AN14259

SDK Example to Write CFPA with Monotonic Counter for MCXN947

Rev. 2.0 — 6 February 2025 Application note

Document information

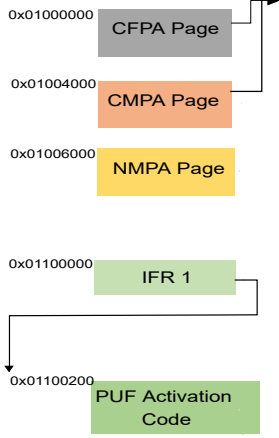
Information	Content
Keywords	AN14259, monotonic counter, CFPA
Abstract	This application note provides an example of how the CFPA bit field can be changed through the ROM APIs.

1 Introduction

This document provides an example of changing the Customer Field Programmable Area (CFPA) bit field using the ROM APIs. However, it can also be done using the MCUXpresso Secure Provisioning Tool, see MCUXpresso Secure Provisioning Tool. MCUXpresso Secure Provisioning Tool has a GUI-based interface that provides a streamlined development flow, making it simpler to prepare (change CFPA bit field, generate keys, and so on), flash and fuse images along with the provisioning of the bootable executables on NXP MCU devices. The ROM APIs or the MCUXpresso Secure Provisioning Tool also allows the programming of One Time Programmable (OTP) fuses, which can lock areas of CMPA and CFPA with specific settings. Such locking is necessary for the finished products, which are released using the MCXNx4x device.

The CMPA and CFPA are part of the Protected Flash Region (PFR) of the MCX devices. The PFR region contains settings for the boot configuration, security policy, PRINCE (internal and external), SoC-specific parameters, and so on. PFR has a fixed address in flash memory, which starts at address `0x1000000`. Figure 1 shows different regions of the PFR layout. The PFR contains a CFPA scratch page, CFPA ping and pong pages, CMPA page, customer key store page (part of CMPA), and NXP Manufacturing Programmable Area (NMPA) page.

Note: *The information provided in this document does not lock areas of CMPA and CFPA with specific settings when using OTP.*



Address (Hex)	Name
0x1000000	Header
0x1000004	CFPA_PAGE_VERSION
0x1000008	Secure_FW_Version
0x100000C	NS_FW_Version
0x1000014	SECBOOT_FLAGS
0x1000018	IMAGE_KEY_REVOKE
0x100001C	LP_VECTOR_ADDR
0x1000020	DBG_REVOKE_VU
0x1000024	DCFG_CC_SOCU_NS_PI
0x1000028	DCFG_CC_SOCU_NS_DF
0x1000030	MCTR_NPX_CTX0
...	
0x100003C	MCTR_NPX_CTX3
0x1000040	MCTR_IPED_CTX0
...	
0x100005C	MCTR_IPED_CTX7
0x1000060	ERR_AUTH_FAIL_COUN
0x1000064	ERR_ITRC_COUNT
0x1000068	ERR_CFG_FAIL_COUN
0x100006C	ERR_CONT3
...	
0x100007C	ERR_CONT7
0x1000080	MCTR_CUST_CTR0
...	
0x100009C	MCTR_CUST_CTR7
0x10000A0	MFLAG_CUST_0
...	
0x10000BC	MFLAG_CUST_7
0x10000C0	FLASH_ACL_0_7
...	
0x10000DC	FLASH_ACL_55_63
0x10000E0	SBL_IMG0_CMA
0x10000F0	IMG1_CMAC
0x1000100	DICE Certificate
0x1000190	IPED_GCM_AAD_CTX0
...	
0x10001AC	IPED_GCM_AAD_CTX7
0x10001EC	CFPA CRC32
0x10001F0	CFPA0 CMAC

Address (Hex)	Name
0x1004000	BOOT_CFG
0x1004004	FLASH_CFG
0x1004008	BOOT_LED_STATUS
0x100400C	BOOT_TIMERS
0x1004010	FLEXSPI_BOOT_CFG(
0x1004014	FLEXSPI_BOOT_CFG'
0x1004018	REC_SPI_FLASH_CFG
0x100401C	REC_SPI_FLASH_CFG
0x1004020	ISP_UART_CFG
0x1004024	ISP_I2C_CFG
0x1004028	ISP_CAN_CFG
0x100402C	ISP_SPI_CFG0
0x1004030	ISP_SPI_CFG1
0x1004034	ISP_USB_ID
0x1004038	ISP_USB_CFG
0x100403C	ISP_MISC_CFG
0x1004040	CC_SOCU_PIN
0x1004044	CC_SOCU_DFLT
0x1004048	VENDOR_USAGE
0x1004050	SECURE_BOOT_CFG
0x1004054	RoTK_USAGE
0x1004060	ROTKH[383:352]
...	
0x100408C	ROTKH[31:0]
0x1004090	NPX_CTX0_WD0
0x1004094	NPX_CTX0_WD1
...	
0x10040A8	NPX_CTX3_WD0
0x10040AC	NPX_CTX3_WD1
0x10040B0	IPED0_START
0x10040B4	IPED0_END
...	
0x10040E8	IPED7_START
0x10040EC	IPED7_END
0x10040F0	REC_IMG_EXT0
0x10040F4	REC_IMG_EXT1
0x1004120	QUICK_SET_GPIO_0
0x1004124	QUICK_CLR_GPIO_0
...	
0x1004148	QUICK_SET_GPIO_5
0x100414C	QUICK_CLR_GPIO_5
0x1004160	CUST_MK_SK key blob
0x10041EC	CMPA _CRC32
0x10041F0	CMPA CMAC

Figure 1. PFR layout

1.1 CMPA

The CMPA region provides the following configuration settings:

1. **Boot configuration:** Contains configurations related to the boot modes, speed, timeout, default boot source, USB product/vendor IDs, and so on.
2. **RoT key table hash:** Contains 48 bytes for Root Key Table Hash (ROTKH), which is an SHA-384 or SHA-256 digest computed over four OEM public keys. Four OEM private-public key pairs are stored so that extra key pairs are available when one private key is compromised. ROTKH is also called 'hash of hashes' as this hash is computed over the four OEM public key hashes.
3. **Debug configuration:** Configurations such as enabling/disabling debug access are also present. These configurations are used to determine the life cycle and debug authentication status.
4. **PRINCE configuration:** Allows runtime decryption of encrypted flash code and data. Contains external flash memory Inline Prince Encryption/Decryption (IPED) and internal flash memory PRINCE-related settings.
5. **FlexSPI configuration:** Contains configuration settings for the FlexSPI interface, which connects to external SPI flash memory, 1, 2, 4, and 8 bits are supported. FlexSPI supports Execute-In-Place (XIP) and on-the-fly decryption using an IPED module.

The CMPA also contains fields for the calculated CMAC/CRC32 for the CMPA region allowing an integrity check over the data in the CMPA region. The calculated value is written to the CMPA by the ROM when the `CMAC_UPD[CMPA_UPD]` field in the CFPA area is either set to `0b010` or `0b011`. The ROM also write this value to the `CMPA_CMAC` OTP if the `CMAC_UPD[CMPA_UPD]` bit is set to `0b011`, and after the `CMPA_CMAC` fuses are burned, the user cannot change the CMPA bit fields. The `CMPA_CMAC` fuses can generally be burned if a transition in the development stage of the MCU is required or if the chip is ready to be deployed in the field.

The CMPA also provides a customer key store page that can be used to store different keys required for the customers application purposes along with some preshared keys that must be supplied externally into Crypto Sub System (CSS), for example, the `CUST_MK_SK` key that is stored in the form of RFC3394 blob at address `0x1004160`. For more details, see *MCXNx4x Reference Manual* (document MCXNX4XRM) and its attachments *MCXNx4x_IFR.xlsx*.

1.2 CFPA

The CFPA scratch, ping, and pong pages have the same structure. However, the CFPA ping and pong pages are used to select the actual CFPA page (the one with the higher version number), while the scratch page is used to update the CFPA page. It means that whenever application code uses FLASH APIs to update the CFPA page, data is first written onto the scratch page, and then after the core reset, the contents of the scratch page are copied to either ping or pong page based on the CFPA page version number. The one with the lower version gets updated to the newest version.

The CFPA provides the following configuration settings:

1. **ROTKH revocation:** There can be a maximum of four RoT keys, each of which can be revoked. When trying to boot images that are signed using a revoked RoT key, they get rejected during the authentication process and fail to boot if *SEC_BOOT_EN* is set to boot only signed images.
2. **Image revocation:** An image key revocation ID is present, which can be used to revoke an image. For more details, see Chapter "Secure Boot ROM" Section "CFPA page" in *MCXNx4x Security Reference Manual* (document MCXNx4xSRM).
3. **Storage for PRINCE region IV codes:** Includes monotonic erase counters for internal and external PRINCE regions. However, these fields are entirely handled by the ROM, and the user must not write anything in these fields. For more details, see Chapter "Secure Boot ROM" Section "CFPA page" in *MCXNx4x Security Reference Manual* (document MCXNx4xSRM).

4. **Debug configuration:** Configurations such as enabling/disabling debug access are also present. These configurations are used to determine the life cycle and debug authentication status. For details, see Chapter "Debug Subsystem" in *MCXNx4x Reference Manual* (document MCXNX4XRM).

Apart from the above-mentioned configurations, the CFPA page also contains fields for various versions, such as the CFPA page version and the secure and nonsecure firmware versions. Fields for storing the CMAC values for the dual boot images that can be stored in the flash are also present. For more details, see *MCXNx4x Reference Manual* (document MCXNX4XRM) and its attachments *MCXNx4x_IFR.xlsx*.

1.2.1 Monotonic counter

Specific fields such as various versions, vendor usage, erase count for internal and external PRINCE regions, and customer-defined monotonic counter fields are implemented using a monotonic (incrementing) counter. A monotonic counter is implemented to produce incrementing or decrementing values. For the implementation of the incrementing values, once the count value changes to a higher number, it must not thereafter exhibit any value less than the higher number. For either implementation, the monotonic nature of the count value must be maintained throughout the life of the device, in which the monotonic counter operates, including across any number of power ON and power OFF cycles. Causing a monotonic counter not to maintain its count value and revert to an earlier value can result in a compromise in the device security.

1.3 NMPA

The NMPA is the last area of the PFR, which contains die and chip-specific data such as the PUF activation code, NXP-issued device genuineness proof certificates, UUID. For more details, see *MCXNx4x Reference Manual* (document MCXNX4XRM) and its attachments *MCXNx4x_IFR.xlsx*.

1.4 ROM API structure

The ROM API structure is shown in Figure 2. It contains several API drivers that contain absolute ROM API function addresses, which can be called using function pointers. Some of the functionalities provided by the ROM API are providing support for both the programmable FLASH region (store application code and data) and the FLASH firewall region (store device configurations, boot configuration, in-field programmable data). It also enables external serial NOR FLASH, on-chip OTP-eFUSE read and programming, support for crypto functions, in-application programming, and so on. The FLASH APIs are used through the FLASH API driver to update both the programmable flash area and specify the parameters in the CMPA and CFPA areas.

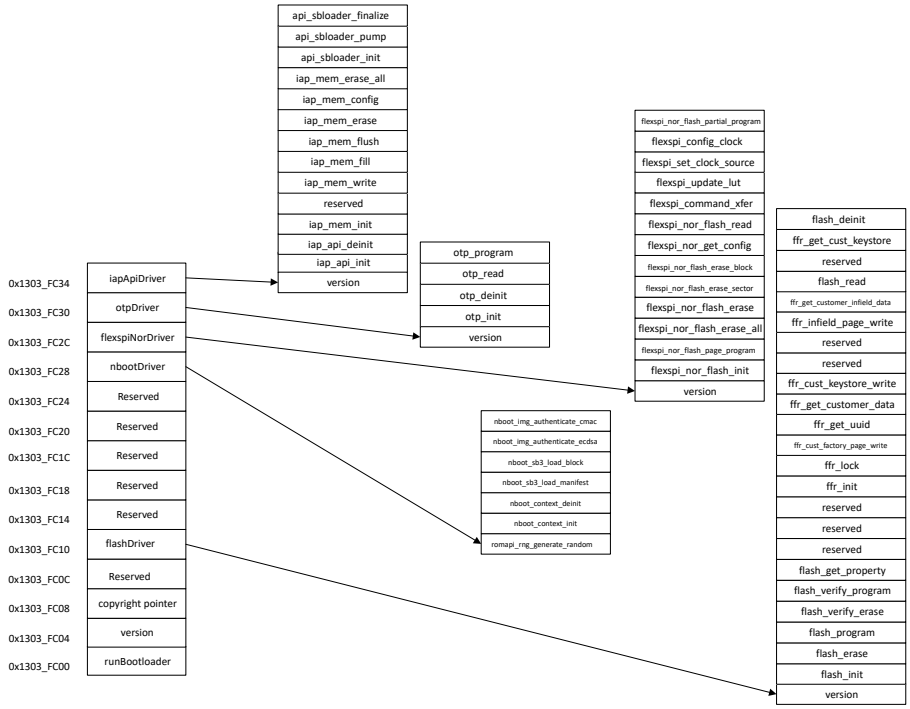


Figure 2. ROM API structure

1.5 FLASH API driver

The FLASH API set enables the following features:

1. Initialize the FLASH controller.
2. Erase and verify the specified FLASH area.
3. Program and verify the specified FLASH page.
4. Retrieve FLASH properties.
5. Initialize or lock the FFR.
6. Program and read the CMPA.
7. Program and read the CFPA.
8. Nonblocking FLASH erase/status check API for timing-critical use cases.

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

For more details on different APIs, see Chapter "ROM API" Section "FLASH APIs" in *MCXNx4x Reference Manual* (document MCXNX4XRM). The whole FLASH API wrapper is also available in the SDK release package.

2 Setup and SDK example

This section provides information about the hardware and software setup, SDK example, and output.

2.1 Hardware setup

Figure 3 shows the host computer connected to FRDM-MCXN947 board at MCU-Link USB J17. The J17 utilizes the MCU-Link VCOM output, which acts as a serial-to-USB bridge to the host computer and provides the CMSIS-DAP debug interface.

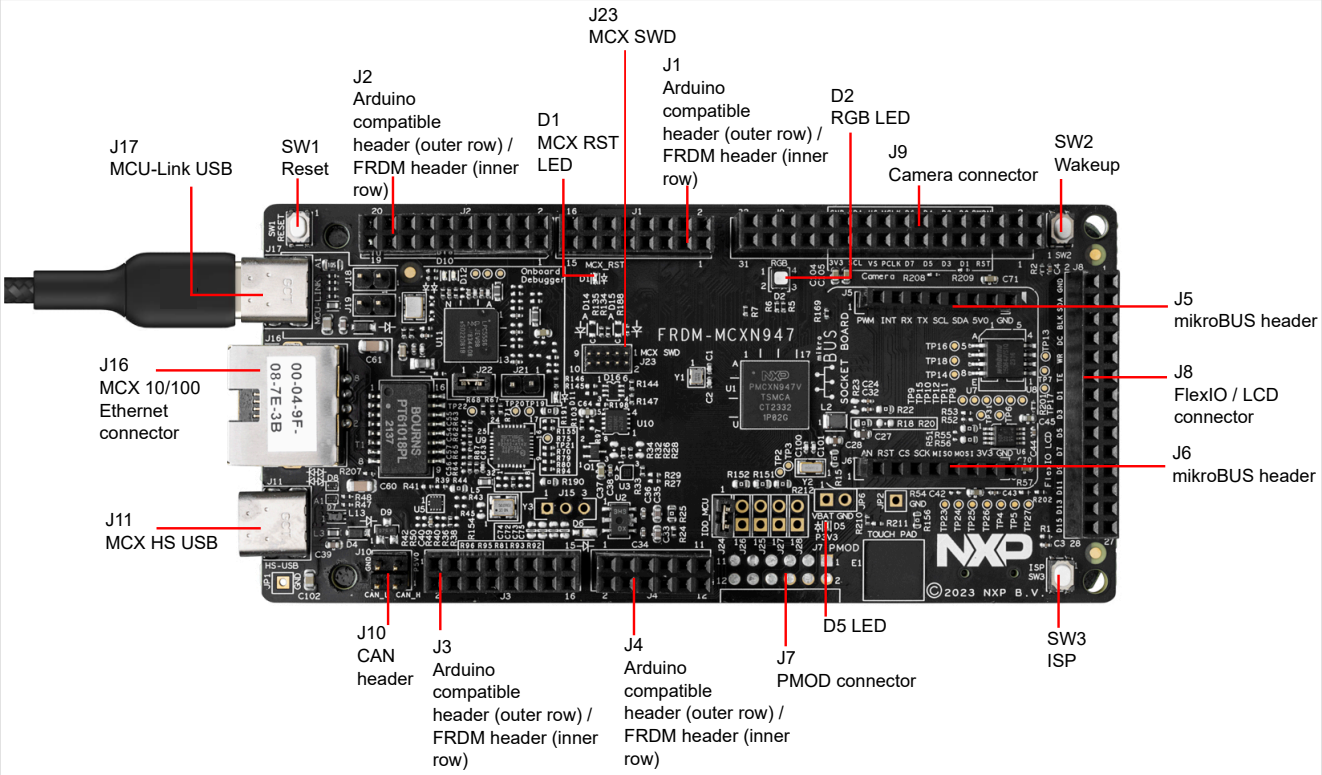


Figure 3. FRDM-MCXN947 board connect with MCU-LINK-USB at J17

2.2 Software setup

The following application example uses MCUXpresso IDE v11.8.0 to download, see MCUXpresso Integrated Development Environment (IDE). An SDK for MCX-N9XX-EVK is also required to build and debug the code. SDK_2.13.1_MCX-N9XX-EVK is used for the following example.

Note: *The output can be displayed on the Tera Term application.*

The following settings must be used while using Tera Term or any other terminal application:

- 115200 baud rate
- No parity
- 8 data bits
- 1 stop bits

2.3 SDK example

The SDK example project is embedded in this document. To import the SDK example project (archived folder):

1. Go to **File -> Import -> General -> Existing Projects**.
2. Into the workspace, click **Next**.
3. Select `.zip` archive file.
4. Click **Finish**.
5. The project must be reflected under the **Project Explorer** window.
6. **To build the project:** Click **Build** button from the **MCUXpresso IDE – Quickstart Panel** located on the bottom-left side of the IDE or click on the hammer icon located on the top panel.
7. **To flash the project onto the MCU:** Right-click **Project** tab **Run as -> 1 Local C/C++ Application**.

8. **To debug the project:** Select **Debug** button from the **MCUXpresso IDE – Quickstart Panel** or click **Bug** button from the top panel (highlighted in Blue).

In the following SDK example, three options are provided:

1. Display UUID and contents of the CFPA page.
2. Increase the CFPA page version.
3. Increase the vendor usage.

The CFPA page version and the vendor usage fields are implemented as a monotonic counter. Therefore, once their values are changed, the new values must always be higher than the old values. It is essential that every time a change is made to the CFPA page, the user must ensure that the CFPA page version is incremented (monotonic counter). Since CFPA changes are made to the CFPA scratch page, there must be a core reset for the new values to be copied from the CFPA scratch page onto the main CFPA page (either ping or pong page, based on the page version).

The example project uses functions from the FLASH API wrapper (available in the SDK release package as the `fsl_flash.h` header file and `fsl_flash.c` source file), which uses the underlying FLASH APIs. The following subsections provide information on some of the wrapper APIs used.

Note: *The function calls use a maximum of 2 kB of stack to read-modify-write PFR pages. Adjust the stack size accordingly.*

2.3.1 FFR_GetUUID

1. Underlying FLASH API: `ffr_get_uuid`.
2. Prototype: `status_t FFR_GetUUID(flash_config_t *config, uint8_t *uuid).`
3. Parameters:
   - `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`.
4. Code snippets from the example project:

```
volatile uint32_t g_UUID[4];
/* Initialize flash driver */
FLASH_Init(&flashInstance);
if (FFR_Init(&flashInstance) == kStatus_Success)
 PRINTF("Flash init successfull!!. Halting...\r\n");
else
 error_trap();

status =  FFR_GetUUID(&flashInstance, (uint8_t *)g_UUID);
PRINTF("\nUUID 0x%8x 0x%8x 0x%8x 0x%8x\r\n", g_UUID[0], g_UUID[1], g_UUID[2],
 g_UUID[3]);
```

2.3.2 FFR_GetCustomerInfieldData

1. Underlying FLASH API: `ffr_get_customer_infield_data`.
2. Prototype: `status_t FFR_GetCustomerInfieldData(flash_config_t *config, uint8_t *pData, uint32_t offset, uint32_t len).`
3. Parameters:
   - `config` - Pointer to `flash_config_t` data structure in memory to store driver runtime state.
   - `pData` - Point to the destination buffer of data that stores data read from the customer In-field page.
   - `offset` - Pointer to the offset value based on the CFPA address `0x3dc00` of the device.
   - `len` - Point to the length of data to read, and the offset + len <= 512 B.

4. Code snippets from the example project:

```
flash_config_t flashInstance;
uint32_t g_CFPAData[FLASH_FFR_MAX_PAGE_SIZE / sizeof(uint32_t)];

/* Read data stored on the Customer Factory page */
status = FFR_GetCustomerInfieldData(&flashInstance, (uint8_t *)g_CFPAData,
 0x0,
FLASH_FFR_MAX_PAGE_SIZE);

PRINTF("\r\nCFPA page data:\r\n");
PRINTF("Header 0x%08x ,          Version  0x%08x , SecureFW Version 0x%08x ,
NonSecureFW Version    0x%08x\r\n", g_CFPAData[0], g_CFPAData[1],
 g_CFPAData[2],
g_CFPAData[3]);
PRINTF("ImageKey Revoke 0x%08x , RothRevoke 0x%08x , Vendor Usage   0x%08x\r
\n",
g_CFPAData[4], g_CFPAData[20], g_CFPAData[21]);
PRINTF("NS PIN 0x%08x ,          NS DFLT  0x%08x ,   Enable FA Mode 0x%08x\r
\n",
g_CFPAData[22], g_CFPAData[23], g_CFPAData[24]);
```

2.3.3 FFR_InfieldPageWrite

1. Underlying FLASH API: `ffr_infield_page_write`.
2. Prototype: `status_t FFR_InfieldPageWrite(flash_config_t *config, uint8_t *page_data, uint32_t valid_len)`.
3. Parameters:
   - `config` - Pointer to `flash_config_t` data structure in memory to store driver runtime state.
   - `page_data` - Pointer to the source buffer of data that is to be programmed into the in-field page.
   - `valid_len` - The length in bytes to be programmed, the length must equal the page size.
4. Code snapshot from the example project:

```
flash_config_t flashInstance;
uint32_t g_CFPAData[FLASH_FFR_MAX_PAGE_SIZE / sizeof(uint32_t)];

status_t markStatus = FFR_InfieldPageWrite(&flashInstance, (uint8_t
 *)g_CFPAData,
 FLASH_FFR_MAX_PAGE_SIZE);
if (kStatus_FLASH_Success == markStatus)
{
 status = kStatus_Success;
 PRINTF("CFPA Write Done!\r\n");
}
else
{
 status = kStatus_Fail;
 PRINTF("CFPA Write Failed!\r\n");
}

/* Core reset to allow updating the main CFPA page */
NVIC_SystemReset();
break;
```

2.4 Output

The output is displayed on the Tera Term application using the UART COM5 port. After the successful build and flashing of the code onto the MCU, the user gets a prompt with three options, see Section 2.3.

1. If the user chooses the first option by typing `0`, the output is shown in Figure 4. The output includes the device-specific UUID and the contents of the CFPA page displayed.

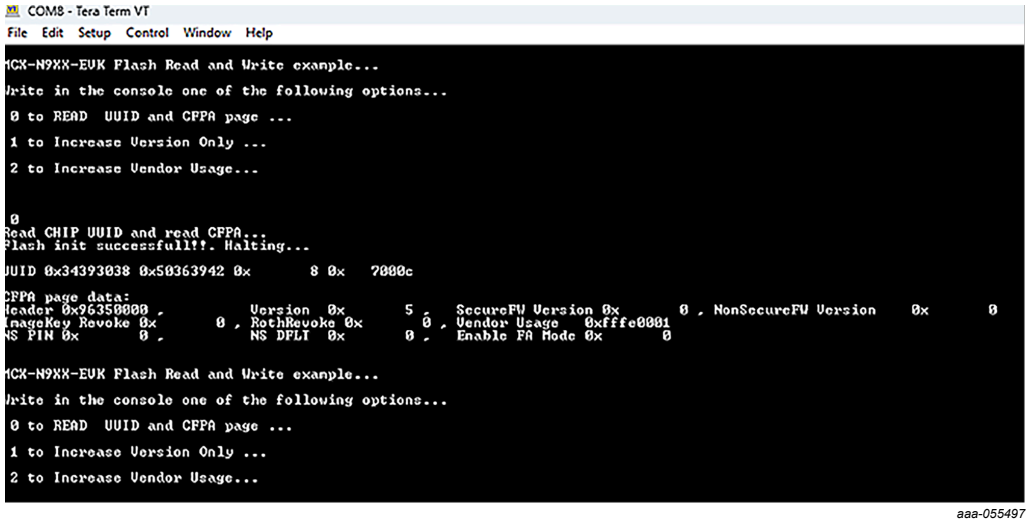


Figure 4. Generated output displayed on Tera Term for option `0`

2. If the user chooses the second option by typing `1`, the new version of the CFPA page that is written into the version field is shown in Figure 5. After the confirmation message is printed ("*CFPA Write Done!*"), the user can type `0` again to see the updated CFPA page contents.

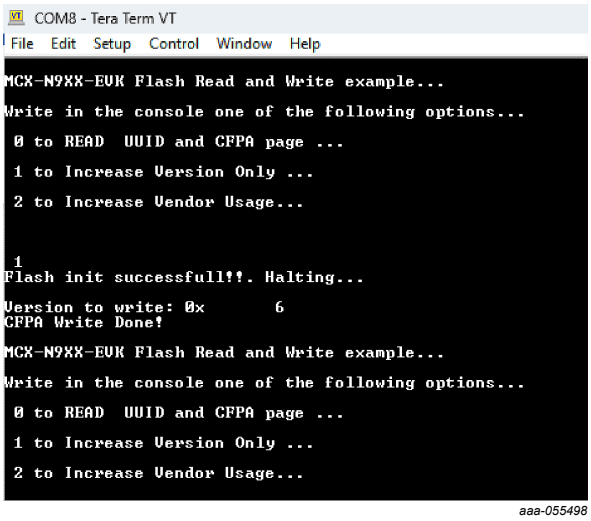


Figure 5. Generated output displayed on Tera Term for option `1`

3. If the user chooses the third option by typing `2`, the new version of the vendor usage page that will be written into the corresponding field is shown in Figure 6 along with the updated CFPA page version. After the confirmation message is printed `CFPA Write Done!`, the user can type "0" again to see the updated CFPA page contents.

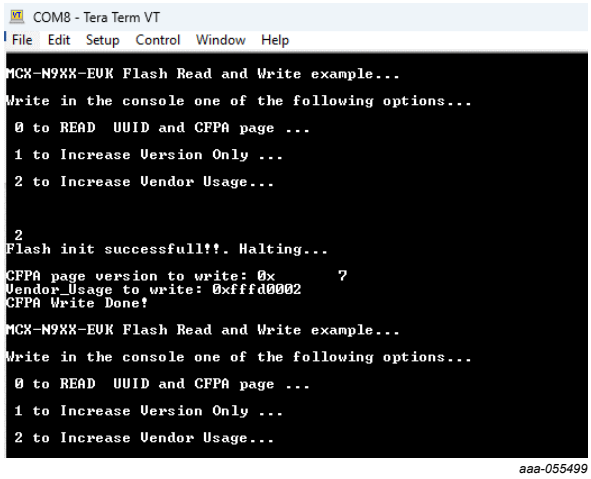


Figure 6. Generated output displayed on Tera Term for option 2

The CFPA page is updated based on the user input, therefore displaying the successful working of the software.

3 References

Table 1 lists additional documents and resources that can be referred to for more information. Some of the documents listed below may be available only under a non-disclosure agreement (NDA). To request access to these documents, contact local field applications engineer (FAE) or sales representative.

Table 1. Related documentation/resources

Document	Link/how to access
MCUXpresso Secure Provisioning Tool	MCUXPRESSO-SECURE-PROVISIONING
MCUXpresso Integrated Development Environment (IDE)	MCUXpresso-IDE
MCXNx4x Reference Manual (document MCXNX4XRM)	MCXNX4XRM
MCXNx4x Security Reference Manual (document MCXNx4x SRM)	Contact local FAE or sales representative

4 Acronyms

Table 2 defines the acronyms used in this document.

Table 2. Acronyms

Acronym	Definition
API	Application Programming Interface
CFPA	Customer Field Programmable Area
CMPA	Customer Manufacturing/Factory Programmable Area
COM	Communications Port
FAE	Field Applications Engineer
GUI	Graphical User Interface
IPED	Inline Prince Encryption/Decryption
MCU	Microcontroller Unit

Table 2. Acronyms...*continued*

Acronym	Definition
NDA	Non-Disclosure Agreement
NMPA	NXP Manufacturing Programmable Area
NOR	Negative-OR
OEM	Original Equipment Manufacturer
OTP	One Time Programmable
PFR	Protected Flash Region
ROTKH	Root Key Table Hash
ROM	Read-Only Memory
SDK	Software Development Kit
System-on-Chip	SoC
SPI	Serial Peripheral Interface
UART	Universal Asynchronous Receiver Transmitter
USB	Universal Serial Bus
VCOM	Virtual COM
XIP	Execute-In-Place

5 Note about the source code in the document

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

Copyright 2024-2025 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.

6 Revision history

Table 3 summarizes revision to this document.

Table 3. Revision history

Document ID	Release date	Description
AN14259 v.2.0	6 February 2025	Updated the Title and Introduction section of this document.
AN14259 v.1.0	18 July 2024	Initial public version

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.

Contents