AN1085: Using the Gecko Bootloader with Silicon Labs Connect

This version of AN1085 has been deprecated with the release of Simplicity SDK Suite 2025.6.1 For the latest version, see docs.silabs.com.

****

KEY POINTS

- Gecko Bootloader overview
- Using the Gecko standalone bootloader
- Using the Gecko application bootloader

This application note includes detailed information on using the Silicon Labs Gecko Bootloader with the Silicon Labs Connect stack, part of the Silicon Labs Flex SDK (Software Development Kit). It supplements the general Gecko Bootloader implementation information provided in *UG266: Silicon Labs Gecko Bootloader User's Guide*. If you are not familiar with the basic principles of performing a firmware upgrade or want more information about upgrade image files, refer to *UG103.6: Bootloading Fundamentals*.

The *Connect User's Guide* is a series of documents that provides in-depth information for developers who are using the Silicon Labs Connect Stack for their application development. Refer to *UG435.06: Bootloading and OTA with Silicon Labs Connect* to learn about the bootloader options (standalone, application, and Over the Air (OTA)) available for use within Connect-based applications.

Proprietary is supported on all EFR32FG devices. For others, check the device's data sheet under Ordering Information > Protocol Stack to see if Proprietary is supported. In Proprietary SDK version 2.7.n, Connect is not supported on EFR32xG22

 Copyright © 2025 by Silicon Laboratories

1 Overview

The Silicon Labs Gecko Bootloader is a common bootloader for all the newer MCUs and wireless MCUs from Silicon Labs. The Gecko Bootloader can be configured to perform a variety of bootload functions, from device initialization to firmware upgrades. The Gecko Bootloader uses a proprietary format for its upgrade images, called GBL (Gecko Bootloader). These images are produced with the file extension ".gbl". Additional information on the GBL file format is provided in *UG103.6: Bootloading Fundamentals*.

The Gecko Bootloader has a two-stage design, where a minimal first stage bootloader is used to upgrade the main bootloader. The first stage bootloader only contains functionality to read from and write to fixed addresses in internal flash. To perform a main bootloader upgrade, the running main bootloader verifies the integrity and authenticity of the bootloader upgrade image file. The running main bootloader then writes the upgrade image to a fixed location in flash and issues a reboot into the first stage bootloader. The first stage bootloader verifies the integrity of the main bootloader firmware upgrade image, by computing a CRC32 checksum before copying the upgrade image to the main bootloader location.

The Gecko Bootloader can be configured to perform firmware upgrades in standalone mode (also called a standalone bootloader) or in application mode (also called an application bootloader), depending on the component configuration. Components can be enabled and configured through the Simplicity Studio IDE.

A standalone bootloader uses a communications channel to get a firmware upgrade image. NCP (network co-processor) devices always use standalone bootloaders. Standalone bootloaders perform firmware image upgrades in a single-stage process that allows the application image to be placed into flash memory, overwriting the existing application image, without the participation of the application itself. In general, the only time that the application interacts with a standalone bootloader is when it requests to reboot into the bootloader. Once the bootloader is running, it receives packets containing the firmware upgrade image by a physical connection such as UART or SPI. To function as a standalone bootloader, a component providing a communication interface such as UART or SPI must be configured.

An application bootloader relies on the application to acquire the firmware upgrade image. The application bootloader performs a firmware image upgrade by reprogramming the device's flash with the firmware upgrade image stored in a region of flash memory referred to as the download space. The application transfers the firmware upgrade image to the download space in any way that is convenient (UART, over-the-air, and so on). The download space is either an external memory device such as an EEPROM or dataflash or a section of the chip's internal flash. The Gecko Bootloader can partition the download space into multiple storage slots, and store multiple firmware upgrade images simultaneously. To function as an application bootloader, a component providing a bootloader storage implementation has to be configured.

OTA (over the air) is often confused with application bootloader, since application bootloader can be used to flash an image that was distributed OTA. However, OTA is an application feature. For example, an NCP gateway device typically uses the standalone bootloader, so the host can update it, but the Host-NCP application can work as an OTA server.

This document describes how to use both standalone and application models with Silicon Labs Connect.

2 Using the Gecko Standalone Bootloaders

A Gecko Bootloader-based standalone bootloader receives an application image onto a target device by serial transfer via SPI or UART. If using UART, you can establish a serial connection between a source device and a target device's serial interface and upload a new software image to it using the XModem protocol. If you need information on the XModem protocol, a good place to start is http://en.wikipedia.org/wiki/XMODEM, which should have a brief description and up-to-date links to protocol documentation.

2.1 Performing a Serial Upload – UART XMODEM Bootloader

Serial upload can be performed with any source device that provides the expected serial interface method. This can be a Windows-based PC, a Linux or Mac OS-based device, or an embedded MCU with no operating system. UART transfer can be done with a third-party serial terminal program like TeraTerm on Windows or Linux minicom or with user-compiled host code. However, drivers for SPI Master or UART may vary with operating systems, and serial terminal programs may vary in timing and performance, so if you are unsure about what driver or program to use on your source code, please consult Silicon Labs technical support.

To open a serial connection over UART, the source device connects to the target device at 115,200 baud, 8 data bits, no parity bit, and 1 stop bit (8-N-1), with no flow control by default. These options may be changed using the component options in the Bootloader project.

Note: The UART-based serial bootloader configurations do not employ any flow control in the communication channel by default, because the XModem protocol used for image transfer already has built-in flow control mechanisms. However, Silicon Labs' normally-supplied NCP firmware does utilize either hardware-based (RTS/CTS) or software-based (XON/XOFF) flow control, so a host device must take care to temporarily disable the flow control when placing its NCP into serial bootloading mode. Alternatively, application designers can change the options in the provided bootloader project and customize the serial bootloader's handling of the UART to add hardware flow control at their discretion.

Once the connection with a UART-based serial bootloader is established:

1. The target device's bootloader sends output over its serial port after it receives a carriage return from the source device at the expected baud rate. This prevents the bootloader from prematurely sending commands that might be misinterpreted by other devices that are connected to the serial port. Note that serial bootloaders typically don't enforce any timeout when awaiting the initial serial handshake via carriage return, so the bootloader will wait indefinitely in this mode until guided by the source device or until the chip is reset.
2. After the bootloader receives a carriage return from the target device, it displays a menu with the following ASCII-based output, (where X.Y.A corresponds to the major, minor, and sub-minor fields of the bootloader version number, respectively):

```
Gecko Bootloader vX.Y.A
1. upload gbl
2. run
3. ebl info
BL >
```

Note: The third menu option has no effect. While current menu options should remain functionally unchanged, the menu title and options text is liable to change, and new options might be added.

After listing the menu options, the bootloader's "BL >" prompt displays, and the ASCII character corresponding to the number of each option can then be entered by the source to select the described action, such as '2' (ASCII code 0x32) to run the firmware presently loaded in the application area. Here again, no timeout is enforced by the bootloader, so it will wait indefinitely until a character is received or the chip is reset. Note that while the menu interface is designed for human interaction, the transfer can still be performed programmatically or through a scripted interface, provided the source device sends the expected ASCII characters to the target at appropriate times.

Note: Scripts that interact with the bootloader should use only the "BL >" prompt to determine when the bootloader is ready for input.

Selecting menu option 1 initiates upload of a new software image to the target device, which unfolds as follows:

1. The target device awaits an XModem CRC upload of a GBL file over the expected serial interface, as indicated by the stream of C characters that its bootloader transmits.
2. If no transaction is initiated within 60 seconds, the bootloader times out and returns to the menu.
3. Once uploading begins (first XModem SOH data packet received), the bootloader expects each successive XModem SOH packet within 1 second, or else a timeout error will be generated and the session will abort.
4. After an image successfully uploads, the XModem transaction completes and the bootloader displays 'Serial upload complete' before redisplaying the menu.

2.2 Errors and Status Codes

If an error occurs during the upload, the UART serial bootloader displays the message 'Serial upload aborted,' followed by a more detailed message and a hex error code. Some of the more common errors are shown in the following table. The UART serial bootloader then redisplays the bootloader menu.

The following tables describe the normal status codes, error conditions, and special characters or enumerations used by the Gecko Bootloader. For additional status codes, see the Gecko Bootloader API documentation installed with your SDK in the platform/bootloader/documentation folder.

Table 1. Serial Uploading Statuses and Error Messages

Hex code	Constant	Description
0x00	BL_SUCCESS	Default success status.
0x01	BL_ERR	General error processing packet.
0x1C	BLOCK_TIMEOUT	The bootloader timed out waiting for some part of the XModem frame.
0x21	BLOCKERR_SOH	The bootloader did not find the expected start of header (SOH) character at the beginning of the XModem frame.
0x22	BLOCKERR_CHK	The bootloader detected the sequence check byte of the XModem frame was not the inverse of the sequence byte.
0x23	BLOCKERR_CRCH	The bootloader encountered an error while comparing the high bytes of the received and calculated CRCx of the XModem frame.
0x24	BLOCKERR_CRCL	The bootloader encountered an error while comparing the low bytes of the received and calculated CRCs of the XModem frame.
0x25	BLOCKERR_SEQUENCE	The bootloader did not receive the expected sequence number in the current XModem frame.
0x26	BLOCKERR_PARTIAL	The frame that the bootloader was trying to parse was deemed incomplete (some bytes missing or lost).
0x27	BLOCKERR_DUPLICATE	The bootloader encountered a duplicate of the previous XModem frame.
0x40	BL_ERR_MASK	Bitmask for any bootloader error codes returned in CAN or NAK frame.
0x41	BL_ERR_HEADER_EXP	No GBL header was received when expected.
0x42	BL_ERR_HEADER_WRITE_CRC	Failed to write header or CRC.
0x43	BL_ERR_CRC	File or written image failed CRC check.
0x44	BL_ERR_UNKNOWN_TAG	Unknown tag detected in GBL image.
0x45	BL_ERR_SIG	Invalid GBL header contents.
0x46	BL_ERR_ODD_LEN	Trying to flash odd number of bytes.
0x47	BL_ERR_BLOCK_INDEX	Indexed past end of block buffer.
0x48	BL_ERR_OVWR_BL	Attempt to overwrite bootloader flash.
0x49	BL_ERR_OVWR_SIMEE	Attempt to overwrite SIMEE flash.
0x4A	BL_ERR_ERASE_FAIL	Flash erase failed.

Hex code	Constant	Description
0x4B	BL_ERR_WRITE_FAIL	Flash write failed.
0x4C	BL_ERR_CRC_LEN	End tag CRC wrong length.
0x4D	BL_ERR_NO_QUERY	Received data before query request/response.
0x4E	BL_ERR_BAD_LEN	An invalid length was detected in the upgrade image file.
0x4F	BL_ERR_TAGBUF	Insufficient tag buffer size or an invalid length was found in the GBL image.

Table 2. Special Characters Used in Packet Types

Hex code	Constant	Description
0x01	SOH	Start of Header.
0x03	CTRL_C	Cancel (from sender).
0x04	EOT	End of Transmission.
0x06	ACK	Acknowledged.
0x15	NAK	Not acknowledged.
0x18	CAN	Cancel
0x43	C	ASCII ‘C’.
0x51	QUERY	ASCII ‘Q’.
0x52	QRESP	ASCII ‘R’.

Table 3. Status Codes Returned in a Synchronous Response

Hex code	Constant	Description
0x16	TIMEOUT	Bootloader timed out expecting characters.
0x17	FILEDONE	EOT process successfully.
0x18	FILEABORT	Transfer aborted prematurely.
0x19	BLOCKOK	Data block processed OK.
0x1A	QUERYFOUND	Successful query.

2.3 Running the Application Image

For standalone bootloader variants that utilize an interactive menu, bootloader menu option 2 (run) resets the target device into the uploaded application image. If no application image is present, or an error occurred during a previous upload, the bootloader returns to the menu. For SPI-based variants, which don’t use a menu, the application starts up immediately upon ACKing the EOT frame from the source device.

2.4 Performing a Bootloader Upgrade

To perform a bootloader upgrade with the standalone bootloaders, simply transmit two GBL files in succession. The first GBL file should contain a main bootloader upgrade image. After upload is completed, the device resets to upgrade the main bootloader. For standalone bootloader variants that utilize an interactive menu, bootloader menu option 2 (run) resets the target device into the first stage bootloader to perform the upgrade, before returning to the upgraded main bootloader. The second GBL file, containing an application upgrade image, can then be uploaded.

2.5 Upload Recovery

If an image upload fails, the target node is left without a valid application image. The standalone bootloader will re-enter firmware upgrade mode if it determines that the application image isn't valid. However, the application image may have a valid structure, but contain a bug preventing normal operation. Regardless of the serial interface supported by your standalone bootloader, a GPIO-based trigger can be used to facilitate recovery via serial upload.

You can configure your standalone bootloader to use a software-based GPIO pin check or other schemes of recovery mode activation such as button recovery by configuring the GPIO Activation component.

3 Using the Gecko Application Bootloader

3.1 Acquiring a New Image

The application bootloader relies on application code to obtain new code images. The bootloader itself only knows how to read a GBL image stored in the download space and copy the relevant portions to the main flash block. This approach means that the application developer is free to acquire the new code image in any way that makes sense (serial, OTA, and so on).

Typically, application developers choose to acquire the new code image over-the-air (OTA) since this is readily available on all devices. The acquisition of a new image in Connect is covered extensively in the Connect Users Guide, UG435.

3.2 Performing an Application Upgrade

The application and the bootloader have limited indirect contact. Their only interaction is through passing non-volatile data across module reboots.

Once the application decides to install a new image saved in the download space it calls the Ember HAL API `halAppBootloaderInstallNewImage()`. This call indicates to the bootloader that it should attempt to perform a firmware upgrade from storage slot 0, and reboots the device. This API is backwards-compatible with the legacy Ember bootloader. If performing a firmware upgrade from a different storage slot than slot 0 is desired, the Gecko Bootloader API should be used instead, by calling `bootloader_setBootloadList()`. If the bootloader fails to install the new image, it sets the reset cause to `RESET_BOOTLOADER_BADIMAGE` and resets the module. Upon startup, the application should read the reset cause with `halGetExtendedResetInfo()`. If the reset cause is set to `RESET_BOOTLOADER_BADIMAGE`, the application knows the install process failed and can attempt to obtain a new image. A printable error string can be acquired from calling `halGetExtendedResetString()`. Under normal circumstances, the application bootloader does not print anything on the serial line.

3.3 External Storage Application Bootloader

See *UG266: Silicon Labs Gecko Bootloader User's Guide* for information about memory configuration for external storage bootloaders.

Application bootloaders typically use an external device to store the downloaded application image. This device can be accessed over either an I2C or SPI serial interface. Refer to *UG266: Silicon Labs Gecko Bootloader User's Guide* for a list of supported Dataflash/EEPROM devices. It is important to select a device whose size is at least the size of your flash in order to fit the application being bootloaded.

The default recommendation for external SPI serial flash is the MX25R, as this is available in standard and smaller packages, is supported by standard drivers, and has very low software-enabled sleep current without the need for an external shutdown control circuit. Most radio boards from Silicon Labs are populated with the MX25R8035F for evaluation purposes. In general, customers should use parts that have low sleep current, don't require external shutdown control circuitry, and have software shutdown control. When using components with high idle/sleep current and no software shutdown control, an external shutdown control circuit is recommended to reduce sleep current.

For EFR32xG1, only the serial dataflash option is available; no internal storage application bootloader is presently offered.

Note: Some of these chips have compatible pinouts with others, but there are several incompatible variations. Contact Silicon Labs for details on connecting I2C or other SPI dataflash chips to an EFR32.

Read-Modify-Write pertains to a feature of certain dataflash chips that their corresponding driver exposes, and that is exploited by the bootloader library. Chips without this feature require a page erase to be performed before writing to that page, which precludes random-access writes by an application.

3.4 Internal Storage Application Bootloader

The internal storage bootloader is essentially an application bootloader with a data flash driver that uses a portion of the on-chip flash for image storage instead of an external storage chip. See *UG266: Gecko Bootloader User's Guide* for information about memory configuration for internal storage.

If you don't modify the flash layout of your application, the NVMv3 token storage is at the end of the flash memory. By default, our internal storage bootloader sets up a slot that protects at least the default 36k of token storage (48k on some parts). This configuration however has some drawbacks:

- You might want to optimize your storage space, and use a smaller NVMv3 token storage, in which case the bootloader protects more than what it needs to.
- Apart from the size outputs of the compiler, you won't get notified if you run out of flash space. If you program the device directly, not through the bootloader, you are allowed to overlap your application with OTA storage, which will cause failure the first time you erase the OTA storage (depicted in the middle below)

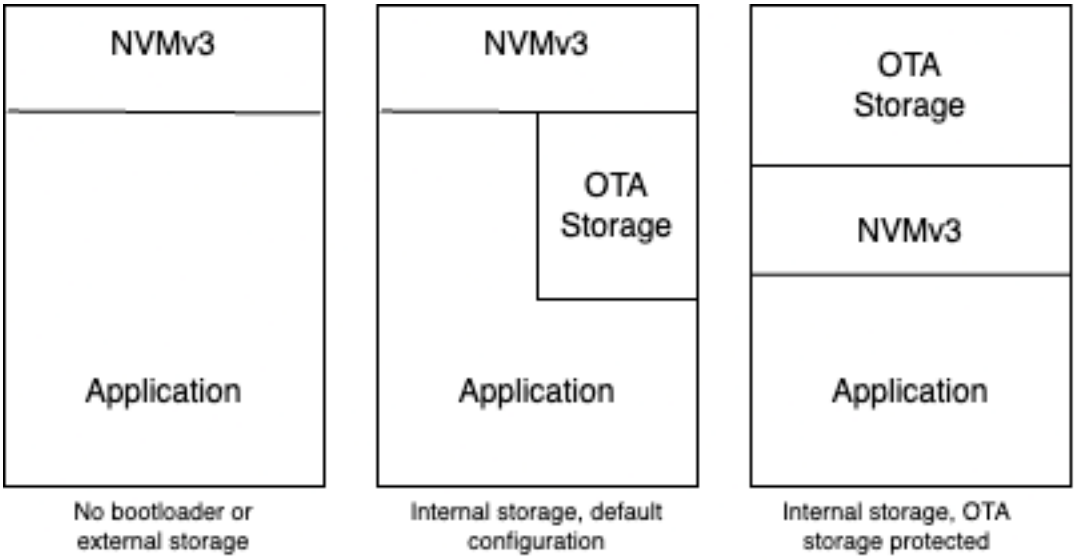

Figure 1. Main Flash Layout for external storage bootloader and two typical internal storage

To avoid these problems, you can modify your main application's memory allocation in the memory editor. For example, a typical application will use half, or a bit more than half, of the flash for the application and NVMv3, while the remainder can be used as OTA storage. The OTA storage can be smaller, because it doesn't include the NVMv3 and because bootloader can be configured to use compressed images.

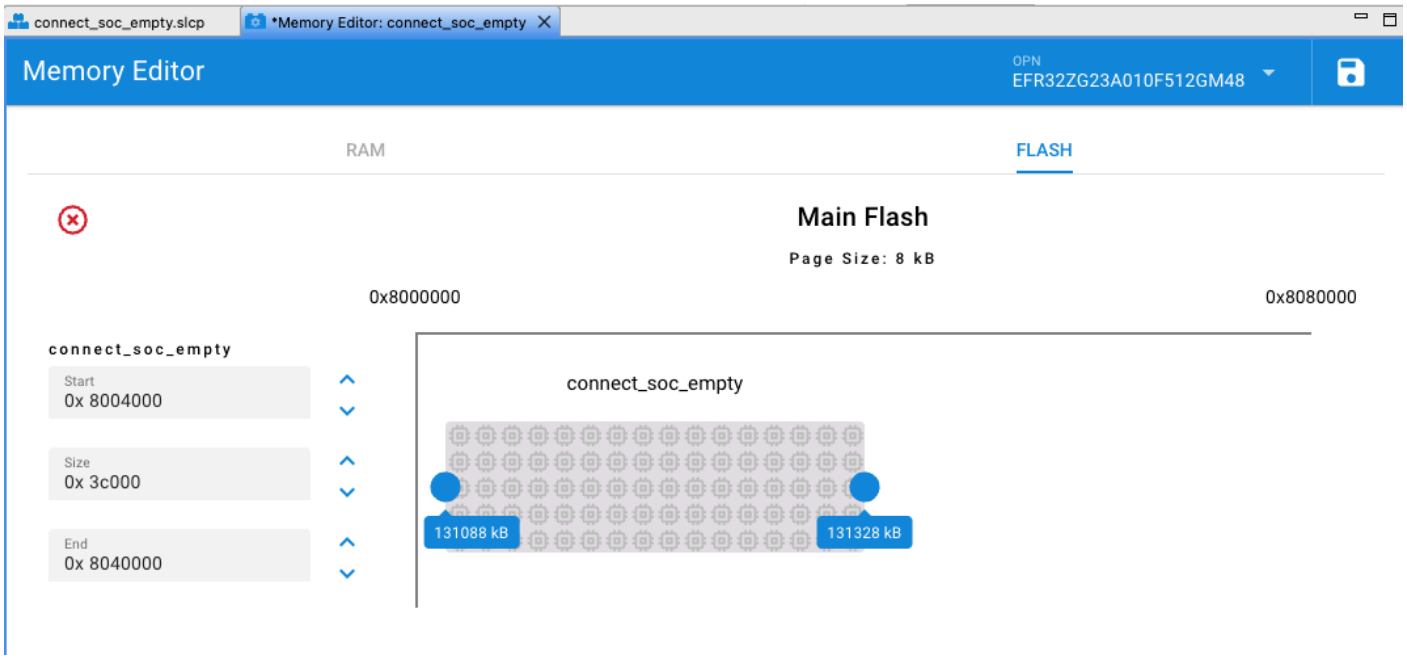

Figure 2. Configuring an application to protect the OTA storage

Finally, you should configure the bootloader project's **Bootloader Storage Slot Setup** component to utilize the space you protected from your application:

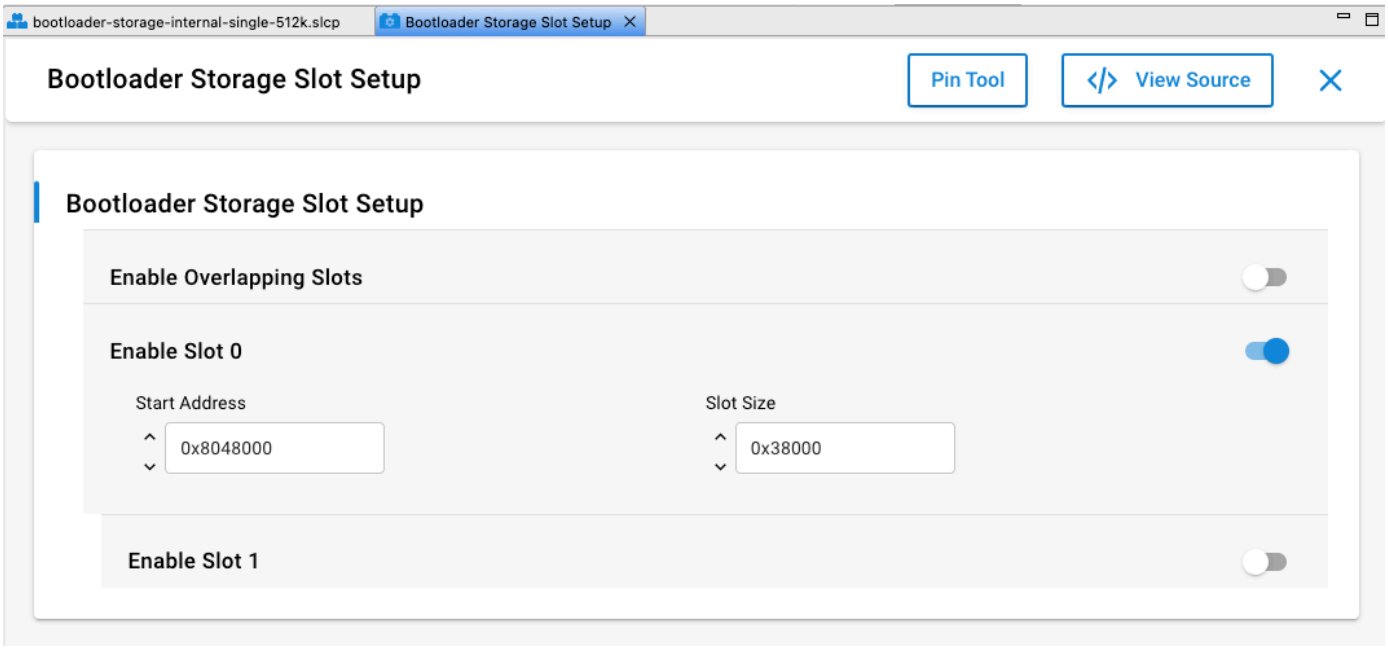

Figure 3. Configuring bootloader slot

Note: On EFR32xG1 and Series 2 devices, the application is offset by 16 kB to accommodate the bootloader at the bottom of main flash. On other EFR32 Series 1 devices, the Gecko Bootloader resides in the Bootloader flash region outside of main flash block.