AN14822

How to Use SmartDMA to Implement SPI Interface on MCXA MCU

Rev. 1.0 — 30 September 2025 Application note

Document information

Information	Content
Keywords	AN14822, SmartDMA, MCXA346, MCXA34, SPI
Abstract	This application note describes how to use SmartDMA to implement the SPI on MCXA series MCUs.

1 Introduction

This application note describes how to use SmartDMA to implement the serial peripheral interface (SPI) on MCXA series MCUs. This document introduces the SPI interface, details the SmartDMA implementation, and demonstrates its use with the ICM42688 6-axis IMU sensor. The MCXA MCU features a coprocessor called SmartDMA, which enables a flexible SPI interface with precise timing control and low CPU overhead.

2 SPI interface

SPI is a synchronous serial communication interface used for short-distance communication, primarily in embedded systems.

SPI devices communicate in full-duplex mode using a controller-target architecture with a single controller. The controller device originates the frame for both reading and writing. It supports multiple target devices, each selected through individual target select (SS) lines.

2.1 SPI signal lines

SPI enables high-speed, full-duplex communication between a controller and target devices using the four key signal lines as shown in Table 1:

Table 1. SPI signals

Signal	Direction	Description
MISO	Target > Controller	Data line from target to controller
MOSI	Controller > Target	Data line from controller to target
SCLK	Controller > Target	Clock signal generated by the controller
SS/CS	Controller > Target	Target selection signal (active low)

2.2 SPI modes

SPI defines four distinct communication modes, as shown in Table 2, by combining the following two configurable parameters:

- CPOL
- CPHA

Table 2. SPI modes

Mode	CPOL	CPHA	Clock Idle State	Data Sampling Edge
0	0	0	Low	Rising edge
1	0	1	Low	Falling edge
2	1	0	High	Falling edge
3	1	1	High	Rising edge

3 SmartDMA SPI implementation

SmartDMA is a coprocessor unit within the MCX MCU that can execute a reduced instruction set. It provides precise timing control and can access the GPIO in a single cycle, making it ideal for implementing timing-critical protocols like SPI.

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

© 2025 NXP B.V. All rights reserved.

Following are the performance aspects of SmartDMA for SPI on MCXA MCUs:

- Uses the system clock of the MCU as its clock source.
- Supports configurable SPI clock frequencies up to 25 MHz under a 150 MHz system clock.
- Controls general-purpose input/outputs (GPIOs) at the hardware level, enabling submicrosecond response times.
- Supports various SPI modes, including all clock polarity (CPOL) or clock phase (CPHA) combinations.
- Transfers data efficiently with minimal CPU intervention.

Following are the key characteristics of the SmartDMA SPI implementation:

- Timing synchronization: Both functions maintain identical clock timing patterns to ensure proper controller-target synchronization.
- Data direction control: The transmit function manages MOSI output signals, while the receive function monitors MISO input signals.
- MSB-first protocol: Each byte is transmitted and received starting with the most significant bit, following the standard SPI convention.
- Chip select framing: The CS signal frames each complete byte transaction.
- Timing compliance: Strategic delay insertions ensure compliance with SPI timing specifications for setup and hold requirements.

3.1 SmartDMA SPI timing implementation

The SmartDMA SPI timing implementation follows a methodology similar to MCU-based Arm core approaches, while applying SmartDMA capabilities to enhance performance. The SPI communication process using SmartDMA handles both transmitting and receiving data. This process follows a structured sequence of steps as detailed below to ensure precise timing and efficient data handling.

1. The communication begins by asserting the chip select with CS = 0 for sending, or by initializing the variables received_data = 0 and CS = 0 for receiving.
2. The loop is prepared by setting bit_count = 7 so that transmission and reception start from the most significant bit (MSB first).
3. The data lines are set up, where MOSI is loaded with the value (data >> bit_count) and 1 for sending, while in receiving MOSI is placed in its idle state with MOSI = 1.
4. A setup time is allowed before the clock transition by applying a short delay using delay_us().
5. The rising edge of the clock is triggered by raising SCLK to 1, which causes the target to sample data in sending, or output data in receiving.
6. The clock is then maintained high for the required duration, again using delay_us().
7. During this period, data sampling occurs: in sending, MISO can optionally be read for full duplex operation, and in receiving, MISO is read and stored in the variable received_data |= (MISO << bit_count).
8. The clock is lowered to complete the cycle by setting SCLK back to 0.
9. The bit counter is decremented with bit_count-- to move to the next bit.
10. If bit_count >= 0, the procedure repeats from step 3 to continue processing the remaining bits.
11. When all bits have been processed, communication is finished by deasserting the chip select with CS = 1, which releases the chip and ends the operation.

3.2 Timing pseudocode

The following pseudocodes illustrate the SPI communication protocol with precise timing sequences for both transmit and receive operations.

- Transmit timing pseudocode
- Receive timing pseudocode

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

© 2025 NXP B.V. All rights reserved.

These pseudocodes provide a comprehensive foundation for implementing reliable SPI communication with precise timing control in embedded systems.

3.2.1 Transmit timing pseudocode

The following pseudocode illustrates the bit-by-bit transmission sequence for sending data over the SPI interface:

```
Function: SmartDMA_SPI_SendByte(data)
Input: data - 8-bit data to transmit
Output: void (or optionally return received data for full-duplex operation)
BEGIN
1. CS = 0 // Assert chip select to initiate communication
2. FOR bit_count = 7 DOWN TO 0 DO // Iterate through 8 bits, MSB transmitted
 first
3. IF (data & (1 << bit_count)) THEN
MOSI = 1 // Set MOSI high for logical '1'
ELSE
MOSI = 0 // Set MOSI low for logical '0'
END IF
4. delay_us() // Setup time delay to meet timing requirements
5. SCLK = 1 // Generate clock rising edge, slave samples data
6. delay_us() // Clock high hold time for stable sampling
7. SCLK = 0 // Generate clock falling edge
8. delay_us() // Clock low hold time (optional recovery period)
END FOR
9. CS = 1 // Deassert chip select to terminate communication
END
```

3.2.2 Receive timing pseudocode

The following pseudocode demonstrates the bit-by-bit reception sequence for reading data from the SPI interface:

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

© 2025 NXP B.V. All rights reserved.

```
Function: SmartDMA_SPI_ReceiveByte()
Input: void
Output: received_data - 8-bit received data
BEGIN
1. received_data = 0 // Initialize receive data buffer
2. CS = 0 // Assert chip select to initiate communication
3. MOSI = 1 // Set MOSI to idle high state (optional)
4. FOR bit_count = 7 DOWN TO 0 DO // Iterate through 8 bits, MSB received first
5. delay_us() // Setup time delay before clock transition
6. SCLK = 1 // Generate clock rising edge, slave outputs data
7. delay_us() // Clock high hold time for stable data output
8. IF (MISO == 1) THEN
received_data |= (1 << bit_count) // Capture and store received bit
END IF
9. SCLK = 0 // Generate clock falling edge
10. delay_us() // Clock low hold time (optional recovery period)
END FOR
11. CS = 1 // Deassert chip select to terminate communication
12. RETURN received_data // Return the complete received byte
END
```

3.3 SmartDMA SPI performance advantages

This section describes the performance of SmartDMA-based SPI communication, in comparison to traditional GPIO-based SPI methods. While maintaining the same logical sequence as traditional GPIO-based SPI, SmartDMA provides:

- Hardware-accelerated timing: Delivers timing control without requiring CPU intervention.
- DMA buffer management: Automatically handles multibyte transfers.
- Reduced CPU overhead: Offloads SPI operation to background processing capabilities.
- Enhanced throughput: Achieves higher data transfer rates compared to bit-banging methods.

Practical testing demonstrates that SmartDMA delivers 25 MHz SPI clock rates at 180 MHz core frequency, with optimization potential for higher speeds.

4 Demo code

This demo code implements a SmartDMA-driven SPI interface for the ICM42688 sensor. It features firmware-based initialization, parameter configuration, and half-duplex read/write operations. It also enables complete sensor control functions for accelerometer and gyroscope data acquisition with hardware-accelerated communication. This demo code also demonstrates the practical application of the transmit and receive pseudocode from Section 3.2. It also provides a working example of how SmartDMA enhances SPI communication in real sensor use cases.

4.1 SmartDMA Initialization

SmartDMA initialization is straightforward, requiring clock enablement and reset state release like other peripherals. The application encapsulates SmartDMA instructions into an array. It assigns the array address to the Boot register, allowing SmartDMA to boot and execute code from the array upon startup. As the application functions as an SPI controller, relevant parameter data must be provided before execution.

The following is the initialization code snippet:

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

© 2025 NXP B.V. All rights reserved.

```
// Initialize SmartDMA without firmware
SMARTDMA_InitWithoutFirmware();
// Install firmware from array
SMARTDMA_InstallFirmware(SMARTDMA_SPI_MEM_ADDR, s_smartdmaSPIFirmware,
SMARTDMA_SPI_FIRMWARE_SIZE);
// Install callback and configure interrupt
SMARTDMA_InstallCallback(SmartDMA_spi_callback, NULL);
NVIC_EnableIRQ(SMARTDMA_IRQn);
NVIC_SetPriority(SMARTDMA_IRQn, 3);
// Configure SmartDMA parameters
smartdmaParam.smartdma_stack = (uint32_t*)g_samrtdma_stack;
smartdmaParam.p_reg_addr = (uint32_t*)&g_register_address;
smartdmaParam.p_write_buf = (uint32_t*)g_write_buffer;
smartdmaParam.p_read_buf = (uint32_t*)g_read_buffer;
smartdmaParam.p_data_length = (uint32_t*)&g_datalength;
smartdmaParam.p_frq_div = (uint32_t*)&g_fre_divider;
// Boot SmartDMA with firmware
g_spi_complete_flag = 1;
SMARTDMA_Boot(kSMARTDMA_spi, &smartdmaParam, 0x2);
```

The `SMARTDMA_InitWithoutFirmware` function enables the clock of the SmartDMA and releases reset. `SMARTDMA_InstallFirmware` assigns array values to SRAMX, as SmartDMA code must execute in SRAMX for efficiency. The `SMARTDMA_InstallCallback` installs callback functions when a frame of SPI data ends, SmartDMA triggers an Arm interrupt. SmartDMA implements SPI controller functionality through the firmware array `s_smartdmaSPIFirmware`, which is defined in the `fsl_smartdma_mcxa.c` file.

4.2 SmartDMA parameter configuration

The smartdma_param variable defines the required parameters for configuring SmartDMA.

Table 3. SmartDMA function parameter

Parameter	Type	Description
smartdma_stack	uint32_t*	SmartDMA stack pointer: Points to the stack memory area allocated for SmartDMA.
p_debugger	uint32_t*	GPIO register buffer pointer: Provides access to GPIO registers for debugging purposes.
p_reg_addr	uint32_t*	Register address pointer: Points to the SPI target device register address.
p_write_buf	uint32_t*	Write buffer pointer: Points to the data buffer to be sent to the SPI target. device
p_read_buf	uint32_t*	Read buffer pointer: Points to the buffer for storing data received from SPI target device.
p_data_length	uint32_t*	Data length pointer: Points to the variable defining SPI transmission data length.
p_op_cmd	uint32_t*	Operation command pointer: Points to SPI operation command configuration.
p_frq_div	uint32_t*	Frequency divider pointer: Points to the variable that controls the SPI clock frequency division ratio.

4.3 SPI operation functions

This application primarily implements SPI read and write operations at the underlying level. Currently, the read and write operations are half-duplex, with separate read and write functions.

4.3.1 SPI read function brief summary

This function reads data from a specified register address on an SPI target device using SmartDMA.

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

© 2025 NXP B.V. All rights reserved.

The following are the operational characteristics of the SPI read function:

- Parameter validation: Checks the validity of the data buffer and data length.
- Blocking operation: Waits for the previous operation to complete before executing a new one.
- SmartDMA-driven: Uses SmartDMA to perform actual SPI communication.
- Data transfer: Copies received data from the global buffer to the user buffer after completion.
- Status return: Returns the status of the operation, indicating success or error.

Execution flow

Following the above characteristics, the SPI read function operates through the following sequence:

1. Checks the parameters.
2. Waits for any ongoing operation to complete.
3. Sets the required parameters.
4. Triggers SmartDMA to start the SPI transfer.
5. Copies the received data into the user buffer.
6. Copies the received data into the user buffer.
7. Returns the status.

4.3.2 SPI write function brief summary

This function writes data from a specified register address on an SPI target device using SmartDMA.

The following are the operational characteristics of the SPI write function:

- Parameter validation: Checks the validity of the data buffer and data length.
- Nonblocking operation: Returns immediately after triggering SmartDMA, unlike the read function.
- SmartDMA-driven: Uses SmartDMA to perform actual SPI communication.
- Data preparation: Copies user data to a global write buffer before transmission.
- Status return: Returns the status of the operation, indicating success or error status.

Execution Flow

Following the above characteristics, the SPI write function operates through the following sequence:

1. Check the parameters.
2. Wait for any previous operation to complete.
3. Copy user data into the global write buffer.
4. Set the required parameters.
5. Trigger SmartDMA to start the SPI transfer.
6. Return the status.

4.4 ICM42688 sensor operation functions

Table 4 provides register-level access and configuration support for the ICM42688 sensor, enabling initialization, data readout, and parameter setup.

Table 4. Sensor operation routines

Function	Description
ICM42688_WriteReg	Clears MSB for write operation and sends 1 byte to the specified register.
ICM42688_ReadReg	Sets MSB for read operation and reads 1 byte from a specified register.
ICM42688_ReadMultipleReg	Sets MSB for read operation and reads multiple bytes starting from a specified register.

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

© 2025 NXP B.V. All rights reserved.

Table 4. Sensor operation routines*...continued*

Function	Description
ICM42688_Init	Reset the device, check the WHO_AM_I register, configure power settings, and set sensor parameters to complete initialization.
ICM42688_Reset	Performs a soft reset and waits for its completion.
ICM42688_SetAccelConfig	Set the full-scale range and output data rate for the accelerometer.
ICM42688_SetGyroConfig	Set the full-scale range and output data rate for the gyroscope.
ICM42688_ReadAccelData	Read 6 bytes and convert them into 16-bit X, Y, and Z accelerometer values.
ICM42688_ReadGyroData	Read 6 bytes and convert them into 16-bit X, Y, and Z gyroscope values.

5 Demo application

This demo application demonstrates how to interface an ICM42688 six-axis sensor with an MCU using SmartDMA to simulate SPI controller communication. The system initializes the sensor with a 2 MHz clock frequency and an 8-bit data configuration. It reads accelerometer and gyroscope data, then converts the raw values into actual measurements using sensitivity calculations. Finally, it outputs real-time acceleration and angular velocity data through UART for verification and debugging purposes.

5.1 Hardware setup

The demo application uses the following pin configuration: FRDM-MCXA346 board connects to the ICM42688 sensor through the SPI interface with SmartDMA simulating SPI controller communication.

Table 5. Hardware connection

FRDM-MCXA346 header	Header signals	SmartDMA	Sensor module	Function
J2-6	P3_11	SmartDMA_PIO11	CS	Chip select
J2-8	P3_8	SmartDMA_PIO8	MOSI	Data line from controller to target
J2-10	P3_9	SmartDMA_PIO9	MISO	Data line from target to controller
J2-12	P3_10	SmartDMA_PIO10	CLK	Clock
J2-14	GND	-	GND	Ground
J2-16	VCC	-	VCC	Power supply

This project requires an ICM42688 sensor module. If the specific model is unavailable, you can choose a functionally similar six-axis sensor module as an alternative. If the pin configuration of the new module does not match the FRDM-MCXA346 development board, use jumper wires for connection. Make sure to follow the SPI requirements—CS, MOSI, MISO, CLK, and power lines—to ensure correct pin-to-pin correspondence.

To order the FRDM-MCXA346 development board and obtain relevant specifications and pricing information, visit the NXP official website.

The image below shows the physical setup of the development board and sensor module connected to a computer through USB.

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

© 2025 NXP B.V. All rights reserved.

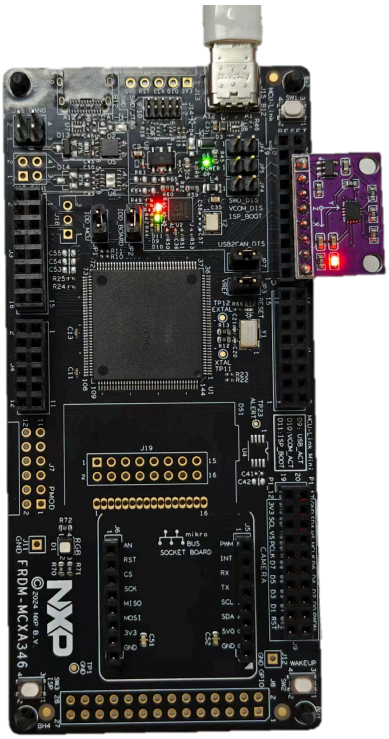

Figure 1. Demo hardware

5.2 Software setup and demo run

To run the demo successfully, follow the steps below:

1. Open MCUXpresso IDE.
2. Import a project from the GitHub repository.
3. Locate and open the `an-mcxa346-spi-interface-implemented-by-smartdm` project.
4. Compile the project and download it to the FRDM-MCXA346 development board.
5. Open a serial port assistant and set the baud rate to 230400.
6. To start the program, click reset. The system first calibrates and initializes the sensor, then print six-axis sensor values every second.
7. To observe changes in acceleration and angular velocity values, rotate the development board.

Below are some of the serial port output results.

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

© 2025 NXP B.V. All rights reserved.

```
=== SmartDMA SPI ICM42688 Example ===
Initializing ICM42688...
ICM42688 initialized successfully!
Please keep the sensor stationary and horizontal for calibration...
Auto-calibrating accelerometer... Keep sensor stationary!
Raw Z average: 2028.7 LSB (from 100 samples)
Calculated scale factor: 2028.7 LSB/g
=== Scale Factor Verification ===
Gyro range: ±2000 dps, Scale factor: 16.4 LSB/dps
Accel range: ±4g, Scale factor: 2048.0 LSB/g
Expected: Gyro ~0 dps when stationary, Accel Z ~±1g when horizontal
================================
Starting sensor data reading (1Hz)...
[0001] Accel(g): X= 0.012 Y=-0.041 Z= 1.000 | Gyro(dps): X= -0.18 Y= -0.12 Z= 0
[0002] Accel(g): X= 0.011 Y=-0.040 Z= 1.000 | Gyro(dps): X= -0.24 Y= -0.12 Z= 0
[0003] Accel(g): X= 0.013 Y=-0.041 Z= 1.000 | Gyro(dps): X= -0.24 Y= -0.06 Z= 0
```

6 Acronyms

Table 6 lists the acronyms used in this document along with their description.

Table 6. Acronyms

Acronym	Description
SPI	Serial peripheral interface
SmartDMA	Smart coprocessor
GPIO	General-purpose input/output
CPOL	Clock polarity
CPHA	Clock phase
SCLK	Serial clock
MOSI	Controller out target in
MISO	Controller in target out
SS/CS	Target select/chip select

7 Note about the source code in the document

Example code shown in this document has the following copyright and BSD-3-Clause license:

Copyright 2025 NXP Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials must be provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT

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

© 2025 NXP B.V. All rights reserved.

SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

8 Revision history

Table 7 summarizes revisions to this document.

Table 7. Revision history

Document ID	Release date	Description
AN14822 v1.0	30 September 2025	Initial public release

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

© 2025 NXP B.V. All rights reserved.

Legal information

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.

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.

Terms and conditions of commercial sale — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at https://www.nxp.com/profile/terms, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

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.

Suitability for use in non-automotive qualified products — Unless this document expressly states that this specific NXP Semiconductors product is automotive qualified, the product is not suitable for automotive use. It is neither qualified nor tested in accordance with automotive testing or application requirements. NXP Semiconductors accepts no liability for inclusion and/or use of non-automotive qualified products in automotive equipment or applications.

In the event that customer uses the product for design-in and use in automotive applications to automotive specifications and standards, customer (a) shall use the product without NXP Semiconductors' warranty of the product for such automotive applications, use and specifications, and (b) whenever customer uses the product for automotive applications beyond NXP Semiconductors' specifications such use shall be solely at customer's own risk, and (c) customer fully indemnifies NXP Semiconductors for any liability, damages or failed product claims resulting from customer design and use of the product for automotive applications beyond NXP Semiconductors' standard warranty and NXP Semiconductors' product specifications.

HTML publications — An HTML version, if available, of this document is provided as a courtesy. Definitive information is contained in the applicable document in PDF format. If there is a discrepancy between the HTML document and the PDF document, the PDF document has priority.

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

Security — Customer understands that all NXP products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately.

Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP.

NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

NXP B.V. — NXP B.V. is not an operating company and it does not distribute or sell products.

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.

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

© 2025 NXP B.V. All rights reserved.

Contents

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

© 2025 NXP B.V. **All rights reserved.**