Freescale Semiconductor
Application Note

Document Number: AN5112
Rev. 0, 4/2015

Using the Kinetis Flash Execute-Only Access Control Feature

Contents

1 Introduction

Some new Kinetis family devices include a feature that adds access controls to the flash memory. Flash access controls (FAC) are a configurable memory protection scheme designed to allow end users to utilize software libraries while offering programmable restrictions to these libraries. This allows Freescale or third-party vendors to pre-program software libraries into a chip and distribute parts to end customers who can use the pre-programmed software libraries. The software is distributed inside the chip, but does not provide end customers the capability of reading the code from the device.

The FAC feature can be used to mark segments of the on-chip flash memory as execute-only and/or supervisor/privileged-only access. This document discusses usage of the FAC feature to create execute-only regions in the flash, limitations of the implementation, and recommendations to ensure limitations are not exploited.

2 How flash access controls work

The flash access control feature adds new registers to the flash module. There are eight 8-bit XACC registers that define which program flash segments are execute-only, and eight 8-bit SACC registers that define which program flash segments

are supervisor-only. Each bit in the registers corresponds to one segment of memory that can be protected. By default, all XACC and SACC registers are 0xFF. When a bit in the XACC or SACC is cleared, it marks the corresponding flash segment as execute-only or supervisor-only.

The size of flash memory in the device determines the number of access-controlled segments. For devices with 128 KB or less of program flash, 32 equal sized segments are used. For devices with more than 128 KB of program flash, 64 equal sized segments are used. Read the FTFA_FACSS and FTFA_FACSN to confirm the number and size of the segments implemented on a particular device.

Devices with 32 segments only use the FTFA_XACCL[3:0] registers (FTFA_XACCHn values are ignored). Devices with 64 segments use all eight of the registers—both FTFA_XACCH[3:0] and FTFA_XACCL[3:0]. The eight XACC registers are referred to as FTFA_XACCn throughout the rest of this document.

The flash memory controller (FMC) performs a cycle-by-cycle evaluation of access rights for each transaction routed to the on-chip flash memory. During the address phase of every attempted flash transfer, the supervisor access (FTFA_SACCn) and execute access (FTFA_XACCn) registers are examined to either allow or deny access. If a data access to an execute-only region is attempted, the access is aborted and terminates with a bus error. The read data is also zeroed.

3 Marking a segment as execute-only

The FTFA_XACCn registers are read only. The registers get their values from locations in the program flash (P-Flash) IFR space. There are two 64-bit locations in the P-Flash IFR--XACCA and XACCB. At reset, the XACCA and XACCB locations are read from the IFR, and the logical AND of XACCA and XACCB is used to load the FTFA_XACCn registers. The Program Once flash command can be used to write the XACC1 and XACC2 values in the P-Flash IFR. Refer to the device reference manual for the program once indexes for XACCA and XACCB.

Because the XACCA and XACCB locations are ANDed, XACCB can be used to add protection for additional segments even after XACCA has been programmed. A segment's access controls can be changed from data read and execute (XAn =1) to execute-only (XAn =0). Having two XACC IFR locations supports two levels of vendors adding their proprietary software libraries to a device.

In order to mark a segment as execute-only, follow these steps:

1. Program and verify library code in areas that will be marked as execute-only.
2. Use the Program Once flash command to write the XACCA value in the P-Flash IFR to mark the segment or segments where the library code resides as execute-only.
3. If a second library also needs to be protected, deliver parts to the owner of the second library to program and verify the second library.
4. Use the Program Once flash command to write the XACCB value in the P-Flash IFR to mark the segment or segments where the second library resides as execute-only.
5. If there is not a second library, program the XACCB with the same value used for XACCA.

NOTE

When marking segments as execute-only, it is important to program and verify the code before configuring the XACCA or XACCB program once values. If you program the XACCA or XACCB first, at next reset, the segment becomes execute-only and you will not be able to program the region without performing an Erase All Blocks command to unlock the execute-only regions for programming.

The figure below shows an example where XACCA and XACCB are used to protect two code libraries in a device with 512 KB of flash.

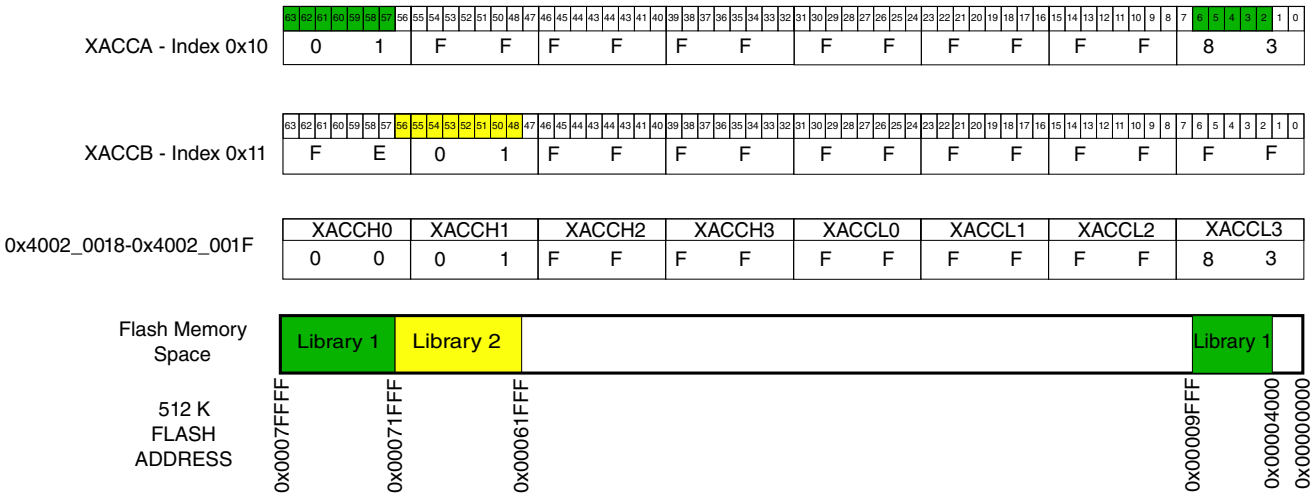

Figure 1. XACCx Protection Example

4 Other effects of marking a region as execute-only

The main effect of marking a region as execute-only is, with the exception of PC-relative loads for constants, the region cannot be read as data using the core, debug, or any of the other masters on the chip. Marking a region as execute-only has an impact on FMC cache behavior and some flash command operations as well.

4.1 FMC cache

Normally, contents of the FMC cache are visible in the FMC register space in a number of FMC tag and data registers (exact number of registers can vary from device to device). If any segment is marked as execute-only, the FMC cache is hidden from the user. The tag registers become read only and cannot be written, and the data registers cannot be read or written. Writes to the tag and data arrays are ignored and reads of the data array return 0's.

4.2 Flash command impact

The FAC functionality is intended to be used with a software library programmed into a device and never changed after the code is marked as execute-only. For this reason, any code distributed as a pre-programmed library in execute-only areas of flash should endure rigorous testing before distribution.

If a situation arises where the code in an execute-only region needs to be updated, it is possible, but there are restrictions on flash programming and erase operations in execute-only regions that need to be taken into account.

4.2.1 Program commands (program longword/phrase/section/check)

If the targeted flash location is in an execute-only protected segment, program commands are not allowed unless a Read 1s All Blocks or Erase All Blocks command completes and returns with a pass code (which means the part is fully erased). After a Read 1s All Blocks or Erase All Blocks command completes and returns a pass code, execute-only segments are opened to program commands, including the program check command. This means that after a library has been programmed and

marked as execute-only, the only way to update that execute-only space is to erase the entire chip. Attempts to program or program check in a protected segment when not open to program commands set the flash protection violation error flag (FTFA_FSTAT[FPVIOL]).

In order to re-lock execute only segments after they've been updated, reset the device or execute a Read 1s All Blocks command again. The Read 1s All Blocks would be expected to fail at this point (chip is not fully erased). This closes off the execute-only segments from programming.

4.2.2 Erase commands (erase block/sector)

If the selected flash block or sector contains an execute-only protected segment, the Erase Block and Erase Sector commands are not allowed unless a Read 1s All Blocks or Erase All Blocks command completes and returns with a pass code (which means the part is fully erased). After the Read 1s All Blocks command completes and returns with a pass code, then the execute-only segment is open to the Erase Block and Erase Sector commands. This means block and sector erase commands cannot be used to erase just execute-only segments. The whole chip must be erased to modify execute-only segments. After the entire chip is erased, execute-only segments can be reprogrammed. While the execute-only area is open to program commands, it is also open to erase commands. However, the chip must be erased to get to this point anyway, so in most cases Erase Block and Erase Sector are not useful if execute-only segments need to be modified. Attempts to run Erase Block or Erase Sector in a protected segment when not open to the erase commands sets the flash protection violation error flag (FTFA_FSTAT[FPVIOL]).

The process for closing off the execute-only space for erase commands is the same used to close off the execute only segments from programming. To close off execute-only space from erasing, the device must be reset or a Read 1s All Blocks command must complete with a fail result (device is no longer fully erased).

WARNING

The Erase All Blocks command is always allowed and erases segments even if they are marked as execute-only. End user code should never use the Erase All Blocks command on a device with one or more pre-programmed libraries that are execute-only. If the Erase All Blocks is used, the pre-programmed library code will be lost.

5 Reasons for the inability to change an execute-only segment back to data and code access

After a segment is marked as execute-only, there is no way to put it back to code and data access. The P-Flash IFR locations used to load the XACCn registers are program ONCE. After XACCA and XACCB are programmed, they cannot be modified. No flash command can be used to erase them, including the Erase All Blocks command or mass erase. After XACC1 and XACC2 are programmed, they cannot be reprogrammed to add more execute-only segments either. Attempts to use the Program Once command to write an IFR location that already has a non-FFFF value return an access error (FTFA_FSTAT[ACCERR] = 1).

6 Special considerations and limitations of flash access controls

Because the ARM® Cortex® M4 and M0+ cores do not have awareness of the flash access control, there are some important limitations to the functionality that need to be taken into consideration. Software written for a system that is using the flash access control needs some special considerations to ensure correct operation and protection of the execute-only software.

6.1 Debugging

The most significant limitation on the FAC functionality is that the debug port was not designed to take this functionality into account. Flash segments marked as execute-only cannot be read directly by a debugger. However, the debugger could still serve as a backdoor to allow someone to reverse engineer code, even if the code is not directly readable.

In order to ensure execute-only code is truly safe from being reconstructed, Freescale recommends:

- Enabling the flash security feature to disable debug access to the device (FTFA_FSEC[SEC]).
- Optionally using the FTFA_FPROT*n* to explicitly protect execute-only code areas.
- Optionally disable the mass erase capability to prevent accidental erasure of the code stored in execute-only segments through a debugger (FTFA_FSEC[MEEN]).

6.1.1 How flash security and protection features interact with FAC

As described in the flash command impact sections, execute-only regions have some protection from accidental erasure and modification. However, if an end customer used an Erase All Blocks command, the code in execute-only regions could still be lost. The Erase All Blocks command aborts if any regions of the flash are protected by FTFA_FPROT*n*. Explicitly marking execute-only regions as protected using FTFA_FPROT*n* can be used to make it more difficult to accidentally erase or modify execute-only regions. In order to reprogram execute-only regions, the flash configuration field containing the FTFA_FPROT*n* settings would need to be erased and reprogrammed with the protection disabled, then the part must be reset for the new protection settings to take effect. After the chip has booted with no regions protected, the Erase All Blocks command could be used to erase the whole chip including execute-only regions.

NOTE

The size of the region protected by FPROT bits and the size of the execute-only segments are not always equal.

NOTE

Freescale recommends NOT protecting the first region of the flash using FTFA_FPROT*n*. If the first region is protected, the only way to modify the protection scheme later is using a mass erase from the debugger.

The mass erase command run from a debugger is not impacted by the FTFA_FPROT*n* settings. When security is enabled, the FTFA_FSEC[MEEN] field can be used to disable the debugger mass erase. Enabling securty and disabling mass erase can be used to prevent accidental erasure of the chip using a debugger. To erase or program the MCU through the debugger, the processor would need to be unsecured using the Backdoor Key Access.

All of these controls bits are located in flash locations 0x0400 - 0x040F. Sector erasing and reprogramming these protections is possible if access is gained through the debug port or bootloader. Refer to *Using Kinetis Security and Protection Features* (document AN4507) for more information.

6.2 Execute-only code is visible to other execute-only segments

The ARM instruction set architecture relies heavily on PC-relative loads. These operations, which look like ordinary data reads, must be allowed to execute-only spaces. The flash access control logic analyzes all accesses to flash memory and allows only instruction fetches and PC-relative loads originating from execute-only code to access execute-only regions. The logic treats all execute-only regions equally; therefore, any segment marked as execute-only can access any other execute-only segment using PC-relative accesses.

For single pre-loads where only one set of software libraries is being loaded into a part, both levels of access control must be programmed to protect the pre-loaded code. This can be done by writing XACCA with the same value as XACCB. This prevents the end user from loading new code onto the chip, marking it as execute-only, and using that code to read out code from the original execute-only regions.

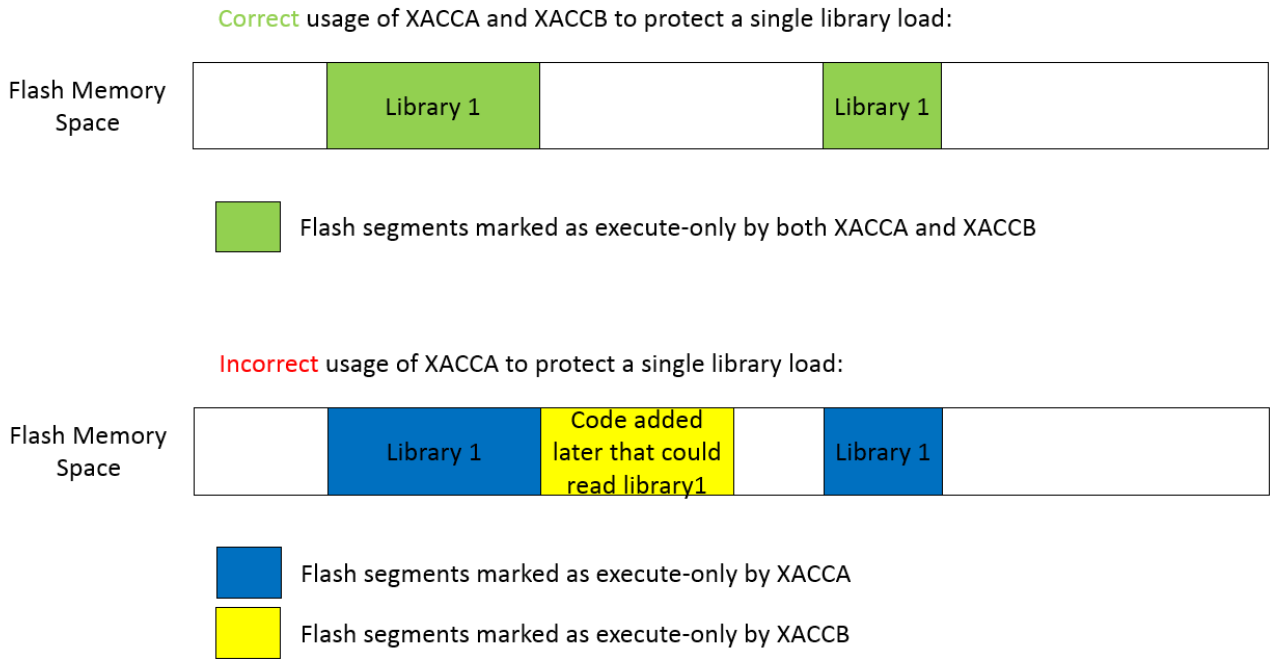

Figure 2. Protecting a Single Library

If two pre-loads of libraries are going to be used, then when the second library is loaded, the first library could be seen. For example, company A loads one library into the device and programs XACC1 to mark segments as execute-only, and sends the part to company B. Company B loads a second software library into the part and writes XACC2 to mark the segments for the second library as execute-only. Company B must be a trusted entity because they could access company A's software library. Cooperation should already be established between the companies to determine who secures the device and provides some system-level functionality. NDAs and legal agreements to protect the software libraries should be put in place to protect both companies while working together.

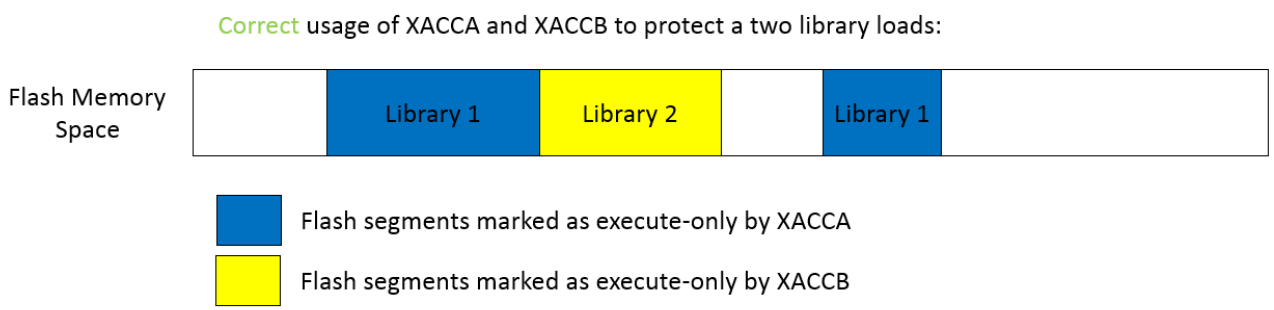

Figure 3. Protecting Two Libraries

6.3 Entry into execute-only code on the ARM Cortex-M4 core

There is not a determinate method of detecting PC-relative loads on the ARM Cortex-M4 core. In order to keep execute-only space protected, a delay happens during transitions from nonexecute-only to execute-only space before state information shows the core is in execute-only space. During the delay period, PC-relative loads from the execute-only space are not allowed. If a PC-relative data load from an execute-only segment happens during this time, it triggers an access error (the same as if the execute-only segment were read from a code and data flash region).

In order to avoid unwanted access errors, the entry point into execute-only code should be controlled:

- Execute-only functions that can be called from non-exeute-only code should not include a PC-relative load in the first six instructions. The simplest way to implement this is to add six NOP instructions at the beginning of execute-only functions. It is important to ensure the compiler does not optimize out these NOP instructions.
- Execute-only functions that are only called from other execute-only functions (functions that are not exposed in the library's API) do not need a special entry sequence.
- All interrupts should be routed through a special piece of execute-only code. If a single default handler is going to be used for all interrupts, the default handler should start with six NOP instructions (or other instructions that do not perform any PC-relative loads from execute-only space). After the safe entry to execute-only space, the handler can jump to the specific handler code for the associated interrupt. When the interrupt specific handler returns to the default handler, another six NOPs (or other non PC-relative load instructions) should be executed. At this point the default handler has met the requirements for entry into execute-only space, so it can return normally. This way if the interrupted code is in execute-only space, the interrupt handler follows the same entry point rules as normal execute-only function entry points.

NOTE

The ARM Cortex-M0+ architecture allows for a determinate method of detecting PC-relative loads. On Kinetis devices with an ARM Cortex-M0+ core and flash access control functionality, the entry points into execute-only code do NOT need to be controlled, so the information in this section does not apply for Cortex-M0+ processors.

6.4 Vector table on ARM Cortex-M4 devices

On devices with a ARM Cortex-M4 core, if a default handler is being used for all interrupt vectors, the CPU's vector table can be left in flash memory and there is no need to modify it at run time to add customer interrupts. Instead, a separate handler jump table needs to be used to keep track of the interrupt specific handler the default handler needs to call. Implementation can vary, but in most cases there would be a default version of the jump table stored in flash. At startup, the table can be copied to RAM so that additional interrupt handlers can be registered while the application is running.

NOTE

To allow the core to fetch vectors from the vector table, the first segment of flash should NOT be marked as execute-only.

6.5 Flash programming

If debugger access is disabled, end customers are unable to use EzPort or a debugger to program their own code onto the device. Instead, flash programming capabilities need to be provided and pre-programmed into the part (the flash programming capability can also be in an execute-only area). The Kinetis bootloader software can be included to allow customers to program their own code into the part. See www.freescale.com/kboot for more information on the Kinetis bootloader.

The flash programming/bootloader code should be configured to execute by default (the initial PC value in the vector table should point to the flash programming/bootloader routines). This is required to allow the end customer to load their code on a new device they have not touched before. It is recommended to leave the flash programming/bootloader code on the device (do not allow end user to overwrite the flash programming software). This means the bootloader code runs by default. The

bootloader can use a timeout, sample a pin, or use another mechanism to determine when it should attempt to branch to the user code. In order for the bootloader to branch to the user code, there should be a predefined start address for the user code configured in the bootloader and in the end user application.

7 References

For additional information, refer to the following:

- Reference manual for your specific Kinetis family device: www.freescale.com/kinetis
- Kinetis bootloader: www.freescale.com/kboot
- *Using Kinetis Security and Protection Features* (document AN4507)

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Document Number AN5112
Revision 0, 4/2015