AN14562

LPC553x/LPC55S3x FLASH APIs Implementation

Rev. 1.0 — 28 February 2025 Application note

Document information

Information	Content
Keywords	AN14562, LPC553x/LPC55S3x, API
Abstract	The ROM API in LPC553x/LPC55S3x devices helps manage and program the programmable Flash region and the Flash firewall region.

1 Introduction

The ROM API in LPC553x/LPC55S3x devices helps manage and program two main areas:

- The ***programmable Flash region***, which stores the application code and data.
- The ***Flash firewall region***, which holds device configurations and settings.

It enables Serial NOR FLASH programming through the FlexSPI NOR API and supports reading and programming One-Time Programmable (OTP) settings. The ROM API also includes built-in cryptographic functions for secure applications.

The structure of the API includes several components:

- The FLASH API is used to update the FLASH area.
- The FLEXSPI API supports various Serial NOR devices.
- The OTP API manages critical one-time programmable parameters.
- The NBOOT APIs help generate random numbers and verify the integrity of application images.
- The In-Application Programming (IAP) API provides flexibility for programming while an application is running, using either a unified memory interface or a dedicated secondary bootloader.

2 ROM API structure

The ROM API table locates at address `0x1302fc00`.

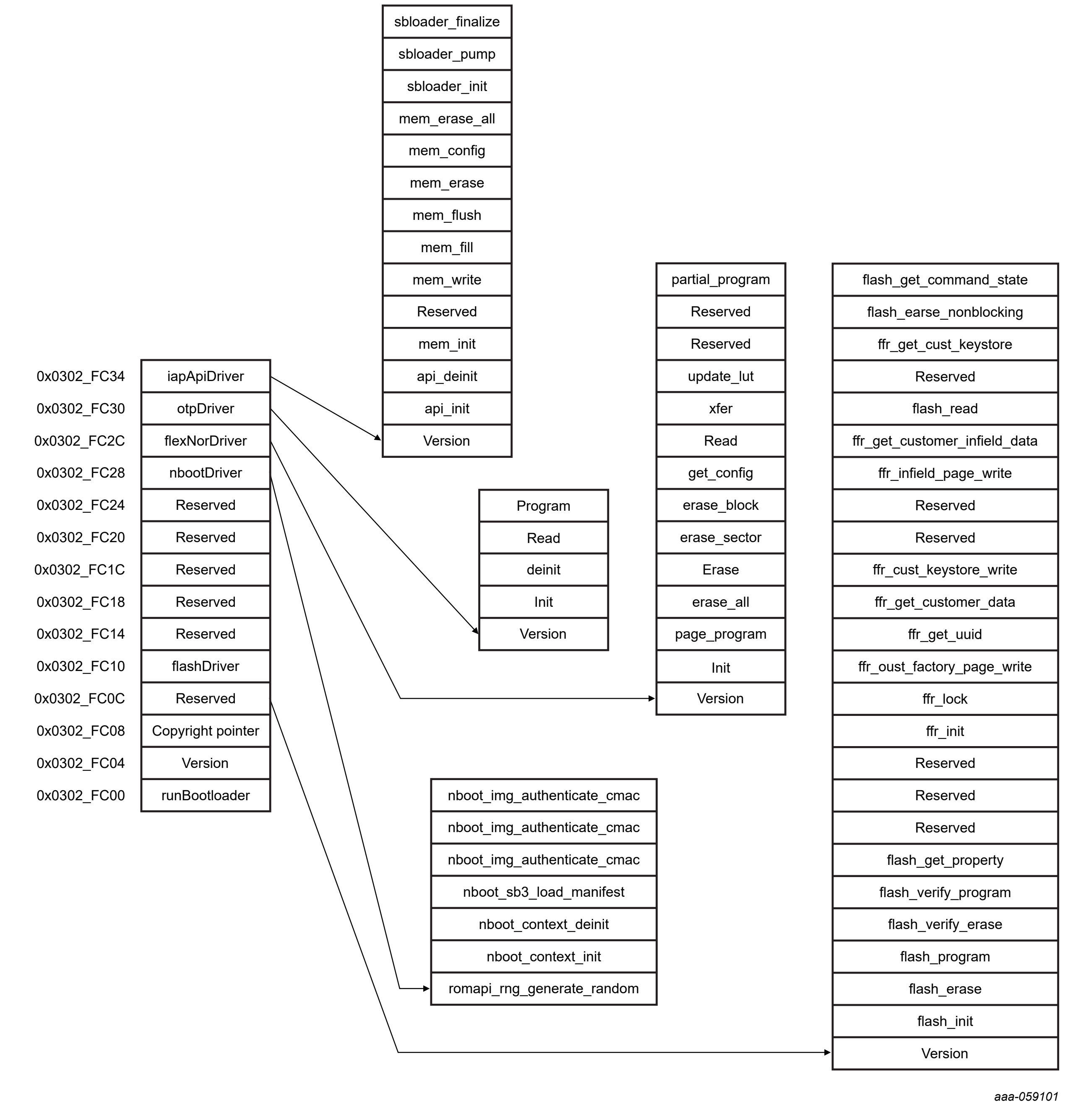

Figure 1. ROM API structure

3 FLASH APIs

The FLASH API set enables the following features:

- Initialize the FLASH controller.
- Erase and verify the specified FLASH area.
- Program and verify the specified FLASH page.
- Retrieve FLASH properties.
- Initialize or Lock the FFR.
- Program and Read CMPA.
- Program and Read CFPA.
- Non-blocking FLASH Erase/Status Check API for timing-critical use cases.

3.1 FLASH driver API interface

The FLASH APIs are organized in the FLASH Driver API Interface structure.

3.1.1 FLASH driver API prototypes

```
typedef struct FLASHDriverInterface
{
standard_version_t version; //!< flash driver API version number.
// FLASH driver
status_t (*flash_init)(flash_config_t *config);
status_t (*flash_erase)(flash_config_t *config, uint32_t start, uint32_t lengthInBytes,
 uint32_t key);
status_t (*flash_program)(flash_config_t *config, uint32_t start, uint8_t *src,uint32_t
 lengthInBytes);
status_t (*flash_verify_erase)(flash_config_t *config, uint32_t start, uint32_t
 lengthInBytes);
status_t (*flash_verify_program)(flash_config_t *config,
uint32_t start,
uint32_t lengthInBytes,
const uint8_t   *expectedData,
uint32_t *failedAddress,
uint32_t *failedData);
status_t (*flash_get_property)(flash_config_t *config, flash_property_tag_t
 whichProperty,uint32_t *value);
uint32_t reserved0[3];
// FLASH FFR driver
status_t (*ffr_init)(flash_config_t *config);
status_t (*ffr_lock)(flash_config_t *config);
status_t (*ffr_cust_factory_page_write)(flash_config_t *config, uint8_t *page_data, bool
 seal_part);
status_t (*ffr_get_uuid)(flash_config_t *config, uint8_t *uuid);
status_t (*ffr_get_customer_data)(flash_config_t *config, uint8_t *pData, uint32_t
 offset,uint32_t len);
status_t (*ffr_cust_keystore_write)(flash_config_t *config, ffr_key_store_t *pKeyStore);
status_t reserved1;
status_t reserved2;
status_t (*ffr_infield_page_write)(flash_config_t *config, uint8_t *page_data, uint32_t
 valid_len);
status_t (*ffr_get_customer_infield_data)(flash_config_t *config, uint8_t *pData,
 uint32_t offset,uint32_t len);
status_t (*flash_read)(flash_config_t *config, uint32_t start, uint8_t *dest, uint32_t
 lengthInBytes);
status_t reserved3;status_t (*flash_get_cust_keystore)(flash_config_t *config, uint8_t
 *pData, uint32_t offset,uint32_t len);
status_t (*flash_erase_non_blocking)(flash_config_t *config, uint32_t start, uint32_t
 lengthInBytes,uint32_t key);
status_t (*flash_get_command_state)(flash_config_t *config);
} flash_driver_interface_t;
```

3.1.2 FLASH configuration structure

Each FLASH API depends on a common FLASH context structure named `flash_config_t` to perform the proper FLASH operation.

```
/*! @brief FLASH driver state information.
*
* An instance of this structure is allocated by the user of the flash driver and
* passed into each of the driver APIs.
*/
typedef struct
{
uint32_t PFlashBlockBase; /*!< A base address of the first PFlash block */
uint32_t PFlashTotalSize; /*!< The size of the combined PFlash block. */
uint32_t PFlashBlockCount; /*!< A number of PFlash blocks. */
uint32_t PFlashPageSize; /*!< The size in bytes of a page of PFlash. */
uint32_t PFlashSectorSize; /*!< The size in bytes of a sector of PFlash. */
flash_ffr_config_t ffrConfig;
flash_mode_config_t modeConfig;
uint32_t *nbootCtx;
} flash_config_t;
```

4 Implementing FLASH APIs in MCUXpresso

First, define the following macros to facilitate the access to ROM and flash memory APIs in a firmware context.

```
#define ROM_API_TREE ((uint32_t *)0x1302FC00U)
#define DO_DEINIT 0
#define FLASH_API_TREE ((flash_driver_interface_t*)ROM_API_TREE[4])
```

4.1 `version`

`version` field in the FLASH API table indicates the current FLASH API version in the ROM bootloader.

Prototype:

```
standard_version_t version;
```

Parameter	Description
`version`	Pointer to version structure to store current Flash driver version information

Implementation:

```
/*Flash version api call*/
uint32_t FlashDriverVersion = FLASH_API_TREE-> version.version;
```

Output: It gives the version of the Flash driver.

4.2 ffr_lock

This API is used for initializing the FFR controller and the `flash_ffr_config` context. It must be called before calling other FFR APIs.

Flash FFR initialization must be done by invoking the `ffr_init` API of the MCUXpresso SDK before calling the `ffr_lock`.

Prototype:

```
status_t (*ffr_lock)(flash_config_t *config);
```

Parameter	Description
config	Pointer to `flash_config_t` data structure in memory to store driver runtime state.

Implementation:

```
/*FRR init*/
status_t (*ffr_init)(flash_config_t *config);
uint32_t status_t = FLASH_API_TREE-> ffr_init (&flashConfig);
```

Output: If the status is `kStatus_FLASH_Success` return value, it means that the FFR region has been locked.

4.3 ffr_cust_factory_page_write

The API is used for writing the CMPA data into the CMPA region, and the API should be called after the `flash_init` and `ffr_init`.

Prototype:

```
status_t (*ffr_cust_factory_page_write)(flash_config_t *config, uint8_t *page_data, bool seal_part);
```

Parameter	Description
config	Pointer to `flash_config_t` data structure in memory to store driver runtime state.
page_data	Pointer to a value address that will be written to the destination address.
seal_part	If set as true or the `page_data` includes the non-zero CMAC data, the CMPA CMAC will be calculated and programed into the CMPA region.

Implementation:

```
/*ffr_cust_factory_page write api*/
uint32_t cmpa_buffer_cust[512] = {0};
uint32_t status_cmpa = FLASH_API_TREE-> ffr_cust_factory_page_write
(&flashConfig, (uint8_t *)cmpa_buffer_t, false);
```

Output: If the status is `kStatus_FLASH_Success` return value, it means that the `cmpa_buffer` data has been programmed into the CMPA region.

4.4 ffr_get_uuid

The API is used for getting the UUID data of the device, and the API should be called after the `flash_init` and `ffr_init`.

Prototype:

```
status_t (*ffr_get_uuid)(flash_config_t *config, uint8_t *uuid);
```

Parameter	Description
`config`	Pointer to `flash_config_t` data structure in memory to store driver runtime state.
`uuid`	Pointer to value address, the value is read back from the `nmpa` configuration `uuid`.

Implementation:

```
/*UUId api*/
uint32_t uuid_buffer [4];
uint32_t status_uuid = FLASH_API_TREE-> ffr_get_uuid(&flashConfig,
(uint8_t *)uuid_buffer);
```

Output: If the status is `kStatus_FLASH_Success` return value, it means that the UUID data has been got from the UUID of the device.

4.5 `ffr_get_customer_data`

This API is used to read data stored in the Customer Factory page, and the API should be called after `flash_init` and `ffr_init`.

Prototype:

```
status_t (*ffr_get_customer_data)(flash_config_t *config, uint8_t *pData, uint32_t
 offset, uint32_t len);
```

Parameter	Description
`config`	Pointer to `flash_config_t` data structure in memory to store driver runtime state.
`pData`	Point to the destination buffer of date that stores data read from the Customer Factory Page.
`offset`	Point to the offset value based on the CMPA address (`0x3e200`) of the device.
`len`	The length in bytes to be read back, and the offset + len <= 512 B.

Implementation:

```
/*ffr_get_customer_data api*/
uint32_t cmpa_buffer[4];
uint32_t offset = 0;
uint32_t status_custdata = FLASH_API_TREE->ffr_get_customer_data
(&flashConfig, (uint8_t *)cmpa_buffer, offset, sizeof(cmpa_buffer));
```

Output: If the status is `kStatus_FLASH_Success` return value, it means that the CMPA data has been successfully read from the CMPA region and stored into the `cmpa_buffer`.

4.6 `ffr_cust_keystore_write`

The API is used for programing the customer key store data into the customer key store region (from `0x3e400` to `0x3e600`), and the API should be called after `flash_init` and `ffr_init`.

Prototype:

```
status_t (*ffr_cust_keystore_write)(flash_config_t *config, ffr_key_store_t *pKeyStore);
```

Parameter	Description
`config`	Pointer to `flash_config_t` data structure in memory to store driver runtime state.
`pKeyStore`	Pointer to the customer key store data buffer, which will be programed into the customer key store region.

Implementation:

```
/*ffr_cust_keystore_write api*/
uint32_t cust_keystore_buffer={4};
uint32_t status_custkeywrite = FLASH_API_TREE->ffr_cust_keystore_write
(&flashConfig, (ffr_key_store_t *) cust_keystore_buffer);
```

Output: If the status is `kStatus_FLASH_Success` return value, it means that the customer key store data has been programmed into the customer key store region.

4.7 `flash_read`

The API is used for getting internal flash data, including the FLASH and FFR data, and the API should be called after the `flash_init`.

Prototype:

```
status_t (*flash_read)(flash_config_t *config, uint32_t start, uint8_t *dest, uint32_t
 lengthInBytes);
```

Parameter	Description
`config`	Pointer to `flash_config_t` data structure in memory to store driver runtime state.
`start`	Point to the start address where will be read.
`dest`	Pointer to the buffer used for storing the read data.
`lengthInBytes`	Point to the read data length

Implementation:

```
/*Flash read api call*/
uint32_t start_addr = 0x1000;
uint8_t read_buffer[512] = {0};
uint32_t length_b = 512;
uint32_t status_flashread = FLASH_API_TREE-> flash_read(&flashConfig,
start_addr, (uint8_t *) read_buffer, length);
```

Output: If the status is `kStatus_FLASH_Success` return value, it means that the expected read region has been loaded into the `read_buffer`.

4.8 `ffr_get_cust_keystore`

API is used for getting the customer key store data from the customer key store region (from `0x3e400` to `0x3e600`), and the API should be called after `flash_init` and `ffr_init`.

Prototype:

```
status_t (*flash_get_cust_keystore)(flash_config_t *config, uint8_t *pData, uint32_t
 offset, uint32_t len);
```

Parameter	Description
`config`	Pointer to `flash_config_t` data structure in memory to store driver runtime state.
`pData`	Pointer to the customer key store data buffer, which got from the customer key store region.
`offset`	Point to the offset value based on the customer key store address (`0x3e400`) of the device.
`len`	Point to the length of the expected to get customer key store data, and the offset + len <= 512 B.

Implementation:

```
/*ffr_get_cust_keystore api*/
uint8_t cust_keystore_buffers[512] = {0};
uint32_t offset_b = 0;
uint32_t length = 512;
uint32_t status_get_cust_key = FLASH_API_TREE-> flash_get_cust_keystore
(&flashConfig, (uint8_t *) cust_keystore_buffers, offset, length);
```

Output: If the status is `kStatus_FLASH_Success` return value, it means that the customer key store data has been got from the customer key store region and stored into the `cust_keystore_buffer`.

5 Acronyms

Table 1. Acronyms

Abbreviations	Full Forms
ROM	Read Only Memory
FlexSPI	Flexible Serial Peripheral Interface
IAP	In-Application Programming
OTP	One Time Program
CMPA	Customer Manufacturing Programming Area
CFPA	Customer Factory Programming Area
UUID	Universal Unique Identifier
FFR	Another name for PFR (Protected Flash Region)

6 References

- *LPC553x Reference Manual* (document LPC553xRM)

7 Note about the source code in the document

The 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 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 2 summarizes the revisions to this document.

Table 2. Revision history

Document ID	Release date	Description
AN14562 v1.0	28 February 2025	Initial public release

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.

AMBA, Arm, Arm7, Arm7TDMI, Arm9, Arm11, Artisan, big.LITTLE, Cordio, CoreLink, CoreSight, Cortex, DesignStart, DynamIQ, Jazelle, Keil, Mali, Mbed, Mbed Enabled, NEON, POP, RealView, SecurCore, Socrates, Thumb, TrustZone, ULINK, ULINK2, ULINK-ME, ULINK-PLUS, ULINKpro, µVision, Versatile — are trademarks and/or registered trademarks of Arm Limited (or its subsidiaries or affiliates) in the US and/or elsewhere. The related technology may be protected by any or all of patents, copyrights, designs and trade secrets. All rights reserved.

Microsoft, Azure, and ThreadX — are trademarks of the Microsoft group of companies.

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.**

For more information, please visit: https://www.nxp.com

Document feedback

Date of release: 28 February 2025
Document identifier: AN14562