AN1303: Programming Series 2 Devices using the Debug Challenge Interface (DCI) and Serial Wire Debug (SWD)

This application note describes how to provision and configure Series 2 devices through the dedicated Debug Challenge Interface (DCI). The process to use the Serial Wire Debug (SWD) interface for programming the internal flash memory of Series 2 devices is also included.

For details on how to use the SWD interface to program devices, see AN0062: Programming Internal Flash over the Serial Wire Debug Interface.

KEY POINTS

- DCI overview
- SWD interface overview
- DCI programming examples to provision and configure Series 2 devices
- SWD programming examples to program Series 2 devices
- How to add a new Series 2 device to the programmer

1. Series 2 Device Security Features

Protecting IoT devices against security threats is central to a quality product. Silicon Labs offers several security options to help developers build secure devices, secure application software, and secure paths of communication to manage those devices. Silicon Labs' security offerings were significantly enhanced by the introduction of the Series 2 products that included a Secure Engine. The Secure Engine is a tamper-resistant component used to securely store sensitive data and keys and to execute cryptographic functions and secure services.

On Series 1 devices, the security features are implemented by the TRNG (if available) and CRYPTO peripherals.

On Series 2 devices, the security features are implemented by the Secure Engine and CRYPTOACC (if available). The Secure Engine may be hardware-based, or virtual (software-based). Throughout this document, the following abbreviations are used:

- HSE - Hardware Secure Engine
- VSE - Virtual Secure Engine
- SE - Secure Engine (either HSE or VSE)

Additional security features are provided by Secure Vault. Three levels of Secure Vault feature support are available, depending on the part and SE implementation, as reflected in the following table:

Level (1)	SE Support	Part (2)
Secure Vault High (SVH)	HSE only (HSE-SVH)	Refer to UG103.05 for details on supporting devices.
Secure Vault Mid (SVM)	HSE (HSE-SVM)	"
"	VSE (VSE-SVM)	"
Secure Vault Base (SVB)	N/A	"

Note:

1. The features of different Secure Vault levels can be found in https://www.silabs.com/security.
2. UG103.05.

Secure Vault Mid consists of two core security functions:

- Secure Boot: Process where the initial boot phase is executed from an immutable memory (such as ROM) and where code is authenticated before being authorized for execution.
- Secure Debug access control: The ability to lock access to the debug ports for operational security, and to securely unlock them when access is required by an authorized entity.

Secure Vault High offers additional security options:

- Secure Key Storage: Protects cryptographic keys by "wrapping" or encrypting the keys using a root key known only to the HSE-SVH.
- Anti-Tamper protection: A configurable module to protect the device against tamper attacks.
- Device authentication: Functionality that uses a secure device identity certificate along with digital signatures to verify the source or target of device communications.
- Advanced cryptographic acceleration (i.e., ChaCha20/Poly1305)

A Secure Engine Manager and other tools allow users to configure and control their devices both in-house during testing and manufacturing, and after the device is in the field.

1.1 User Assistance

In support of these products Silicon Labs offers whitepapers, webinars, and documentation. The following table summarizes the key security documents:

Document	Summary	Applicability
AN1190: Series 2 Secure Debug	How to lock and unlock Series 2 debug access, including background information about the SE	Secure Vault Mid and High
AN1218: Series 2 Secure Boot with RTSL	Describes the secure boot process on Series 2 devices using SE	Secure Vault Mid and High
AN1222: Production Programming of Series 2 Devices	How to program, provision, and configure security information using SE during device production	Secure Vault Mid and High
AN1247: Anti-Tamper Protection Configuration and Use	How to program, provision, and configure the anti-tamper module	Secure Vault High
AN1268: Authenticating Silicon Labs Devices using Device Certificates	How to authenticate a device using secure device certificates and signatures, at any time during the life of the product	Secure Vault High
AN1271: Secure Key Storage	How to securely “wrap” keys so they can be stored in non-volatile storage.	Secure Vault High
AN1311: Integrating Crypto Functionality Using PSA Crypto Compared to Mbed TLS	How to port firmware from the Mbed TLS cryptographic library to PSA Crypto.	Secure Vault Mid and High
AN1374: Series 2 TrustZone	How to TrustZone to separate trusted and non-trusted firmware.	Secure Vault Mid and High

1.2 Key Reference

Public/Private keypairs along with other keys are used throughout Silicon Labs security implementations. Because terminology can sometimes be confusing, the following table lists the key names, their applicability, and the documentation where they are used.

Key Name	Customer Programmed	Purpose	Used in
Public Sign key (Sign Key Public)	Yes	Secure Boot binary authentication and/or OTA upgrade payload authentication	AN1218 (primary), AN1222
Public Command key (Command Key Public)	Yes	Secure Debug Unlock or Disable Tamper command authentication	AN1190 (primary), AN1222, AN1247
OTA Decryption key (GBL Decryption key) aka AES-128 Key	Yes	Decrypting GBL payloads used for firmware upgrades	AN1222 (primary), UG266/UG489
Attestation key aka Private Device Key	No	Device authentication for secure identity	AN1268

1.3 SE Firmware

Silicon Labs strongly recommends installing the latest SE firmware on Series 2 devices to support the required security features. Refer to AN1222 for the procedure to upgrade the SE firmware and UG103.05 for the latest SE Firmware shipped with Series 2 devices and modules.

2. Introduction

The latest SE firmware image (`.seu` and `.hex`) and release notes can be found in the Windows folder below.

For GSDK v3.2 and lower:

```
C:\SiliconLabs\SimplicityStudio\v5\developer\sdks\gecko_sdk_suite\<GSDK VERSION>\util\se_release\public
```

For GSDK v4.0 and higher:

```
C:\Users\<PC USER NAME>\SimplicityStudio\SDKs\gecko_sdk\util\se_release\public
```

Silicon Labs provides Custom Part Manufacturing Service (CPMS) to customize the users' security features and settings.

The Debug Challenge Interface (DCI) is used to configure the security features of the Series 2 devices, whereas the Serial Wire Debug (SWD) interface is used to program the flash memory of Series 2 devices. A general overview of the DCI and SWD programming steps is described in the following sections.

3. Debug Challenge Interface (DCI)

Interaction with the SE is performed over a command interface that is available through a dedicated Debug Challenge Interface (DCI). The DCI is intended to be used for various SE commands. The DCI is open while the SE is running.

3.1 DCI Connection

The DCI is made available through a connection on the Serial Wire Debug (SWD) port. The steps involved in connecting the DCI through the SWD port are as follows.

1. Send the JTAG-to-SWD switching sequence.
2. Read the IDCODE register (SWD DP register 0) to retrieve an identification value. On the Series 2 devices with a Cortex-M33 core, this value is `0x6BA02477`.
3. Use the ABORT register (SWD DP register 0 = `0x0000001E`) to clear the error and sticky flag conditions.
4. Use the STAT register (SWD DP register 1 = `0x50000000`) to generate a system and debug domain power-up request.
5. Use the SELECT register (SWD DP register 2 = `0x01000000`) to set the SWD interface in the chip to communicate with the DCI.
6. Use the CSW register (SWD AP register 0 = `0x22000002`) to set the transfer size to 32-bit.

See AN0062: Programming Internal Flash over the Serial Wire Debug Interface for more information about the SWD port registers.

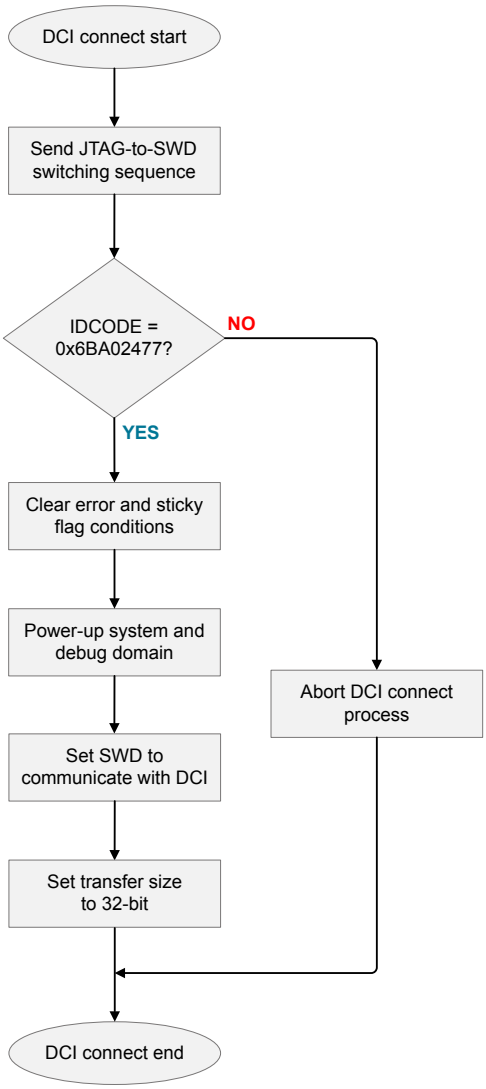


Figure 3.1. DCI Connection Flowchart

3.2 DCI Registers

The following table lists three registers to interact with the DCI through the SWD interface.

Table 3.1. DCI Register

Register	Description	Fields	Address	Remarks
DCI_WDATA	Write data to the DCI	WDATA [31:0]	0x1000	Command
DCI_RDATA	Read data from the DCI	RDATA [31:0]	0x1004	Response
DCI_STATUS	Status of DCI accesses	Bit 0 = WPENDING	0x1008	Write Request to the DCI is pending. Additional writes to DCI_WDATA are discarded when this bit is asserted.
"	"	Bit 8 = RDATAVALID	"	Response from the DCI is valid when this bit is asserted. Cleared on a read of DCI_RDATA.

3.3 DCI Calls

3.3.1 Command

All DCI calls start by writing a 32-bit word containing the length of the DCI data followed by the 32-bit Command ID into the DCI. The length includes the length word itself, so the minimum value is 8 (4 bytes of length and 4 byes of Command ID). Then a variable length command payload (if applicable) is transferred into the DCI.

Table 3.2. DCI Command

Command Word	Description
Word 0	Length of packet in bytes (including word 0)
Word 1	Command ID
Words 2 – N	Command payload if applicable

3.3.2 Response

On completion, a 32-bit word consisting of the response length [15:0] in bytes and response code [31:16] is read from the DCI followed by the response payload, if present.

Table 3.3. DCI Response

Response Word	Description
Word 0	Total response length (including word 0) and response code: [15:0] – Total length in bytes; [31:16] – Response code
Words 1 – N	Response payload if applicable

All executed commands return a response code that classifies the result of the operation. The basic meaning of these response codes is given in the following table.

Table 3.4. DCI Response Codes

Response Code	Status	Description
0	SE_RESPONSE_OK	Command executed successfully or signature was successfully validated.
1	SE_RESPONSE_INVALID_COMMAND	Command was not recognized as a valid command, or is not allowed in the current context.
2	SE_RESPONSE_AUTHORIZATION_ERROR	User did not provide the required credentials to be allowed to execute the command.
3	SE_RESPONSE_INVALID_SIGNATURE	Signature validation command failed to verify the given signature as being correct.
4	SE_RESPONSE_BUS_ERROR	A command started in non-secure mode is trying to access secure memory.
5	SE_RESPONSE_INTERNAL_ERROR	Internal SE error.
6	SE_RESPONSE_CRYPTO_ERROR	Error in crypto operation.
7	SE_RESPONSE_INVALID_PARAMETER	One of the passed parameters is deemed invalid (for example, out of bounds), or the number of parameters is incorrect.
8	SE_RESPONSE_INTEGRITY_ERROR	Operation cannot be completed due to the SE having an invalid internal state.
9	SE_RESPONSE_SECUREBOOT_ERROR	The host application failed secure boot check.
10	SE_RESPONSE_SELFTEST_ERROR	Failure during self-test.
11	SE_RESPONSE_NOT_INITIALIZED	Feature or item is not present or not initialized.

3.4 DCI Operation

3.4.1 DCI Write

The user can write the DCI_WDATA register to pass command to the SE. The steps involved in writing a command to DCI are as follows.

1. Connect to DCI according to Figure 3.1 DCI Connection Flowchart on page 5.
2. For each word in the command, follow these steps in sequence:
   a. Set the DCI to read from DCI_STATUS by setting SWD AP register 1 to 0x1008.
   b. Read DCI_STATUS by reading from SWD AP register 3.
   c. If WPENDING is high, go back to step (a). If WPENDING is low, continue. If RDATAVALID is high, then the SE has started issuing a reply and the current command needs to be aborted.
   d. Set the DCI to write to DCI_WDATA by setting SWD AP register 1 to 0x1000.
   e. Write the command word to SWD AP register 3.

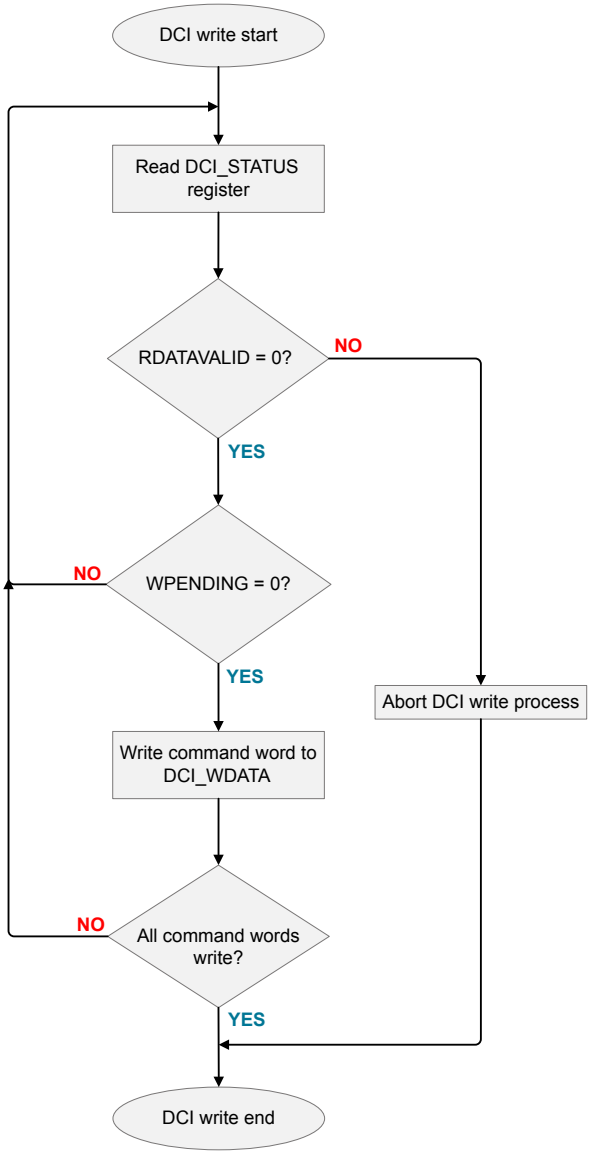


Figure 3.2. DCI Write Flowchart

3.4.2 DCI Read

The user can read the DCI_RDATA register to retrieve responses from the SE to a previously written command. If a user has sent a command according to Figure 3.2 DCI Write Flowchart on page 8, the steps to read the response from the DCI are as follows.

1. Set the DCI to read from DCI_STATUS by setting SWD AP register 1 to 0x1008.
2. Read DCI_STATUS by reading from SWD AP register 3.
3. If RDATAVALID is high, a response word is available to be read from DCI_RDATA. If RDATAVALID is low, go back to step (2) because the SE has not begun to reply.
4. Set the DCI to read from DCI_RDATA by setting SWD AP register 1 to 0x1004.
5. Read the first response word from the command to SWD AP register 3.
6. Total length of the response (including the first word) is in the lower 16 bits of that word. If this is larger than 4, for each response word:
   a. Set the DCI to read from DCI_STATUS by setting SWD AP register 1 to 0x1008.
   b. Read DCI_STATUS by reading from SWD AP register 3.
   c. If RDATAVALID is high, a response word is available to be read from DCI_RDATA. If RDATAVALID is low, go back to step (a) because the process is polling faster than the SE can send data.
   d. Set the DCI to read from DCI_RDATA by setting SWD AP register 1 to 0x1004.
   e. Read the sequential response word to SWD AP register 3.

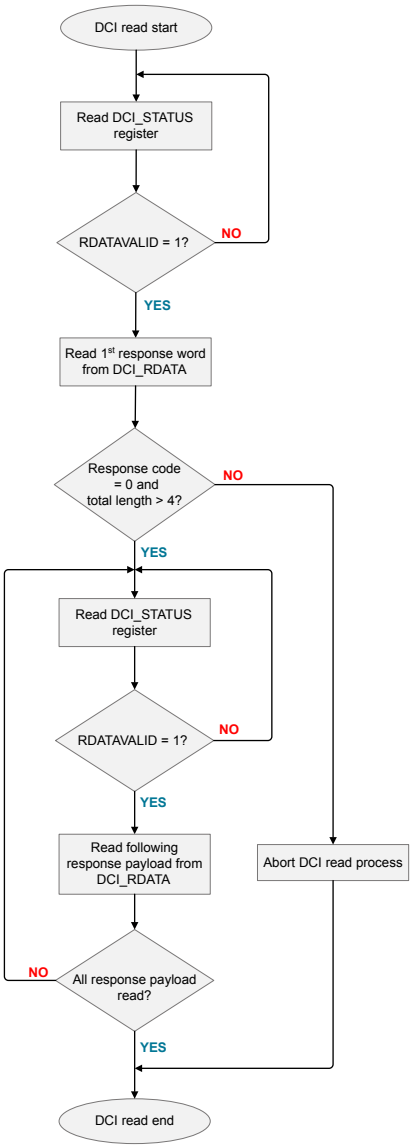


Figure 3.3. DCI Read Flowchart

4. Serial Wire Debug (SWD) Interface

The flash memory on Series 2 devices is divided into two blocks: the main block and the information block. Program code is normally written to the main block. The information block is available for special user data. Throughout this document, the flash or main flash is referred to as the main block and user data is referred to as the information block.

Program the Series 2 flash memory by writing directly to the device's Memory System Controller (MSC) registers over the Serial Wire Debug (SWD) interface. This method is simple and easy to upgrade to support new Series 2 devices.

The program must set the MSC bit in the CMU CLKEN1 register (if available) to enable the clock source for the Memory System Controller (MSC).

Series 2 devices with a Cortex-M33 core return the value `0x84770001` when reading the AP Identification Register (IDR) through SWD.

See AN0062: Programming Internal Flash over the Serial Wire Debug Interface for more information on how to access the SWD interface of the target device and how to use this interface to program devices.

4.1 Flash Erase

There are two ways to erase the flash of the target device through SWD interface:

Page Erase

A page erase can be initiated from software using the ERASEPAGE bit in the MSC WRITECMD register. The page erase operations require that the address of main flash or user data is written into the MSC ADDRB register. To reduce the time needed for the flash erase process, the program should avoid erasing the target main flash page by page, especially for devices with larger flash memories.

Mass Erase

A mass erase can be initiated from software using the ERASEMAIN0 bit in the MSC WRITECMD register. This erases the entire flash (excluding the user data).

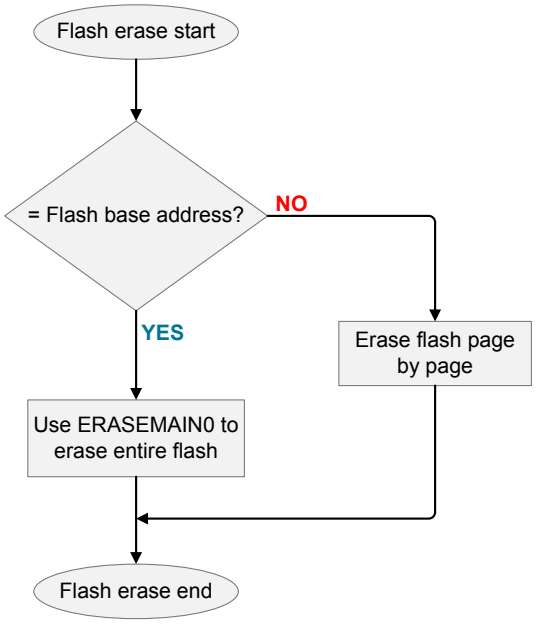


Figure 4.1. Flash Erase Flowchart

The flash base address of the Series 2 device is either `0x00000000` or `0x08000000`. The timing for page erase or mass erase on each Series 2 device might vary. Refer to the device-specific datasheets for details.

4.2 Flash Write

The write operation requires that the address be written into the MSC ADDRB register. After each 32-bit word is written, the internal address register is incremented automatically by 4. When a word is written to the MSC WDATA register, the WDATAREADY bit of the MSC STATUS register is cleared. When this status bit is set, software can write the next word.

The flash program time (32-bit word) of each Series 2 device might vary. Refer to the device-specific datasheets for details.

To reduce the time for the flash write process, the program can omit polling the WDATAREADY bit in the MSC STATUS register after writing each 32-bit word, because the register read process is time consuming. The alternative is to add a fixed microseconds delay between each write to make sure the maximum write time can be met.

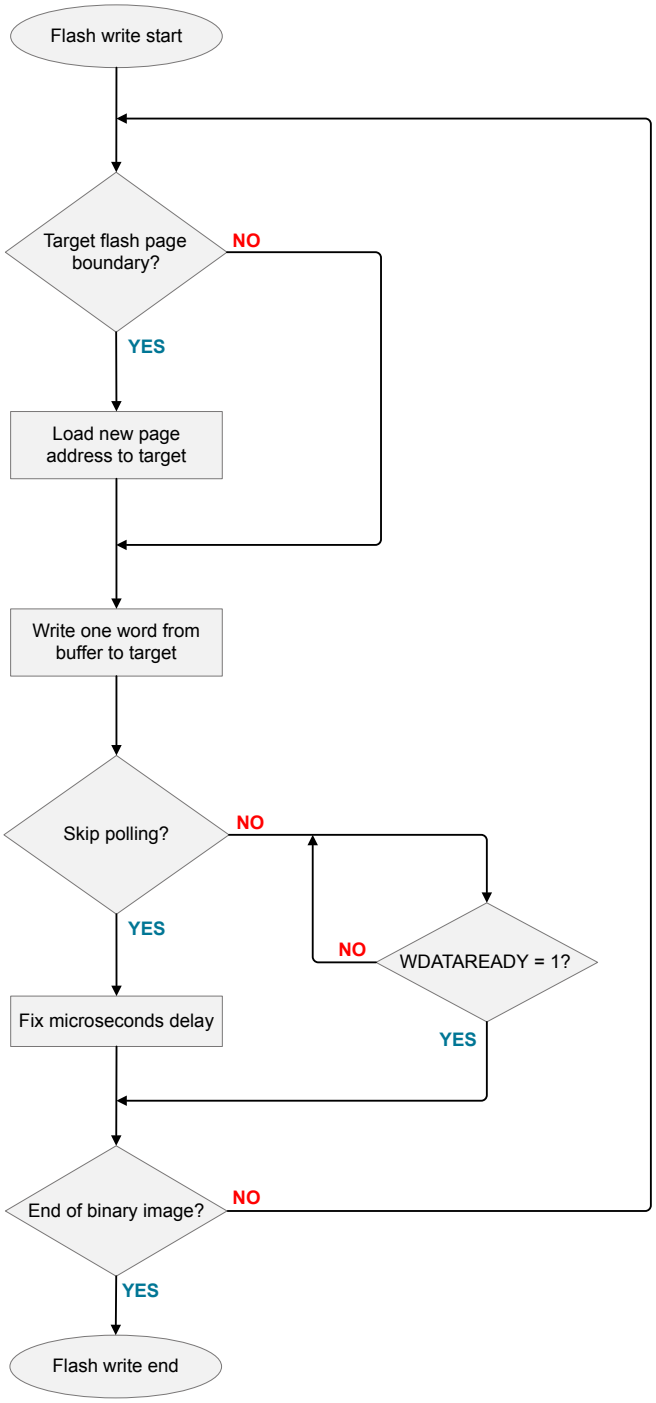


Figure 4.2. Flash Write Flowchart

4.3 Flash Verify

The program verifies the target flash contents with a buffer to make sure that no errors occurred during the programming process. The autoincrement of the Transfer Address Register (TAR) is for burst reads within the TAR wraparound boundary. The TAR must be initialized at every TAR wraparound boundary to set up the next flash read address.

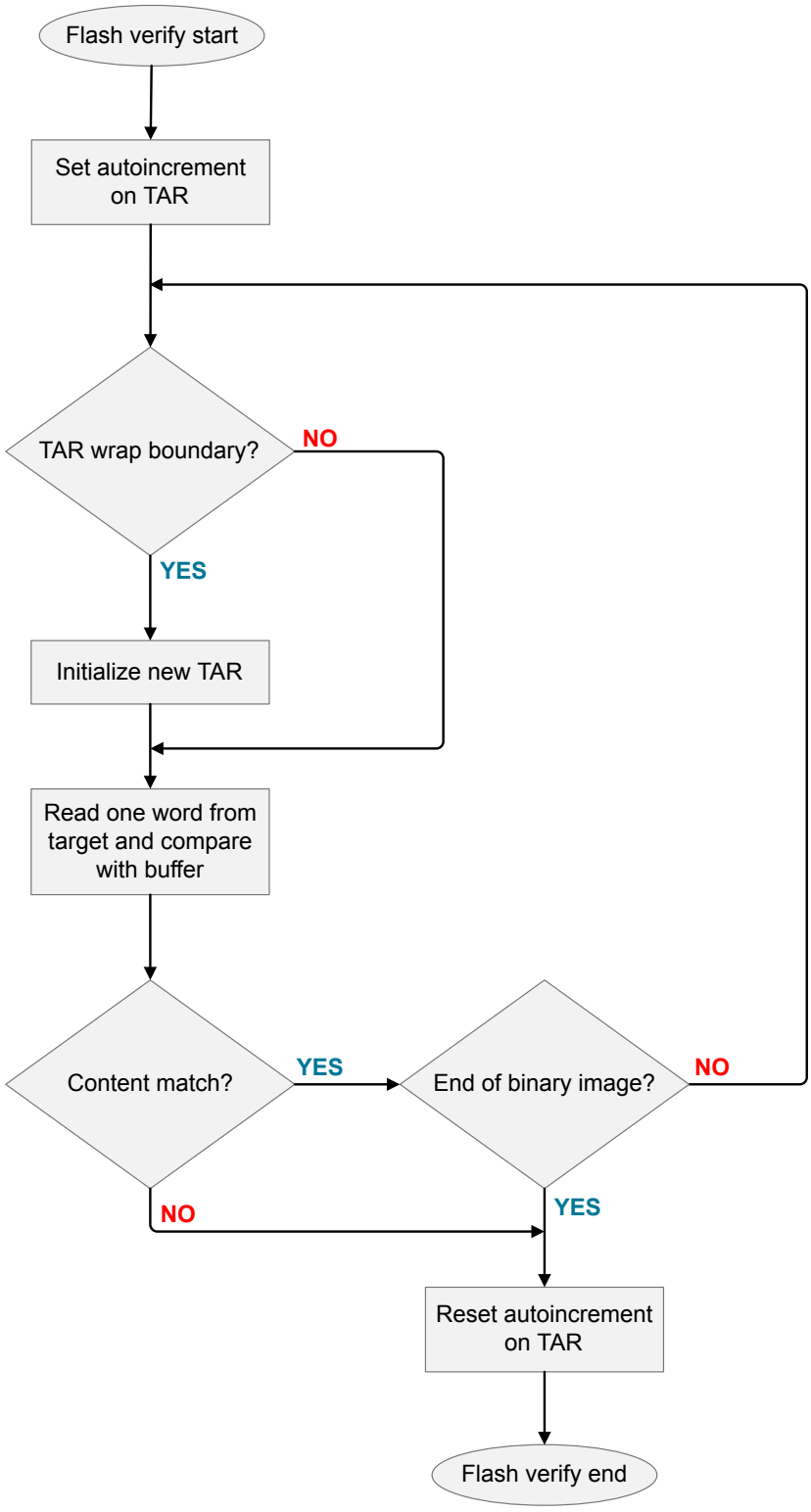


Figure 4.3. Flash Verify Flowchart

5. SE Command List

This application note does not include a complete list of commands for DCI. The following sections contain information about each command's operation and arguments for DCI production programming. The command and response payload may be device-specific (e.g., Initialize OTP and Get Status).

For more information about secure boot, see AN1218: Series 2 Secure Boot with RTSL. For more information about debug lock and secure debug, see AN1190: Series 2 Secure Debug.

5.1 SE Image Check

This command can be used to check the SE image before starting the upgrade process, in order to be able to abort early if the image is invalid or inapplicable.

Note: This command is only available on SE firmware version ≥ v1.2.2 (xG21 or xG22 devices).

Table 5.1. SE Image Check Command

ID [31:16]	Option 1 [15:8]	Option 2 [7:0]	Command payload	Response payload
0x4302	0x00	0x00	Address in internal flash where the SE upgrade image is stored - 4 bytes:	None

5.2 SE Image Apply

This command can be used to perform an upgrade of the SE firmware where the existing firmware will be overwritten with the one stored in the internal flash if the upgrade image is valid and applicable. The system is restarted and no response code is returned if the SE image is successfully upgraded.

Note: This command is only available on SE firmware version ≥ v1.2.2 (xG21 or xG22 devices).

Table 5.2. SE Image Apply Command

ID [31:16]	Option 1 [15:8]	Option 2 [7:0]	Command payload	Response payload
0x4303	0x00	0x00	Address in internal flash where the SE upgrade image is stored - 4 bytes:	None

5.3 Apply Lock

This command enables the debug lock for the part.

Table 5.3. Apply Lock Command

ID [31:16]	Option 1 [15:8]	Option 2 [7:0]	Command payload	Response payload
0x430C	0x00	0x00	None	None

5.4 Enable Secure Debug

This command enables the secure debug functionality. This command must be used before the debug port is locked and will fail if executed after locking debug access.

Table 5.4. Enable Secure Debug Command

ID [31:16]	Option 1 [15:8]	Option 2 [7:0]	Command payload	Response payload
0x430D	0x00	0x00	None	None

5.5 Disable Secure Debug

This command disables the secure debug functionality and is available even after the debug port has been locked.

Table 5.5. Disable Secure Debug Command

ID [31:16]	Option 1 [15:8]	Option 2 [7:0]	Command payload	Response payload
0x430E	0x00	0x00	None	None

5.6 Erase Device

This command performs a device mass erase and resets the debug configuration to its initial unlocked state. It is only available if the Disable Device Erase command has not been executed.

This command clears and verifies the main flash and RAM of the system, excluding the user data and one-time programmable (OTP) commissioning information in the SE.

Table 5.6. Erase Device Command

ID [31:16]	Option 1 [15:8]	Option 2 [7:0]	Command payload	Response payload
0x430F	0x00	0x00	None	None

5.7 Disable Device Erase

This command disables the Erase Device command. This command does not lock the debug interface to the part, but it is a permanent action for the part. This is a one-time command.

Table 5.7. Disable Device Erase Command

ID [31:16]	Option 1 [15:8]	Option 2 [7:0]	Command payload	Response payload
0x4310	0x00	0x00	None	None

5.8 Read Serial Number

This command is used to read the Silicon Labs-provisioned serial number of the device.

Table 5.8. Read Serial Number Command

ID [31:16]	Option 1 [15:8]	Option 2 [7:0]	Command payload	Response payload
0xFE00	0x00	0x00	None	16 bytes serial number

5.9 Get Status

This command is used to read out the status information from the SE.

Table 5.9. Get Status Command

ID [31:16]	Option 1 [15:8]	Option 2 [7:0]	Command payload	Response payload
0xFE01	0x00	0x00	None	Varies by SE type, see following.

VSE-SVM - total 20 bytes:

- Boot status - 4 bytes
- VSE firmware version - 4 bytes
- MCU firmware version - 4 bytes
- Debug lock status - 4 bytes
- Secure boot configuration - 4 bytes

HSE-SVM or HSE-SVH - total 36 bytes:

- 16 bytes for HSE-SVM: Reserved
- 16 bytes for HSE-SVH:
  - Tamper status - 4 bytes
  - Tamper time stamp - 4 bytes
  - Tamper raw status - 4 bytes
  - Time stamp - 4 bytes
- Boot status - 4 bytes
- HSE firmware version - 4 bytes
- MCU firmware version - 4 bytes
- Debug lock status - 4 bytes
- Secure boot configuration - 4 bytes

Note:

- Tamper status is a set of 32 flags that indicate which tamper events have occurred.
- Tamper time stamp is a HSE timer counter value for the last tamper event.
- Tamper raw status is encoded the same as tamper status but is an immediate value of the tamper event sources.
- The time stamp is a HSE timer counter value.
- Boot status:
  - Bit [7:0] - 0x20 if boot is successful.
  - Bit [15:8] (for xG21 or xG22 devices only) - The response code if SE firmware version ≥ v1.2.0
- VSE or HSE firmware version:
  - Bit [7:0] - Patch version
  - Bit [15:8] - Minor version
  - Bit [23:16] - Major version
  - Bit [31:24] - Series 2 device family (0 for xG21, 1 for xG22, 2 for xG23, etc.)
- MCU firmware version: Bit [31:0] - The MCU firmware version is not available if all set to 1 (0xFFFFFFFF)
- Debug lock status:
  - Bit [0] - Debug lock (configuration status) is enabled if set.
  - Bit [1] - Device erase is enabled if set.
  - Bit [2] - Secure debug is enabled if set.
  - Bit [5] - Debug lock (hardware status) is enabled if set.
- Secure boot configuration:
  - Bit [31:0] - SE OTP is not yet configured if all set to 1 (0xFFFFFFFF)
  - Bit [31:0] - SE OTP has been configured if Bit [31:1] are 0, secure boot is enabled if Bit [0] is set

5.10 Read User Configuration

This command is used to read non-reconfigurable user settings on the SE OTP for secure boot and tamper response.

Note: This command is only available on SE firmware versions ≥ v1.2.2 (xG21 or xG22 devices).

Table 5.10. Read User Configuration Command

ID [31:16]	Option 1 [15:8]	Option 2 [7:0]	Command payload	Response payload
0xFE04	0x00	0x00	None	Varies by SE type, see following.

VSE-SVM - total 4 bytes:
- MCU flags - 4 bytes

HSE-SVM - total 24 bytes:
- MCU flags - 4 bytes
- Reserved - 20 bytes

HSE-SVH - total 24 bytes:
- MCU flags - 4 bytes
- Tamper response levels (2 signals per byte) - 16 bytes
- Filter reset period - 1 byte
- Filter trigger threshold - 1 byte
- Tamper flags - 1 byte
- Tamper reset threshold - 1 byte

5.11 Initialize OTP

This command is used during factory initialization, to upload device-specific settings to the SE OTP. This is a one-time command.

Table 5.11. Initialize OTP Command

ID [31:16]	Option 1 [15:8]	Option 2 [7:0]	Command payload	Response payload
0xFF00	0x00	0x01	Varies by SE type, see following.	None

VSE-SVM - total 12 bytes:
1. Parity (equal to item 3) - 4 bytes
2. Length of the following content - 4 bytes
3. MCU flags - 4 bytes

HSE-SVM - total 32 bytes:
1. Parity (the XOR of 32-bit words from item 3 and 4) - 4 bytes
2. Length of the following content - 4 bytes
3. MCU flags - 4 bytes
4. Reserved - 20 bytes

HSE-SVH - total 32 bytes:
1. Parity (the XOR of 32-bit words from item 3 to 8) - 4 bytes
2. Length of the following content - 4 bytes
3. MCU flags - 4 bytes
4. Tamper response levels (2 signals per byte) - 16 bytes
5. Filter reset period - 1 byte
6. Filter trigger threshold - 1 byte
7. Tamper flags - 1 byte
8. Tamper reset threshold - 1 byte

5.11.1 MCU Flags

The parameters of the MCU flags are described in the following tables.

Table 5.12. Parameters of MCU Flags

Fields	Description
Bit [15:0]	Reserved
Bit [16]	SECURE_BOOT_ENABLE
Bit [17]	SECURE_BOOT_VERIFY_CERTIFICATE
Bit [18]	SECURE_BOOT_ANTI_ROLLBACK
Bit [19]	SECURE_BOOT_PAGE_LOCK_NARROW
Bit [20]	SECURE_BOOT_PAGE_LOCK_FULL
Bit [31:21]	Reserved

Table 5.13. MCU Flags for Series 2 Devices

Name	Description
SECURE_BOOT_ENABLE	If set, verifies the image on the Cortex-M33 before releasing the Cortex-M33 from reset.
SECURE_BOOT_VERIFY_CERTIFICATE	If set, requires certificate-based signing of the host application.
SECURE_BOOT_ANTI_ROLLBACK	If set, prevents secure upgrading to a host image with a lower version than the image that is currently stored in flash.
SECURE_BOOT_PAGE_LOCK_NARROW	If set, locks flash pages that have been validated by the secure boot process to prevent re-flashing by means other than through the SE.
"	Write/erase locks pages from 0 through the page where the secure boot signature of the application is located, not including the last page if the signature is not on a page boundary.
SECURE_BOOT_PAGE_LOCK_FULL	If set, locks flash pages that have been validated by the secure boot process to prevent re-flashing by means other than through the SE.
"	Write/erase locks pages from 0 through the page where the secure boot signature of the application is located, including the last page if the signature is not on a page boundary.

5.11.2 Anti-Tamper Configuration

The 16 bytes of tamper response levels on HSE-SVH devices are described in the following tables.

Table 5.14. Tamper Source Response Level on HSE-SVH (xG21B) Devices

Tamper Response Level																															
Byte [Bit]	15 [7:4]	15 [3:0]	14 [7:4]	14 [3:0]	13 [7:4]	13 [3:0]	12 [7:4]	12 [3:0]	11 [7:4]	11 [3:0]	10 [7:4]	10 [3:0]	9 [7:4]	9 [3:0]	8 [7:4]	8 [3:0]	7 [7:4]	7 [3:0]	6 [7:4]	6 [3:0]	5 [7:4]	5 [3:0]	4 [7:4]	4 [3:0]	3 [7:4]	3 [3:0]	2 [7:4]	2 [3:0]	1 [7:4]	1 [3:0]	0 [7:4]	0 [3:0]
Tamper source	SE ICACHE	Digital glitch	SE debug	Secure lock	Voltage glitch rising	Voltage glitch falling	Temperature sensor	Decouple BOD	PRS7	PRS6	PRS5	PRS4	PRS3	PRS2	PRS1	PRS0	TRNG monitor	Self test	Reserved	OTP read	DCI authorization	Mailbox authorization	User secure boot	SE secure boot	SE software assertion	Reserved	SE hard fault	SE RAM CRC	Reserved	SE watchdog	Filter counter	Reserved
Level	7 only	1/2/4/7	1/2/4/7	7 only	1/2/4/7	1/2/4/7	1/2/4/7	7 only	1/2/4/7	1/2/4/7	1/2/4/7	1/2/4/7	1/2/4/7	1/2/4/7	1/2/4/7	1/2/4/7	1/2/4/7	7 only	—	7 only	1/2/4/7	1/2/4/7	1/2/4/7	7 only	7 only	—	7 only	7 only	—	7 only	1/2/4/7	—

Table 5.15. Tamper Source Response Level on Other HSE-SVH Devices

Tamper Response Level																															
Byte [Bit]	15 [7:4]	15 [3:0]	14 [7:4]	14 [3:0]	13 [7:4]	13 [3:0]	12 [7:4]	12 [3:0]	11 [7:4]	11 [3:0]	10 [7:4]	10 [3:0]	9 [7:4]	9 [3:0]	8 [7:4]	8 [3:0]	7 [7:4]	7 [3:0]	6 [7:4]	6 [3:0]	5 [7:4]	5 [3:0]	4 [7:4]	4 [3:0]	3 [7:4]	3 [3:0]	2 [7:4]	2 [3:0]	1 [7:4]	1 [3:0]	0 [7:4]	0 [3:0]
Tamper source	PRS6	PRS5	PRS4	PRS3	PRS2	PRS1	PRS0	DPLL rise	DPLL fall	Temperature sensor	BOD	SE RAM ECC1	SE ICACHE	Voltage glitch	Digital glitch	Secure lock	TRNG monitor	Self test	Reserved	OTP read	DCI authorization	Mailbox authorization	User secure boot	SE secure boot	SE software assertion	Reserved	SE hard fault	SE RAM ECC2	Reserved	SE watchdog	Filter counter	Reserved
Level	1/2/4/7	1/2/4/7	1/2/4/7	1/2/4/7	1/2/4/7	1/2/4/7	1/2/4/7	1/2/4/7	1/2/4/7	1/2/4/7	7 only	1/2/4/7	7 only	1/2/4/7	1/2/4/7	7 only	1/2/4/7	7 only	—	7 only	1/2/4/7	1/2/4/7	1/2/4/7	7 only	7 only	—	7 only	7 only	—	7 only	1/2/4/7	—

Other anti-tamper settings on HSE-SVH devices are described in the following table.

Table 5.16. Anti-Tamper Configuration Settings

Setting	Value
Filter reset period	0 to 31
Filter trigger threshold	0 to 7
Tamper flag	Bit [1] - Digital glitch detector is always on if set
Tamper flag (not available on EFR32xG21B devices)	Bit [2] - Tamper module keeps running at sleep mode if set
Tamper reset threshold	0 to 255

For more information about tamper source and anti-tamper configuration, see sections "*Tamper Sources*" and "*Anti-Tamper Configuration*" in AN1247: Anti-Tamper Protection Configuration and Use.

5.12 Initialize Public Key

This command is used to initialize the user public key(s) to the SE OTP. This is a one-time command.

Table 5.17. Initialize Public Key Command

ID [31:16]	Option 1 [15:8]	Option 2 [7:0]	Command payload	Response payload
0xFF07	Key type, see following	0x01	See following	None

Key type:

- 0x01 - Public Sign Key
- 0x02 - Public Command Key

Command payload - total 68 bytes:

1. Parity (the XOR of 32-bit words from item 2) - 4 bytes
2. Public key in option 1 - 64 bytes

5.13 Read Public Key

This command can be used to read out one of the public keys that are permanently stored in SE OTP.

Table 5.18. Read Public Key Command

ID [31:16]	Option 1 [15:8]	Option 2 [7:0]	Command payload	Response payload
0xFF08	Key type, see following	0x01	None	64 bytes: public key in option 1

Key type:

- 0x01 - Public Sign Key
- 0x02 - Public Command Key

5.14 Initialize AES Key

This command is used to initialize a 128-bit symmetric key to the SE OTP. This is a one-time command.

Note: This command is only available on HSE devices.

Table 5.19. Initialize AES Key Command

ID [31:16]	Option 1 [15:8]	Option 2 [7:0]	Command payload	Response payload
0xFF0B	Key type, see following	0x01	See following	None

Key type:

- 0x05 - AES-128 key

Command payload - total 20 bytes:

1. Parity (the XOR of 32-bit words from item 2) - 4 bytes
2. Symmetric key in option 1 - 16 bytes

5.15 Set Debug Restrictions

This command is used to set the restrictions for the debug port.

Table 5.20. Set Debug Restrictions Command

ID [31:16]	Option 1 [15:8]	Option 2 [7:0]	Command payload	Response payload
0x4312	0x00	0x00	See following	None

Command Payload – total 4 bytes. Debug restriction bit mask is defined in the table below.

Table 5.21. Debug Port Restriction Bits

Bit	Name	Description
0	DBGLOCK	Non-secure, invasive debug lock. If this bit is set, it is not possible to debug the non-secure state in a way that is intrusive to program execution.
1	NIDLOCK	Non-secure, non-invasive debug lock. If this bit is set, it is not possible to observe the non-secure state of the M33 using trace.
2	SPIDLOCK	Secure, invasive debug lock. If this bit is set, it is not possible to debug the secure state in a way that is intrusive to program execution.
3	SPNIDLOCK	Secure, non-invasive debug lock. If this bit is set, it is not possible to observe the secure state of the M33 using trace. (If SPIDLOCK is open, SPNIDLOCK will also remain open.)

5.16 Read Lock Status

This command is used to read the lock status of the debug port.

Table 5.22. Read Lock Status Command

ID [31:16]	Option 1 [15:8]	Option 2 [7:0]	Command payload	Response payload
0x4311	0x00	0x00	See following	None

Debug port lock status – total 4 bytes:

- Bit [0] - Debug lock (configuration status) is enabled if set
- Bit [1] - Device erase enabled is enabled if set
- Bit [2] – Secure debug lock enabled is enabled if set
- Bit [3:4] – Reserved
- Bit [5] – Debug lock hardware status is enabled if set
- Bit [6] – Invasive debug lock is enabled if set
- Bit [7] – Non-invasive debug lock is enabled if set
- Bit [8] - Secure invasive debug lock is enabled if set
- Bit [9] - Secure non-invasive debug lock is enabled if set

6. Series 2 DCI and SWD Programming Examples

The programming examples in this application note use the Series 2 DCI and SWD Programming platform example of GSDK v4.0.1. The hardware and software implementation may be different on other versions of the GSDK.

For more information about production programming steps, see section "*Overview*" in AN1222: Production Programming of Series 2 Devices.

6.1 Hardware Overview

The Wireless Starter Kit (WSTK) with the BRD4182A Radio Board (EFR32MG22C224F512IM40) is used as the hardware platform of the DCI and SWD programmer.

The programmer uses GPIO to emulate the DCI and SWD to program the target device. The UART interface handles the user interface. The VCOM_TX and VCOM_RX are routed to the WSTK virtual COM port by setting the VCOM_ENABLE line high.

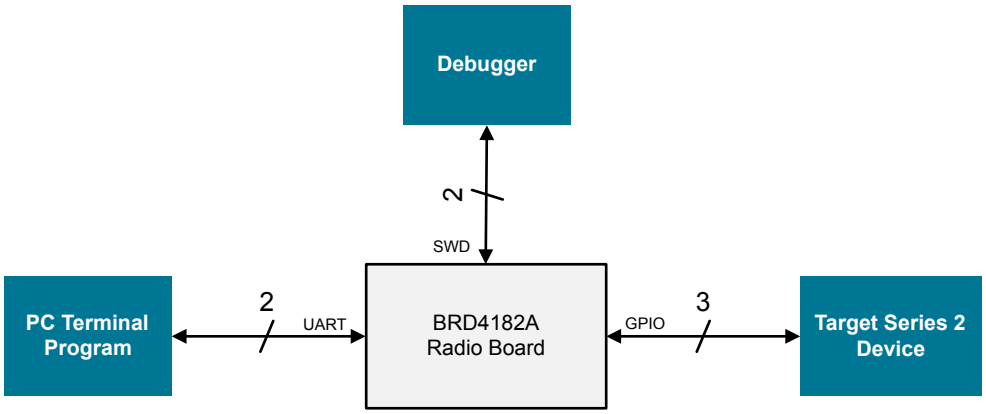


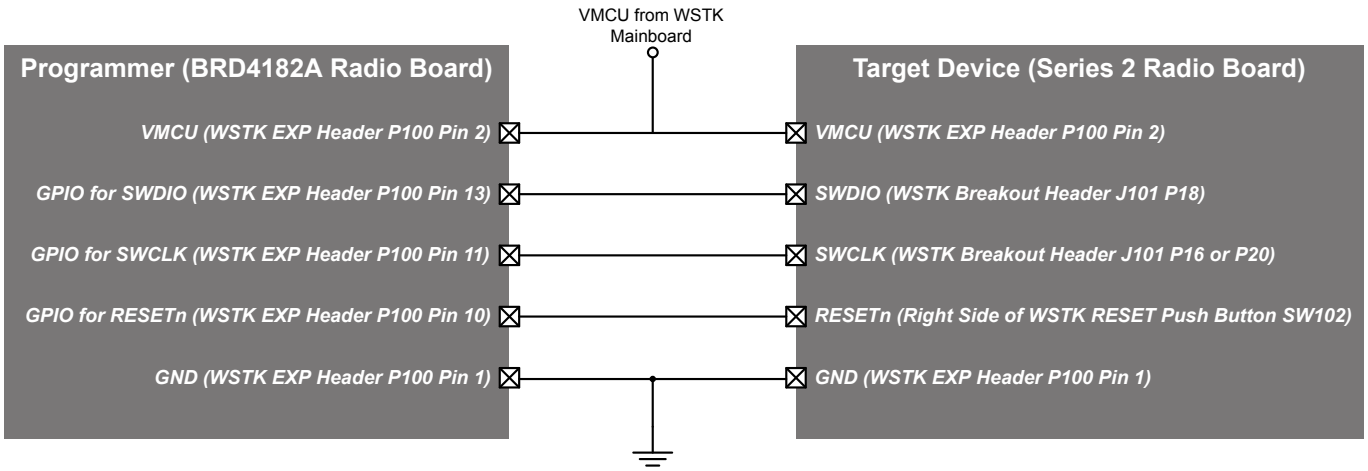
Figure 6.1. Block Diagram of DCI and SWD Programmer

Table 6.1. Resources of BRD4182A Used by Programmer

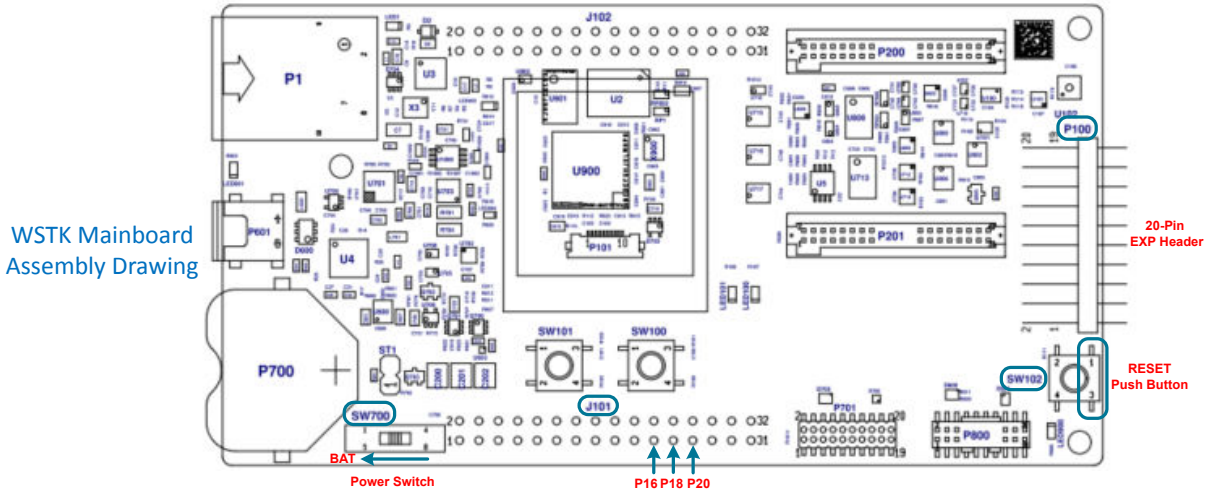
GPIO (SoC Peripheral)	WSTK Peripheral	Function	EXP Header Connection
PA01 (DBG_SWCLK)	DBG_TCK_SWCLK	Programmer debug SWCLK	—
PA02 (DBG_SWDIO)	DBG_TCK_SWDIO	Programmer debug SWDIO	—
PA05 (US1_TX)	VCOM_TX	WSTK Virtual COM port TX	Pin 12
PA06 (US1_RX)	VCOM_RX	WSTK Virtual COM port RX	Pin 14
PB04	VCOM_ENABLE	WSTK Virtual COM port enable	—
PC03	—	Target device RESETn	Pin 10
PD02	UIF_LED0	Target device SWCLK (shared with WSTK LED0)	Pin 11
PD03	UIF_LED1	Target device SWDIO (shared with WSTK LED1)	Pin 13

The following figure shows the interconnection between programmer and target device. The programmer is the WSTK with the BRD4182A Radio Board and the target device is the WSTK with one of the Series 2 radio boards below.

- EFR32MG21A — BRD4181A (J101 P16 for SWCLK)
- EFR32MG21B — BRD4181C (J101 P16 for SWCLK)
- EFR32MG22 — BRD4182A (J101 P16 for SWCLK)
- EFR32FG23A — BRD4263B (J101 P20 for SWCLK)
- EFR32FG23B — BRD4263C (J101 P20 for SWCLK)



The RESETn pin of the target device has an internal pullup to the DVDD supply. If external circuitry drives RESETn above DVDD, additional current may flow into the pin due to this pullup.



The programmer powers the target device on the Series 2 radio board through VMCU. Therefore power switch (SW700) of the target device WSTK should be in the BAT position.

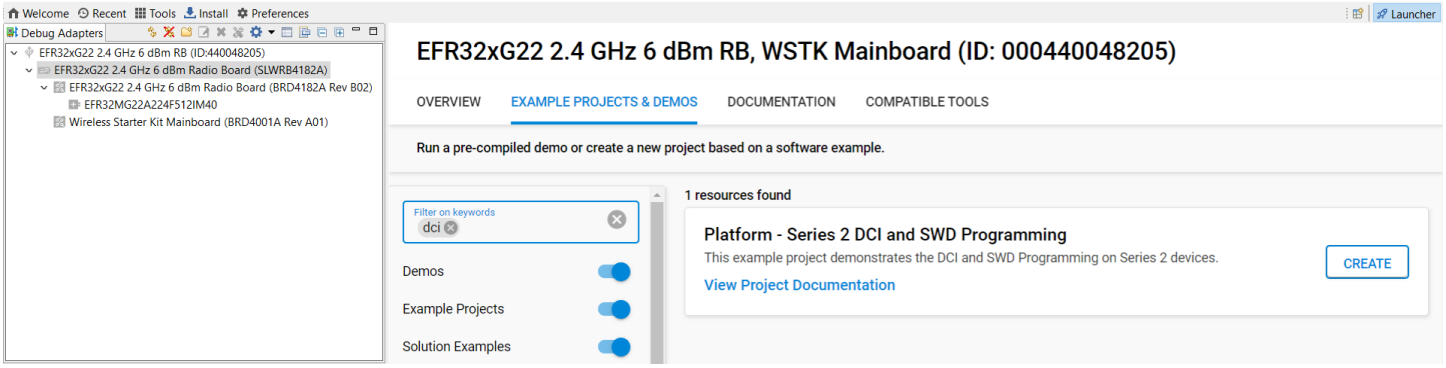
Figure 6.2. Programmer Connection Diagram

The programmer can work on other Series 2 radio boards or kits. However, the following limitations may apply.

- Some firmware images (`xg2*_app_image[]` and `xg2*_signed_imag[]`) in Table 6.4 Hard-coded Firmware Images on page 26 are hardware-dependent (GPIOs for LEDs, push buttons, and UART). Therefore, they may not function properly on other radio boards or kits.
- The SWCLK and SWDIO of the target radio boards or kits may be on different pins or connectors. Therefore, the connection diagram above may not apply (refer to the board-specific schematics for details).

6.2 Software Overview

Users must select the BRD4182A radio board to access the `Series 2 DCI and SWD Programming` platform example in Simplicity Studio 5. Click the `View Project Documentation` link to open the `readme` file. This file includes the procedures to create the project and run the example.



6.2.1 Bit-Bang

The main overhead on writing directly to MSC registers is to emulate the SWCLK and SWDIO signals by bit-banging GPIO pins. The programmer must use the GPIOs in the same port group (0-7 or 8-15) for SWCLK and SWDIO to speed up this emulation process. The software writes to the entire port at once when bit-banging the SWCLK and SWDIO signals. Therefore, the programmer cannot use the spare pins of this port for other purposes.

6.2.2 Security Keys

The following table describes the security keys (hard-coded in `app_dci_task.c`) used in the programming examples. Users can modify these keys to adapt to the application requirements. The files (`.prv` and `.pem`) of security keys can be found in the Windows folder `C:\SiliconLabs\SimplicityStudio\v5\developer\adapter_packs\secmgr\scripts\offline`.

Table 6.2. Hard-coded Security Keys

Array	Usage	Source
aes_key[]	Decrypt GBL payloads	A 16 bytes AES-128 key in encrypt-unsafe-key.prv (binary file).
public_sign_key[]	Secure boot	A 64 bytes Public Sign Key, the corresponding Private Sign Key in root-sign-unsafe-privkey.pem.
public_command_key[]	Secure debug unlock and Disable tamper	A 64 bytes Public Command Key, the corresponding Private Command Key in cmd-unsafe-privkey.pem.

The public key can be derived from the private key by using OpenSSL.

```
openssl ec -in rootsign-unsafe-privkey.pem -pubout -text > public_sign_key.txt
```

```
openssl ec -in cmd-unsafe-privkey.pem -pubout -text > public_command_key.txt
```

6.2.3 OTP Settings

The following table describes the OTP settings (hard-coded in `app_dci_task.c`) used in the programming examples. Users can modify these values to adapt to the application requirements.

Table 6.3. Hard-coded OTP Settings

Array	OTP Settings	Value
vse_svm_conf[]	VSE-SVM	MCU flags: Secure boot enable and Secure boot anti-rollback (0x00050000)
hse_svm_conf[]	HSE-SVM	MCU flags: Secure boot enable and Secure boot anti-rollback (0x00050000)
"	"	Reserved: 0x00000000, 0x00000000, 0x00000000, 0x00000000
hse_svh_xg21b_conf[]	HSE-SVH (xG21B)	MCU flags: Secure boot enable and Secure boot anti-rollback (0x00050000)
"	"	Tamper source levels: 0x40440410, 0x14040104, 0x77442211, 0x42042224
"	"	Filter reset period: 10
"	"	Filter trigger threshold: 6
"	"	Tamper flags: 0
"	"	Tamper reset threshold: 5
hse_svh_other_conf[]	HSE-SVH (others)	MCU flags: Secure boot enable and Secure boot anti-rollback (0x00050000)
"	"	Tamper source levels: 0x40440410, 0x14040104, 0x22414224, 0x74422112
"	"	Filter reset period: 10
"	"	Filter trigger threshold: 6
"	"	Tamper flags: 0
"	"	Tamper reset threshold: 5

Note: Refer to the tables below for details about OTP settings.

- Table 5.12 Parameters of MCU Flags on page 17
- Table 5.14 Tamper Source Response Level on HSE-SVH (xG21B) Devices on page 18
- Table 5.15 Tamper Source Response Level on Other HSE-SVH Devices on page 18
- Table 5.16 Anti-Tamper Configuration Settings on page 18

6.2.4 Firmware images

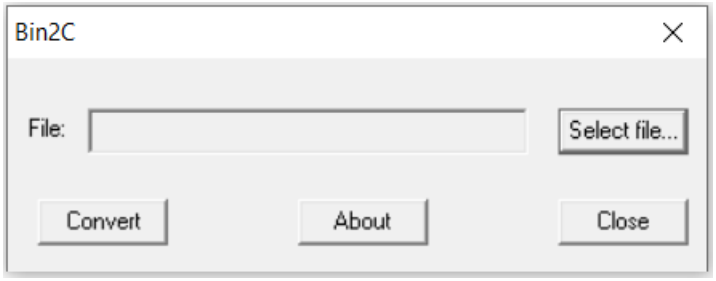
The following table describes the firmware images (hard-coded in `app_firmware_image.c`) used in the programming examples. Users can modify these images to adapt to the application requirements.

Table 6.4. Hard-coded Firmware Images

Array	Size (Bytes)	Usage
xg21_hse_image[]	41125	The xG21 HSE upgrade firmware image (v1.2.9)
xg21_app_image[]	9212	The xG21 application firmware image (Blink Bare-metal platform example)
xg21_signed_image[]	11192	A signed xG21 UART XMODEM Bootloader image (v1.12.0)
erase_xg21_userdata[]	3068	Application firmware image to erase xG21 user data
write_xg21_userdata[]	4228	Application firmware image to program xG21 user data
prog_xg21_hse_upgrade[]	6956	Application firmware image to upgrade xG21 HSE firmware
xg22_vse_image[]	16549	The xG22 VSE upgrade firmware image (v1.2.7)
xg22_app_image[]	10756	The xG22 application firmware image (Blink Bare-metal platform example)
xg22_signed_image[]	14128	A signed xG22 UART XMODEM Bootloader image (v1.12.0)
prog_xg22_vse_upgrade[]	9320	Application firmware image to upgrade xG22 VSE firmware
xg23_hse_image[]	88221	The xG23 HSE upgrade firmware image (v2.1.4)
xg23_app_image[]	13560	The xG23 application firmware image (Blink Bare-metal platform example)
xg23_signed_image[]	16248	A signed xG23 UART XMODEM Bootloader image (v1.12.0)
prog_xg23_hse_upgrade[]	11696	Application firmware image to upgrade xG23 HSE firmware

Note:

- The HSE or VSE upgrade firmware image must be stored to the device's internal flash in .seu format.
- The firmware image (`.seu` format) can be converted to a C source file using the SEGGER free utility `Bin2C.exe`. The last **NULL** (`0x00`) character in the converted firmware image array should be discarded.



- The programmer uses the bootloader image signed by the Private Sign Key (rootsign-unsafe-privkey.pem) to recover a secure boot failure device.
- Refer to 7. Add a New Series 2 Device to the Programmer for instructions on adding firmware images for newer devices.

6.2.5 Compile Options

The following tables describe the compile options in header files to set up the software and hardware environment for the programming examples.

Table 6.5. Compile Options in app_dci_swd.h

Parameter	Usage	Default Setting
RESET_PULSE	Pin reset pulse width in microseconds	1000 µs (1 ms)
RESET_DELAY	Delay in microseconds after issuing a soft or pin reset	50000 µs (50 ms)
DCI_RETRY_COUNT	Number of times to retry a DCI read or write operation. It must be high enough to receive a response from the SE command.	1001000
SWCLK_PORT	GPIO port for SWCLK	3 (Port D)
SWCLK_PIN	GPIO pin for SWCLK	2 (PD02)
SWDIO_PORT	GPIO port for SWDIO	3 (Port D)
SWDIO_PIN	GPIO pin for SWDIO	3 (PD03)
RESET_PORT	GPIO port for RESETn	2 (Port C)
RESET_PIN	GPIO pin for RESETn	3 (PC03)

Table 6.6. Compile Options in app_firmware_image.h

Parameter	Usage	Default Setting
SE_START_ADDR	SE upgrade firmware image start address (aligned with 8 kB flash page size)	0x00060000
SE_UPGRADE_DELAY	Delay in microseconds after issuing a command to upgrade the SE firmware	2500000 µs (2.5 s)
USER_DATA_DELAY	Delay in microseconds after issuing a soft reset to run the application to erase or write the user data on xG21 devices	1000000 µs (1 s)

Note: Make sure the device has enough flash space to accommodate the SE firmware image when setting the SE_START_ADDR.

Table 6.7. Compile Options in app_swd_task.h

Parameter	Usage	Default Setting
ERASE_DELAY	Delay in microseconds after a flash page erase or mass erase	12000 µs (12 ms)
ERASE_LOOPCNT	It must be high enough to cover the page erase or mass erase time	1000
SKIP_POLLING	Skip polling WDATAREADY bit in the MSC STATUS register after writing 32-bit word to target device flash	1 (Skip)
WRITE_DELAY	Fix delay in microseconds after writing a 32-bit word to target device flash (if SKIP_POLLING = 1)	11 µs

Note:

- Refer to 4.1 Flash Erase for details about page erase and mass erase.
- Refer to 4.2 Flash Write for details about `SKIP_POLLING` and `WRITE_DELAY`.

6.3 Menu Operation

The SPACE and ENTER presses from the terminal program are used to manipulate the menu of the programmer.

Interface Menu

```
Series 2 DCI and SWD Programming Examples - Core running at 80000 kHz.
  . Current interface selection is DEBUG CHALLENGE INTERFACE (DCI).
  + Press SPACE to pick an interface (DCI/SWD), press ENTER to select the task of the chosen interface.
```

DCI Menu

```
. Current DCI task is GET SE STATUS.
  + Press SPACE to cycle through the DCI tasks, press ENTER to run the selected DCI task.
  + Current DCI task is READ SE OTP CONFIGURATION.
  + Current DCI task is READ SERIAL NUMBER.
  + Current DCI task is READ PUBLIC SIGN KEY.
  + Current DCI task is READ PUBLIC COMMAND KEY.
  + Current DCI task is READ LOCK STATUS.
  + Current DCI task is SET DEBUG RESTRICTIONS.
  + Current DCI task is ENABLE SECURE DEBUG.
  + Current DCI task is DISABLE SECURE DEBUG.
  + Current DCI task is LOCK DEVICE.
  + Current DCI task is ERASE DEVICE (UNLOCK).
  + Current DCI task is RECOVER SECURE BOOT FAILURE DEVICE.
  + Current DCI task is UPGRADE SE FIRMWARE THROUGH DCI.
  + Current DCI task is INITIALIZE AES-128 KEY (HSE).
  + Current DCI task is INITIALIZE PUBLIC SIGN KEY.
  + Current DCI task is INITIALIZE PUBLIC COMMAND KEY.
  + Current DCI task is INITIALIZE SE OTP.
  + Current DCI task is DISABLE DEVICE ERASE.
```

From task ENABLE SECURE DEBUG to task UPGRADE SE FIRMWARE THROUGH DCI:

```
  + Current DCI task is ENABLE SECURE DEBUG.
  + Press ENTER to confirm or press SPACE to abort.
```

From task INITIALIZE AES-128 KEY (HSE) to task DISABLE DEVICE ERASE:

```
  + Current DCI task is DISABLE DEVICE ERASE.
  + Warning: This is a ONE-TIME command and the operation is IRREVERSIBLE!
  + Press ENTER to confirm or press SPACE to abort.
```

SWD Interface Menu

```
Series 2 DCI and SWD Programming Examples - Core running at 80000 kHz.
  . Current interface selection is DEBUG CHALLENGE INTERFACE (DCI).
  + Press SPACE to pick an interface (DCI/SWD), press ENTER to select the task of the chosen interface.
  + Current interface selection is SERIAL WIRE DEBUG (SWD) INTERFACE.

  . Current SWD task is ERASE MAIN FLASH.
  + Press SPACE to cycle through the SWD tasks, press ENTER to run the selected SWD task.
  + Current SWD task is PROGRAM MAIN FLASH.
  + Current SWD task is ERASE USER DATA.
  + Current SWD task is PROGRAM USER DATA.
  + Current SWD task is UPGRADE SE FIRMWARE THROUGH APPLICATION FIRMWARE.
```

```
  + Current SWD task is ERASE MAIN FLASH.
  + Press ENTER to confirm or press SPACE to abort.
```

Error Handling

```
  + Press ENTER to issue a pin reset to the target, press SPACE to skip.
```

6.4 DCI Programming Examples

The following DCI programming examples are based on EFR32MG21B (BRD4181C) as the target device (Figure 6.2 Programmer Connection Diagram on page 23). Software is compiled with -O2 optimization in Simplicity IDE of Simplicity Studio 5.

Get SE Status

- To get the current SE status.

```
. Current DCI task is GET SE STATUS.
+ Press SPACE to cycle through the DCI tasks, press ENTER to run the selected DCI task.

. Read target device SE status... OK
+ SE firmware version  : 00010209
+ MCU firmware version : NA
+ Debug lock           : Disabled
+ Debug lock state     : False
+ Device Erase         : Enabled
+ Secure debug         : Disabled
+ Secure boot          : Disabled
+ Boot status          : 0x20 - Command successful.
```

Read SE OTP Configuration

- To read the current SE OTP configuration.

```
. Current DCI task is READ SE OTP CONFIGURATION.
+ Press SPACE to cycle through the DCI tasks, press ENTER to run the selected DCI task.

. Read target device user (SE OTP) configuration... OK
+ Secure boot                    : Disabled
+ Secure boot verify certificate : Disabled
+ Secure boot anti-rollback      : Disabled
+ Secure boot page lock narrow   : Disabled
+ Secure boot page lock full     : Disabled
+ Tamper source level
  Filter counter         :  1
  SE watchdog            :  4
  SE RAM CRC             :  4
  SE hard fault          :  4
  SE software assertion  :  4
  SE secure boot         :  4
  User secure boot       :  0
  Mailbox authorization  :  1
  DCI authorization      :  0
  OTP read               :  4
  Self test              :  4
  TRNG monitor           :  1
  PRS0                   :  1
  PRS1                   :  1
  PRS2                   :  2
  PRS3                   :  2
  PRS4                   :  4
  PRS5                   :  4
  PRS6                   :  7
  PRS7                   :  7
  Decouple BOD           :  4
  Temperature sensor     :  2
  Voltage glitch falling :  2
  Voltage glitch rising  :  2
  Secure lock            :  4
  SE debug               :  0
  Digital glitch         :  2
  SE ICACHE              :  4
+ Reset period for the tamper filter counter: ~32 ms x 1024
+ Activation threshold for the tamper filter: 4
+ Digital glitch detector always on: Disabled
+ Tamper reset threshold: 5
```

- SE will return the SE_RESPONSE_INVALID_COMMAND code if the SE OTP data has not been initialized or SE firmware version is less than v1.2.2 (xG21 or xG22 devices).

```
. Read target device user (SE OTP) configuration... Failed - Unsupported command.
+ Press ENTER to issue a pin reset to the target, press SPACE to skip.
```

Read Serial Number

- To read the serial number of the Series 2 device.

```
. Current DCI task is READ SERIAL NUMBER.
+ Press SPACE to cycle through the DCI tasks, press ENTER to run the selected DCI task.

. Read target device serial number... OK
+ The serial number (16 bytes): 000000000000000014B457FFFE0F77CE
```

Read Public Sign Key

- To read the Public Sign Key in SE OTP.

```
  . Current DCI task is READ PUBLIC SIGN KEY.
  + Press SPACE to cycle through the DCI tasks, press ENTER to run the selected DCI task.

  . Read target device public sign key... OK
  + The public sign key (64 bytes): C4AF4AC69AAB9512DB50F7A26AE5B4801183D85417E729A56DA974F4E08A562C
                                    DE6019DEA9411332DC1A743372D170B436238A34597C410EA177024DE20FC819
```

- SE will return the SE_RESPONSE_INTERNAL_ERROR code if the Public Sign Key has not been provisioned.

```
  . Read target device public sign key... Failed - Internal SE error.
  + Press ENTER to issue a pin reset to the target, press SPACE to skip.
```

Read Public Command Key

- To read the Public Command Key in SE OTP.

```
  . Current DCI task is READ PUBLIC COMMAND KEY.
  + Press SPACE to cycle through the DCI tasks, press ENTER to run the selected DCI task.

  . Read target device public command key... OK
  + The public command key (64 bytes): B1BC6F6FA56640ED522B2EE0F5B3CF7E5D48F60BE8148F0DC08440F0A4E1DCA4
                                       7C04119ED6A1BE31B7707E5F9D001A659A051003E95E1B936F05C37EA793AD63
```

- SE will return the SE_RESPONSE_INTERNAL_ERROR code if the Public Command Key has not been provisioned.

```
  . Read target device public command key... Failed - Internal SE error.
  + Press ENTER to issue a pin reset to the target, press SPACE to skip.
```

Read Lock Status

- To read the lock state of the debug port.

```
+ Current DCI task is READ LOCK STATUS.
 . Debug port lock state
 + Debug Locked, config status    : Disabled.
 + Device erase                   : Enabled.
 + Secure debug unlock            : Enabled.
 + Debug port                     : Unlocked.
 + Invasive Debug Lock            : Unlocked.
 + Non-invasive Debug Lock        : Unlocked.
 + Secure Invasive Debug Lock     : Unlocked.
 + Secure Non-invasive Debug Lock : Unlocked.
```

Set Debug Restrictions

- To set debug restrictions.

```
  + Setting the lockbit for Secure, Non-Invasive Debug

  . Set the debug restriction bits of the target device... OK

  . Read target device SE status... OK
  + SE firmware version  : 0001020E
  + MCU firmware version : NA
  + Debug lock           : Disabled
  + Debug lock state     : False
  + Device Erase         : Enabled
  + Secure debug         : Enabled
  + Secure boot          : Disabled and SE OTP is not configured
  + Boot status          : 0x20 - Command successful.

Series 2 DCI and SWD Programming Examples - Core running at 80000 kHz.
  . Current interface selection is DEBUG CHALLENGE INTERFACE (DCI).
  + Press SPACE to pick an interface (DCI/SWD), press ENTER to select the task of the chosen interface.
```

Enable Secure Debug

- To enable the secure debug functionality.

```
+ Current DCI task is ENABLE SECURE DEBUG.
+ Press ENTER to confirm or press SPACE to abort.

. Enable secure debug of the target device... OK

. Read target device SE status... OK
+ SE firmware version  : 00010209
+ MCU firmware version : NA
+ Debug lock           : Disabled
+ Debug lock state     : False
+ Device Erase         : Enabled
+ Secure debug         : Enabled
+ Secure boot          : Disabled
+ Boot status          : 0x20 - Command successful.
```

- SE will return the SE_RESPONSE_INVALID_PARAMETER code if the Public Command Key has not been provisioned.

```
. Enable secure debug of the target device... Failed - Parameters are invalid or buffer is too small.
+ Press ENTER to issue a pin reset to the target, press SPACE to skip.
```

- SE will return the SE_RESPONSE_INVALID_COMMAND code if the secure debug has already enabled.

```
. Enable secure debug of the target device... Failed - Unsupported command.
+ Press ENTER to issue a pin reset to the target, press SPACE to skip.
```

Disable Secure Debug

- To disable the secure debug functionality.

```
+ Current DCI task is DISABLE SECURE DEBUG.
+ Press ENTER to confirm or press SPACE to abort.

. Disable secure debug of the target device... OK

. Read target device SE status... OK
+ SE firmware version  : 00010209
+ MCU firmware version : NA
+ Debug lock           : Disabled
+ Debug lock state     : False
+ Device Erase         : Enabled
+ Secure debug         : Disabled
+ Secure boot          : Disabled
+ Boot status          : 0x20 - Command successful.
```

Lock Device

- To enable the debug lock of the Series 2 device.

```
+ Current DCI task is LOCK DEVICE.
+ Press ENTER to confirm or press SPACE to abort.

. Lock target device... OK

. Read target device SE status... OK
+ SE firmware version  : 00010209
+ MCU firmware version : NA
+ Debug lock           : Enabled
+ Debug lock state     : True
+ Device Erase         : Enabled
+ Secure debug         : Disabled
+ Secure boot          : Disabled
+ Boot status          : 0x20 - Command successful.
```

Erase Device (Unlock)

- To perform a device mass erase and unlock the standard debug lock of the Series 2 device.

```
+ Current DCI task is ERASE DEVICE (UNLOCK).
+ Press ENTER to confirm or press SPACE to abort.

. Erase (unlock) target device... OK

. Read target device SE status... OK
+ SE firmware version  : 00010209
+ MCU firmware version : NA
+ Debug lock           : Disabled
+ Debug lock state     : False
+ Device Erase         : Enabled
+ Secure debug         : Disabled
+ Secure boot          : Disabled
+ Boot status          : 0x20 - Command successful.
```

- SE will return the SE_RESPONSE_INVALID_COMMAND code if the Device Erase is disabled.

```
. Erase (unlock) target device... Failed - Unsupported command.
+ Press ENTER to issue a pin reset to the target, press SPACE to skip.
```

Recover Secure Boot Failure Device

- To get the current SE status.

```
. Current DCI task is GET SE STATUS.
+ Press SPACE to cycle through the DCI tasks, press ENTER to run the selected DCI task.

. Read target device SE status... OK
+ SE firmware version  : 00010209
+ MCU firmware version : NA
+ Debug lock           : Disabled
+ Debug lock state     : False
+ Device Erase         : Enabled
+ Secure debug         : Disabled
+ Secure boot          : Enabled
+ Boot status          : 0x14 - Failure while checking the host for secure boot.
```

- Issue a device erase to unlock the target device through DCI.

```
+ Current DCI task is RECOVER SECURE BOOT FAILURE DEVICE.
+ Press ENTER to confirm or press SPACE to abort.

. Issue a device erase through DCI.
+ Erase target device... OK
```

- A correctly-signed firmware image xg21_signed_image[] is programmed to the target device main flash through the SWD interface to recover a secure boot failure device.

```
+ Initialize DP... OK - IDCODE = 0x6BA02477
+ Read AP... OK - IDR = 0x84770001
+ Set up AHB-AP and halt target... OK
+ Get device information... OK - Target device is EFR32MG21B010F1024 and unique ID is 0x14B457FFFE0F7762

. Program a correctly-signed image to recover device.
+ The xG21 signed firmware image size is 11192 bytes and start address is 0x00000000.
+ Erase-Program-Verify the xG21 main flash for signed firmware image... OK (cycles: 13616519 time: 170 ms)

. Read target device SE status... OK
+ SE firmware version  : 00010209
+ MCU firmware version : 010C0000
+ Debug lock           : Disabled
+ Debug lock state     : False
+ Device Erase         : Enabled
+ Secure debug         : Disabled
+ Secure boot          : Enabled
+ Boot status          : 0x20 - Command successful.
```

- The device cannot be recovered if the image is not correctly-signed.

```
. Read target device SE status... OK
+ SE firmware version  : 00010209
+ MCU firmware version : 010C0000
+ Debug lock           : Disabled
+ Debug lock state     : False
+ Device Erase         : Enabled
+ Secure debug         : Disabled
+ Secure boot          : Enabled
+ Boot status          : 0x12 - Failure while checking the host for secure boot.
```

Upgrade SE Firmware Through DCI

- A SE upgrade firmware image xg21_hse_image[] is programmed to the target device main flash at SE_START_ADDR through the SWD interface. It will overwrite the data from `SE_START_ADDR` to `SE_START_ADDR` + the size of `xg21_hse_image[]`.

```
+ Current DCI task is UPGRADE SE FIRMWARE THROUGH DCI.
+ Press ENTER to confirm or press SPACE to abort.

. Program a SE firmware image to the target device.
+ Initialize DP... OK - IDCODE = 0x6BA02477
+ Read AP... OK - IDR = 0x84770001
+ Set up AHB-AP and halt target... OK
+ Get device information... OK - Target device is EFR32MG21B010F1024 and unique ID is 0x588E81FFFE7034E5
+ The xG21 HSE firmware image version: 00010209
+ The xG21 HSE firmware image size is 41125 bytes and start address is 0x00060000.
+ Erase-Program-Verify the xG21 main flash for HSE firmware image... OK (cycles: 52365323 time: 654 ms)
```

- Validate the SE upgrade firmware image and start the upgrade process if the image is valid.

```
. Validate SE firmware image in the target device... OK

. Upgrade SE firmware image, delay few seconds to check SE status... Done

. Read target device SE status... OK
+ SE firmware version  : 00010209
+ MCU firmware version : NA
+ Debug lock           : Disabled
+ Debug lock state     : False
+ Device Erase         : Enabled
+ Secure debug         : Disabled
+ Secure boot          : Disabled
+ Boot status          : 0x20 - Command successful.
```

- SE will return the SE_RESPONSE_INVALID_PARAMETER code if the SE upgrade firmware image is invalid.

```
. Validate SE firmware image in the target device... Failed - Parameters are invalid or buffer is too small.
+ Press ENTER to issue a pin reset to the target, press SPACE to skip.
```

- SE will return the SE_RESPONSE_INVALID_COMMAND code if the current SE firmware version is less than v1.2.2 (xG21 or xG22 devices).

```
. Validate SE firmware image in the target device... Failed - Unsupported command.
+ Press ENTER to issue a pin reset to the target, press SPACE to skip.
```

Initialize AES-128 Key (HSE)

- To initialize the aes_key[] to HSE OTP.

```
+ Current DCI task is INITIALIZE AES-128 KEY (HSE).
+ Warning: This is a ONE-TIME command and the operation is IRREVERSIBLE!
+ Press ENTER to confirm or press SPACE to abort.

. Initialize AES-128 key of the target device... OK
```

- SE will return the SE_RESPONSE_INVALID_PARAMETER code if the AES-128 key has already been initialized.

```
. Initialize AES-128 key of the target device... Failed - Parameters are invalid or buffer is too small.
+ Press ENTER to issue a pin reset to the target, press SPACE to skip.
```

Initialize Public Sign Key

- To initialize the public_sign_key[] to SE OTP.

```
+ Current DCI task is INITIALIZE PUBLIC SIGN KEY.
+ Warning: This is a ONE-TIME command and the operation is IRREVERSIBLE!
+ Press ENTER to confirm or press SPACE to abort.

. Initialize public sign key of the target device... OK

. Read target device public sign key... OK
+ The public sign key (64 bytes): C4AF4AC69AAB9512DB50F7A26AE5B4801183D85417E729A56DA974F4E08A562C
                                   DE6019DEA9411332DC1A743372D170B436238A34597C410EA177024DE20FC819
```

- SE will return the SE_RESPONSE_INVALID_PARAMETER code if the Public Sign Key has already been initialized.

```
. Initialize public sign key of the target device... Failed - Parameters are invalid or buffer is too small.
+ Press ENTER to issue a pin reset to the target, press SPACE to skip.
```

Initialize Public Command Key

- To initialize the public_command_key[] to SE OTP.

```
+ Current DCI task is INITIALIZE PUBLIC COMMAND KEY.
+ Warning: This is a ONE-TIME command and the operation is IRREVERSIBLE!
+ Press ENTER to confirm or press SPACE to abort.

. Initialize public command key of the target device... OK

. Read target device public command key... OK
+ The public command key (64 bytes): B1BC6F6FA56640ED522B2EE0F5B3CF7E5D48F60BE8148F0DC08440F0A4E1DCA4
                                      7C04119ED6A1BE31B7707E5F9D001A659A051003E95E1B936F05C37EA793AD63
```

- SE will return the SE_RESPONSE_INVALID_PARAMETER code if the Public Command Key has already been initialized.

```
. Initialize public command key of the target device... Failed - Parameters are invalid or buffer is too small.
+ Press ENTER to issue a pin reset to the target, press SPACE to skip.
```

Initialize SE OTP

- To initialize the hse_svh_xg21b_conf[] to the HSE OTP.

```
+ Current DCI task is INITIALIZE SE OTP.
+ Warning: This is a ONE-TIME command and the operation is IRREVERSIBLE!
+ Press ENTER to confirm or press SPACE to abort.

. Initialize SE OTP of the target device... OK

. Read target device user (SE OTP) configuration... OK
+ Secure boot                    : Enabled
+ Secure boot verify certificate : Disabled
+ Secure boot anti-rollback      : Enabled
+ Secure boot page lock narrow   : Disabled
+ Secure boot page lock full     : Disabled
+ Tamper source level
  Filter counter         :  1
  SE watchdog            :  4
  SE RAM CRC             :  4
  SE hard fault          :  4
  SE software assertion  :  4
  SE secure boot         :  4
  User secure boot       :  0
  Mailbox authorization  :  1
  DCI authorization      :  0
  OTP read               :  4
  Self test              :  4
  TRNG monitor           :  1
  PRS0                   :  1
  PRS1                   :  1
  PRS2                   :  2
  PRS3                   :  2
  PRS4                   :  4
  PRS5                   :  4
  PRS6                   :  7
  PRS7                   :  7
  Decouple BOD           :  4
  Temperature sensor     :  2
  Voltage glitch falling :  2
  Voltage glitch rising  :  2
  Secure lock            :  4
  SE debug               :  0
  Digital glitch         :  2
  SE ICACHE              :  4
+ Reset period for the tamper filter counter: ~32 ms x 1024
+ Activation threshold for the tamper filter: 4
+ Digital glitch detector always on: Disabled
+ Tamper reset threshold: 5
```

- SE will return the SE_RESPONSE_INVALID_COMMAND code if the SE OTP has already programmed.

```
. Initialize SE OTP of the target device... Failed - Unsupported command.
+ Press ENTER to issue a pin reset to the target, press SPACE to skip.
```

- SE will return the SE_RESPONSE_INVALID_PARAMETER code if secure boot option is enabled and the Public Sign Key has not been provisioned.

```
. Initialize SE OTP of the target device... Failed - Parameters are invalid or buffer is too small.
+ Press ENTER to issue a pin reset to the target, press SPACE to skip.
```

Disable Device Erase

- To disable the Erase Device command.

```
+ Current DCI task is DISABLE DEVICE ERASE.
+ Warning: This is a ONE-TIME command and the operation is IRREVERSIBLE!
+ Press ENTER to confirm or press SPACE to abort.

. Disable device erase of the target device... OK

. Read target device SE status... OK
+ SE firmware version  : 00010209
+ MCU firmware version : NA
+ Debug lock           : Disabled
+ Debug lock state     : False
+ Device Erase         : Disabled
+ Secure debug         : Enabled
+ Secure boot          : Disabled
+ Boot status          : 0x20 - Command successful.
```

6.5 SWD Programming Examples

The following SWD programming examples are based on EFR32MG21B (BRD4181C) and EFR32MG22 (BRD4182A) as target devices (Figure 6.2 Programmer Connection Diagram on page 23). Software is compiled with -O2 optimization in the Simplicity IDE of Simplicity Studio 5.

If the secure boot option is enabled in the SE OTP or bootloader, the application firmware must be signed.

Some operations are implemented by the SE Manager APIs in an application firmware that is programmed to the target device.

- For more information about SE Manager, see AN1190: Series 2 Secure Debug.
- The SE Manager is available in Gecko SDK Suite 3.0.0 or later.
- The SE Manager APIs are fully described in the Silicon Labs online documentation located at https://docs.silabs.com/gecko-platform/latest/service/api/group-sl-se-manager.

Erase Main Flash

- This example demonstrates the mass erase operation through the SWD interface.

```
+ Current SWD task is ERASE MAIN FLASH.
+ Press ENTER to confirm or press SPACE to abort.

. Connect to the target device through the SWD interface.
+ Initialize DP... OK - IDCODE = 0x6BA02477
+ Read AP... OK - IDR = 0x84770001
+ Set up AHB-AP and halt target... OK
+ Get device information... OK - Target device is EFR32MG21B010F1024 and unique ID is 0x14B457FFFE0F77CE

. Erase main flash of the target device... OK (cycles: 1511120 time: 18889 us)
```

Program Main Flash

- This example demonstrates the flash erase, flash write, and flash verify operations through the SWD interface. In this example, an application firmware image xg21_app_image[] is programmed to the target device main flash.

```
+ Current SWD task is PROGRAM MAIN FLASH.
+ Press ENTER to confirm or press SPACE to abort.

. Connect to the target device through the SWD interface.
+ Initialize DP... OK - IDCODE = 0x6BA02477
+ Read AP... OK - IDR = 0x84770001
+ Set up AHB-AP and halt target... OK
+ Get device information... OK - Target device is EFR32MG21B010F1024 and unique ID is 0x14B457FFFE0F77CE

. Program an application firmware image to the target device.
+ The xG21 application firmware image size is 9212 bytes and start address is 0x00000000.
+ Erase-Program-Verify the xG21 main flash for application firmware image... OK (cycles: 11388469 time: 142 ms)
+ Issue a soft reset to run the application firmware... OK
```

Erase User Data (xG21 Devices)

- For xG21 devices (BRD4181C), the user data can only be erased by issuing a command to the HSE through the SE Manager API.
- In this example, an application firmware image erase_xg21_userdata[] (see source code below) is programmed to the target device main flash.

```
#include "em_chip.h"
#include "em_cmu.h"
#include "sl_se_manager.h"
#include "sl_se_manager_util.h"

/***************************************************************************//**
 * Main function
 ******************************************************************************/
int main(void)
{
  // Command context
  sl_se_command_context_t cmd_ctx;

  // Switch SYSCLK to 38 MHz HFRCO
  CMU_HFRCODPLLBandSet(cmuHFRCODPLLFreq_38M0Hz);

  // Initialize SE Manager
  sl_se_init();

  // Erase user page
  sl_se_erase_user_data(&cmd_ctx);

  while (1) ;
}
```

- The original application firmware image on the application start address will be overwritten.
- Issue a soft reset to run the application above to erase the user data.

```
  + Current SWD task is ERASE USER DATA.
  + Press ENTER to confirm or press SPACE to abort.

  . Connect to the target device through the SWD interface.
  + Initialize DP... OK - IDCODE = 0x6BA02477
  + Read AP... OK - IDR = 0x84770001
  + Set up AHB-AP and halt target... OK
  + Get device information... OK - Target device is EFR32MG21B010F1024 and unique ID is 0x14B457FFFE0F77CE

  . Erase xG21 user data through application firmware.
  + The xG21 application firmware image size is 3068 bytes and start address is 0x00000000.
  + Erase-Program-Verify the xG21 main flash for application firmware image... OK (cycles: 4473314 time: 55916 us)
  + Issue a soft reset to run the application firmware to erase the xG21 user data... OK
```

Erase User Data (non xG21 Devices)

- For non xG21 devices (BRD4182A), the user data can be erased the same way as any page in the main flash.

```
  + Current SWD task is ERASE USER DATA.
  + Press ENTER to confirm or press SPACE to abort.

  . Connect to the target device through the SWD interface.
  + Initialize DP... OK - IDCODE = 0x6BA02477
  + Read AP... OK - IDR = 0x84770001
  + Set up AHB-AP and halt target... OK
  + Get device information... OK - Target device is EFR32MG22C224F512 and unique ID is 0x680AE2FFFE287808

  . Erase xG22 user data... OK (cycles: 2027306 time: 25341 us)
```

Program User Data (xG21 Devices)

- For xG21 devices (BRD4181C), the user data can only be written by issuing a command to the HSE through the SE Manager API.
- In this example, an application firmware image write_xg21_userdata[] (see source code below) is programmed to the target device main flash.

```
#include "em_chip.h"
#include "em_cmu.h"
#include "sl_se_manager.h"
#include "sl_se_manager_util.h"

// User data
SL_ALIGN(4) static const uint32_t user_data[256] SL_ATTRIBUTE_ALIGN(4) = {
  0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55,
  0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55,
  0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55,
  0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55,
  0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55,
  0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55,
  0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55,
  0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55,
  0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55,
  0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55,
  0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55,
  0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55,
  0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55,
  0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55,
  0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55,
  0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55,
  0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55,
  0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55,
  0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55,
  0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55,
  0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55,
  0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55,
  0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55,
  0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55,
  0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55,
  0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55,
  0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55,
  0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55,
  0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55,
  0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55,
  0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55,
  0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55, 0x55AA55AA, 0xAA55AA55,
};

/***************************************************************************//**
 * Main function
 ******************************************************************************/
int main(void)
{
  // Command context
  sl_se_command_context_t cmd_ctx;

  // Switch SYSCLK to 38 MHz HFRCO
  CMU_HFRCODPLLBandSet(cmuHFRCODPLLFreq_38M0Hz);

  // Initialize SE Manager
  sl_se_init();

  // Erase user data
  sl_se_erase_user_data(&cmd_ctx);

  // Write user data
  sl_se_write_user_data(&cmd_ctx, 0, (uint32_t *)user_data, sizeof(user_data));

  while (1) ;
}
```

- The original application firmware image on the application start address will be overwritten.
- Issue a soft reset to run the application above to write the user data.

```
+ Current SWD task is PROGRAM USER DATA.
+ Press ENTER to confirm or press SPACE to abort.

. Connect to the target device through the SWD interface.
+ Initialize DP... OK - IDCODE = 0x6BA02477
+ Read AP... OK - IDR = 0x84770001
+ Set up AHB-AP and halt target... OK
+ Get device information... OK - Target device is EFR32MG21B010F1024 and unique ID is 0x14B457FFFE0F77CE

. Program xG21 user data through application firmware.
+ The xG21 application firmware image size is 4228 bytes and start address is 0x00000000.
+ Erase-Program-Verify the xG21 main flash for application firmware image... OK (cycles: 5777749 time: 72221 us)
+ Issue a soft reset to run the application firmware to program the xG21 user data... OK
```

Program User Data (non xG21 Devices)

- For non xG21 devices (BRD4182A), the user data can be written the same way as any page in the main flash.

```
+ Current SWD task is PROGRAM USER DATA.
+ Press ENTER to confirm or press SPACE to abort.

. Connect to the target device through the SWD interface.
+ Initialize DP... OK - IDCODE = 0x6BA02477
+ Read AP... OK - IDR = 0x84770001
+ Set up AHB-AP and halt target... OK
+ Get device information... OK - Target device is EFR32MG22C224F512 and unique ID is 0x680AE2FFFE287808

. Program xG22 user data.
+ User data size is 1024 bytes and start address is 0x0FE00000.
+ Erase-Program-Verify the xG22 user data... OK (cycles: 2179418 time: 27242 us)
```

Upgrade SE Firmware Through Application Firmware

- For xG21 or xG22 target devices, the SE firmware can only be upgraded by issuing a command to the SE through the SE Manager API if the SE firmware version is less than v1.2.2.
- Connect to the target device through the SWD interface.
- An application firmware image prog_xg21_hse_upgrade[] (see source code below) is programmed to the target device main flash.

```
#include "em_chip.h"
#include "em_cmu.h"
#include "sl_se_manager.h"
#include "sl_se_manager_util.h"

// SE firmware image start address
#if defined(_SILICON_LABS_32B_SERIES_2_CONFIG) \
  && (_SILICON_LABS_32B_SERIES_2_CONFIG < 3)
#define SE_START_ADDR           (0x00060000UL)
#else
#define SE_START_ADDR           (0x08060000UL)
#endif

// Current SE firmware version
static uint32_t current_version;

// Upgrade SE firmware version
static volatile uint32_t upgrade_version;

/*******************************************************************************//**
 * Main function
 ******************************************************************************/
int main(void)
{
  // Command context
  sl_se_command_context_t cmd_ctx;

  // Switch SYSCLK to 38 MHz HFRCO
  CMU_HFRCODPLLBandSet(cmuHFRCODPLLFreq_38M0Hz);

  // Initialize SE Manager
  sl_se_init();

  // Get current SE firmware version
  if (sl_se_get_se_version(&cmd_ctx, &current_version) != 0) {
    goto exit;
  }

  // Get upgrade SE firmware version
  upgrade_version = *((uint32_t *)SE_START_ADDR + 3);

  // Check if upgrade version > current version
  if (upgrade_version <= current_version) {
    goto exit;
  }

#if !defined(CRYPTOACC_PRESENT)
  // Validate the SE firmware image
  if (sl_se_check_se_image(&cmd_ctx, (uint32_t *)SE_START_ADDR) != 0) {
    goto exit;
  }
#endif

  // Upgrade the SE firmware image
  sl_se_apply_se_image(&cmd_ctx, (uint32_t *)SE_START_ADDR);

exit:
  while (1) ;
}
```

- The original application firmware image on the application start address will be overwritten.

```
+ Current SWD task is UPGRADE SE FIRMWARE THROUGH APPLICATION FIRMWARE.
+ Press ENTER to confirm or press SPACE to abort.

. Connect to the target device through the SWD interface.
+ Initialize DP... OK - IDCODE = 0x6BA02477
+ Read AP... OK - IDR = 0x84770001
+ Set up AHB-AP and halt target... OK
+ Get device information... OK - Target device is EFR32MG21B010F1024 and unique ID is 0x60A423FFFEA0773C

. Program an application firmware image to the target device to upgrade the SE firmware.
+ The xG21 HSE upgrade application firmware image size is 6956 bytes and start address is 0x00000000.
+ Erase-Program-Verify the xG21 main flash for application to upgrade
  HSE firmware... OK (cycles: 8840993 time: 110 ms)
```

- A SE upgrade firmware image xg21_hse_image[] is programmed to the target device main flash at SE_START_ADDR. The application start address cannot overlap with SE_START_ADDR. This means the SE_START_ADDR must be greater than the application start address plus the size (6956 bytes) of the application firmware image (prog_xg21_hse_upgrade[]).

```
. Program a SE firmware image to the target device.
+ The xG21 HSE firmware image version: 00010209
+ The xG21 HSE firmware image size is 41125 bytes and start address is 0x00060000.
+ Erase-Program-Verify the xG21 main flash for HSE firmware image... OK (cycles: 52363897 time: 654 ms)
```

- Issue a pin reset to run the application firmware in prog_xg21_hse_upgrade[].
- Check the SE firmware version after a few seconds to verify the SE firmware has been upgraded.

```
+ Issue a pin reset to run the application to upgrade the SE firmware.
+ Delay few seconds to check SE status... Done

. Read target device SE status... OK
+ SE firmware version  : 00010209
+ MCU firmware version : NA
+ Debug lock           : Disabled
+ Debug lock state     : False
+ Device Erase         : Enabled
+ Secure debug         : Disabled
+ Secure boot          : Disabled and SE OTP is not configured
+ Boot status          : 0x20 - Command successful.
```

- The SE firmware cannot be downgraded so the upgrade will be ignored if SE firmware with the same or a lower version is applied to the device.

6.6 Benchmark

The application firmware in Table 6.4 Hard-coded Firmware Images on page 26 is replaced by a 256 kB image for benchmarking. The test results in the following table are based on the conditions below.

- Series 2 DCI and SWD Programming platform example of GSDK v4.0.1
- Erase, program, and verify the main flash
- Compile options are set to default values
- Software is compiled with -O2 in Simplicity IDE (GNU ARM v10.2.1) of Simplicity Studio 5
- The programmer (EFR32MG22C224F512IM40) is running at 80 MHz

Table 6.8. Erase-Program-Verify Time for Different Target Devices

Target Device	Application Firmware Image Size	Erase-Program-Verify Time
EFR32MG21B (BRD4181C)	256 kB (xg21_app_image[])	3.68 s
EFR32MG22 (BRD4182A)	256 kB (xg22_app_image[])	3.68 s
EFR32FG23B (BRD4263C)	256 kB (xg23_app_image[])	3.74 s

Note: The following section describes how to add firmware images for newer devices. Users requiring performance data for devices not listed here can add their own 256 kb image to perform the test.

7. Add a New Series 2 Device to the Programmer

The programmer in this application note uses EFR32MG22C224F512IM40 as a host controller. The defines in the following table for EFR32MG22C224F512 should usually apply to the new Series 2 devices.

Table 7.1. Defines for EFR32MG22C224F512

Item	Define	Value
Main flash base address	FLASH_MEM_BASE in efr32mg22c224f512im40.h	0x00000000
User data base address	USERDATA_BASE in efr32mg22c224f512im40.h	0x0FE00000
MSC base address	MSC_BASE in efr32mg22c224f512im40.h	0x40030000
Offset and bitfields of MSC registers	MSC Register Map in xG22 reference manual	—
DEVINFO base address	DEVINFO_BASE in efr32mg22c224f512im40.h	0x0FE08000
Offset and bitfields of DEVINFO registers	DEVINFO Register Map in xG22 reference manual	—
CMU base address	CMU_BASE in efr32mg22c224f512im40.h	0x40008000
Offset of CMU_CLKEN1_SET register	CMU Register Map in xG22 reference manual	0x00001068
Bitmask to enable MSC clock	CMU_CLKEN1_MSC in efr32mg22_cmu.h	0x00020000
AP IDR (Cortex-M33)	S2_AHBAP_ID in app_dci_swd.h	0x84770001
TAR wrap mask (Cortex-M33)	TAR_WRAP_1K in app_swd_task.h	0x3FF

Users need to define new items if any defines in the table above do not match the new Series 2 device. And users may require tuning the settings in 6.2.5 Compile Options for the new Series 2 device.

The following procedures describe how to add the xG23 device to the programmer.

1. Add `XG23_FAMILY` (`0x17` for 23) to `app_swd_task.h`. It is `0x15` for xG21, `0x16` for xG22, `0x17` for xG23, etc.

```
/// Device family of xG23
#define XG23_FAMILY     (0x00170000UL)
```

2. The items below are different from the EFR32MG22C224F512 after checking the xG23 header files (e.g., efr32fg23b010f512im48.h) and reference manual.
   - Main flash base address — Add `FLASH_BASE_XG23` to `app_swd_task.h`.

```
/// Flash start address of xG23
#define FLASH_BASE_XG23 (0x08000000UL)
```

   - Bitmask to enable MSC clock — Add `CLKEN1_MSC_XG23` to `app_swd_task.h`.

```
/// MSC bit of xG23 CMU_CLKEN1_SET
#define CLKEN1_MSC_XG23 (0x00010000UL)
```

3. Add code to `app_swd_task.c` to enable MSC clock and set flash base address.

```
// Enable MSC clock if device is xG23
if (buf0 == XG23_FAMILY) {
  write_mem((uint32_t)&(CMU->CLKEN1_SET), CLKEN1_MSC_XG23);
  // Check UDLOCKBIT on xG23
  if (read_mem((uint32_t)&(MSC->MISCLOCKWORD)) & MSC_MISCLOCKWORD_UDLOCKBIT) {
    RAISE(SWD_ERROR_USERDATA_LOCK);
  }
  // Flash start address for xG23
  flash_start_addr = FLASH_BASE_XG23;
}
```

4. Add `DEVICE_XG23` (ASCII code of character "3") to `app_process.h`. It is "1" for xG21, "2" for xG22, "3" for xG23, etc.

```
/// xG23 device
#define DEVICE_XG23             (0x33)
```

5. The hse_svm_conf[] can apply to xG23A (HSE-SVM) devices. Add `|| defined(DEVICE_XG23)` to `hse_svm_conf[]` in `app_dci_task.c`.

```
#if defined(DEVICE_XG21) || defined(DEVICE_XG23)
/// HSE-SVM user configuration
static const uint32_t hse_svm_conf[HSE_USER_CONF_SIZE] = {
  0x00000018,                                     // 24 bytes data below
  SECURE_BOOT_ENABLE_MASK + ANTI_ROLLBACK_MASK, // MCU settings
  0x00000000,                                     // 20 bytes reserved data
  0x00000000,
  0x00000000,
  0x00000000,
  0x00000000
};
#endif
```

6. The hse_svh_xg21b_conf[] cannot apply to xG23B (HSE-SVH) devices. Add hse_svh_other_conf[] to `app_dci_task.c`.

```
#if defined(DEVICE_XG23)
/// Other HSE-SVH user configuration
static const uint32_t hse_svh_other_conf[HSE_USER_CONF_SIZE] = {
  0x00000018,                                     // 24 bytes data below
  SECURE_BOOT_ENABLE_MASK + ANTI_ROLLBACK_MASK, // MCU settings
  0x40440410,                                     // Tamper settings
  0x14040104,
  0x22414224,
  0x74422112,
  0x0500060A
};
#endif
```

7. Add code (based on `case DEVICE_XG21:`, replace `hse_svh_xg21b_conf` with `hse_svh_other_conf`) to `app_dci_task.c` to set up a command buffer for OTP configuration.

```
#if defined(DEVICE_XG23)
        case DEVICE_XG23:
          // Check SVM or SVH
          if (*(cmd_buf + 2) == 'A') {
            *cmd_buf = INIT_HSE_OTP_LENGTH;
            *(++cmd_buf) = COMMAND_INIT_OTP;
            // Calculate parity of OTP configuration
            *(++cmd_buf) = 0;
            for (i = 1; i < HSE_USER_CONF_SIZE; i++) {
              *cmd_buf ^= hse_svm_conf[i];
            }
            // Copy OTP configuration to buffer
            memcpy((uint32_t *)(++cmd_buf), (uint32_t *)hse_svm_conf,
                   INIT_HSE_OTP_LENGTH);
          } else if (*(cmd_buf + 2) == 'B') {
            *cmd_buf = INIT_HSE_OTP_LENGTH;
            *(++cmd_buf) = COMMAND_INIT_OTP;
            // Calculate parity of OTP configuration
            *(++cmd_buf) = 0;
            for (i = 1; i < HSE_USER_CONF_SIZE; i++) {
              *cmd_buf ^= hse_svh_other_conf[i];
            }
            // Copy OTP configuration to buffer
            memcpy((uint32_t *)(++cmd_buf), (uint32_t *)hse_svh_other_conf,
                   INIT_HSE_OTP_LENGTH);
          } else {
            RAISE(SWD_ERROR_UNKNOWN_DEVICE);
          }
          break;
#endif
```

8. Add firmware images to `app_firmware_image.c` and `app_firmware_image.h`.

```
#if defined(DEVICE_XG23)
/// xG23 HSE firmware image
SL_ALIGN(4) static const uint8_t xg23_hse_image[88221UL] SL_ATTRIBUTE_ALIGN(4) = {

/// xG23 application firmware image
SL_ALIGN(4) static const uint8_t xg23_app_image[13560UL] SL_ATTRIBUTE_ALIGN(4) = {

/// xG23 signed firmware image
SL_ALIGN(4) static const uint8_t xg23_signed_image[16248UL] SL_ATTRIBUTE_ALIGN(4) = {

/// xG23 application firmware image to upgrade HSE firmware
SL_ALIGN(4) static const uint8_t prog_xg23_hse_upgrade[11696UL] SL_ATTRIBUTE_ALIGN(4) = {
#endif
```

9. Add functions to `app_firmware_image.c` and `app_firmware_image.h` to get the address and size of firmware images.

```
#if defined(DEVICE_XG23)
/*****************************************************************************//**
const uint8_t * get_xg23_hse_addr(void)
{
  return(xg23_hse_image);
}

/*****************************************************************************//**
uint32_t get_xg23_hse_size(void)
{
  return(sizeof(xg23_hse_image));
}

/*****************************************************************************//**
const uint8_t * get_xg23_app_addr(void)
{
  return(xg23_app_image);
}

/*****************************************************************************//**
uint32_t get_xg23_app_size(void)
{
  return(sizeof(xg23_app_image));
}

/*****************************************************************************//**
const uint8_t * get_xg23_signed_addr(void)
{
  return(xg23_signed_image);
}

/*****************************************************************************//**
uint32_t get_xg23_signed_size(void)
{
  return(sizeof(xg23_signed_image));
}

/*****************************************************************************//**
const uint8_t * get_xg23_hse_upgrade_app_addr(void)
{
  return(prog_xg23_hse_upgrade);
}

/*****************************************************************************//**
uint32_t get_xg23_hse_upgrade_app_size(void)
{
  return(sizeof(prog_xg23_hse_upgrade));
}
#endif
```

10. The `tamper_source_xg21b[]` cannot be applied to xG23B (HSE-SVH) devices. Add `tamper_source_other[]` to `app_process.c`.

```
#if defined(DEVICE_XG23)
/// Strings for tamper sources of other HSE-SVH devices
static const char *tamper_source_other[TAMPER_SIGNAL_NUM] = {
  NULL,
  "Filter counter        : ",
  "SE watchdog           : ",
  NULL,
  "SE RAM ECC 2          : ",
  "SE hard fault         : ",
  NULL,
  "SE software assertion : ",
  "SE secure boot        : ",
  "User secure boot      : ",
  "Mailbox authorization : ",
  "DCI authorization     : ",
  "OTP Read              : ",
  NULL,
  "Self test             : ",
  "TRNG monitor          : ",
  "Secure lock           : ",
  "Digital glitch        : ",
  "Voltage glitch        : ",
  "SE ICACHE             : ",
  "SE RAM ECC 1          : ",
  "BOD                   : ",
  "Temperature sensor    : ",
  "DPLL lock fail low    : ",
  "DPLL lock fail high   : ",
  "PRS0                  : ",
  "PRS1                  : ",
  "PRS2                  : ",
  "PRS3                  : ",
  "PRS4                  : ",
  "PRS5                  : ",
  "PRS6                  : "
};
#endif
```

11. Add `|| defined(DEVICE_XG23)` and code (based on `case DEVICE_XG21:`, replace `tamper_source_xg21b` with `tamper_source_other`) to function `print_otp_conf()` in `app_process.c` to print out the tamper configuration.

```
#if defined(DEVICE_XG21) || defined(DEVICE_XG23)
  uint32_t i;
  uint32_t j;
  uint32_t k;
#endif
...
#if defined(DEVICE_XG23)
      case DEVICE_XG23:
        for (i = 0, j = 0, k = 2; i < TAMPER_SIGNAL_NUM; i++, j += 4) {
          if (j == 32) {
            j = 0;
            k++;
          }
          cmd_resp_buf[8] = (cmd_resp_buf[k] >> j) & 0x0f;
          if (tamper_source_other[i] != NULL) {
            printf("    %s %lu\n", tamper_source_other[i], cmd_resp_buf[8]);
          }
        }
        break;
#endif
...
#if defined(DEVICE_XG21) || defined(DEVICE_XG23)
    // Common tamper parameters
    printf("  + Reset period for the tamper filter counter: ~32 ms x %u\n",
           1 << (cmd_resp_buf[6] & COUNTER_PERIOD_MASK));
    printf("  + Activation threshold for the tamper filter: %d\n",
           256 / (1 << ((cmd_resp_buf[6] & COUNTER_THRESHOLD_MASK) >> COUNTER_THRESHOLD_SHIFT)));
    if (cmd_resp_buf[6] & GLITCH_DETECTOR_MASK) {
      printf("  + Digital glitch detector always on: Enabled\n");
    } else {
      printf("  + Digital glitch detector always on: Disabled\n");
    }
    if (device_name[DEVICE_INDEX] > DEVICE_XG22) {
      if (cmd_resp_buf[6] & SLEEP_ALIVE_MASK) {
        printf("  + Keep tamper alive during sleep: Enabled\n");
      } else {
        printf("  + Keep tamper alive during sleep: Disabled\n");
      }
    }
    printf("  + Tamper reset threshold: %lu\n", cmd_resp_buf[6] >> TAMPER_RESET_SHIFT);
#endif
```

12. Add code (take `case DEVICE_XG21:` as reference) to function `prog_main_flash_app()` in `app_process.c`.

```
#if defined(DEVICE_XG23)
    case DEVICE_XG23:
      printf("  + The xG23 application firmware image size is %lu bytes "
             "and start address is 0x%08lX.\n", get_xg23_app_size(),
             get_flash_start_addr());
      printf("  + Erase-Program-Verify the xG23 main flash for "
             "application firmware image... ");
      cmd_resp_buf[0] = prog_flash(get_flash_start_addr(),
                                   get_xg23_app_size(),
                                   (uint32_t *)get_xg23_app_addr());
      print_cycle_time();
      break;
#endif
```

13. Add code (take `case DEVICE_XG21:` as reference) to function `prog_main_flash_se()` in `app_process.c`.

```
#if defined(DEVICE_XG23)
    case DEVICE_XG23:
      printf("  + The xG23 HSE firmware image version: %08lX\n",
             *((uint32_t *)get_xg23_hse_addr() + 3));
      printf("  + The xG23 HSE firmware image size is %lu bytes and start "
             "address is 0x%08lX.\n", get_xg23_hse_size(),
             SE_START_ADDR + get_flash_start_addr());
      printf("  + Erase-Program-Verify the xG23 main flash for HSE "
             "firmware image... ");
      cmd_resp_buf[0] = prog_flash(SE_START_ADDR + get_flash_start_addr(),
                                   get_xg23_hse_size(),
                                   (uint32_t *)get_xg23_hse_addr());
      print_cycle_time();
      break;
#endif
```

14. Add code (take `case DEVICE_XG21:` as reference) to function `prog_main_flash_se_app()` in `app_process.c`.

```
#if defined(DEVICE_XG23)
    case DEVICE_XG23:
      printf("  + The xG23 HSE upgrade application firmware image size is "
             "%lu bytes and start address is 0x%08lX.\n",
             get_xg23_hse_upgrade_app_size(), get_flash_start_addr());
      printf("  + Erase-Program-Verify the xG23 main flash for "
             "application to upgrade \n");
      printf("    HSE firmware... ");
      cmd_resp_buf[0] = prog_flash(get_flash_start_addr(),
                                   get_xg23_hse_upgrade_app_size(),
                                   (uint32_t *)get_xg23_hse_upgrade_app_addr());
      print_cycle_time();
      break;
#endif
```

15. Add code (take `case DEVICE_XG21:` as reference) to function `prog_main_flash_signed()` in `app_process.c`.

```
#if defined(DEVICE_XG23)
    case DEVICE_XG23:
      printf("  + The xG23 signed firmware image size is %lu bytes "
             "and start address is 0x%08lX.\n",
             get_xg23_signed_size(),
             get_flash_start_addr());
      printf("  + Erase-Program-Verify the xG23 main flash for "
             "signed firmware image... ");
      cmd_resp_buf[0] = prog_flash(get_flash_start_addr(),
                                   get_xg23_signed_size(),
                                   (uint32_t *)get_xg23_signed_addr());
      print_cycle_time();
      break;
#endif
```

16. For xG23 devices, the user data can be erased the same way as any page in the main flash. Add code (take `case DEVICE_XG22:` as reference) to function `erase_user_data()` in `app_process.c`.

```
#if defined(DEVICE_XG23)
    case DEVICE_XG23:
      printf("\n  . Erase xG23 user data... ");
      cmd_resp_buf[0] = erase_flash(true);
      print_cycle_time();
      hard_reset_target();
      break;
#endif
```

17. For xG23 devices, the user data can be written the same way as any page in the main flash. Add code (take `case DEVICE_XG22:` as reference) to function `prog_user_data()` in `app_process.c`.

```
#if defined(DEVICE_XG23)
    case DEVICE_XG23:
      printf("\n  . Program xG23 user data.\n");
      printf("  + User data size is %lu bytes and start address is 0x%08lX.\n",
             get_userdata_size(), USERDATA_BASE);
      printf("  + Erase-Program-Verify the xG23 user data... ");
      cmd_resp_buf[0] = prog_flash(USERDATA_BASE,
                                   get_userdata_size(),
                                   (uint32_t *)get_userdata_addr());
      print_cycle_time();
      hard_reset_target();
      break;
#endif
```

18. There is no need to change the default settings in 6.2.5 Compile Options for xG23 devices.

Note: The 512 kB flash of EFR32MG22C224F512 is not enough to store the firmware images when the programmer needs to support more Series 2 devices. Users can selectively comment out the defines in `app_process.h` to save memory for the required Series 2 devices.

```
/// xG21 device
#define DEVICE_XG21             (0x31)

/// xG22 device
#define DEVICE_XG22             (0x32)

/// xG23 device
#define DEVICE_XG23             (0x33)
```

8. Use J-Link Commander to Run DCI Command

J-Link Commander from Segger is a command line-based utility that supports simple commands to verify and communicate with the target connection. There are two ways to run the DCI command through the J-Link Commander.

1. J-Link Command File (`*.jlink`)
2. J-Link Script File (`*.JLinkScript`)

This application note uses J-Link Commander v7.66g. The J-Link Commander's command line interface is invoked by `JLink.exe` located in `C:\Program Files\SEGGER\JLink_V766g` (Windows).

J-Link Command File

The J-Link Command file below is an example to invoke the J-Link Commander in batch processing mode for the Erase Device operation. The delay (`sleep 500`) in milliseconds after writing the command word is device dependent. It may be shorter or longer.

```
connect

swdwritedp 2 0x01000000    ; Select DCI AP
swdwriteap 1 0x1008        ; DCI STATUS
swdreadap 3
swdreaddp 3                ; Poll till DCI_STATUS.WPENDING (bit 0) is low

swdwriteap 1 0x1000        ; DCI WDATA
swdwriteap 3 0x08          ; Command word 0
sleep 500                  ; Delay if necessary

swdwriteap 1 0x1008        ; DCI STATUS
swdreadap 3
swdreaddp 3                ; Poll till DCI_STATUS.WPENDING (bit 0) is low

swdwriteap 1 0x1000        ; DCI WDATA
swdwriteap 3 0x430F0000    ; command word 1 (Erase Device)
sleep 500                  ; Delay if necessary

swdwriteap 1 0x1008        ; DCI STATUS
swdreadap 3
swdreaddp 3                ; Poll till DCI_STATUS.RDATAVALID (bit 8) is high

swdwriteap 1 0x1004        ; DCI RDATA
swdreadap 3                ; First word
swdreaddp 3                ; Status code is upper 16 bits, total length is lower 16 bits
exit
```

The example below is to run the J-Link Command file (assume `EraseDevice.jlink` is in the J-Link Commander folder) via the Windows DOS command prompt. The target device is EFR32MG21A, and the target interface is 1000 kHz SWD. The SWD speed is hardware dependent; for example, the length of the wires between the programmer and the device debug pins. The speed (frequency) can be higher for shorter wires and lower for longer wires.

```
JLink.exe -device EFR32MG21AXXXF1024 -if SWD -speed 1000 -CommandFile EraseDevice.jlink
```

```
SEGGER J-Link Commander V7.66g (Compiled Jul  7 2022 10:44:29)
DLL version V7.66g, compiled Jul  7 2022 10:42:43


J-Link Command File read successfully.
Processing script file...
J-Link>connect
J-Link connection not established yet but required for command.
Connecting to J-Link via USB...O.K.
Firmware: Silicon Labs J-Link Pro OB compiled Sep 16 2020 17:10:58
Hardware version: V4.00
S/N: 440048205
License(s): RDI, FlashBP
IP-Addr: DHCP (no addr. received yet)
VTref=3.326V
Device "EFR32MG21AXXXF1024" selected.


Connecting to target via SWD
```

```
Found SW-DP with ID 0x6BA02477
DPv0 detected
CoreSight SoC-400 or earlier
Scanning AP map to find all available APs
AP[3]: Stopped AP scan as end of AP map has been reached
AP[0]: AHB-AP (IDR: 0x84770001)
AP[1]: APB-AP (IDR: 0x54770002)
AP[2]: AHB-AP (IDR: 0x84770001)
Iterating through AP map to find AHB-AP to use
AP[0]: Core found
AP[0]: AHB-AP ROM base: 0xE00FE000
CPUID register: 0x410FD213. Implementer code: 0x41 (ARM)
Feature set: Mainline
Found Cortex-M33 r0p3, Little endian.
FPUnit: 8 code (BP) slots and 0 literal slots
Security extension: implemented
Secure debug: enabled
CoreSight components:
ROMTbl[0] @ E00FE000
[0][0]: E00FF000 CID B105100D PID 000BB4C9 ROM Table
ROMTbl[1] @ E00FF000
[1][0]: E000E000 CID B105900D PID 000BBD21 DEVARCH 47702A04 DEVTYPE 00 Cortex-M33
[1][1]: E0001000 CID B105900D PID 000BBD21 DEVARCH 47701A02 DEVTYPE 00 DWT
[1][2]: E0002000 CID B105900D PID 000BBD21 DEVARCH 47701A03 DEVTYPE 00 FPB
[1][3]: E0000000 CID B105900D PID 000BBD21 DEVARCH 47701A01 DEVTYPE 43 ITM
[1][5]: E0041000 CID B105900D PID 002BBD21 DEVARCH 47724A13 DEVTYPE 13 ETM
[1][6]: E0042000 CID B105900D PID 000BBD21 DEVARCH 47701A14 DEVTYPE 14 CSS600-CTI
[0][1]: E0040000 CID B105900D PID 000BBD21 DEVARCH 00000000 DEVTYPE 11 Cortex-M33
[0][2]: E00FD000 CID B105F00D PID 001BB101 TSG
Cortex-M33 identified.
J-Link>swdwritedp 2 0x01000000    ; Select DCI AP
Write DP register 2 = 0x01000000
J-Link>swdwriteap 1 0x1008        ; DCI STATUS
Write AP register 1 = 0x00001008
J-Link>swdreadap 3
Read AP register 3 = 0x000000B1
J-Link>swdreaddp 3                ; Poll till DCI_STATUS.WPENDING (bit 0) is low
Read DP register 3 = 0x00000000
J-Link>swdwriteap 1 0x1000        ; DCI WDATA
Write AP register 1 = 0x00001000
J-Link>swdwriteap 3 0x08          ; Command word 0
Write AP register 3 = 0x00000008
J-Link>sleep 500                  ; Delay if necessary
Sleep(500)
J-Link>swdwriteap 1 0x1008        ; DCI STATUS
Write AP register 1 = 0x00001008
J-Link>swdreadap 3
Read AP register 3 = 0x00000000
J-Link>swdreaddp 3                ; Poll till DCI_STATUS.WPENDING (bit 0) is low
Read DP register 3 = 0x00000000
J-Link>swdwriteap 1 0x1000        ; DCI WDATA
Write AP register 1 = 0x00001000
J-Link>swdwriteap 3 0x430F0000    ; command word 1 (Erase Device)
Write AP register 3 = 0x430F0000
J-Link>sleep 500                  ; Delay if necessary
Sleep(500)
J-Link>swdwriteap 1 0x1008        ; DCI STATUS
Write AP register 1 = 0x00001008
J-Link>swdreadap 3
Read AP register 3 = 0x00000000
J-Link>swdreaddp 3                ; Poll till DCI_STATUS.RDATAVALID (bit 8) is high
Read DP register 3 = 0x00000100
J-Link>swdwriteap 1 0x1004        ; DCI RDATA
Write AP register 1 = 0x00001004
J-Link>swdreadap 3                ; First word
Read AP register 3 = 0x00000100
J-Link>swdreaddp 3                ; Status code is upper 16 bits, total length is lower 16 bits
Read DP register 3 = 0x00000004
J-Link>exit

Script processing completed.
```

J-Link Script File

The J-Link Script file below is an example to customize some actions performed by the J-Link Commander for the Read Serial Number operation.

```
// Function to select DCI AP register bank 0
int SelectDciAp(void)
{
  int stat;
  int value;

  JLINK_SYS_Report("- Select DCI AP register bank 0\n");
  value = 0x01000000;
  stat = JLINK_CORESIGHT_WriteDP(2, value);
  if (stat < 0) {
    JLINK_SYS_Report1("- DP 2 write failed: ", value);
  }
  return stat;
}

// Function to write command to DCI_WDATA (0x1000)
int WriteCommandWord(int command)
{
  int stat;
  int value;

  // Poll DCI_STATUS (0x1008) WPENDING bit (bit 0)
  do {
    value = 0x00001008;
    stat = JLINK_CORESIGHT_WriteAP(1, value);
    if (stat < 0) {
      JLINK_SYS_Report1("- AP 1 write failed: ", value);
      return stat;
    }

    JLINK_CORESIGHT_ReadAP(3);
    value = JLINK_CORESIGHT_ReadDP(3);
    if (value == -1) {
      JLINK_SYS_Report("- DP 3 read failed");
      return value;
    }
    if ((value & 0x0100) != 0) {
      JLINK_SYS_Report("- RDATAVALID is high, command aborted");
      return -1;
    }
  } while ((value & 0x01) != 0);

  // Write command
  value = 0x00001000;
  stat = JLINK_CORESIGHT_WriteAP(1, value);
  if (stat < 0) {
    JLINK_SYS_Report1("- AP 1 write failed: ", value);
    return stat;
  }

  value = command;
  stat = JLINK_CORESIGHT_WriteAP(3, value);
  if (stat < 0) {
    JLINK_SYS_Report1("- AP 3 write failed: ", value);
    return stat;
  }
  JLINK_SYS_Report1("- Write command word ", command);
  return stat;
}

// Function to read response from DCI_RDATA (0x1004)
int ReadResponse(void)
{
  int stat;
  int value;
  int count;
```

```
// Poll DCI_STATUS (0x1008) RDATAVALID bit (bit 8)
do {
  value = 0x00001008;
  stat = JLINK_CORESIGHT_WriteAP(1, value);
  if (stat < 0) {
    JLINK_SYS_Report1("- AP 1 write failed: ", value);
    return stat;
  }

  JLINK_CORESIGHT_ReadAP(3);
  value = JLINK_CORESIGHT_ReadDP(3);
  if (value == -1) {
    JLINK_SYS_Report("- DP 3 read failed");
    return value;
  }
} while ((value & 0x0100) != 0x0100);

// Read first 32-bit response word
value = 0x00001004;
stat = JLINK_CORESIGHT_WriteAP(1, value);
if (stat < 0) {
  JLINK_SYS_Report1("- AP 1 write failed: ", value);
  return stat;
}

JLINK_CORESIGHT_ReadAP(3);
value = JLINK_CORESIGHT_ReadDP(3);
if (value == -1) {
  JLINK_SYS_Report("- DP 3 read failed");
  return value;
}

// Get response count
count = value & 0x00FF;
count = count >> 2;
JLINK_SYS_Report1("- Number of 32-bit response word: ", count);

// Get response code
JLINK_SYS_Report1("- Response: ", value);
stat = value >>16;
if (stat != 0) {
  JLINK_SYS_Report1("- Command error: ", stat);
  return -1;
}

// Read following 32-bit response word
count = count - 1;
while (count != 0) {
  // Poll DCI_STATUS (0x1008) RDATAVALID bit (bit 8)
  do {
      value = 0x00001008;
    stat = JLINK_CORESIGHT_WriteAP(1, value);
    if (stat < 0) {
      JLINK_SYS_Report1("- AP 1 write failed: ", value);
      return stat;
    }

    JLINK_CORESIGHT_ReadAP(3);
    value = JLINK_CORESIGHT_ReadDP(3);
    if (value == -1) {
      JLINK_SYS_Report("- DP 3 read failed");
      return value;
    }
  } while ((value & 0x0100) != 0x0100);

  // Read 32-bit response word
  value = 0x00001004;
  stat = JLINK_CORESIGHT_WriteAP(1, value);
  if (stat < 0) {
    JLINK_SYS_Report1("- AP 1 write failed: ", value);
    return stat;
  }
```

```
    JLINK_CORESIGHT_ReadAP(3);
    value = JLINK_CORESIGHT_ReadDP(3);
    if (value == -1) {
      JLINK_SYS_Report("- DP 3 read failed");
      return value;
    }
    JLINK_SYS_Report1("- Response: ", value);
    count = count - 1;
  }
  return 0;
}

// Function to connect DCI
int ConnectDci(void)
{
  int stat;
  int value;

  JLINK_SYS_Report("\n*****************************************************");
  JLINK_SYS_Report("Connect to Series 2 Device DCI\n");

  JLINK_SYS_Report("- Select SWD by sending SWD switching sequence\n");
  stat = JLINK_CORESIGHT_Configure("");
  if (stat < 0) {
    JLINK_SYS_Report("- SWD connection failed");
    return stat;
  }

  // Manually select debug interface
  JLINK_SYS_Report("- Clear sticky error flags\n");
  value = 0x0000001E;
  stat = JLINK_CORESIGHT_WriteDP(0, value);
  if (stat < 0) {
    JLINK_SYS_Report1("- DP 0 write failed: ", value);
    return stat;
  }

  JLINK_SYS_Report("- Power up system and debug\n");
  value = 0x50000000;
  stat = JLINK_CORESIGHT_WriteDP(1, value);
  if (stat < 0) {
    JLINK_SYS_Report1("- DP 1 write failed: ", value);
    return stat;
  }

  value = JLINK_CORESIGHT_ReadDP(0);
  if (value != 0x6BA02477) {
    JLINK_SYS_Report1("- The connected device does not have a Secure Element (ID Code): ", value);
    return -1;
  }
  JLINK_SYS_Report1("- Read SWD-DP ID code: ", value);

  JLINK_SYS_Report("- Select DCI AP register bank 0\n");
  value = 0x01000000;
  stat = JLINK_CORESIGHT_WriteDP(2, value);
  if (stat < 0) {
    JLINK_SYS_Report1("- DP 2 write failed: ", value);
    return stat;
  }

  JLINK_SYS_Report("- Set up AP defaults\n");
  value = 0x22000002;
  stat = JLINK_CORESIGHT_WriteAP(0, value);
  if (stat < 0) {
    JLINK_SYS_Report1("- AP 0 write failed: ", value);
    return stat;
  }

  JLINK_SYS_Report("Connect to Series 2 Device DCI OK\n");
  JLINK_SYS_Report("*****************************************************");
  return 0;
}
```

```
// Function to run DCI command
void RunDciCommand(void)
{
  JLINK_SYS_Report("\n*****************************************************");
  JLINK_SYS_Report("Read Serial Number\n");
  if (SelectDciAp() == -1) {
    JLINK_SYS_Report("- Select DCI AP failed\n");
    JLINK_SYS_Report("*****************************************************\n\n");
    return;
  }
  if (WriteCommandWord(0x00000008) == -1) {
    JLINK_SYS_Report("- Write DCI command (length) failed\n");
    JLINK_SYS_Report("*****************************************************\n\n");
    return;
  }
  if (WriteCommandWord(0xFE000000) == -1) {
    JLINK_SYS_Report("- Write DCI command (ID) failed\n");
    JLINK_SYS_Report("*****************************************************\n\n");
    return;
  }
  if (ReadResponse() == -1) {
    JLINK_SYS_Report("- Read DCI command response failed\n");
    JLINK_SYS_Report("*****************************************************\n\n");
    return;
  }
  JLINK_SYS_Report("Read Serial Number Done\n");
  JLINK_SYS_Report("*****************************************************\n\n");
}

// Replace ConfigTargetSettings() in J-Link DLL
void ConfigTargetSettings(void)
{
  if (ConnectDci() == -1) {
    JLINK_SYS_Report("- Connect to DCI failed\n");
    JLINK_SYS_Report("*****************************************************\n\n");
    return;
  }
  RunDciCommand();
}
```

The example below is to run the J-Link Script file (assume `ReadSerialNo.JLinkScript` is in the J-Link Commander folder) via the Windows DOS command prompt. The target device is EFR32MG21A, and the target interface is 1000 kHz SWD. The SWD speed is hardware dependent; for example, the length of the wires between the programmer and the device debug pins. It can be higher for shorter wires and lower for longer wires.

```
JLink -device EFR32MG21AXXXF1024 -if SWD -speed 1000 -autoConnect 1 -JLinkScriptFile ReadSerialNo.JlinkScript
```

```
SEGGER J-Link Commander V7.66g (Compiled Jul  7 2022 10:44:29)
DLL version V7.66g, compiled Jul  7 2022 10:42:43

Connecting to J-Link via USB...O.K.
Firmware: Silicon Labs J-Link Pro OB compiled Sep 16 2020 17:10:58
Hardware version: V4.00
S/N: 440048205
License(s): RDI, FlashBP
IP-Addr: DHCP (no addr. received yet)
VTref=3.328V
Device "EFR32MG21AXXXF1024" selected.


Connecting to target via SWD
ConfigTargetSettings() start

*************************************************
Connect to Series 2 Device DCI
- Select SWD by sending SWD switching sequence
- Clear sticky error flags
- Power up system and debug
- Read SWD-DP ID code: 0x6BA02477
- Select DCI AP register bank 0
- Set up AP defaults
```

```
Connect to Series 2 Device DCI OK
**************************************************

**************************************************
Read Serial Number
- Select DCI AP register bank 0
- Write command word 0x00000008
- Write command word 0xFE000000
- Number of 32-bit response word: 0x00000005
- Response: 0x00000014
- Response: 0x00000000
- Response: 0x00000000
- Response: 0xFF818E58
- Response: 0xE53470FE
Read Serial Number Done
**************************************************

ConfigTargetSettings() end
```

9. Revision History

Revision 0.8

October 2023

- Updated description of ‘Secure Vault High offers additional security options:” in 1. Series 2 Device Security Features.
- Updated table in 1.1 User Assistance to include AN1374 and AN1311.
- Corrected bit definitions in 5.16 Read Lock Status.
- Reworded the instructions in 7. Add a New Series 2 Device to the Programmer, step 10.

Revision 0.7

June 2023

- Updated supported target boards in 6.1 Hardware Overview.
- Updated the description of other boards supported in 6.1 Hardware Overview.
- Added 5.15 Set Debug Restrictions and 5.16 Read Lock Status in 5. SE Command List.

Revision 0.6

December 2022

- Fixed a typo (`.sec` to `.seu`) in 2. Introduction and 6.2.4 Firmware images.
- Updated 3.1 DCI Connection.
- Updated 5. SE Command List to clarify the command list in this application note is incomplete.
- Updated figure in 6.2 Software Overview.
- Added 8. Use J-Link Commander to Run DCI Command.

Revision 0.5

June 2022

- Updated table and note in 1. Series 2 Device Security Features.
- Replaced Device Compatibility with SE Firmware in 1. Series 2 Device Security Features.
- Removed Table 2.1 in (this table is moved to UG103.05).

Revision 0.4

March 2022

- Added digit 4 to Note 3 in 1. Series 2 Device Security Features.
- Updated Device Compatibility and moved it under 1. Series 2 Device Security Features.

Revision 0.3

January 2022

- Added UG489 to the table in 1.2 Key Reference.
- Updated Windows folder in 2. Introduction for GSDK v4.0 or higher.
- Updated Shipped SE Firmware Version in Table 2.1.
- Added CPMS information to 2. Introduction.
- Added keep tamper alive during sleep flag to Table 5.16 Anti-Tamper Configuration Settings on page 18.
- Updated to GSDK v4.0.1 in 6. Series 2 DCI and SWD Programming Examples and 6.6 Benchmark.
- Added keep tamper alive during sleep flag to 7. Add a New Series 2 Device to the Programmer step 11.

Revision 0.2

December 2021

- Added 1. Series 2 Device Security Features and use the terminology defined in this section throughout the document.
- Updated Device Compatibility.
- Removed terminology in 2. Introduction and updated Table 2.1.
- Replaced Table 4.1 with a sentence.
- Added Table 5.13 MCU Flags for Series 2 Devices on page 17.
- Updated Table 5.14 Tamper Source Response Level on HSE-SVH (xG21B) Devices on page 18 and added Table 5.15 Tamper Source Response Level on Other HSE-SVH Devices on page 18.
- Updated Table 5.17 Initialize Public Key Command on page 19, 5.13 Read Public Key, and Table 5.19 Initialize AES Key Command on page 19.
- Removed the push buttons in 6.1 Hardware Overview and added radio boards for EFR32FG23A and EFR32FG23B.
- Updated Figure 6.2 Programmer Connection Diagram on page 23.
- Updated Table 6.3 Hard-coded OTP Settings on page 25 for other HSE-SVH devices.
- Updated Table 6.4 Hard-coded Firmware Images on page 26 for xG23 devices.
- Updated 6.3 Menu Operation, 6.4 DCI Programming Examples, 6.5 SWD Programming Examples, and 6.6 Benchmark to the latest Series 2 DCI and SWD Programming platform example.
- Added 7. Add a New Series 2 Device to the Programmer.

Revision 0.1

April 2021

- Initial Revision.

Simplicity Studio

One-click access to MCU and wireless tools, documentation, software, source code libraries & more. Available for Windows, Mac and Linux!

IoT Portfolio
www.silabs.com/IoT

SW/HW
www.silabs.com/simplicity

Quality
www.silabs.com/quality

Support & Community
www.silabs.com/community

SILICON LABS

Silicon Laboratories Inc.
400 West Cesar Chavez
Austin, TX 78701
USA

www.silabs.com