AN961: Bringing Up Custom Devices for the Mighty Gecko and Flex Gecko Families

This application note describes how to initialize a piece of custom hardware (a “device”) based on the Mighty Gecko and Flex Gecko families so that it interfaces correctly with a network stack. The same procedures can be used to restore devices whose settings have been corrupted or erased.

KEY POINTS

- Information required before board bring-up
- Setting manufacturing tokens
- Bootloading
- Performing functional testing

1. Introduction

Before an EFR32-based product such as a member of the Wireless Gecko or Flex Gecko families (hereafter EFR32) can be initialized and tested, manufacturing tokens within the EFR32 User Data Page and the Lock Bits Page must be configured. Similarly, before any application specific code can be programmed into the EFR32 flash, a board header needs to be created either manually or from the Application Builder tool set. In order to perform these tasks, the product design team must know how the EFR32 is to be used in the wireless system.

In particular, the design team must know the following:

- PCB Manufacturing-specific information (serial number, product numbers, EUI, and so on)
- Bootloader architecture (serial dataflash)
- External components in the RF Path (PA, LNA, and so on)
- 38.4 MHz crystal oscillator specification and a CTUNE value to match your crystal so you hit the center frequency
- Application security tokens (Keys, certificates, and so on)

Note: Even though the EFR32 flash is fully tested during production test, the flash contents in the main flash block are not set to a known state before shipment. The User Data and Lock Bits pages are erased to all 0xFF, except in kits where they might be preloaded with configuration values such as CTUNE.

2. EFR32 Wireless System

Once the hardware design of a custom device has been completed, the assembled product is ready for test and functional validation. Before testing, developers must understand how the device will operate at both the device-level and system-level. The following table describes the different items that developers should know before board bring-up.

Table 2.1. Information Needed Before Board Bring-Up

Information	Required or Optional	Description
Device Level		
Product Information	Optional	Most products have unique serial numbers as well as generic product codes that might be stored in the EFR32. This information might include items such as where and when the device was assembled, a product serial number, and product name.
Custom EUI	Optional	IEEE 64-bit unique number. Each EFR32 comes with an EUI programmed into the Device Information Page. Customers that have their own IEEE address block can use it in place of the Device Information Page's EUI.
RF Component Information	Required	If the product uses external PAs or LNAs, then developers must know the pin-connections between the off-chip components and the EFR32. They should also understand the LNA Gain as it will be used to offset the clear channel assessment (CCA) threshold.
Protocol-Specific Information	Optional	If developing a product using the ZigBee networking protocol, obtain a ZigBee-assigned manufacturer code before testing the application.
System Level		
Bootloader Option	Required	Silicon Labs offers several bootloading options for different system designs. For more information, see *UG103.6: Application Development Fundamentals: Bootloading*.
System Security	Optional	If device-level security is required for the product, determine necessary security settings and keys before setting up the device.

As detailed in the above table, Silicon Labs offers its customers an opportunity to guarantee a device's uniqueness on the network. In addition, it allows customers a way to store product descriptions, manufacturer-specific information and device information.

3. Setting Manufacturing Tokens

EFR32 chips are delivered to customers with all memory erased. Exceptions to this rule are chips that come with kits. Before the EFR32 chips can be used to run applications for a networking stack, the customer or a contract manufacturer/test house must prepare them. Preparation includes programming the proper application and bootloader, if necessary, into the Main flash block, as well as programming customer manufacturing tokens in the User Data block and Lock Bits block.

Manufacturing tokens are values programmed into special, non-volatile storage area of flash. The User Data page and Lock Bits page contain data that manufacturers of EFR32-based devices can program. The Device Information Page also contains manufacturing tokens, but these tokens are fixed values that cannot be modified.

Note: Applications and the stack can read any manufacturing tokens at any time.

Simplicity Commander is a single, all-purpose tool to be used in a production environment. It is invoked using a simple Command-Line Interface (CLI) that is also scriptable. Simplicity Commander enables customers to complete these essential tasks:

- Flash their own applications.
- Configure their own applications.
- Create binaries for production.

For more information, refer to *UG162: Simplicity Commander Reference Guide*.

The following two tables identify the User Data manufacturing tokens that OEMs and CMs may want to program at manufacturing time. Refer to `platform\base\hal\micro\cortexm3\efm32\token-manufacturing.h` for the token definition, because it may differ from these tables depending on the stack release version.

Note: Using tokens to set up manufacturing data in the Silicon Labs Flex SDK only works if you use the Silicon Labs Connect stack. If you're interfacing the RAIL library directly, you can set up similar values using the board-specific header files.

Silicon Labs recommends that User Data and Lock Bits page tokens be written using Simplicity Commander at the same time as programming the main flash. Simplicity Commander also allows for patching and reprogramming the manufacturing blocks as many times as necessary. (See *UG162: Simplicity Commander Reference Guide* for details.) The following two tables describe the location of each manufacturing token as an offset to the starting address of the relevant block. For the most accurate and specific information about where the flash regions begin in the address map of your chip, please consult your IC's Reference Manual or Data Sheet.

Some situations though, may require that a manufacturing token be programmed at runtime from code running on the chip itself. For wireless mesh network SDKs, the manufacturing token module of the HAL provides a token API to write the manufacturing tokens. However, this API only writes manufacturing tokens that are in a completely erased state. If a manufacturing token must be reprogrammed, you must use an external utility.

- The API on SoC platforms is `halCommonSetMfgToken(token, data)`. The parameters for this API are the same as the API `halCommonSetToken(token, data )`.
- For EmberZNet PRO EZSP NCP platforms, the host API is `ezspSetMfgToken(tokenId, tokenDataLength, tokenData)`. (See *UG100: EZSP Reference Guide* for details.)
- For Silicon Labs Thread, the API is `emberSetMfgToken(tokenId, tokenDataLength, tokenData)`.

Table 3.1. User Data Manufacturing Tokens for the EFR32

Offset from User Data starting address	Size (Bytes)	Name	Description
0x0000	2	TOKEN_MFG_CUSTOM_VERSION	Version number to signify which revision of User Data manufacturing tokens you are using. This value should match CURRENT_MFG_CUSTOM_VERSION which is currently set to `0x01FE`. CURRENT_MFG_CUSTOM_VERSION is defined in `\hal\micro\cortexm3\efm32\token-manufacturing.h`. *Usage*: Recommended for all devices using User Data manufacturing tokens.

Offset from User Data starting address	Size (Bytes)	Name	Description
0x0002	8	TOKEN_MFG_CUSTOM_EUI_64	IEEE 64-bit address, unique for each radio. Entered and stored in little-endian. Setting a value here overrides the EUI64 stored in the Device Information Page. This is for customers who have purchased their own address block from IEEE. **WARNING:** If this value is set "live," it may not propagate to all levels of the stack and some outgoing messages may continue to use the default value. Silicon Labs recommends forcing a reset after setting EUI64 and before starting the networking stack. *Usage:* Optionally set by device manufacturer if using custom EUI64 address block.
0x001A	16	TOKEN_MFG_STRING	Optional device-specific string, for example, the serial number. *Usage:* Optionally set by device manufacturer to identify device.
0x002A	16	TOKEN_MFG_BOARD_NAME	Optional string identifying the board name or hardware model. *Usage:* Optionally set by device manufacturer to identify device.
0x003A	2	TOKEN_MFG_MANUF_ID	16-bit ID denoting the manufacturer of this device. When you are programming with EmberZNet PRO, Silicon Labs recommends setting this value to match your ZigBee-assigned manufacturer code, such as in the stack's emberSetManufacturerCode()API call. *Usage:* Recommended for EmberZNet PRO devices utilizing the stand-alone bootloader.
0x003C	2	TOKEN_MFG_PHY_CONFIG	Reserved for future use; should be left un-programmed (0xFFFF).
0x003E	40	TOKEN_MFG_ASH_CONFIG	ASH configuration information. *Usage:* Optional for EmberZNet PRO devices acting as network coprocessors (NCPs) for EZSP-UART. Not used by Thread NCPs or any SoC use cases.
0x00F0	2	TOKEN_MFG_SYNTH_FREQ_OFFSET	Reserved for future use; should be left un-programmed (0xFFFF). Radio synthesizer frequency adjustments should be made using the `TOKEN_MFG_CTUNE` token instead.

Offset from User Data starting address	Size (Bytes)	Name	Description
0x00F6	2	TOKEN_MFG_CCA_THRESHOLD	Threshold(s) used for energy detection Clear Channel Assessment (CCA). You may want to override the default CCA threshold(s) by setting this token if your design uses a Low-Noise Amplifier (LNA). An LNA changes the gain on the radio input which results in the radio “seeing” a different energy level than if no LNA was used. Bits 0-7: Set to the two’s complement representation of the 2.4GHz band CCA threshold in dBm below which a 2.4GHz channel will be considered clear. Valid values are -128 through +126, inclusive. +127 is **not** valid and **must not** be used. Bit 8: Set to 0 if the threshold in Bits 0-7 is valid and should be used. Set to 1 (the erased state) if those bits are invalid or have not been set; in this case, the default 2.4GHz band threshold of -75 dBm will be used. Bits 9-15: Applies only to SubGHz-capable PHYs. These bits can be used to override the default CCA threshold that is applied to channels in the SubGHz band (for example, in GB868, the default SubGHz CCA threshold is -87 dBm). The interpretation of the SubGHz CCA threshold value in bits 9..15 is as a 7-bit unsigned value in the range 1..126 which is negated to be the CCA threshold to apply (that is, token values 1..126 in bits 15..9 map to SubGHz CCA thresholds of -1..-126 dBm respectively). The values 0 and 127 (the 7 bits all 0s or all 1s) are interpreted as no override—the default SubGHz CCA threshold will be used. **Example 1:** A 2.4GHz design uses an LNA that provides a gain of +12 dB. Add that gain to the 2.4GHz default threshold of -75 dBm to get the dBm value for the token: -75 + 12 = -63 dBm. The two’s complement signed representation of -63 is 0xC1 so the complete token value to be programmed is 0xFEC1. **Example 2:** A GB868 SubGHz design uses an LNA that provides a gain of +12 dB. Add that gain to the GB868 SubGHz default threshold of -87 dBm to get the negated dBm value for the token: -87 + 12 = -75 dBm. Negating, the value 75 or 0x4B goes into bits 9..15 of the token so the complete token value to be programmed is 0x97FF. **Example 3:** A dual-PHY design has an LNA like Example 1 for 2.4GHz and an LNA as in Example 2 for GB868 SubGHz. The token values in those examples combine to be 0x96C1 for this design.
0x00F8	8	TOKEN_MFG_EZSP_STORAGE	An 8-byte, general-purpose token that can be set at manufacturing time and read by the host microcontroller via EZSP’s getMfgToken command frame. Can also be accessed by Thread host applications using emberGetMfgToken. *Usage:* Not required. Device manufacturer may populate or leave empty as desired.
0x0100	2	TOKEN_MFG_CTUNE	This token is for tuning the EFR32 system XTAL and consequently also tunes the radio synthesizer frequency. *Usage:* Optional. Only necessary if additional crystal tuning is desired to reach optimal timing and frequency output.

Offset from User Data starting address	Size (Bytes)	Name	Description
0x0102	2	TOKEN_MFG_XO_TUNE	This token is for tuning an attached Si446x transceiver's system crystal and consequently also tunes its radio synthesizer frequency. A value in the least-significant byte in the range 0x00 to 0x7F will be applied to the Si446x's GLOBAL_XO_TUNE property. All other values are currently ignored and reserved for future use.

Table 3.2. Lock Bits Data Manufacturing Tokens for the EFR32

Offset from Lock Bits starting address	Size (Bytes)	Name	Description
0x0204	16	TOKEN_MFG_BOOTLOAD_AES_KEY	Reserved for future use; should be left un-programmed (all 0xFF bytes).
0x0214	92	TOKEN_MFG_CBKE_DATA	Defines the security data necessary for Smart Energy devices. It is used for Certificate Based Key Exchange to authenticate a device on a Smart Energy network. The first 48 bytes are the device's implicit certificate, the next 22 bytes are the Root Certificate Authority's Public Key, the next 21 bytes are the device's private key (the other half of the public/private key pair stored in the certificate), and the last byte is a flags field. The flags field should be set to 0x00 to indicate that the security data is initialized. For more information on the Smart Energy tokens, see document *AN708: Setting Smart Energy Certificates for Zigbee Devices*. *Usage:* Required by Smart Energy Profile certified devices.

Offset from Lock Bits starting address	Size (Bytes)	Name	Description
0x0270	20	TOKEN_MFG_INSTALLATION_CODE	Defines the installation code for zigbee devices. The installation code is used to create a pre-configured link key for initially joining a zigbee 3.0 or Zigbee Smart Energy (ZSE) network. The first 2 bytes are a flags field, the next 16 bytes are the installation code, and the last 2 bytes are a CRC. Valid installation code sizes for ZSE devices are 6, 8, 12, or 16 bytes in length, but zigbee 3.0 devices are required to use 16-byte installation codes. All unused bytes should be 0xFF. The flags field should be set as follows depending on the size of the install code: 6 bytes = 0x0000 8 bytes = 0x0002 12 bytes = 0x0004 16 bytes = 0x0006 For more information on working with zigbee installation codes, see document *AN1089: Using Installation Codes with Zigbee Devices*. For more information on the Smart Energy tokens, see document *AN708: Setting Smart Energy Certificates for Zigbee Devices*. *Usage*: Required by zigbee 3.0 and Zigbee Smart Energy compliant devices.
0x0284	2	TOKEN_MFG_SECURITY_CONFIG	(EmberZNet PRO only) Defines the security policy for application calls into the stack to retrieve key values. The API calls emberGetKey() and emberGetKeyTableEntry() are affected by this setting. This prevents a running application from reading the actual encryption key values from the stack. This token may also be set at runtime with emberSetMfgSecurityConfig() (see that API for more information). The stack utilizes the emberGetMfgSecurityConfig() to determine the current security policy for encryption keys. The token values are mapped to the EmberKeySettings stack data type (defined in ember-types.h). See the following table for the mapping of token values to the stack values. *Usage:* Optional for EmberZNet NCP devices wishing to limit access to security data over the serial interface.
0x0286	16	TOKEN_MFG_SECURE_BOOTLOADER_KEY	This token holds the 128 bit key used by the secure bootloader to decrypt encrypted Ember Bootloader (EBL) and Gecko Bootloader (GBL) files. A value of all F's is considered an invalid key and will not be used by the secure bootloader. *Usage:* Required only if using one of the Ember bootloaders with the "secure" prefix, or the Gecko Bootloader with support for encrypted GBL files enabled. See *UG266: Gecko Bootloader User Guide*, for more information on the Gecko Bootloader.

Offset from Lock Bits starting address	Size (Bytes)	Name	Description
0x0296	148	TOKEN_MFG_CBKE_283K1_DATA	Defines the security data necessary for Smart Energy 1.2 devices using the ECC 283k1 curve. It is used for Certificate Based Key Exchange to authenticate a device on a Smart Energy network. The first 74 bytes are the device's implicit certificate, the next 37 bytes are the Root Certificate Authority's Public Key, the next 36 bytes are the device's private key (the other half of the public/private key pair stored in the certificate), and the last byte is a flags field. The flags field should be set to 0x00 to indicate that the security data is initialized. For more information on the Smart Energy tokens, see document *AN708: Setting Smart Energy Certificates for Zigbee Devices*. *Usage:* Required for ZigBee Smart Energy 1.2 devices.
0x34A	32	TOKEN_MFG_SIGNED_BOOTLOADER_KEY_X	These tokens hold the 256-bit X and Y components of the ECDSA P-256 public key used by the Gecko bootloader to perform cryptographic signature verification of signed GBL files ("secure bootload") and signed application images ("secure boot"). A value of all F's is considered an invalid key. See *UG266: Gecko Bootloader User Guide*, for more information on the Gecko Bootloader. *Usage*: Required only if using the Gecko bootloader with secure boot or support for signed GBL files is enabled.
0x36A	32	TOKEN_MFG_SIGNED_BOOTLOADER_KEY_Y	
0x38A	34	TOKEN_MFG_THREAD_JOIN_KEY	This token holds the random join key used to join a Thread Network. This token is for use with the Silicon Labs Thread stack only. This token contains 32 bytes of space for a random join key which needs to be null-terminated and two bytes that contain the length of the join key. The programmed length includes the null terminator (0x00) and any bytes beyond the programmed length are ignored (no special padding values required). *Usage*: Optional for a Thread Network

Table 3.3. Mapping of EmberKeySettings to `TOKEN_MFG_SECURITY_CONFIG`

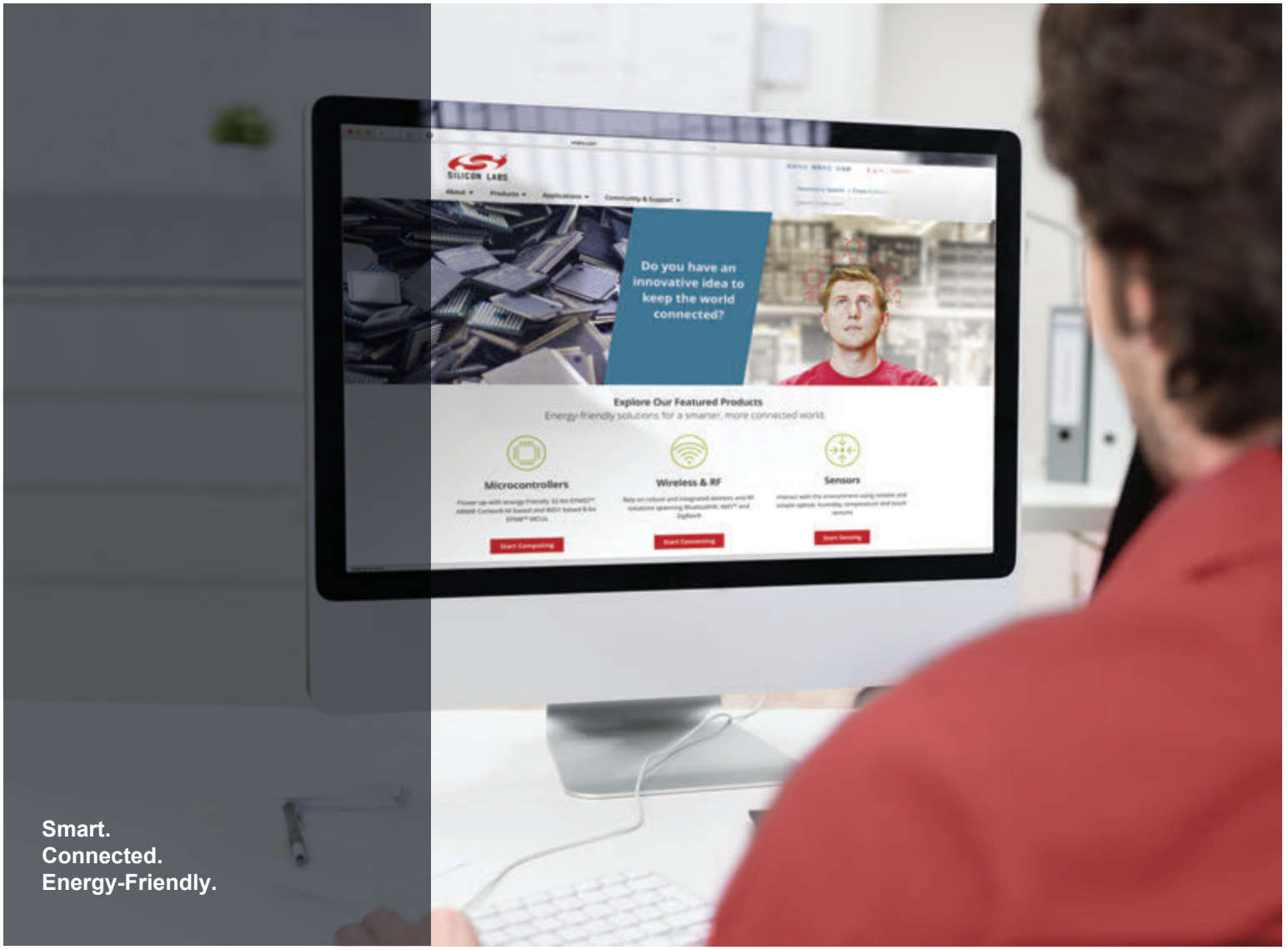
EmberKeySettings Value	TOKEN_MFG_SECURITY_CONFIG Value
`EMBER_KEY_PERMISSIONS_NONE`	0x0000
`EMBER_KEY_PERMISSIONS_READING_ALLOWED`	0x00FF
`EMBER_KEY_PERMISSIONS_HASHING_ALLOWED`	0xFF00

4. Testing

At this point, you may want to perform a simple send/receive test on the new device to determine its range and generally test its radio functionality.

For customers who are using mature applications with the EmberZNet PRO or Silicon Labs Thread SDKs, Silicon Labs recommends using the manufacturing test library (also known as “mfglib”), a set of APIs or NCP serial commands provided for IEEE 802.15.4-based functional testing of the radio from within a normal EmberZNet PRO or Silicon Labs Thread application for an SoC or host platform. This library is typically used in the low-volume and high-volume phases of testing and manufacturing, where additional programming steps for a dedicated RF test application add unnecessary test time. For SoC applications based on EmberZNet PRO or Silicon Labs Thread, as well as for custom zigbee NCP builds through the NCP Framework, a “Manufacturing Library” plugin is provided as part of the corresponding application framework to enable this functionality in the build. The user must then add code to the application to invoke these functions (described in stack/include/mfglib.has well as *UG100: EZSP Reference Guide*) in the desired context. A serial command line-driven interface to these functions is available in EmberZNet PRO SoC and Host applications via the Manufacturing Library CLI plugin, and an over-the-air (OTA)-driven interface is available in EmberZNet PRO SoC applications via the Manufacturing Library OTA plugin.

If the new device fails to successfully transmit or receive packets with the known good device, you may want to attach the new device to a signal generator or network analyzer to verify that generated packets on the target frequency can be received and that the new device can transmit accurately at the center frequency of the selected channel. Other tests may be required for FCC or CE compliance testing.



Products
www.silabs.com/products

Quality
www.silabs.com/quality

Support and Community
community.silabs.com

SILICON LABS®

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

http://www.silabs.com