AN14178

MCXNx4x Flash Command Example

Rev. 1 — 24 January 2024 Application note

Document information

Information	Content
Keywords	AN14178, MCXNx4x, MCX N, MCX N Series, MCXNx4x Flash Command Controller, Flash IAP, Flash Programming, Arm Cortex-M33, General Purpose MCU
Abstract	This document explains how to use the flash command controller to perform flash read and write operations, which can be more efficient than using calls to the ROM API.

1 Introduction

This document explains how to use the flash command controller to perform flash read and write operations, which can be more efficient than using calls to the ROM API. In some complex applications, it is required to have non-blocking flash operations. However, the command write sequence can be more difficult to use. The purpose of this document is to provide instructions on how to program internal flash on MCXNx4x using the command write sequence.

2 Overview

The process follows a generic flash command write sequence, as shown in Figure 1.

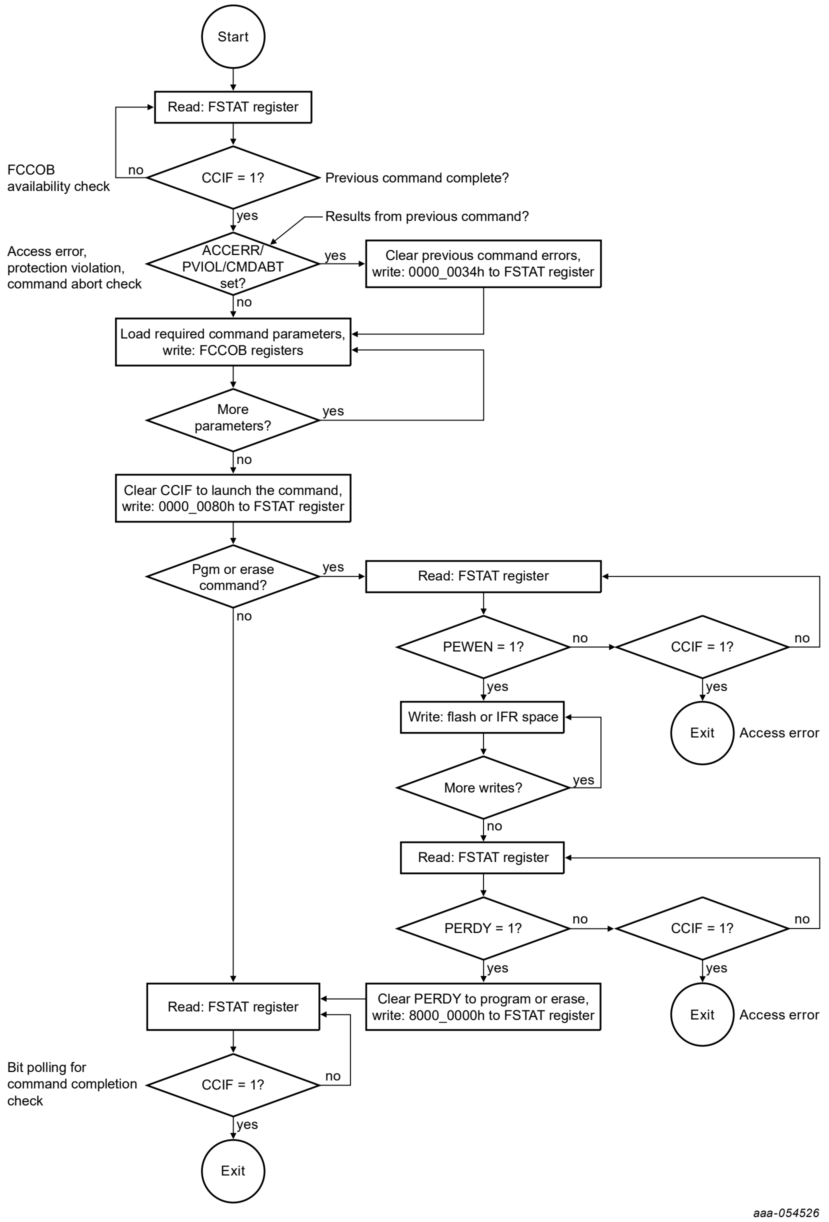


Figure 1. Command write sequence

2.1 High-level overview

Following is the high-level overview of the steps used:

1. Initialize the necessary clocks and registers.
2. Erase `0x10_0000` -> `0x1F_FFFF` one sector `8192 Bytes` of internal flash at a time using the erase sector command.
3. Program `0x10_0000` -> `0x1F_FFFF` one page `128 bytes` at a time using the program page command.
4. Verify that the values stored match the expected values.
5. Additionally, between each command, check the `FSTAT` registers for error handling and wait for `CCIF` to be set before continuing with the next command.

For more details, see Figure 2.

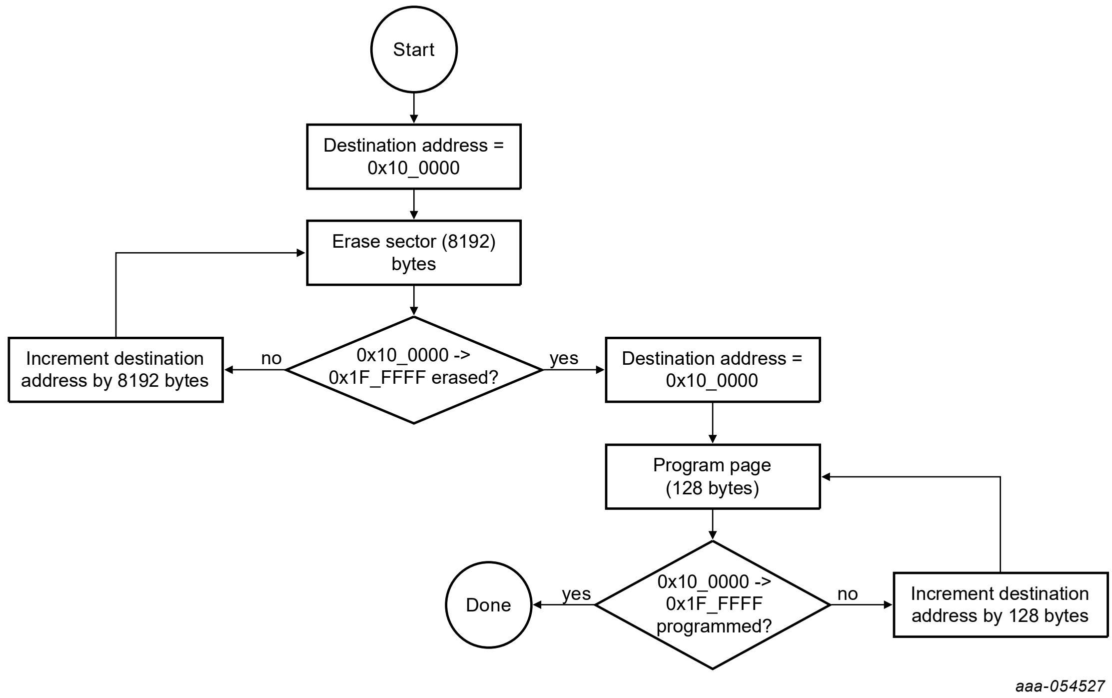


Figure 2. Flash erase-write flow

3 Use case example

An example use case is provided, which includes an MCUXpresso project that erases and programs the second half of flash, size 1 MB. The example can be found in the associated software package of this application note.

As outlined in Section 1, this process follows the generic command write sequence. The following subsections highlight the commands used in this example.

3.1 Erase sector

These steps show the process for erasing one sector `8192` bytes. For the example project, the process gets repeated until the entire second half of the flash is erased. And, it begins with a destination address `destAdrss = 0x10_0000`, the first index in the second half of flash.

1. Check `FMU FSTAT` register to ensure that `CCIF` is set. The previous command is completed.

```
if (((FMU0->FSTAT & FMU_FSTAT_CCIF(1)) >> FMU_FSTAT_CCIF_SHIFT) == 1)
```

```
{
  //continue with programming
}
```

If the CCIF register is not set, then we cannot continue with the operation and must wait until the previous operation is completed before starting another flash controller command. In the example code, a while loop is used to accommodate for a wait until the CCIF register is set. However, it is up to the developer to consider whether the application needs to be running other tasks in parallel.

2. Handle and clear any error flags present in FMU FSTAT register.

```
//clear previous errors
FMU0->FSTAT = 0x34;
```

The value for FSTAT_CLEARERR is 0x34.

3. Specify the command as erase sector by setting FMU FCCOB[0] to 0x42 (ERSSCR).

```
//42h is erase sector command ERSSCR
//specify command
FMU0->FCCOB[0] = 0x42;
```

4. Clear CCIF register to launch the command.

```
//clear ccif to launch
FMU0->FSTAT = 0x80;
```

The value for FSTAT_CLEARCCIF is 0x80. This writes a 1 to FSTAT[CCIF] bit, which clears it.

5. Check FMU FSTAT PEWEN == 1, writes are enabled for one phrase.

```
if (((FMU0->FSTAT & FMU_FSTAT_PEWEN(value)) >> FMU_FSTAT_PEWEN_SHIFT) == 1)
{
//continue
}
```

We cannot continue with the operation of the erase sector command until FSTAT PEWEN is equal to 1. In the example code, a while loop is used to accommodate for a wait until the PEWEN register is set. However, it is up to the developer to consider whether the application needs to be running other tasks in parallel.

6. Write four consecutive words to the flash, with the first write being phrase or sector aligned.
Note: *The contents of these writes are insignificant, as the sector is to be erased, but we must perform four consecutive writes for the command to execute per the implementation of the erase sector command.*
The destination address at the beginning of the example is 0x100000. This is the first index in the second half of flash.

```
*(volatile uint32_t *)(destAdrss) = 0x0;
*(volatile uint32_t *)(destAdrss + 4) = 0x0;
*(volatile uint32_t *)(destAdrss + 8) = 0x0;
*(volatile uint32_t *)(destAdrss + 12) = 0x0;
```

7. Check for PERDY == 1, the operation is ready to execute.

```
if (((FMU0->FSTAT & FMU_FSTAT_PERDY(1)) >> FMU_FSTAT_PERDY_SHIFT) == 1)
{
//continue
}
```

We cannot continue with this operation unless PERDY is set to 1, which means that the operation is ready to execute.
The PERDY must get set to one directly after the fourth consecutive *(volatile uint32_t *)(destAdrss + 12) = 0x0 write in the sequence of step 6. In the example code, a while loop is used to accommodate for a wait until the PERDY register is set. However, it is up to the developer to consider whether the application needs to be running other tasks in parallel.

8. Clear `PERDY` by writing `1` to it. The operation stalls until it is cleared.

```
//controller should erase AND verify after we clear PERDY
FMU0->FSTAT = 0x80000000;
```

9. Check for any errors in `FSTAT` register.

```
if (((FMU0->FSTAT & FMU_FSTAT_ACCERR(1)) >> FMU_FSTAT_ACCERR_SHIFT) == 1)
{
   PRINTF("\r\n Access Error \r\n");
}
else if (((FMU0->FSTAT & FMU_FSTAT_PVIOL(1)) >> FMU_FSTAT_PVIOL_SHIFT) == 1)
{
   PRINTF("\r\n Protection Violation \r\n");
}
else if (((FMU0->FSTAT & FMU_FSTAT_CMDABT(1)) >> FMU_FSTAT_CMDABT_SHIFT) ==
 1)
{
   PRINTF("\r\n Operation Is Aborted \r\n");
}
else if(((FMU0->FSTAT & FMU_FSTAT_FAIL(1)) >> FMU_FSTAT_FAIL_SHIFT) == 1)
{
   PRINTF("\r\n Command Failed \r\n");
}
```

10. Before continuing with another command controller operation, ensure that `FSTAT CCIF` is set. This command is completed.

```
if (((FMU0->FSTAT & FMU_FSTAT_CCIF(1)) >> FMU_FSTAT_CCIF_SHIFT) == 1)
{
//continue with programming
}
```

In the example code, a while loop is used to accommodate for a wait until the `CCIF` register is set. However, it is up to the developer to consider whether the application needs to be running other tasks in parallel.

3.2 Program page command

The following steps demonstrate the process for executing one program page command. The example project continues to perform the program page command until `0x10_0000 -> 0x1F_FFFF` is successfully programmed.

1. Check `FMU FSTAT` register to ensure that `CCIF` is set. This signifies that the previous command has been completed.

```
if (((FMU0->FSTAT & FMU_FSTAT_CCIF(1)) >> FMU_FSTAT_CCIF_SHIFT) == 1)
{
//continue with programming
}
```

The `CCIF` register must be set to `1` for us to continue with a new operation. In the example code, a while loop is used to accommodate for a wait until the `CCIF` register is set. However, it is up to the developer to consider whether the application needs to be running other tasks in parallel.

2. Handle and clear any error flags present in `FMU FSTAT` register.

```
//clear previous errors
FMU0->FSTAT = 0x34;
```

The value for `FSTAT_CLEARERR` is `0x34`.

3. Specify the command as program page by setting `FMU FCCOB[0]` to `0x23 (PGMPG)`.

```
//only need to specify command at call time
```

```
FMU0->FCCOB[0] = PGMPG;
```

4. Clear `CCIF` register to launch the command.

```
//clear ccif to launch
FMU0->FSTAT = 0x80;
```

Clear `CCIF` register by writing `1` to it, and launch the command.

5. Check for `FMU_FSTAT_PEWEN == 2`, writes are enabled for page programming - one page.

```
if (((FMU0->FSTAT & FMU_FSTAT_PEWEN(value)) >> FMU_FSTAT_PEWEN_SHIFT) == 2)
{
//continue
}
```

The `FSTAT_PEWEN` must be set to `2` to continue with the operation. In the example code, a while loop is used to accommodate for a wait until the `PEWEN` register is set. However, it is up to the developer to consider whether the application needs to be running other tasks in parallel.

6. Write `32` consecutive words to flash space.

```
//write 32 consecutive words to flash space
//one word = 4 bytes
for (int i = 0; i < 32; i++)
{
*(volatile uint32_t *)(destAdrss + index + (i*4)) = 0x12345678;
}
```

7. Check for `FMU_FSTAT_PERDY == 1`, the program command operation ready to execute.

```
if (((FMU0->FSTAT & FMU_FSTAT_PERDY(1)) >> FMU_FSTAT_PERDY_SHIFT) == 1)
{
//continue
}
```

In the example code, a while loop is used to accommodate for a wait until `PERDY` register is set. However, it is up to the developer to consider whether the application needs to be running other tasks in parallel.

Note: *Before we execute the command, the `FSTAT_PERDY` must be set to `1`.*

8. Clear `FMU_FSTAT_PERDY` by writing `1` to it, otherwise, the operation remain stalled.

```
//clear PERDY
FMU0->FSTAT = 0x80000000;
```

9. Check for errors in `FSTAT` register.

```
if (((FMU0->FSTAT & FMU_FSTAT_ACCERR(1)) >> FMU_FSTAT_ACCERR_SHIFT) == 1)
{
  PRINTF("\r\n Access Error \r\n");
}
else if (((FMU0->FSTAT & FMU_FSTAT_PVIOL(1)) >> FMU_FSTAT_PVIOL_SHIFT) == 1)
{
  PRINTF("\r\n Protection Violation \r\n");
}
else if (((FMU0->FSTAT & FMU_FSTAT_CMDABT(1)) >> FMU_FSTAT_CMDABT_SHIFT) ==
 1)
{
  PRINTF("\r\n Operation Is Aborted \r\n");
}
else if(((FMU0->FSTAT & FMU_FSTAT_FAIL(1)) >> FMU_FSTAT_FAIL_SHIFT) == 1)
{
  PRINTF("\r\n Command Failed \r\n");
}
```

10. Before continuing with another command controller operation, ensure that `FSTATCCIF` is set. This command is completed.

```
if (((FMU0->FSTAT & FMU_FSTAT_CCIF(1)) >> FMU_FSTAT_CCIF_SHIFT) == 1)
{
//continue with programming
}
```

4 Run demo

Requirements:

1. MCUXpresso 11.7.1 or newer
2. MCXNx4x EVK or FRDM
3. USB cable
4. SDK version 2.13.0

Steps:

1. Download the associated software package.
2. Import the project to MCUXpresso IDE - Quickstart Panel. Click **Import project(s) from file system…**, see Figure 3.

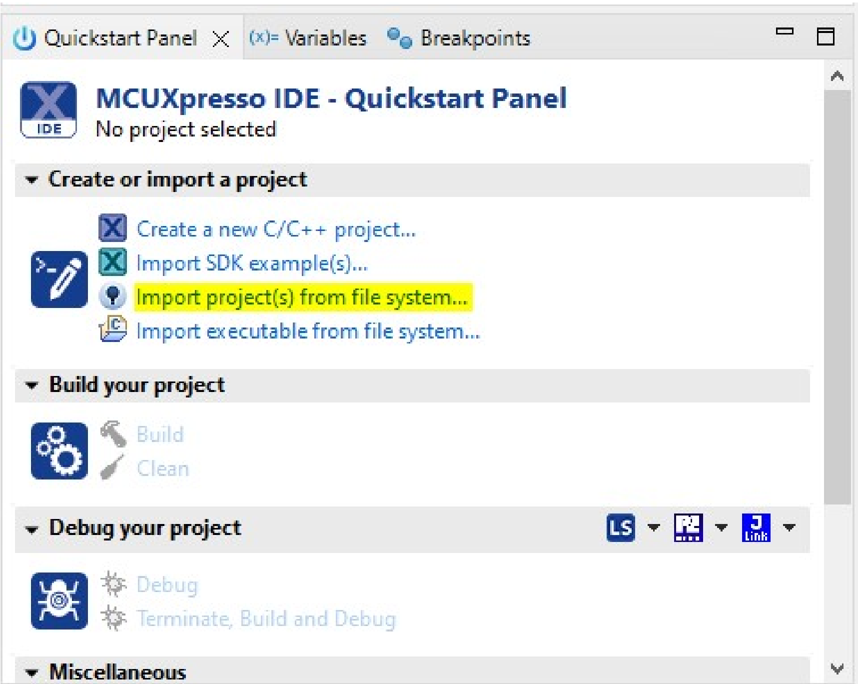


Figure 3. Quickstart panel - import project

3. Click **Browse…**, see Figure 4.

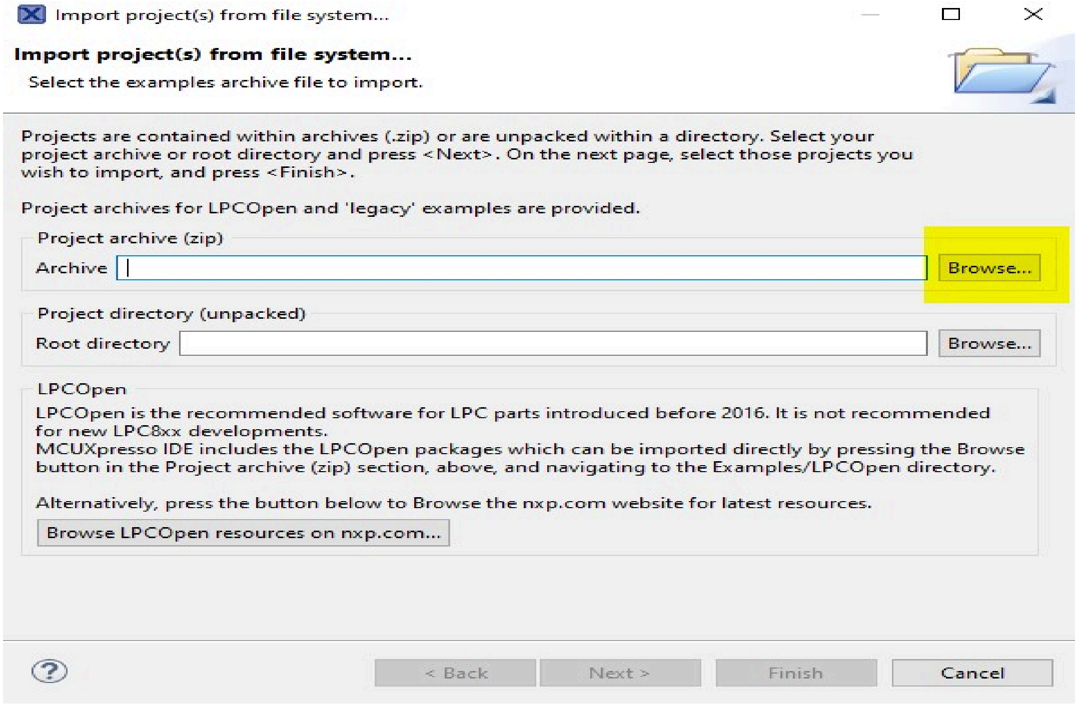


Figure 4. Import archived project

4. Navigate through the file browser and select the downloaded `IAP_Flash_Commands.zip`.

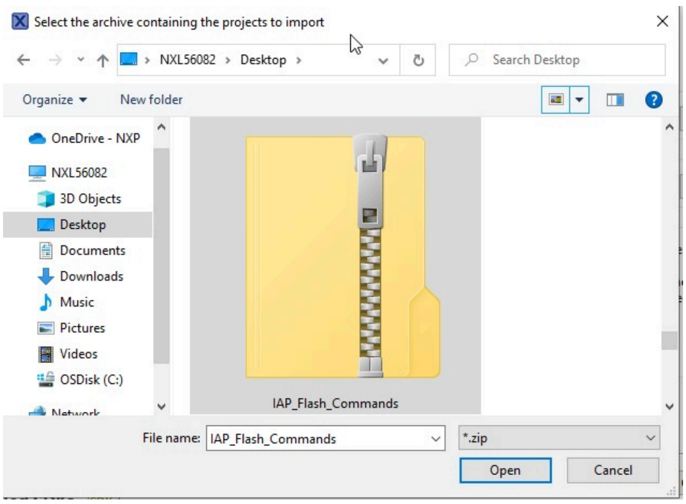


Figure 5. Selecting file from file browser

5. Click **Open**, see Figure 5.
6. Click **Next**, see Figure 6.

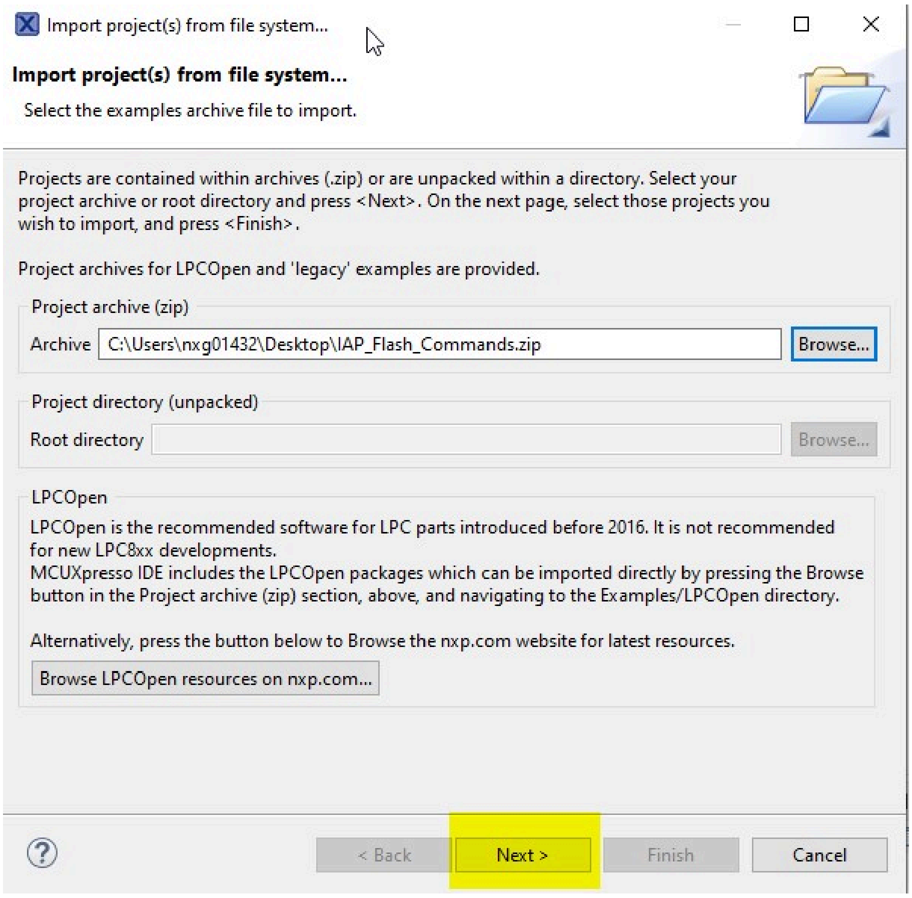


Figure 6. Importing `IAP_Flash_Commands.zip`

7. Click **Finish**, see Figure 7,

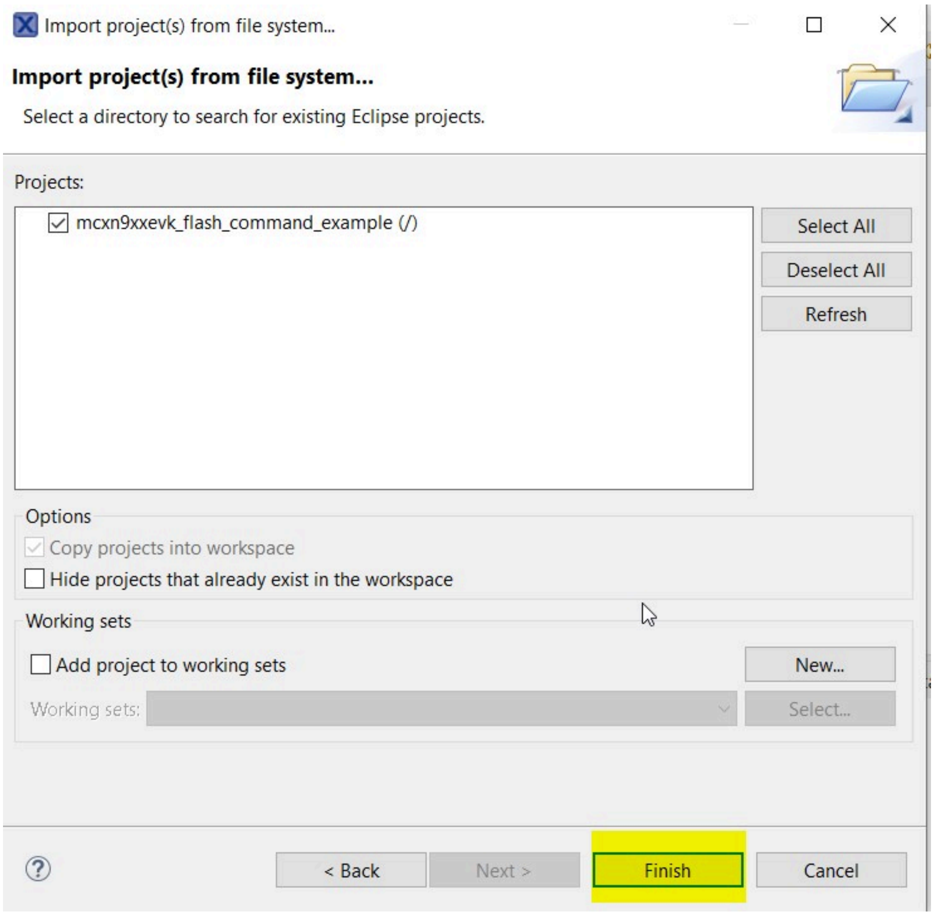


Figure 7. Import complete

Once the project is downloaded and imported into MCUXpresso, connect a micro-USB cable between the PC host and the MCU-Link USB port `J5` on the board when using MCX-N9XX-EVK, `J17` when using FRDM-MCXN947.

Open a serial terminal with the following settings:

- `115200` baud rate
- `8` data bits
- No parity
- One stop bit
- No flow control

1. Click **Launch Serial Terminal** option from the toolbar, see Figure 8.

Figure 8. Launch serial terminal

2. **Launch Terminal** windows pop up.
3. From the drop-down list of Choose terminal -> Select **Serial Terminal**, see Figure 9.

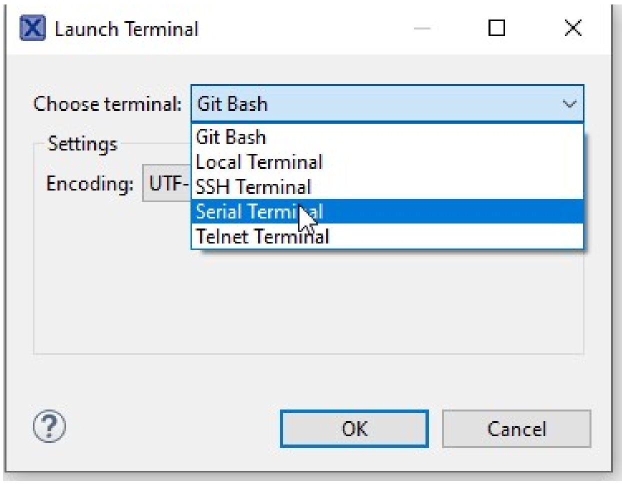


Figure 9. Select and launch serial terminal

4. Select **Serial port** associated with the connected device, see Figure 10.

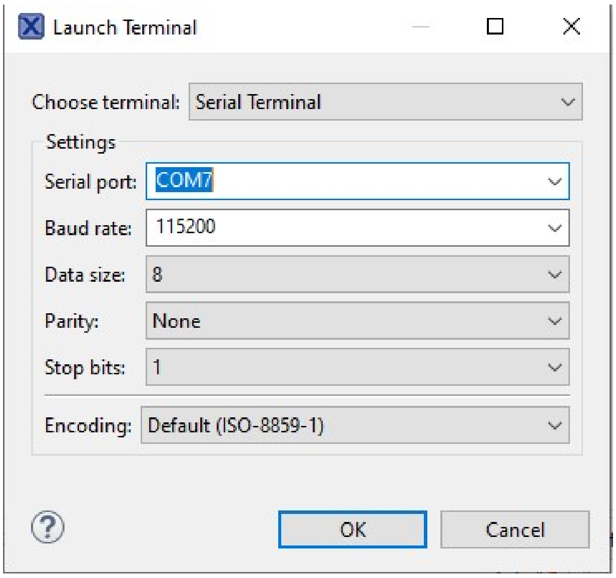


Figure 10. Launch terminal

Note: *The serial port differs for each user device.*

5. Select the following settings, see Figure 10.
   - **Baud rate** -> `115200`.
   - **Data size** -> `8`.
   - **Parity** -> None.
   - **Stop bits** -> `1`.
6. Click **OK**.
7. Click **Build** in Quickstart Panel, see Figure 11.

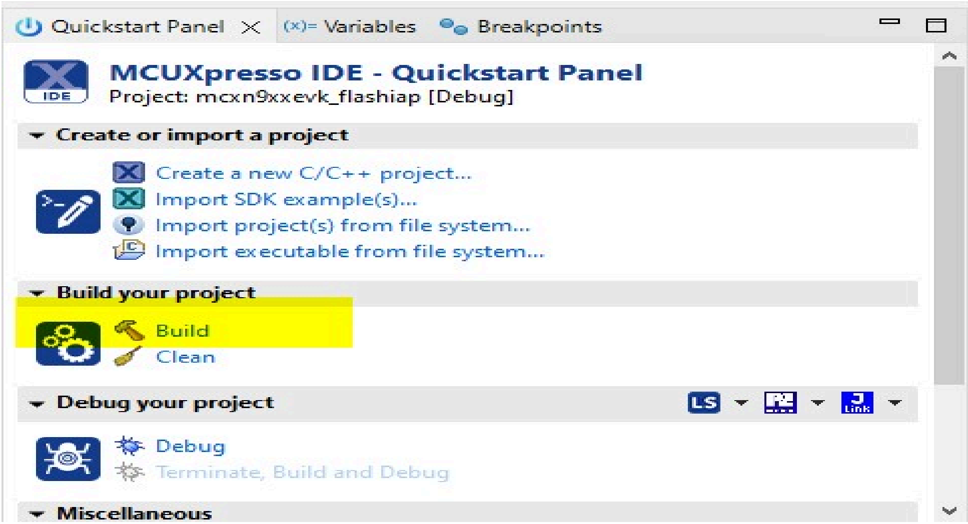


Figure 11. Quickstart panel - build project

8. Click **Debug**, see Figure 12.

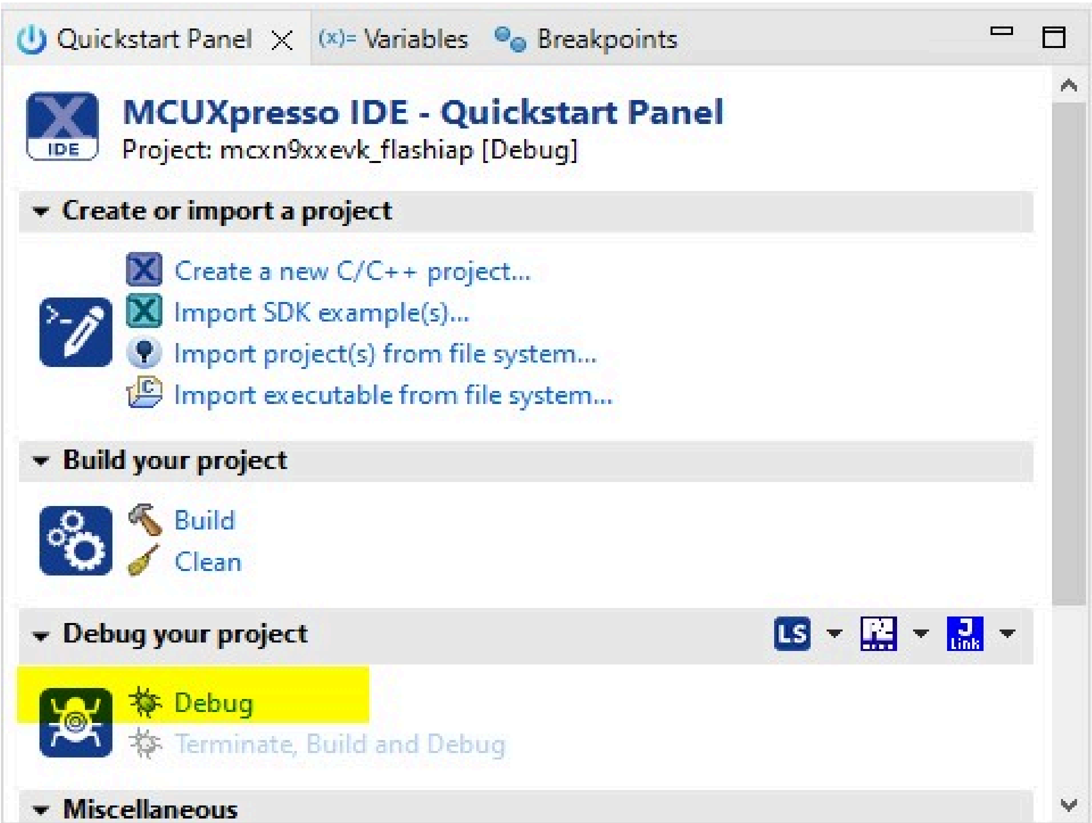


Figure 12. Quickstart panel - debug project

9. Click **OK**, see Figure 13.

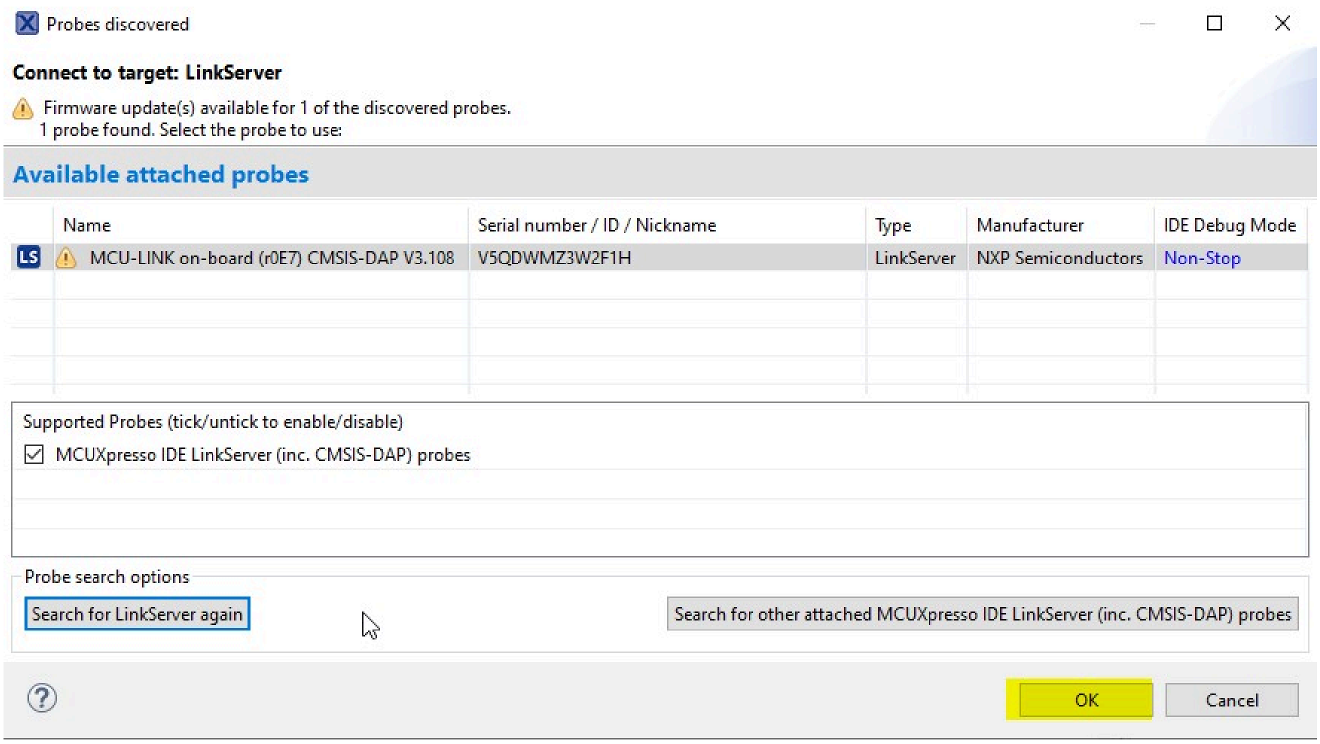


Figure 13. Confirm debug probe selection

10. Now, you must be able to step through the code. Click **Step Over** option in the toolbar, see Figure 14.

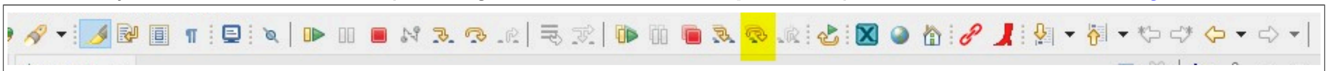


Figure 14. Step through icon

11. Step through line **215**, see Figure 15.

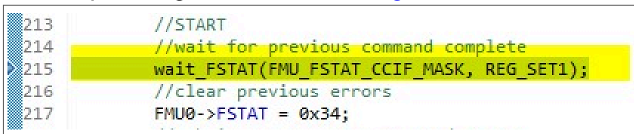


Figure 15. Execution on line 215

12. Now that we have reached the first step of the erase sector command, open the peripheral viewer. Click **Peripherals+** tab, see Figure 16.

Figure 16. Peripheral viewer tab

13. Expand **FMU0**, see Figure 17.

FMU0			0x40043000	Flash
FSTAT	0x00000080	RW	0x40043000	Flash Status Register
FCNFG	0xff000000	RW	0x40043004	Flash Configuration Register
FCTRL	0x00000003	RW	0x40043008	Flash Control Register
FCCOB0	0x00000000	RW	0x40043010	Flash Common Command Object Registers
FCCOB1	0x00000000	RW	0x40043014	Flash Common Command Object Registers
FCCOB2	0x00000000	RW	0x40043018	Flash Common Command Object Registers
FCCOB3	0x00000000	RW	0x4004301c	Flash Common Command Object Registers
FCCOB4	0x00000000	RW	0x40043020	Flash Common Command Object Registers
FCCOB5	0x00000000	RW	0x40043024	Flash Common Command Object Registers
FCCOB6	0x00000000	RW	0x40043028	Flash Common Command Object Registers
FCCOB7	0x00000000	RW	0x4004302c	Flash Common Command Object Registers

Figure 17. Peripheral viewer `FMU0`

14. We can see that the `FSTATCCIF` register is set to 1, meaning that no commands are still being executed, and we can execute a command using the command controller, see Figure 18.

FMU0			0x40043000	Flash
FSTAT	0x00000080	RW	0x40043000	Flash Status Register
FAIL	fail0	R	[0]	Command Fail Flag
CMDABT	cmdabt0	RW	[2]	Command Abort Flag
PVIOL	pviol0	RW	[4]	Command Protection Violation Flag
ACCERR	accerr0	RW	[5]	Command Access Error Flag
CWSABT	cwsabt0	RW	[6]	Command Write Sequence Abort Flag
CCIF	ccif1	RW	[7]	Command Complete Interrupt Flag

Figure 18. `FMU CCIF` register

15. Continue stepping through the code and stop on line **222**, see Figure 19.

```
221         //clear ccif to launch
222         FMU0->FSTAT = 0x80;
```

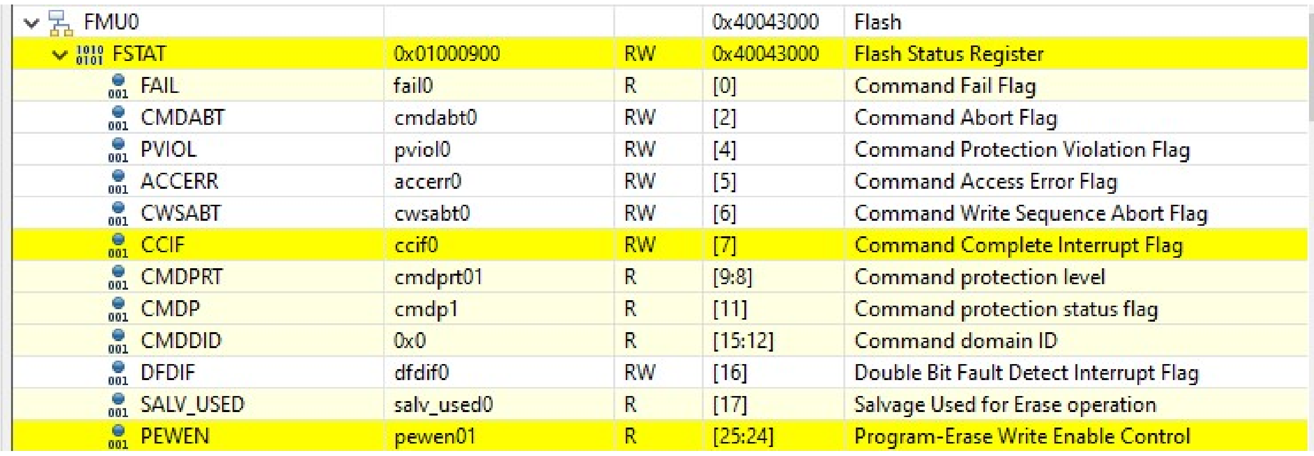
Figure 19. Stop execution on line 222

16. The peripheral viewer shows that we have set `FMU` -> FSTAT -> `FCCOB[0]` to `0x42`, which is the erase sector command, see Figure 20.

FCCOB0	0x00000042	RW	0x40043010	Flash Common Command Object Registers
CCOBn	0x42	RW	[31:0]	CCOBn

Figure 20. `FCCOB0` register

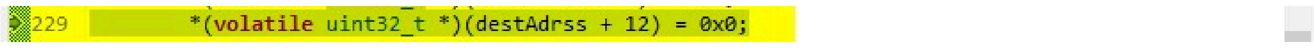
17. Step through the code one additional step, and we can see that we have cleared `CCIF`, causing the command to execute, see Figure 21.



FMU0			0x40043000	Flash
FSTAT	0x01000900	RW	0x40043000	Flash Status Register
FAIL	fail0	R	[0]	Command Fail Flag
CMDABT	cmdabt0	RW	[2]	Command Abort Flag
PVIOL	pviol0	RW	[4]	Command Protection Violation Flag
ACCERR	accerr0	RW	[5]	Command Access Error Flag
CWSABT	cwsabt0	RW	[6]	Command Write Sequence Abort Flag
CCIF	ccif0	RW	[7]	Command Complete Interrupt Flag
CMDPRT	cmdprt01	R	[9:8]	Command protection level
CMDP	cmdp1	R	[11]	Command protection status flag
CMDDID	0x0	R	[15:12]	Command domain ID
DFDIF	dfdif0	RW	[16]	Double Bit Fault Detect Interrupt Flag
SALV_USED	salv_used0	R	[17]	Salvage Used for Erase operation
PEWEN	pewen01	R	[25:24]	Program-Erase Write Enable Control

Figure 21. `CCIF` **and** `PEWEN` **register**

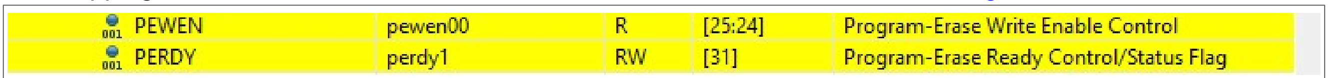
18. Continue stepping through and stop on line **229**. Recall that we need to perform four writes, with the first being sector-aligned. We have stopped on the fourth write in the sequence, see Figure 22.



```
229        *(volatile uint32_t *)(destAdrss + 12) = 0x0;
```

Figure 22. Stop execution on line 229

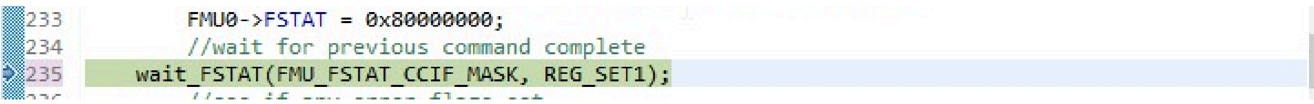
19. Stepping over this, we must see that `PEWEN` is cleared and `PERDY` is set, see Figure 23



PEWEN	pewen00	R	[25:24]	Program-Erase Write Enable Control
PERDY	perdy1	RW	[31]	Program-Erase Ready Control/Status Flag

Figure 23. `PEWEN` **and** `PERDY` **register**

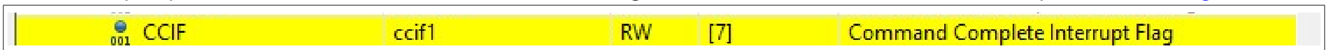
20. Step over line **233**, which clears `PERDY`, see Figure 24.



```
233        FMU0->FSTAT = 0x80000000;
234        //wait for previous command complete
235     wait_FSTAT(FMU_FSTAT_CCIF_MASK, REG_SET1);
```

Figure 24. Step over line 233

21. In the peripheral viewer, `CCIF` is set to `1`, meaning that the command has been completed, see Figure 25.



CCIF	ccif1	RW	[7]	Command Complete Interrupt Flag

Figure 25. `CCIF` **set to** `1` **– command complete**

Note: *The erase starts at* `0x10_0000` *and erases one sector.*

22. On **Peripherals+** tab, Click three vertical dots and select **Add memory monitor** -> **`PROGRAM_FLASH1`**, see Figure 26.

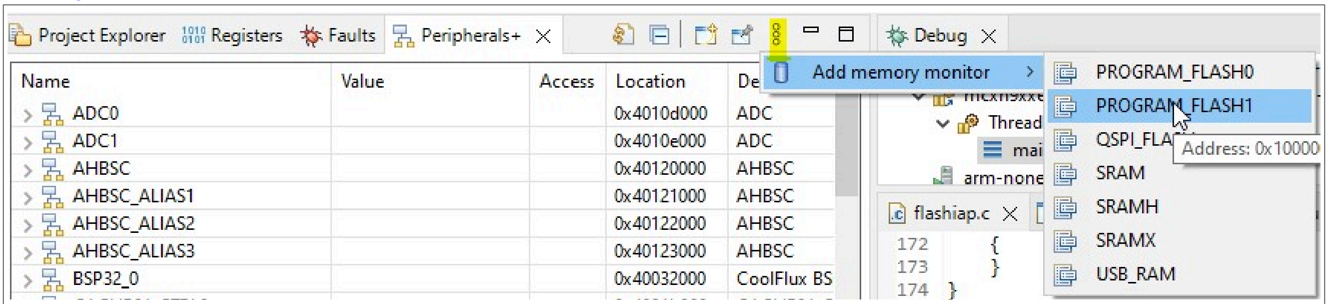
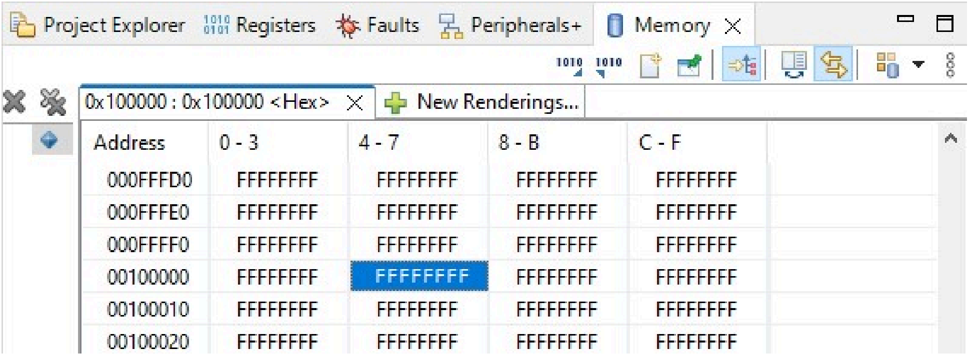


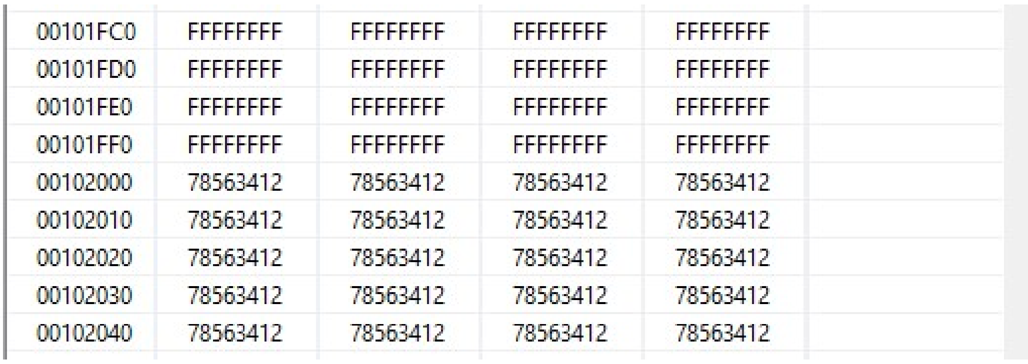
Figure 26. Open memory monitor – `PROGRAM_FLASH1`

23. After completing the erase sector command, on **Memory->0x100000: 0x100000 <Hex>** tab, we must find `FFFFFFFF` and continues until `0x102000`, which means one sector of flash has been erased, see Figure 27 and Figure 28.



Address	0 - 3	4 - 7	8 - B	C - F
000FFFD0	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF
000FFFE0	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF
000FFFF0	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF
00100000	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF
00100010	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF
00100020	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF

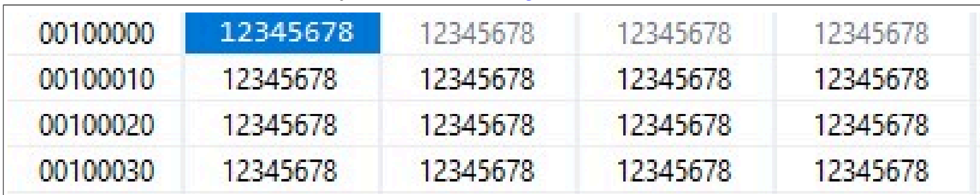
Figure 27. Sector erased in memory viewer



00101FC0	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF
00101FD0	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF
00101FE0	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF
00101FF0	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF
00102000	78563412	78563412	78563412	78563412
00102010	78563412	78563412	78563412	78563412
00102020	78563412	78563412	78563412	78563412
00102030	78563412	78563412	78563412	78563412
00102040	78563412	78563412	78563412	78563412

Figure 28. End of erased sector in memory viewer

24. You may now choose to continue stepping through the code or terminate the debug session as the flash program command follows a similar sequence.
25. After all the flash commands have been executed, the memory monitor must be filled with `0x1234_5678` hexadecimal in each 4-byte area, see Figure 29.



00100000	12345678	12345678	12345678	12345678
00100010	12345678	12345678	12345678	12345678
00100020	12345678	12345678	12345678	12345678
00100030	12345678	12345678	12345678	12345678

Figure 29. Second half of flash program

26. The following message is displayed in the terminal window, which confirms that the example code runs successfully.

```
Flash Command Erase / Programming example:
This application erases the flash area from 0x0010_0000 -> 0x001F_FFFF and
 then programs with 0x1234_5678.
Begin erase:   Success!
Begin Program: Success!

End of Flash Programming Example!
```

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 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 1 summarizes the revisions to this document.

Table 1. Revision history

Document ID	Release date	Description
AN14178 v.1.0	24 January 2024	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.

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.

MCX — is a trademark of NXP B.V.

Contents

1	**Introduction**	**2**
2	**Overview**	**2**
2.1	High-level overview	3
3	**Use case example**	**3**
3.1	Erase sector	3
3.2	Program page command	5
4	**Run demo**	**7**
5	**Note about the source code in the document**	**16**
6	**Revision history**	**17**
	Legal information	**18**

Date of release: 24 January 2024
Document identifier: AN14178