nRF9160 Production Programming

nAN-41

Application Note

v1.0

Contents

Revision history

Date	Version	Description
June 2020	1.0	First release

1 nRF9160 Production Programming

This document provides information on downloading software to nRF9160 devices and is intended for developers of flash download tools.

It is meant to serve as a starting point for nRF9160 device support in production tools and is intended to accelerate the engineering process of supporting nRF9160 devices. This document describes a robust way to program devices and in many cases, steps can be skipped. For example, if the device has never been programmed before and its flash is completely erased or if the device is unprotected.

2 Related documentation

In addition to the information in this document, you may need to consult other documents.

Nordic documentation

The following sections in the nRF9160 Product Specification contain relevant information for programming *System on Chip (SoC)*s:

- Memory – nRF9160 devices use flash-based *Non-volatile Memory (NVM)* in the code flash, *User Information Configuration Registers (UICR)*, and *Factory Information Configuration Registers (FICR)* memory regions. This section also has information about flash page size and number.
- NVMC — Non-volatile memory controller – has detailed specifications about timing for write/erase operations. For example, it takes 43 µs (max) to write one word in flash and it takes 173 ms (max) to erase all flash. This is theoretical because it does not take real-world overhead into consideration, such as an external tester interface, algorithm executed in RAM, or serial wire debug port speed.
- Debug and trace – provides access to the on-chip debug functionality. This is a standard two-pin *Serial Wire Debug (SWD)* interface as defined by Arm®.

The nWP034 - nRF9160 Hardware Verification Guidelines provides guidelines for the hardware verification testing of nRF9160 devices during development as well as production.

Other documentation

See the ARM Infocenter for more information on general concepts such as Arm CoreSight™ or *SWD*.

3 Programming flow

The diagram shows the flow of production programming under normal circumstances.

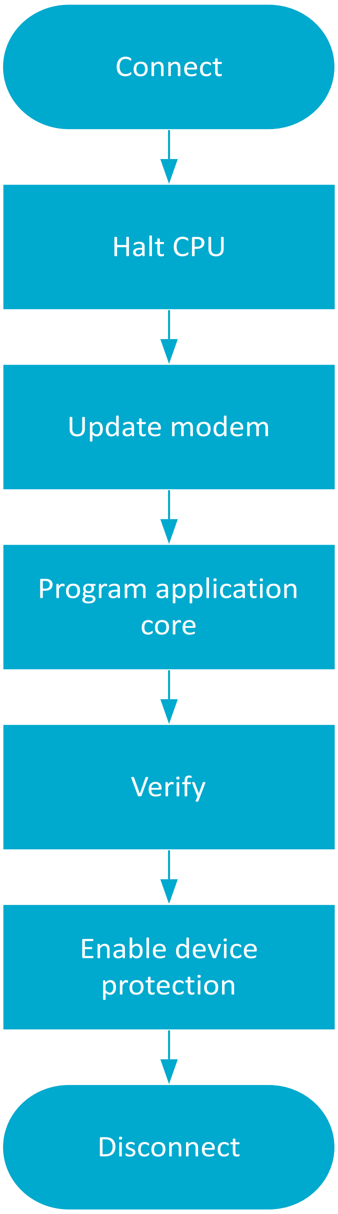

Figure 1: Normal programming flow

While the diagram describes the normal programming flow for initial programming of new devices, the following factors may affect production programming:

- *Access Port Protection (APPROTECT)* was enabled by a previously programmed application on the device that needs to be overwritten for any reason. See APPROTECT and ERASEPROTECT are enabled on page 15 for more information.
- *System Protection Unit (SPU)* memory protection was enabled by a previously programmed application on the device that may need to be overwritten or a different region of flash may need to be programmed while keeping the existing program unchanged. See Disabling SPU on page 17 for more information.

- A *watchdog timer (WDT)* was enabled by a previously programmed application on the device and may reset the device. See Reloading the watchdog timer on page 16 for more information.

4 Connecting

Use the standard *SWD* Arm CoreSight *Debug Access Port (DAP)* protocol to enter the nRF9160 Debug interface mode.

Before the external debugger can access the CPU, it must first request and make sure that the appropriate power domains are powered up. This is handled using the built-in CxxxPWRUPREQ and CxxxPWRUPACK feature found in the *DAP*. As long as the debugger is requesting the debug domain or the complete system to be powered up, the device will be in debug interface mode.

> **Note:** The user must request system and debug power. To enable both power domains, write CTRL/STAT.DBGPWRUPREQ and CTRL/STAT.SYSPWRUPREQ. See nRF9160 Revision 1 Errata - Debug and Trace for more information.

5 Updating the modem

nRF9160 devices come with preprogrammed firmware for the modem that must be updated in production to the modem firmware version the end product is certified with.

Modem firmware images are available from nRF9160 modem firmware. To update the modem firmware in production, use the firmware update algorithm described in this chapter.

5.1 Setting up the device

Configure the device to a state where it can communicate with the modem.

1. Write `0x2` to the SPU.PERIPHID[42].PERM (0x500038A8) register to configure IPC to be in non-secure mode .
2. Write the following values to configure IPC HW for *Device Firmware Update (DFU)*.

Register	Address	Value
IPC.SEND_CNF[1]	0x4002A514	0x00000002
IPC.SEND_CNF[3]	0x4002A51C	0x00000008
IPC.GPMEM[0]	0x4002A610	0x21000000
IPC.GPMEM[1]	0x4002A614	0x00000000
IPC.RECEIVE_CNF[0]	0x4002A590	0x00000001
IPC.RECEIVE_CNF[2]	0x4002A598	0x00000004
IPC.RECEIVE_CNF[4]	0x4002A5A0	0x00000010

3. Write `0x7` to all RAM regions n<0:31> 0x50003700+(n*4) SPU.RAMREGION[n].PERM to configure RAM as non-secure.
4. Write the following values to allocate memory in RAM.

Address	Value
0x20000000	0x80010000
0x20000004	0x2100000C
0x20000008	0x0003FC00

5. Power up or reset the modem by doing the following:
   a) Write `0` to POWER.LTEMODEM.STARTN (0x50005610).
   b) Write `1` to POWER.LTEMODEM.FORCEOFF (0x50005614).
   c) Write `1` to POWER.LTEMODEM.STARTN (0x50005610).
   d) Write `0` to POWER.LTEMODEM.FORCEOFF (0x50005614).
   e) Write `0` to POWER.LTEMODEM.STARTN (0x50005610).

5.2 Reading the modem key digest

You must program a modem firmware loader image into the RAM to transfer the modem firmware image from the *SWD* interface to the modem. This image needs to be signed with an authentication key recognized by the preprogrammed firmware in the modem. To select the correct modem firmware loader from the downloaded modem firmware package, get the key from the modem by reading the modem key digest. This digest determines the modem firmware loader you need to upgrade the modem.

To request the modem key digest, do the following steps.

1. Poll IPC.MODEM_CTRL_EVENT by reading the event registers in the following table and doing the corresponding action.
   - FAULT_EVENT (0x4002A100)
   - COMMAND_EVENT (0x4002A108)
   - DATA_EVENT (0x4002A110)

Register	Action
FAULT_EVENT (0x4002A100)	Restart the procedure if this register contains a value other than `0`.
COMMAND_EVENT (0x4002A108)	Proceed to step 2 if this register contains a value other than `0`.
DATA_EVENT (0x4002A110)	Proceed to step 2 if this register contains a value other than `0`.

2. Write `0` to the registers listed in step 1 to acknowledge events and reset event registers.
3. Read the value of address 0x20000010 to 0x20000030 to readout the digest.

   Note: Readout values need to be converted from big endian to little endian (e.g. readout value 0x01234567 is equal to 0x67452301).

4. Compare the first 7 hexadecimal digits of the readout value with the filenames in the nRF9160 modem firmware package. If there is no corresponding file with same hexadecimal digits, check if there are other packages with a valid file.

5.3 Programming the modem firmware loader

To program the modem firmware loader into the system RAM, do the following steps.

1. Unzip the modem firmware package and identify the modem firmware loader file (`<digest identifyer>.ipc_dfu.signed_<version>.ihex`) using the digest identifier retrieved in Reading the modem key digest on page 9.
2. Write the contents of the modem firmware loader file to the RAM.
3. Write `1` to address 0x4002A004.
4. Poll IPC.MODEM_CTRL_EVENT and acknowledge the event.

5.4 Updating the modem firmware

With the modem firmware loader up and running, you can update the modem firmware.

The modem firmware package has several HEX files. The firmware is divided in to several files named `firmware.update.image.segments.[n].hex` that contain separate segments of the firmware.

To successfully update the modem, you must program the firmware segments in correct order.

1. Take the lowest numbered `firmware.update.image.segments.[n].hex` file.
2. Split the content of the hex file into contiguous sections that are page aligned and no larger than X bytes.

3. From the previous step, write the following:
   a) The starting address of the section to 0x20000010.
   b) The length of the section to 0x20000014.
   c) The data of the section to 0x20000018.
   d) Write `0x3` to 0x2000000C.
4. Write `1` to 0x4002A004 to start the IPC transaction.
5. Poll IPC.MODEM_CTRL_EVENT and acknowledge the event.
6. Repeat steps 2 to 5 until all sections have been programmed.
7. Repeat steps 2 to 6 using the next `firmware.update.image.segments.[n].hex` file until all files have been programmed.

5.5 Verifying the modem

Use the verification file in the modem firmware package to check if the modem was updated successfully.

The `firmware.update.image.digest.txt` file contains the hash of all the different contiguous regions of the modem. Use the following procedure to get the hash of these regions from the modem and compare the values.

1. Using either the HEX files or the verification file, find the addresses and length of the segments you want to verify.
2. Write `0x7` to 0x2000000C.
3. Write the number of segments to be validated to 0x20000010.
4. Write the address of segment `n` to 0x20000014+(n*8).
5. Write the length of segment `n` to 0x20000018+(n*8).
6. Write `1` to 0x4002A004 to start the verification.
7. Read out the digest of the requested areas in 0x20000010 – 0x20000030.

   The values are in big endian and must be converted to little endian.
8. Compare the readout with the values in the verification file.
   If the received hash is the same as the hash given in the file, the firmware on the modem is the same as the one provided in the modem firmware package.

6 Programming the application core

When writing is enabled, the *NVM* is written by writing a word to a word-aligned address in the code or *UICR*. Only word-aligned writes are allowed. Byte or half-word-aligned writes result in a hard fault.

6.1 Writing to an empty application

The memory space of the application core is empty when nRF9160 devices are shipped. To program it, use the *SWD* Arm CoreSight *DAP* protocol.

1. Write `0x00000001` to the CONFIG register (0x50039504) of the *Non-volatile Memory Controller (NVMC)*.
   This enables writing to the *NVM*.
2. The READY register (0x50039400) of the *NVMC* will have a value of `0x00000001` when the device is ready for write operations.
3. Write the data to the selected, word-aligned address.

   When the write operation is completed, the READY register (0x50039400) of the *NVMC* will have a value of `0x00000001`.
4. Continue writing and then reading the READY register (0x50039400) as necessary.
5. Write `0x00000000` to the CONFIG register (0x50039504) of the *NVMC*.
   This configures the *NVM* as read-only.

The ranges of writeable addresses are:

- *UICR* addresses (located in addresses 0x00FF8000 through0xFF8FFC)
- All program flash (located in addresses 0x00000000 through ((INFO.CODESIZE * INFO.CODEPAGESIZE) – 0x4)

Different methods can be used to write flash. For nRF9160, a good flash algorithm should take around 10 seconds to write the entire flash. If you cannot achieve this time, contact Nordic Semiconductor for assistance.

6.2 Verifying flash content

To verify the contents of flash after programming, use the standard *SWD* Arm CoreSight *DAP* protocol and read every address written and compare with the expected value.

It is possible that the HEX file being programmed will enable access port (read-back) protection that makes it impossible to verify the contents of flash. This protection takes effect only after a reset is applied. Make sure not to reset between Programming the application core and Verifying.

6.3 Enabling device protection

There are several ways to protect nRF9160. *APPROTECT* secures the access port, *Erase Protection (ERASEPROTECT)* stops the device from being erased, and *Secure Access Port Protection (SECUREAPPROTECT)* stops unauthorized access to the secure domain.

Note: If the device has activated both *APPROTECT* and *ERASEPROTECT*, it cannot be recovered without a proper software solution. See Checking if APPROTECT is enabled on page 15 for more information.

APPROTECT

APPROTECT blocks debugger read/write access to all CPU registers and memory mapped addresses.

To check if *APPROTECT* is already enabled, read the UICR.APPROTECT(0x00FF8000) register. If this register has any other value than `0xFFFFFFFF`, it is protected. If the register shows the device as unprotected, write any value other than `0xFFFFFFFF` to it. The protection will be activated after a reset. If you are activating *ERASEPROTECT* or *SECUREAPPROTECT*, wait to reset until all the protections are set.

ERASEPROTECT

ERASEPROTECT blocks NVMC ERASEALL and CTRL-AP.ERASEALL functionality.

To check if *ERASEPROTECT* is already enabled, read the UICR.ERASEPROTECT(0x00FF8030) register. If this register has any other value than `0xFFFFFFFF`, it is protected. If the register shows the device as unprotected, write any value other than `0xFFFFFFFF` to it. The protection will be activated after a reset. If you are activating *APPROTECT* or *SECUREAPPROTECT*, wait to reset until all the protections are set.

SECUREAPPROTECT

SECUREAPPROTECT blocks debugger read/write access to all secure CPU registers and secure memory mapped addresses.

To check if *SECUREAPPROTECT* is already enabled, read the UICR.SECUREAPPROTECT(0x00FF802C) register. If this register has any other value than `0xFFFFFFFF`, it is protected. If the register shows the device as unprotected, write any value other than 0xFFFFFFFF to it. The protection will be activated after a reset. If you are activating *APPROTECT* or *ERASEPROTECT*, wait to reset until all the protections are set

7 Disconnecting

Use the standard *SWD* Arm CoreSight *DAP* protocol to exit the debug interface mode.

This is handled using the built-in CxxxPWRUPREQ and CxxxPWRUPACK features found in the Arm CoreSight *DAP*. When the debugger stops requesting the debug domain or the complete system to be powered up, the device exits the debug interface mode.

We recommend a hard reset of the device after programming by doing a power cycle.

8 Troubleshooting

nRF9160 devices can be reprogrammed by erasing pages. Reprogramming is also dependent on the current nRF9160 security setting.

8.1 Checking if APPROTECT is enabled

The *Control Access Port (CTRL-AP)* is a custom access port that enables control of the device even if other access ports in the *DAP* are disabled by *APPROTECT*.

If access port protection has been enabled in the *APPROTECT* register (0x00FF8000) of the *UICR*, the debugger's read/write access to all CPU registers and memory mapped addresses is blocked.

For more information about *CTRL-AP*, see nRF9160 CTRL-AP - Control access port.

Use the standard *SWD* Arm CoreSight *DAP* protocol to check if *APPROTECT* is enabled.

1. Select or connect to the control access port.

   This access port is at index 0x04 (See nRF9160 DAP - Debug access port).
2. Read the APPROTECT.STATUS register (0x00C) of the *CTRL-AP*.

   If the bit at position 1 of this register is `0`, *APPROTECT* is enabled.

   If the bit at position 1 of this register is `1`, *APPROTECT* is disabled.
3. Read the ERASEPROTECT.STATUS (0x018) of the *CTRL-AP*.

   If the least significant bit of this register is `0`, *ERASEPROTECT* is enabled.

   If *ERASEPROTECT* is set, it can be disabled if the installed firmware and the debugger both write the same non-zero 32-bit KEY value to ERASPROTECT.DISABLE (0x01C).

   > **Note:** If *APPROTECT* and *ERASEPROTECT* are enabled and the device does not have firmware that allows disabling *ERASEPROTECT*, the device is locked and cannot be recovered.
4. Go to Erase all on page 18.

   If the least significant bit of this register is `1`, access port protection is disabled. Go to Halting CPU on page 16.

8.1.1 APPROTECT and ERASEPROTECT are enabled

If both *APPROTECT* and *ERASEPROTECT* are enabled, access port 0 and the ERASEALL functionality are unavailable.

To unlock the device, it must have compatible firmware that provides a 32-bit non-zero KEY value to ERASEPROTECT.DISABLE. When both the debugger and firmware provide the same 32-bit non-zero KEY value to ERASEPROTECT.DISABLE, the device does a *CTRL-AP* erase all operation. The access port is re-enabled on the next reset once the erase sequence is done.

8.1.2 Only APPROTECT is enabled

If *APPROTECT* is enabled on the device, access port `0` is unavailable.

The only way to reopen or unlock the device is to issue an ERASEALL command through the *CTRL-AP* access port and then issue a pin reset. This will erase the entire code flash, the *UICR* area of the device, and the entire RAM. This method of erasing is slower than performing an *NVMC* erase all since it also must erase all RAM, but if *APPROTECT* is enabled, it is the only way to unlock the device.

8.1.2.1 Erasing all through CTRL-AP

Use the standard *SWD* Arm CoreSight *DAP* protocol to erase all while *CTRL-AP* is still selected by the debug port.

1. Write `0x00000001` to the ERASEALL register (0x004) of *CTRL-AP*.

   This will start the ERASEALL operation which erases all flash and RAM on the device.
2. Read the ERASEALLSTATUS register (0x008) of the *CTRL-AP* until the value read is `0x00` or wait 15 seconds after the ERASEALL write has expired.
3. Issue a pin reset.

8.1.2.2 Halting CPU

Use the standard *SWD* Arm CoreSight *DAP* protocol to issue a halt command to the chip.

An application running on the device that was previously programmed may use the *WDT*. In this case, the *WDT* is paused when the CPU is halted by default.

8.1.2.3 Reloading the watchdog timer

If the *WDT* is configured not to stop on a halt command, it must be reloaded periodically to prevent it from resetting the domain.

1. Read the WDT.RUNSTATUS register (0x50018400) to check if the *WDT* is running.

   If the least significant bit is `1`, the *WDT* is running.
2. Read the **HALT** field of the WDT.CONFIG (0x5001850C) register to check if the *WDT* halts with the CPU.

   If the fourth least significant bit is `1`, the *WDT* does not halt.
3. Reload the *WDT* by doing the following:
   a) Identify an enabled reload request in WDT.RREN (0x50018508).

      If value of the bit `n` in WDT.RREN is `1`, the reload request `n` in WDT.RR[n] (0x50018600 + (n × 0x4)) can be used to reload the watchdog counter to the value written in WDT.CRV (0x50018504).

   b) Reload the watchdog timer by writing `0x6E524635` to the WDT.RR[n] register.

8.1.2.4 Reading FICR

*FICR*s are pre-programmed in the factory and cannot be erased by the user. These registers contain chip-specific information and configuration.

Using the standard *SWD* Arm CoreSight *DAP* protocol:

1. Read the INFO.CODEPAGESIZE register (0x00FF0220) of the *FICR*.

   The value of this register contains the code memory page size in hexadecimal format, so `0x00001000` stored in this register corresponds to a page size of 4096 bytes.
2. Read the INFO.CODESIZE register (0x00FF0224) of the *FICR*.

   The value of this register contains the number of pages in code memory in hexadecimal format, so `0x0000100` stored in this register corresponds to 256 total pages in flash memory.

> **Note:** Total flash memory (in bytes) = INFO.CODEPAGESIZE * INFO.CODESIZE. This information is used later to determine the valid range of addresses to program.

8.1.3 APPROTECT is disabled

The *UICR*s have not been previously configured to enable access port protection.

If the device is in secure protection, you cannot read *FICR*s or disable the *SPU*. To disable *SECUREAPPROTECT* , see Erasing all through CTRL-AP on page 16.

In some cases, you may assume that the entire flash has already been erased. If the flash is already erased and the device has never been programmed before, go to Programming the application core on page 12.

8.1.3.1 Reading FICR

*FICR*s are pre-programmed in the factory and cannot be erased by the user. These registers contain chip-specific information and configuration.

Using the standard *SWD* Arm CoreSight *DAP* protocol:

1. Read the INFO.CODEPAGESIZE register (0x00FF0220) of the *FICR*.

   The value of this register contains the code memory page size in hexadecimal format, so `0x00001000` stored in this register corresponds to a page size of 4096 bytes.
2. Read the INFO.CODESIZE register (0x00FF0224) of the *FICR*.

   The value of this register contains the number of pages in code memory in hexadecimal format, so `0x0000100` stored in this register corresponds to 256 total pages in flash memory.

> **Note:** Total flash memory (in bytes) = INFO.CODEPAGESIZE * INFO.CODESIZE. This information is used later to determine the valid range of addresses to program.

8.1.3.2 Halting CPU

Use the standard *SWD* Arm CoreSight *DAP* protocol to issue a halt command to the chip.

An application running on the device that was previously programmed may use the *WDT*. In this case, the *WDT* is paused when the CPU is halted by default.

8.1.3.3 Reloading the watchdog timer

If the *WDT* is configured not to stop on a halt command, it must be reloaded periodically to prevent it from resetting the domain.

1. Read the WDT.RUNSTATUS register (0x50018400) to check if the *WDT* is running.

   If the least significant bit is `1`, the *WDT* is running.
2. Read the **HALT** field of the WDT.CONFIG (0x5001850C) register to check if the *WDT* halts with the CPU.

   If the fourth least significant bit is `1`, the *WDT* does not halt.
3. Reload the *WDT* by doing the following:
   a) Identify an enabled reload request in WDT.RREN (0x50018508).

      If value of the bit `n` in WDT.RREN is `1`, the reload request `n` in WDT.RR[n] (0x50018600 + (n × 0x4)) can be used to reload the watchdog counter to the value written in WDT.CRV (0x50018504).

   b) Reload the watchdog timer by writing `0x6E524635` to the WDT.RR[n] register.

8.1.3.4 Disabling SPU

The *SPU* protects memories from illegal erases, writes, and accesses.

An attempt to erase, write, or access a protected memory region leads to a secure fault. System protection can be turned off in the debug mode by configuring the relevant FLASHREGION[n].PERM for flash and RAMREGION[n].PERM for RAM.

To disable the *SPU*, the processor must be reset and stopped in the reset vector using the Flash Patch and Breakpoint unit (FPB). This disables any *SPU* protection and prevents any firmware from re-enabling the *SPU*.

See nRF9160 SPU - System protection unit for more information.

8.1.3.5 Check flash region security attribute

For every flash page, there is a security attribute that controls if that region is secure or non-secure.

Check the corresponding SPU.FLASHREGION[n].PERM (0x5003600 + (n*4)), where n is the flash page. If the 4th bit is `0`, the flash region is in non-secure, otherwise it is secure.

8.2 Erasing all or erasing pages

An erase all operation takes the same amount of time as erasing two pages one by one. As there are 256 total pages of flash, erasing all is more efficient than erasing page by page. If a region of the chip has been preprogrammed, you can erase the flash you intend to program page by page and then write those addresses with data leaving pre-programmed flash untouched. If the value of all flash addresses is `0xFFFFFFFF`, skip this procedure.

8.2.1 Erase all

Use the standard *SWD* Arm CoreSight *DAP* protocol to erase all.

1. If *SECUREAPPROTECT* is enabled, erase all is disabled. To erase, see either Erasing page by page on page 18 or Erasing all through CTRL-AP on page 16 to disable *SECUREAPPROTECT*.
2. Write `0x00000002` to the CONFIG register (0x50039504) of the *NVMC*.

   This configures the *NVM* for erasing.
3. Read the READY register (0x50039400) of the *NVMC* until the value is `0x00000001`.

   When this value is read, the *NVMC* is ready and not currently performing any operations.
4. Write `0x00000001` to the ERASEALL register (0x5003950C) of the *NVMC*.

   This erases all *NVM* including *UICR* registers.
5. Read the READY register (0x50039400) of the *NVMC* until the value is `0x00000001` before continuing to ensure the erase all operation has completed.
6. Write `0x00000000` to the CONFIG register (0x50039504) of the *NVMC*.

   This configures the *NVM* back to read-only.

8.2.2 Erasing page by page

Use the standard *SWD* Arm CoreSight *DAP* protocol to erase page by page.

8.2.2.1 SECUREAPPROTECT disabled

Use this procedure to erase page by page if *SECUREAPPROTECT* is disabled.

Before you begin, check *SPU* if the flash page is secure or non-secure (see Check flash region security attribute on page 17).

1. Write `0x00000002` to the CONFIG register (0x50039504) of the *NVMC*.

   This configures the *NVM* for erasing.
2. Read the READY register (0x50039400) of the *NVMC* until the value is `0x00000001`.

   When this value is read, the *NVMC* is ready and not currently performing any operations.
3. Write `0xFFFFFFFF` to the first 32-bit word in the flash page you want to be erased.
4. Read the READY register (0x50039400) of the *NVMC* until the value is `0x00000001`.
5. Repeat steps 3 and 4 until all wanted pages are erased.
6. Write `0x00000000` to the CONFIG register (0x50039504) of the *NVMC*.

   This configures the *NVM* back to read-only.

> **Note:** The *UICR* flash page cannot be erased by using erase page. It can only be erased by an erase all operation.

8.2.2.2 SECUREAPPROTECT enabled

Use this procedure to erase page by page if *SECUREAPPROTECT* is enabled.

1. Write `0x00000002` to the CONFIGNS register (0x40039584) of the *NVMC*.

   This configures the *NVM* for erasing.
2. Read the READY register (0x40039400) of the *NVMC* until the value is `0x00000001`.

   When this value is read, the *NVMC* is ready and not currently performing any operations.
3. Write `0xFFFFFFFF` to the first 32-bit word in the flash page you want to be erased.
4. Read the READY register (0x40039400) of the *NVMC* until the value is `0x00000001`.
5. Repeat steps 3 and 4 until all wanted pages are erased.
6. Write `0x00000000` to the CONFIGNS register (0x40039584) of the *NVMC*.

   This configures the *NVM* back to read-only.

8.3 Writing data - SECUREAPPROTECT disabled

Use the standard *SWD* Arm CoreSight *DAP* protocol to write data into flash.

1. Check the corresponding SPU.FLASHREGION[n].PERM (0x5003600 + (n*4)), where n is the flash page.

   If the 4th bit is `0`, the flash region is non-secure, otherwise it is secure. For non-secure regions, the CONFIG register in the following steps needs to be replaced with CONFIGNS (0x50039584).
2. Write `0x00000001` to the CONFIG register (0x50039504) of the *NVMC*.
   This enables writing to the *NVM*.
3. Read the READY register (0x50039400) of the *NVMC* until the value is `0x00000001`.

   When this value is read, the *NVMC* is ready and not currently performing any operations.
4. Write the data to the desired, word-aligned address.
5. Read the READY register (0x50039400) of the *NVMC* until the value is `0x00000001` before continuing to ensure the write operation has completed.
6. Continue writing and then reading the READY register (0x50039400) as necessary.
7. Write `0x00000000` to the CONFIG register (0x50039504) of the *NVMC*.
   This configures the *NVM* as read-only.

The ranges of writeable addresses are:

- *UICR* addresses (located in addresses 0x00FF8000 through0xFF8FFC)
- All program flash (located in addresses 0x00000000 through ((INFO.CODESIZE * INFO.CODEPAGESIZE) – 0x4)

Different methods can be used to write flash. For nRF9160, a good flash algorithm should take around 10 seconds to write the entire flash. If you cannot achieve this time, contact Nordic Semiconductor for assistance.

8.4 Writing data - SECUREAPPROTECT enabled

Use the standard *SWD* Arm CoreSight *DAP* protocol to write data into flash.

1. Check the corresponding SPU.FLASHREGION[n].PERM (0x4003600 + (n*4)), where n is the flash page.

   If the 4th bit is '0', the flash region is in non-secure, otherwise it is secure. With *SECUREAPPROTECT* enabled, secure flash regions cannot be written to.
2. Write the value 0x00000001 to the CONFIGNS register (0x400395084) of the *NVMC*.

   This configures the *NVM* for writing.

3. Read the READY register (0x40039400) of the *NVMC* until the value is 0x00000001.

   When this value is read, the *NVMC* is ready and not currently performing any operations.
4. Write the data to the desired, word-aligned address.
5. Read the READY register (0x40039400) of the *NVMC* until the value is 0x00000001 before continuing to ensure the write operation has completed.
6. Continue writing and then reading the READY register (0x40039400) as necessary.
7. Write the value 0x00000000 to the CONFIGNS register (0x40039584) of the *NVMC*.

   This configures the *NVM* back to read-only.

The range of writeable addresses are all program flash that is configured to be non-secure. The size and addresses of the non-secure flash are dependent on earlier programmed firmware.

Different methods can be used to write flash. For nRF9160, a good flash algorithm should take around 10 seconds to write the entire flash. If you cannot achieve this time, contact Nordic Semiconductor for assistance.

Glossary

Access Port Protection (APPROTECT)

A register used to prevent read and write access to all CPU registers and memory-mapped addresses.

Control Access Port (CTRL-AP)

A custom access port that enables control of the device even if other access ports in the debug access port are disabled by the access port protection.

Debug Access Port (DAP)

Provides multiple master driving ports, all accessible and controlled through a single external interface port to provide system-wide debug.

Device Firmware Update (DFU)

A mechanism for upgrading the firmware of a device.

Erase Protection (ERASEPROTECT)

A register used to block NVMC ERASEALL and CTRL-AP.ERASEALL functionality.

Factory Information Configuration Registers (FICR)

Pre-programmed registers that contain chip-specific information and configuration. FICRs cannot be erased by users.

Non-volatile Memory (NVM)

Memory that can retrieve stored information even after having been power-cycled.

Non-volatile Memory Controller (NVMC)

A controller used for writing and erasing the internal flash memory and the *UICR*.

Secure Access Port Protection (SECUREAPPROTECT)

A register used to prevent read and write access to all secure CPU registers and secure memory-mapped addresses.

System on Chip (SoC)

A microchip that integrates all the necessary electronic circuits and components of a computer or other electronic systems on a single integrated circuit.

System Protection Unit (SPU)

The central point in the system that controls access to memories, peripherals, and other resources.

Serial Wire Debug (SWD)

A standard two-wire interface for programming and debugging Arm CPUs

User Information Configuration Registers (UICR)

Non-volatile memory registers used to configure user-specific settings.

watchdog timer (WDT)

A timer that causes a system reset if it is not poked periodically.

Acronyms and abbreviations

These acronyms and abbreviations are used in this document.

APPROTECT
Access Port Protection

CTRL-AP
Control Access Port

DAP
Debug Access Port

DFU
Device Firmware Update

ERASEPROTECT
Erase Protection

FICR
Factory Information Configuration Registers

NVM
Non-volatile Memory

NVMC
Non-volatile Memory Controller

SECUREAPPROTECT
Secure Access Port Protection

SoC
System on Chip

SPU
System Protection Unit

SWD
Serial Wire Debug

UICR
User Information Configuration Registers

WDT
watchdog timer

Legal notices

By using this documentation you agree to our terms and conditions of use. Nordic Semiconductor may change these terms and conditions at any time without notice.

Liability disclaimer

Nordic Semiconductor ASA reserves the right to make changes without further notice to the product to improve reliability, function, or design. Nordic Semiconductor ASA does not assume any liability arising out of the application or use of any product or circuits described herein.

Nordic Semiconductor ASA 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. If there are any discrepancies, ambiguities or conflicts in Nordic Semiconductor's documentation, the Product Specification prevails.

Nordic Semiconductor ASA reserves the right to make corrections, enhancements, and other changes to this document without notice.

Life support applications

Nordic Semiconductor products are not designed for use in life support appliances, devices, or systems where malfunction of these products can reasonably be expected to result in personal injury.

Nordic Semiconductor ASA customers using or selling these products for use in such applications do so at their own risk and agree to fully indemnify Nordic Semiconductor ASA for any damages resulting from such improper use or sale.

RoHS and REACH statement

Complete hazardous substance reports, material composition reports and latest version of Nordic's REACH statement can be found on our website www.nordicsemi.com.

Trademarks

All trademarks, service marks, trade names, product names, and logos appearing in this documentation are the property of their respective owners.

Copyright notice

COMPANY WITH
QUALITY SYSTEM
CERTIFIED BY DNV GL
= ISO 9001 =