AN12130

Production Flash Programming Best Practices for S32K1xx MCUs

Rev. 1 — September 2019 Application Note

by: NXP Semiconductors

Contents

1 Introduction

This application note describes recommended steps for production flash programming on S32K 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 Programming interfaces

There is one physical interface that can be used for production flash programming on S32K1xx devices: SWD/JTAG.

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

S32K14x devices support both SWD and JTAG interfaces to the ARM Debug Access Port (DAP). S32K11x devices only support the SWD interface (JTAG is only supported for Boundary Scan Register or BSR). 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 SWD would be used for running control of the flash erasing and programming code execution from the RAM. SWD can 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 be significantly longer if the SWD directly writes the flash FCCOB registers.

3 Programming steps

The following sections describe high level procedures for programming flash.

3.1 Connecting

The first step to program the flash is to establish a connection over the desired interface which is SWD.

3.1.1 SWD connection steps

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

1. Power on the processor, or if power has already been applied, assert the RESET pin to reset the processor. For devices that do not have a RESET pin, write the System Reset Request bit in the MDM-AP control register after establishing communication.
2. 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.
3. Read the MDM-AP status register until the Flash Memory Ready bit sets.
4. 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).
5. Write the MDM-AP control 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.

6. Negate the RESET signal or clear the System Reset Request bit in the MDM-AP control register.

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

3.2 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)
- 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.

3.2.1 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 programming interface.

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 (FTFC_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 FTFC_FSTAT[MGSTAT0] bit is set and the process stops. If the erase verify passes, then the process continues.
4. The FTFC_FSEC[SEC] register field is set to 0b10 (unsecure). This releases security immediately.
5. The Flash 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.

3.2.1.1 SWD mass erase

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

1. Follow steps described in SWD connection steps.

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 Memory Mass Erase in Progress bit clears.
5. When the Flash Memory 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 do not 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 RESET asserted or keeping the System Reset Request set during the mass erase.

3.2.2 Erase all blocks

Erase all blocks is a flash command that performs a similar operation to the SWD 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, 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 FTFC_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.

3.2.3 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 FTFC_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.

3.2.4 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 the flash configuration (interleaved or non-interleaved). See the Program flash memory features in RM to determine the sector size.

The erase flash sector command will abort if the selected sector is protected.

3.2.5 Erase recommendations

While there are a number of options for erasing flash on S32K1xx, most production flash programmers should only need to implement the mass erase and sector erase.

3.2.5.1 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). 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 NXP 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. NXP recommends using sector erase by default, but having a user selectable option to use mass erase instead.

3.2.5.2 Mass erase and CSEc considerations

The Flash module has added features to comply with the SHE9 specification. By using an embedded processor, firmware and hardware assisted AES-128 sub block, the Flash macro enables encryption, decryption and message generation and authentication algorithms for secure messaging applications. This module is called as Cryptographic Services Engine or CSEc.

When CSEc is enabled, some considerations must be taken prior to launch mass erase command. If no CSEc keys have been configured or the part is not partitioned, then the mass erase operation will be allowed to run. If CSEc key size field in the program partition command is set to zero keys (0x00), then you can mass erase at any time and it should always run as long as mass erase is enabled in the flash configuration field. If any CSEc keys have been configured (so the part has been already partitioned) user must finish the complete flow below otherwise the mass eras command will not work:

- When launching program partition command (PGMPART), be sure to allocate CSEc key size to value different than zero.
- Program CSEc keys, at least the MASTER_ECU_KEY.
- Erase Keys by issuing CMD_DBG_CHAL and CMD_DBG_AUTH commands. The CMD_DBG_AUTH command will erase the data IFR thus erasing the partition code and allow mass erase to work again.

NOTE

If any CSEc key is ‘write-protected’ the above procedure will not work thus mass erase cannot be launched.

3.2.5.3 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.

3.3 Programming flash

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

- Program section
- Program phrase (64-bit)

3.3.1 Program section

The program section command enables programming of a larger area of memory than the regular program phrase commands. A section is one fourth the size of the FlexRAM for S32K1xx devices. 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. 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.

3.3.2 Program phrase

The program phrase command can be used to program 64-bits (8 bytes) 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 for programming.

3.3.3 Programming recommendations

To increase programming speed, the program section command is the recommended programming method to use for flash production programming.

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 to determine if the command has completed before moving to the next command.

4 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.

4.1 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. NXP 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. For more information, please refer to Flash configuration field description section in reference manual.

Some users may 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.

Erasing the configuration fields via erase all (ERSALL), erase sector (ERSSCR) and erase block (ERSBLK) without programming back the configuration fields might lead to lockup the device so Flash configuration field must be programmed to an unsecure state prior to reset.

NOTE

Erase All blocks unsecured command (ERSALLU) should be used instead of ERSALL when the user does not intend to secure the device.

4.2 Emulated EEPROM support

S32K1xx devices include a FlexMemory feature. The FlexMemory enables users to configure some of the on-chip flash memory as emulated 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, the production flash programmer would need to include support to configure the EEPROM. EEPROM configuration can be supported over the SWD/JTAG interface only if flash is in its unsecured state.

4.2.1 FlexMemory partitioning

To configure the amount of EEPROM memory that will be used and the amount of flash that will be used to back up 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.

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.

4.2.2 Programming Emulated 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.

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.

4.3 Secured part

The JTAG/SWDinterfacewillbedisabledwhenthepartissecured.Thismeansthatadebugcontrollercannotreadorwriteto SOC memory mapped addresses when the part is in this state. The part is secure when the FTFC_FSEC byte is in a secure state in theflashconfigurationfield.Oncethishappens,you can'trunanyCMD_DBG_CHALandCMD_DBG_AUTHcommandsvia JTAG/ SWD. So, Customer application code must have the flow shown in Mass erase and CSEc considerations embedded intheirapplicationandtriggertheroutinefromadifferentinterfacesuchas CANorUART/Serialinterfacesforexample

5 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's voltage level electrical 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.
- Special care must be taken not to interrupt configuration routines associated with security, as parts can become inadvertently secured / locked up. Prior to resetting the processor, verify the written values.
- You must minimize the duration during the testing of the part in secured state in production line.
- Ensure power supplies are free of noise and within spec during flash programming

6 Further reading

Additional documentation that may be useful includes the following:

- S32K Microcontroller documentation, software, and tools available at www.nxp.com/s32k
- Flash Memory Controller, Flash Memory Module and Debug chapters of the applicable device's reference manual.
- Using S32K1xx EEPROM functionality (document AN11983)
- Using S32K148 FlexNVM memory (document AN12003)
- Getting Started with CSEc Security Module (document AN5401)
- Production Flash Programming Best Practices for Kinetis K- and L-series MCUs (document AN4835)
- 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).

7 Revision history

Table 1. Revision History

Revision	Date	Description
0	February, 2018	Initial release
1	September, 2019	• Updated Secured part. • Updated Common problems.

How To Reach Us

Home Page:

nxp.com

Web Support:

nxp.com/support

Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. NXP reserves the right to make changes without further notice to any products herein.

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including “typicals,” must be validated for each customer application by customer's technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: nxp.com/SalesTermsandConditions.

While NXP has implemented advanced security features, all products may be subject to unidentified vulnerabilities. Customers are responsible for the design and operation of their applications and products to reduce the effect of these vulnerabilities on customer’s applications and products, and NXP accepts no liability for any vulnerability that is discovered. Customers should implement appropriate design and operating safeguards to minimize the risks associated with their applications and products.

For more information, please visit: http://www.nxp.com

For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: September 2019

Document identifier: AN12130

arm