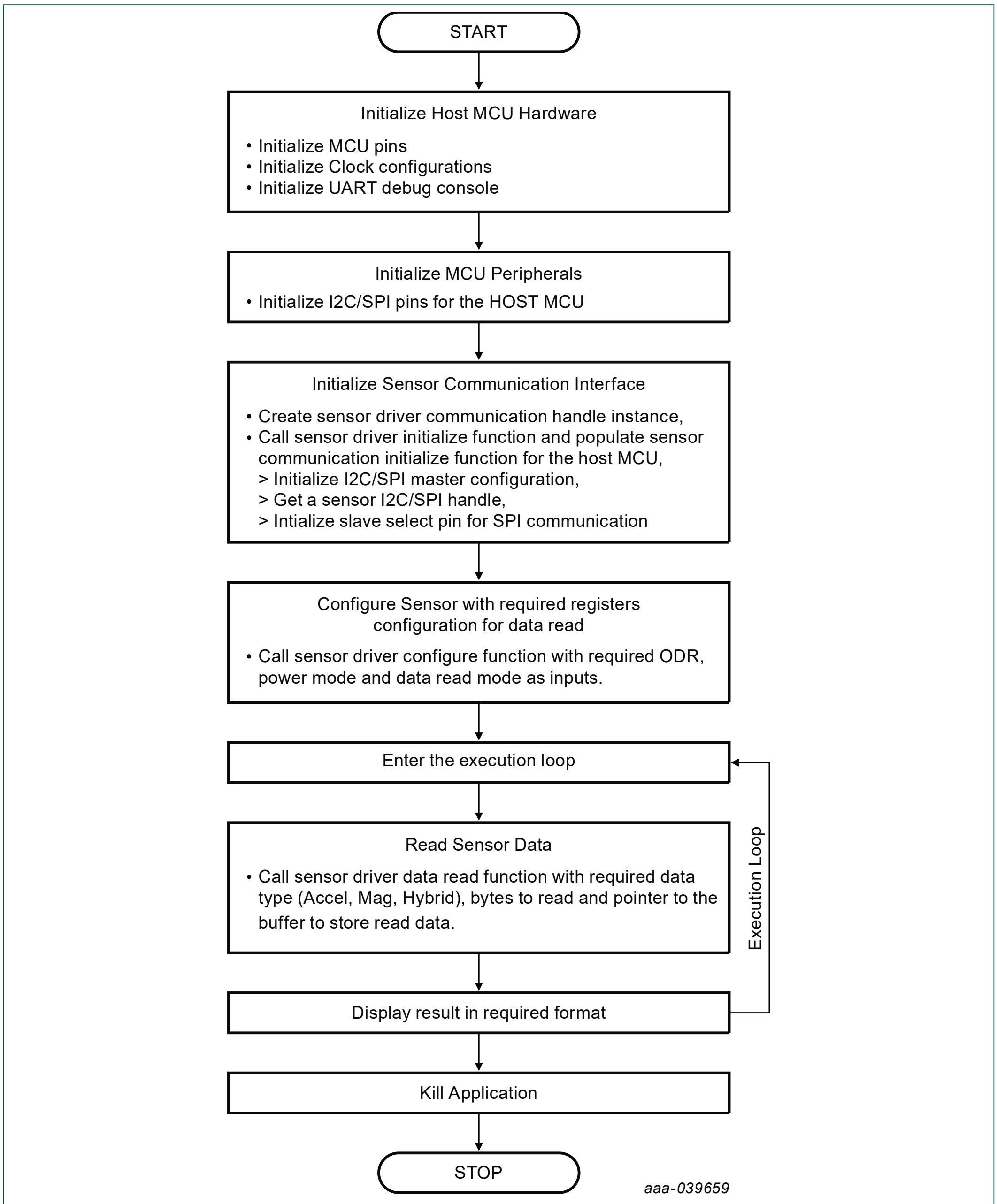
Note: The sensor driver component covers sensor drivers supporting register configurations needed for data read modes and embedded functions described in respective data sheet. In case the use case requires additional sensor configuration, then update <sensor>_config.c/.h.

2. Use the sensor driver component SDK abstraction functions defined in sensor_comm.c and .h to update SDK implementation for underlying host MCU peripherals like $I^2C$, SPI
3. Refer to the example application structure for the similar implementations
4. Cross-compile the project (resolve compilation & linking errors) and use tool chain debugger to load the generated binary to target MCU. Use UART debug console to verify output.

# 4 Creating a sensor data read example

This section provides a typical flow of creating a sensor data read example using sensor driver models for any host MCU.

Figure 2 describes typical code flow for a sensor data read (in polling mode) example that can be created using sensor driver models.



START
Initialize Host MCU Hardware
• Initialize MCU pins
• Initialize Clock configurations
• Initialize UART debug console
Initialize MCU Peripherals
• Initialize I2C/SPI pins for the HOST MCU
Initialize Sensor Communication Interface
• Create sensor driver communication handle instance,
• Call sensor driver initialize function and populate sensor communication initialize function for the host MCU,
> Initialize I2C/SPI master configuration,
> Get a sensor I2C/SPI handle,
> Intialize slave select pin for SPI communication
Configure Sensor with required registers configuration for data read
• Call sensor driver configure function with required ODR, power mode and data read mode as inputs.
Enter the execution loop
Read Sensor Data
• Call sensor driver data read function with required data type (Accel, Mag, Hybrid), bytes to read and pointer to the buffer to store read data.
Display result in required format
Execution Loop
Kill Application
STOP
aaa-039659

**Figure 2. Typical code flow for a sensor data read (polling mode) example**

Figure 3 describes typical code flow for a sensor data read (in interrupt mode) example that can be created using sensor driver models.

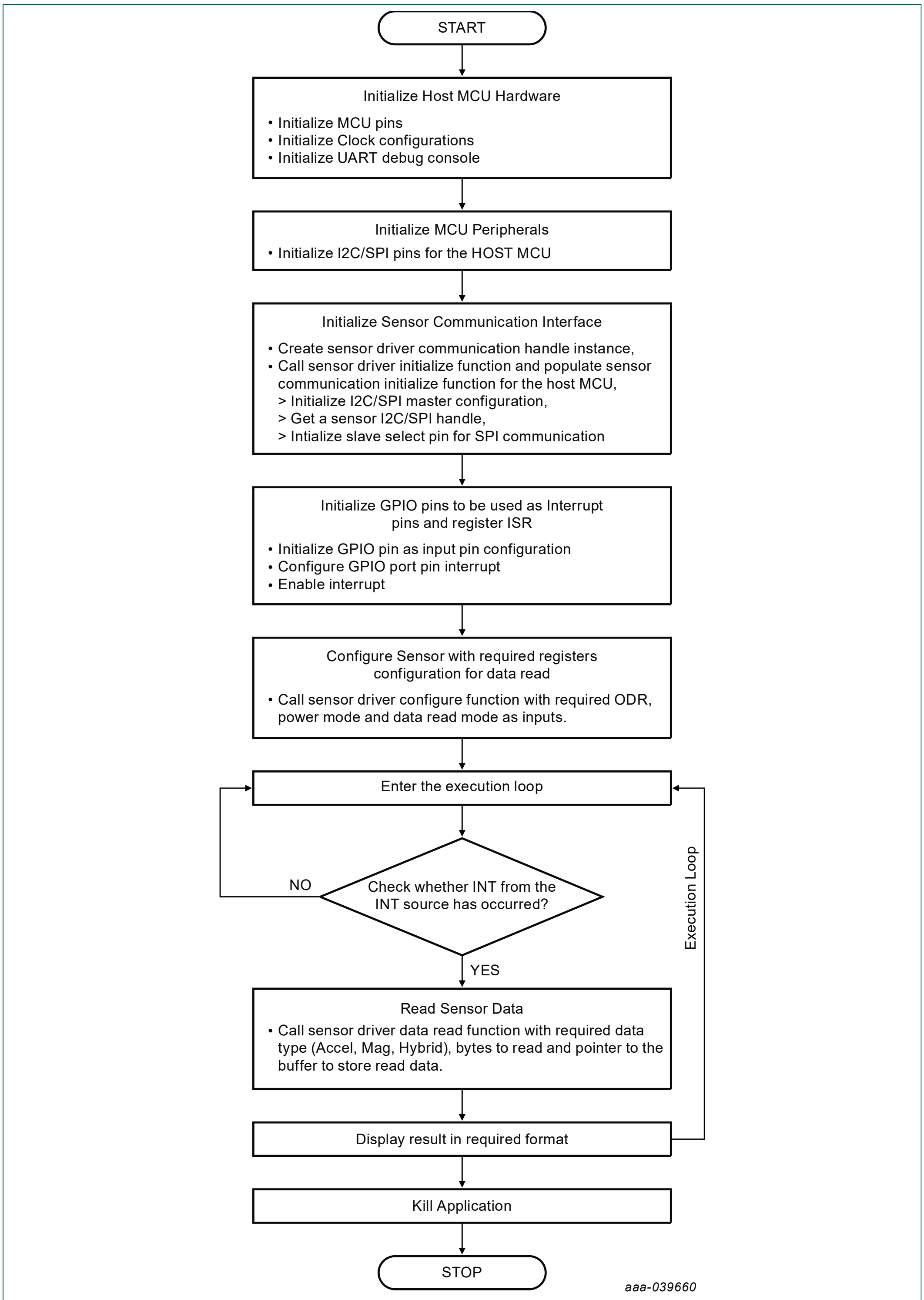

START
Initialize Host MCU Hardware
• Initialize MCU pins
• Initialize Clock configurations
• Initialize UART debug console
Initialize MCU Peripherals
• Initialize I2C/SPI pins for the HOST MCU
Initialize Sensor Communication Interface
• Create sensor driver communication handle instance,
• Call sensor driver initialize function and populate sensor communication initialize function for the host MCU,
> Initialize I2C/SPI master configuration,
> Get a sensor I2C/SPI handle,
> Intialize slave select pin for SPI communication
Initialize GPIO pins to be used as Interrupt pins and register ISR
• Initialize GPIO pin as input pin configuration
• Configure GPIO port pin interrupt
• Enable interrupt
Configure Sensor with required registers configuration for data read
• Call sensor driver configure function with required ODR, power mode and data read mode as inputs.
Enter the execution loop
NO
Check whether INT from the INT source has occurred?
YES
Execution Loop
Read Sensor Data
• Call sensor driver data read function with required data type (Accel, Mag, Hybrid), bytes to read and pointer to the buffer to store read data.
Display result in required format
Kill Application
STOP
aaa-039660

**Figure 3. Typical code flow for a sensor data read (interrupt mode) example**

## 5 Creating a sensor embedded function event read example

This section provides a typical flow of creating a sensor event read example using sensor driver models supporting embedded functions for any host MCU.

Figure 4 describes typical code flow for a sensor event read example that can be created using sensor driver models.

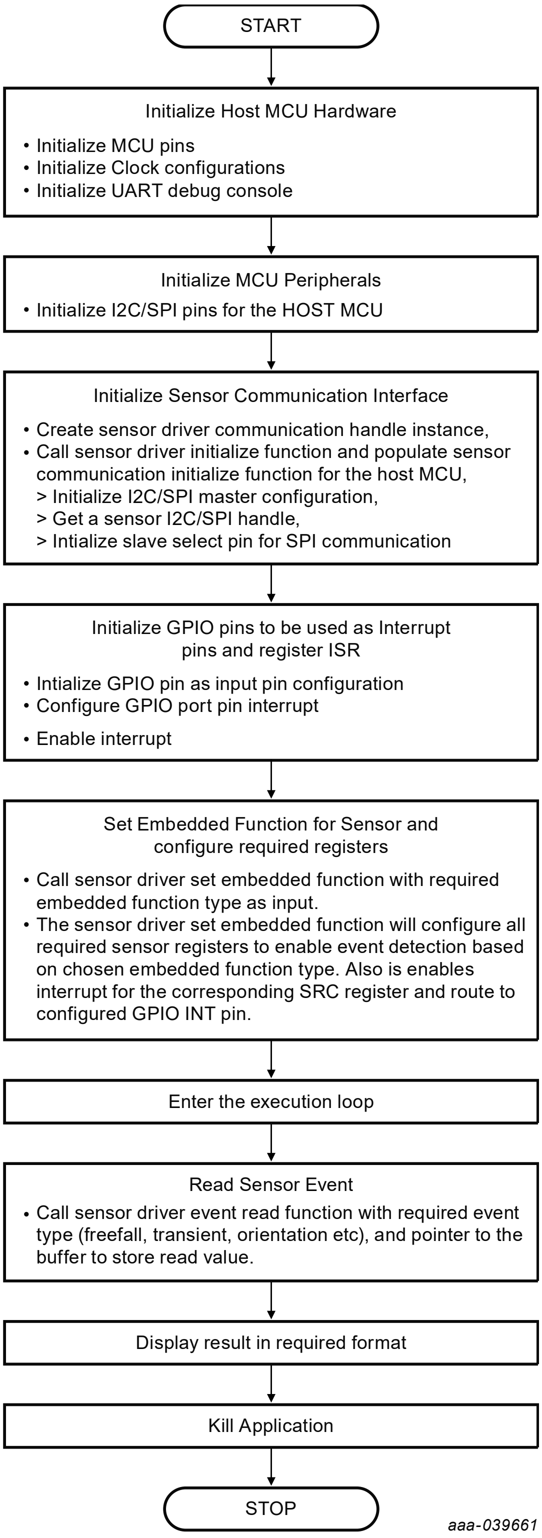

START
Initialize Host MCU Hardware
• Initialize MCU pins
• Initialize Clock configurations
• Initialize UART debug console
Initialize MCU Peripherals
• Initialize I2C/SPI pins for the HOST MCU
Initialize Sensor Communication Interface
• Create sensor driver communication handle instance,
• Call sensor driver initialize function and populate sensor communication initialize function for the host MCU,
> Initialize I2C/SPI master configuration,
> Get a sensor I2C/SPI handle,
> Intialize slave select pin for SPI communication
Initialize GPIO pins to be used as Interrupt pins and register ISR
• Intialize GPIO pin as input pin configuration
• Configure GPIO port pin interrupt
• Enable interrupt
Set Embedded Function for Sensor and configure required registers
• Call sensor driver set embedded function with required embedded function type as input.
• The sensor driver set embedded function will configure all required sensor registers to enable event detection based on chosen embedded function type. Also is enables interrupt for the corresponding SRC register and route to configured GPIO INT pin.
Enter the execution loop
Read Sensor Event
• Call sensor driver event read function with required event type (freefall, transient, orientation etc), and pointer to the buffer to store read value.
Display result in required format
Kill Application
STOP
aaa-039661

**Figure 4. Typical code flow for a sensor embedded function event read example**

# 6 Legal information

## 6.1 Definitions

**Draft** — A draft status on a document indicates that the content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included in a draft version of a document and shall have no liability for the consequences of use of such information.

## 6.2 Disclaimers

**Limited warranty and liability** — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors. In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory. Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

**Right to make changes** — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

**Suitability for use** — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

**Applications** — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification. Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products. NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

**Export control** — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

**Translations** — A non-English (translated) version of a document is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

**Security** — While NXP Semiconductors has implemented advanced security features, all products may be subject to unidentified vulnerabilities. Customers are responsible for the design and operation of their applications and products to reduce the effect of these vulnerabilities on customer's applications and products, and NXP Semiconductors accepts no liability for any vulnerability that is discovered. Customers should implement appropriate design and operating safeguards to minimize the risks associated with their applications and products.

## 6.3 Trademarks

Notice: All referenced brands, product names, service names and trademarks are the property of their respective owners.

**NXP** — wordmark and logo are trademarks of NXP B.V.

# Figures

All information provided in this document is subject to legal disclaimers.

## Contents

Please be aware that important notices concerning this document and the product(s) described herein, have been included in section 'Legal information'.

For more information, please visit: http://www.nxp.com
For sales office addresses, please send an email to: salesaddresses@nxp.com

**Date of release: 2 November 2020**
**Document identifier: UM11516**