Freescale Semiconductor
Application Note

Document Number:AN4835
Rev 1, 05/2014

Production Flash Programming Best Practices for Kinetis K- and L-series MCUs

by: Melissa Hunter

Contents

1 Introduction

This application note describes recommended steps for production flash programming on Kinetis K-series and L-series devices. There are different options available for the physical interface used for communicating with the microcontroller to program the flash and also multiple commands that can be used to modify the flash contents.

This application note will discuss the different physical interfaces, flash commands, and provide recommended sequences to minimize issues when programming the flash and also decrease flash programming times. The target audience is anyone developing production flash programming hardware, production flash programming software, or a debugger interface that will support flash programming. Although some of the information in this document might be useful for software-controlled field firmware updates, that topic is not explicitly covered.

2 Kinetis flash types

Kinetis K- and L-series devices all use the Freescale Thin Film Storage (TFS) flash; however, there are different types of TFS flash used on different devices. Depending on the particular Kinetis device, the flash could be an FTFA, FTFL, or FTFE module.

The table below shows the differences between the flash modules.

Table 1. Comparison of Flash Module Types

	FTFA	FTFL	FTFE
Found on devices	Kinetis L-series	Kinetis K-series processors with 512KB of flash or less	Kinetis K-series processors with 512KB of flash or more
Program commands	Program longword (32-bit)	Program section and program longword (32-bit)	Program section and program phrase (64-bit)
EEE supported?	No	Yes, on some devices	Yes, on some devices
Swap supported?	No	Yes, on some devices	Yes, on some devices
Flash bus interface	Handshake—MCM_PLACR[ESFC] bit can be used to automatically stall flash reads while the flash is busy.	Wait states—Attempting to read a flash block while it is busy will cause an error. Blocks that are not busy can be read while other blocks are processing commands.	Wait states—Attempting to read a flash block while it is busy will cause an error. Blocks that are not busy can be read while other blocks are processing commands.

3 Programming interfaces

There are two main physical interfaces that can be used for production flash programming on Kinetis—SWD/JTAG and EzPort. On devices that support both options, the SWD/JTAG pins and EzPort pins are multiplexed on the processor. The state of the EZP_CS signal is sampled at reset to determine whether the SWD/JTAG pin functions or the EzPort pin functions are enabled on the pins. If EZP_CS is high at reset, then SWD/JTAG functions are enabled on the pins. If EZP_CS is low at reset, then EzPORT pin functions are enabled.

3.1 SWD/JTAG (Serial Wire Debug/Joint Test Action Group)

Kinetis K-series devices support both SWD and JTAG interfaces to the ARM debug access port (DAP). Kinetis L-series devices only support the SWD interface.

SWD and JTAG are different physical interfaces, but the commands and register accesses that would be used for programming the flash are the same. For simplicity the term SWD will be used throughout the remainder of this document, but the operations described can be used over SWD or JTAG.

SWD is the primary debug interface for the processor. In addition to debugging, SWD can be used for flash programming. Typically a flash programming algorithm with software supporting the desired flash commands would be loaded into the microcontroller's on-chip RAM over SWD, then the SWD would be used for run control of the flash erasing and programming code execution from the RAM.

SWD could also be used to write to the flash registers directly to configure them for the desired commands. Sending in SWD commands is slower than executing code natively on the core, so this approach is not normally recommended as the flash programming times will usually be significantly longer if the SWD directly writes the flash FCCOB registers.

3.2 EzPort

EzPort is a serial flash programming interface found on Kinetis K-series microcontrollers. The only function of EzPort is to interface to the flash. The processor is automatically halted in EzPort mode, so code cannot be executed while EzPort is active.

There is an EzPort command that allows for direct interaction with the flash FCCOB registers, but for the most commonly used flash functions, the EzPort implements commands that manage the FCCOB registers automatically.

For full details on the EzPort and the supported EzPort commands, refer to the reference manual for the specific Kinetis K-series device.

4 Programming steps

The following sections describe high level procedures for programming flash.

4.1 Connecting

The first step to program the flash is to establish a connection over the desired interface, either SWD or EzPort.

4.1.1 SWD connection steps

The recommended procedure for establishing a connection to the processor over SWD is listed below.

1. On devices that support EzPort, make sure that EZP_CS is high, so that SWD/JTAG pin functions are enabled.
2. Power on the processor, or if power has already been applied, assert the $\overline{\text{RESET}}$ pin to reset the processor. For devices that do not have a $\overline{\text{RESET}}$ pin, write the System Reset Request bit in the MDM-AP control register after establishing communication.
3. Keep reset low and establish communication with the ARM DAP. The MDM-AP ID register can be read to verify that the connection is working correctly.
4. Read the MDM-AP status register until the Flash Ready bit sets.
5. Read the System Security bit to determine if security is enabled. If System Security = 0, then proceed. If System Security = 1, then communication with the internals of the processor, including the flash, will not be possible without issuing a mass erase command or unsecuring the part through other means (backdoor key unlock).
6. Write the MDM-AP register to set the Debug Request bit. This will prevent the core from attempting to boot when the reset pin is released. NOTE: the Debug Request bit cannot be written if the processor is secured.
7. Negate the $\overline{\text{RESET}}$ signal or clear the System Reset Request bit in the MDM-AP control register.

When the steps above have been completed, debugging or flash programming can be started.

4.1.2 EzPort connection steps

The recommended procedure for establishing a connection to the processor over EzPort is listed below.

1. Make sure that EZP_CS is low, so that EzPort pin functions are enabled.
2. Power on the processor, or if power has already been applied, assert the $\overline{\text{RESET}}$ pin to reset the processor.
3. Release $\overline{\text{RESET}}$ and establish communication with the EzPort.
4. Send the Read Status Register command to read the Flash Security bit (FS). If Flash Security = 0, then proceed. If Flash Security = 1, then communication with the internals of the processor including the flash will not be possible without issuing a mass erase command or unsecuring the part through other means (backdoor key unlock).

When the steps above have been completed flash programming can be started.

4.2 Erasing flash

Before a flash sector can be programmed it must be erased. There are several different flash erase mechanisms that can be used:

- Mass erase (SWD or EzPort)
- Erase all blocks
- Erase flash block
- Erase flash sector

The following sections describe each of the commands and when they might be used by a production flash programmer.

4.2.1 Mass erase

Unlike the other erase options listed above, the mass erase is not a flash command executed using the FCCOB registers. Instead, the mass erase is requested directly by the SWD or EzPort.

Invoking the mass erase will cause the entire flash to be erased even if security is enabled or regions are protected. If security is enabled the mass erase function must also be enabled (FTFx_FSEC[MEEN] does not equal 0b10) or the mass erase will not be allowed. If allowed (mass erase is enabled), then the following steps are performed:

1. All blocks of flash are erased regardless of protection settings. This includes program flash, data flash, the program flash IFR swap indicator address, data flash IFR space (including EEE partition information), EEPROM backup memory (E-flash), and FlexRAM.
2. The flash is read to verify that the erase completed successfully.
3. If the erase verify fails, then the FTFx_FSTAT[MGSTAT0] bit is set and the process stops. If the erase verify passes, then the process continues.
4. The FTFx_FSEC[SEC] register field is set to 0b10 (unsecure). This releases security immediately.
5. The security byte in the flash configuration field is also programmed to 0xFE (unsecure). On subsequent resets, the processor will boot in unsecure mode unless the flash configuration field is modified.

4.2.1.1 SWD mass erase

The recommended procedure for performing a mass erase over the SWD interface is listed below.

1. Follow steps described in the SWD connection steps section.
2. Read the MDM-AP status register Mass Erase Enable bit to determine if the mass erase command is enabled. If Mass Erase Enable = 0, then mass erase is disabled and the processor cannot be erased or unsecured. If Mass Erase Enable = 1, then the mass erase command can be used.
3. Write the MDM-AP control register to set the Flash Mass Erase in Progress bit. This will start the mass erase process.
4. Read the MDM-AP control register until the Flash Mass Erase in Progress bit clears.
5. When the Flash Mass Erase bit clears, the mass erase has completed. As long as the mass erase verify portion of the mass erase completed successfully, security will be released.

It is important to ensure that the processor doesn't receive a new reset or execute any code that could interfere with the operation of the mass erase command while the mass erase is in progress. To avoid this, keep the processor in reset by keeping $\overline{\text{RESET}}$ asserted or keeping the System Reset Request set during the mass erase.

4.2.1.2 EzPort mass erase

The recommended procedure for performing a mass erase using the EzPort is listed below.

1. Follow steps described in the EzPort connection steps section.
2. Send the Read Status Register command to read the Bulk Erase Disable bit (BEDIS). If BEDIS = 0, bulk erase is enabled, so the mass erase can proceed. If BEDIS = 1, mass erase is disabled and the processor cannot be erased.

3. Issue the Flash Bulk Erase EzPort command.
4. Send the Read Status Register command until the write in progress (WIP) bit clears.
5. When the WIP bit clears, the mass erase has completed. As long as the mass erase verify portion of the mass erase completed successfully, security will be released.

It is important to ensure that the processor doesn't receive a new reset request while the mass erase command is in progress.

4.2.2 Erase all blocks

Erase all blocks is a flash command that performs a similar operation to the SWD/EzPort mass erase, but the operations are not identical. Like mass erase, the erase all blocks command will erase all of the user flash contents——that is, program flash, data flash, program flash IFR swap indicator address, data flash IFR (including EEPROM partition configuration), EEPROM backup memory (E-flash), and FlexRAM. There are two important differences between mass erase and the erase all blocks command:

- Unlike mass erase, erase all blocks takes protection settings into account. If any flash or FlexRAM regions are protected, the erase all blocks command will abort.
- If the verify completes, erase all blocks will release security by setting the FTFx_FSEC[SEC] field to unsecured, but the security byte in the flash configuration field is not programmed to 0xFE. On a subsequent reset the processor will be in the secured state unless FSEC[SEC] in the flash configuration field is reprogrammed.

4.2.3 Erase flash block

The erase flash block command can be used to erase an entire block of flash. The erase all blocks command can erase multiple blocks (if the device has more than one flash block), but the erase flash block command only erases a single block. If any of the regions within the block are protected, the erase aborts. Also, if EEPROM is partitioned and the block being erased is data flash, then the erase is aborted. The current security settings are unaffected by executing an erase flash block command, but if the flash configuration field is erased then on a subsequent reset the processor will be in the secured state unless the FSEC[SEC] setting in the flash configuration field is reprogrammed.

NOTE

Devices that contain a single flash block do not support the erase flash block command. If there is only one block of flash on a device, then the erase all blocks command provides the same functionality as the unimplemented erase flash block command.

4.2.4 Erase flash sector

As implied by the name, the erase flash sector command is used to erase an entire sector of flash. The erase flash sector command erases the smallest amount of memory possible for an erase. The size of a sector varies depending on the flash module used and the configuration of flash blocks on the device. Refer to the "Flash Memory Configuration" section of the Chip Configuration chapter in the reference manual for a Kinetis device to determine the sector size.

The erase flash sector command will abort if the selected sector is protected. If swap is being used, the sectors containing the swap indicators (one in each half of the program flash memory space) are implicitly protected unless the swap system is in the update mode and the sector being erased is in the non-active half.

4.2.5 Erase recommendations

While there are a number of options for erasing flash on Kinetis, most production flash programmers should only need to implement the mass erase and sector erase. The EzPort only has direct support for mass erase (bulk erase EzPort command) and sector erase. Other erase instructions could be implemented over EzPort by using the write FCCOB command, but this should not be required.

4.2.5.1 Mass erase recommendations

Mass erase is an important feature that allows for unsecuring of parts. All flash programmers should support execution of the mass erase request for the supported hardware interface (SWD or EzPort). If a device is detected as secured when connecting, then the user should be prompted to ask if they would like to mass erase the entire flash to release security. An option could be included to have the mass erase request issued automatically if a device is detected as secured, but Freescale recommends not mass erasing automatically by default. This way the user is alerted that there might be issues, and if there is already information in the flash that needs to be retained they have the option to abort the operation.

The mass erase causes the loss of all user information in the flash, so it is not recommended for default use by a flash programmer. If mass erase is used exclusively for erase procedures, then any type of multi-stage programming where user's program multiple files at different times could not be supported. Freescale recommends using sector erase by default, but having a user selectable option to use mass erase instead.

4.2.5.2 Sector erase recommendations

In order to avoid loss of pre-programmed information and configuration, production flash programmers should use the sector erase command by default. The preferred method is to execute an erase on a sector to ensure it is truly in the erased state, program the sector, then move to the next sector as needed. This way sectors are only erased if needed, and if there is an error in erasing upper sectors or programming any sectors that leads to an abort of the entire operation, then the minimum amount of flash has been modified.

4.3 Programming flash

After a flash sector (or the entire flash) has been erased, programming can start. There are three different flash programming commands that a device can support:

- Program section
- Program longword (32-bit)
- Program phrase (64-bit)

The program section command is only supported on some devices. Processors will support either the program longword or program phrase command, but not both. Refer to the flash module overview table for more information.

4.3.1 Program section

The program section command enables programming of a larger area of memory than the regular program longword and program phrase commands. A section is the smaller of the flash sector size or half the size of the FlexRAM/programming acceleration RAM for FTFL devices or one fourth the size of the FlexRAM/programming acceleration RAM for FTFE devices. The sector size and FlexRAM size can vary from device to device. Refer to the reference manual for the specific Kinetis device being used to determine sizes. Because the program section command enables programming of more memory, it is the most efficient means to program memory. This is the recommended program command to use for production programming when available.

To use the program section command through SWD, the data to be programmed is written into the FlexRAM/programming acceleration RAM (devices with FlexMemory refer to the RAM as FlexRAM, while devices with program flash only refer to the RAM as programming acceleration RAM). The FlexRAM must be configured for traditional RAM mode instead of EEPROM mode. After the data is loaded into the FlexRAM, the program section command can be executed.

The EzPort will automatically use the program section command for programming. The EzPort will configure the FlexRAM/ programming acceleration RAM for traditional RAM mode when entering EzPort mode. If the RAM mode is changed to EEPROM, then the RAM must be changed back to traditional RAM mode before sending a section program EzPort command or the command will not be accepted.

4.3.2 Program longword/phrase

The program longword or program phrase command can be used to program either 32-bits or 64-bits into the flash. Unlike the program section command, the data to be programmed is written directly into the flash FCCOBn registers, so these commands do not use the FlexRAM/programming acceleration RAM for programming.

FTFA devices are the only parts that don't support EzPort or the program section command. On FTFA parts, the program longword command is the only option for programming.

4.3.3 Programming recommendations

To increase programming speed, the program section command is the recommended programming method to use for flash production programming. On the Kinetis L-series, the program longword must be used instead. This can slow down flash programming time, but as the flash density available on Kinetis L-series devices is smaller than on the K-series parts, so overall flash programming times should still be reasonable.

When executing any flash command, always poll the appropriate status register waiting for completion of the command. Command execution times will be fairly uniform for new devices, but relying on fixed delay loops instead of the status can lead to problems. Instead, poll the flash FSTAT[CCIF] bit or EzPort status register's WIP bit to determine if the command has completed before moving to the next command.

5 Other programming considerations

When programming flash there are some special locations within the flash and some flash features that should be considered. The following sections provide information about flash addresses and features that should be given special treatment by a flash programmer.

5.1 Flash configuration field

The flash configuration field is a 16-byte section of flash memory located at 0x400-0x40F. Values from the flash configuration field are copied to flash registers at reset, so they set the default values for several flash settings, most importantly the flash protection and system security settings.

To prevent accidental securing of devices, the flash configuration field requires special treatment. Freescale recommends that by default flash programmers write the following value to the flash configuration field:

0xFFFF_FFFF

0xFFFF_FFFF

0xFFFF_FFFF

0xFFFF_FFFE

This value places the device in an unsecured state with no flash regions protected.

Some users will want to program the flash configuration field, so an override option should be included to enable users to program other values into the flash configuration field. Note that changes to the flash configuration field will take effect at the next reset. To prevent an updated flash configuration field value that secures the processor and prevents further access to the processor, ensure the programmer completes all programming steps and any verification of values written before resetting the processor.

5.2 Enhanced EEPROM support

Kinetis devices with FlexNVM include a FlexMemory feature. The FlexMemory enables users to configure some of the on-chip flash memory as enhanced EEPROM, additional flash memory, or a combination of the two.

By default a blank device configures all of the FlexNVM as flash memory. To enable customers to configure the EEPROM feature and possibly pre-program starting values to be stored in the EEPROM memory, the production flash programmer would need to include support to configure the EEPROM. EEPROM configuration can be supported over the SWD/JTAG or EzPort interface.

5.2.1 FlexMemory partitioning

To configure the amount of EEPROM memory that will be used and the amount of flash that will be used to backup the EEPROM, the flash must be partitioned. The flash program partition command (PGMPART) is used to setup the memory partitioning and configure the EEPROM for use. The EzPort doesn't have direct support for running the flash PGMPART command, so the EzPort's "Write FCCOB Registers" command would be used to execute the PGMPART command.

When the PGMPART command executes it will cause all of the FlexNVM to be erased (even sectors that will not be used for EEPROM backup). After the FlexNVM is erased, sectors used for EEPROM backup will be formatted for use. Because the FlexNVM is erased and parts of the FlexNVM are no longer accessible after partitioning, the EEPROM configuration should be the first step in programming a device if the EEPROM feature will be used. After the EEPROM is configured and optionally loaded, then code and data can be programmed into the remainder of the flash. Because the partitioning affects the amount of flash available for regular programming, it is a good idea to read and save the partition code from a device so that the user can be warned if they attempt to program flash addresses that are no longer available as traditional flash because of the device partitioning.

NOTE

Partitioning should only be done once. If the flash is re-partitioned for a different configuration, then record data is lost and you will not be guaranteed to get the expected endurance.

NOTE

Partitioning information, EEE data, and EEE location usage information will be lost if a device is mass erased. If the EEE has been used before the mass erase then this could impact the maximum endurance of the EEE.

5.2.2 Programming EEPROM

After the flash has been partitioned, initial EEPROM values can be programmed by writing to the FlexRAM memory space starting at 0x1400_0000. Using SWD this can be done with direct memory writes. The EzPort includes a "Write FlexRAM" command that can be used to write EEPROM values.

NOTE

The FlexRAM must be configured for EEPROM mode when attempting to program EEPROM data. Be sure to switch the FlexRAM mode back to traditional RAM mode before attempting to use the program section command.

5.3 Swap support

Some Kinetis devices include a swap feature that enables the memory base address of flash blocks to be exchanged. This enables users to have two code images that can be booted and enables the swap system to determine which image executes.

The swap feature is typically used for robust updates. The idea is to download the new software image to the inactive flash space without touching the software image that is presently running. If the new firmware downloads successfully, then the swap is completed and the next time the processor boots the new firmware runs. If any errors are detected in the new image it can be discarded and the current software is untouched and continues to boot.

Because the swap feature is typically used for firmware updates, systems that are using swap will usually have support for the swap system implemented within the firmware. This means that most production flash programmers will not usually need to have direct support for flash swap features. However, if swap is enabled on a device the flash swap indicator space is implicitly protected in most of the swap states, so if parts that have previously been configured for swap are being programmed, privilege violations can occur when attempting to erase the flash sectors containing the swap indicators. A mass erase might be required to clear out all swap configuration when reprogramming a device.

5.4 Parallelization

To incrase programming throughput, multiple Kinetis devices can be programmed in parallel using either SWD/JTAG or EzPort. When connecting to multiple devices simultaneously, control signals that are inputs to the processor can be shared, but individual pins from each processor should be monitored for output signals. This is required so that the programmer can monitor the completion of commands on each device individually. It is not safe to assume that multiple processors will complete running commands at exactly the same time, so the status for each processor needs to be tracked. Using unique pins for outputs from the processors is also needed to enable verification of flash contents after programming.

Table 2. Parallelization pin connections

Interface	Shared pins	Individual pins per processor
SWD	SWD_CLK, SWD_DIO (input), Reset	SWD_DIO (output)
JTAG	JTAG_TCLK, JTAG_TMS, JTAG_TDI, Reset	JTAG_TDO
EzPort	EZP_CK, EZP_CS, EZP_D, Reset	EZP_Q

6 Common problems

The list below describes problems that are known to cause issues with programming.

- Not protecting the flash configuration field to prevent accidental programming and/or intentionally programming the flash configuration field with an incorrect value (take into consideration endianness). If the flash configuration field is programmed with a value that enables security and disables mass erase and backdoor key accesses, then flash cannot be programmed.
- Interruption of an erase or program command can lead to corruption of the flash contents. Interruptions could include reset, loss of power, or a conflict with code running on processor.

- Not using the Debug Request bit to prevent code from executing while programming the flash. If the core attempts to run code when establishing a debug connection or modifying flash, then the code can interfere. If there is no code programmed (a blank part), then the processor will periodically reset due to core lockups. If there is code programmed in the device that modifies the power mode, clocking, or configuration of the SWD/JTAG pins, then that can prevent connection and/or cause corruption of the flash when a flash command is in process.
- Applying voltages to I/O pins when the processor is not powered. If the pin VDIO or VIO specifications are violated, the processor can attempt a partial power up and/or puts the flash into an undefined state. This can lead to corruption of flash contents, corruption of flash control logic, or corruption of device configuration and trim values which in turn can lead to the processor reporting as secured (locked device) or failure of the processor to respond to and complete flash commands.

7 Further reading

Additional documentation that may be useful includes the following:

- Kinetis Microcontroller documentation, software, and tools available at http://www.freescale.com/kinetis
- The Chip Configuration, Flash Memory Module, and EzPort chapters of the applicable device's reference manual
- *Using the Kinetis Security and Flash Protection Features* (document AN4507)
- *Using the Kinetis Family Enhanced EEPROM Functionality* (document AN4282)
- *Robust Over-the-Air Firmware Updates Using Program Flash Memory Swap on Kinetis Microcontrollers* (document AN4533)

8 Revision history

Table 3. Revision history

Revision number	Update description
0	Initial release
1	Changed references to core hold feature to Debug Request bit.

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support