
BRINGING UP CUSTOM DEVICES FOR THE EMBER[®] EM35x SoC OR NCP PLATFORM

(Formerly document 120-5064-000)

Before an Ember EM35x-based product can be initialized and tested, SIMEE tokens within the EM35x Customer Information Block (CIB) must be configured. Similarly, before any application specific code can be programmed into the EM35x flash, a board header file (that is, board-header.c) must be created. In order to perform these tasks, the product design team must know how the EM35x 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)
- 24 MHz crystal oscillator
- Application security tokens (Keys, certificates, and so on)

This application note describes how to initialize a piece of custom hardware (a “device”) based on the EM35x so that it interfaces correctly with the EmberZNet PRO network stack. The same procedures can be used to restore Ember development kit devices whose settings have been corrupted or erased.

Even though the EM35x flash is fully tested during production test, the flash contents in the main flash block (MFB) are not set to a known state before shipment. The CIB is left erased with read protection disabled (read protection is an option byte and part of the CIB manufacturing tokens).

New in This Revision

New section 1.

Contents

1	EM35x Wireless System.....	2
1.1	Product Information	2
2	Setting CIB Manufacturing Tokens.....	2
3	Bootloaders.....	7
4	Running the nodetest Application.....	8
4.1	Uploading and Running nodetest	8
4.2	nodetest Commands	8
5	Performing Functional Testing.....	8
6	Setting Stack Tokens.....	9

1 EM35x 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. Table 1 describes the different items that developers should know before board bring-up.

Table 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 EM35x. These might include where and when the device was assembled, a product serial number and product name.
Custom EUI	Optional	IEEE 64-bit unique number. Each EM35x comes with an Ember EUI programmed into the FIB. Customers that have their own IEEE Block can use it in place of the Ember EUI.
RF Component-Information	Required	Most products use external power amplifiers (PAs) or low noise amplifiers (LNAs) in order to maximize the range of the communication link. If the product uses PAs or LNAs, then developers must know the pin-connections between the off-chip components and the EM35x. They should also understand the LNA Gain as it will be used to offset the clear channel assessment (CCA) threshold. In addition to amplification, the EM35x allows using a synthesizer offset to trim the initial frequency tolerance of the 24 MHz crystal and its loading capacitors.
ZigBee-Specific Information	Required	Developers must know the ZigBee-assigned manufacturer code before testing.
System Level		
Bootloader Option	Required	Silicon Labs offers two bootloading options: application bootloader and a stand-alone bootloader.
System Security	Required	ZigBee profiles define specific security protocols for a device to follow.

1.1 Product Information

As detailed in Table 1, 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.

2 Setting CIB Manufacturing Tokens

Ember EM35x chips are delivered to customers with only the fixed information block (FIB) programmed. The FIB contains a serial-link-only monitor, chip identifiers, Ember EUI64, and calibration values. The FIB cannot be modified. Before the EM35x chips can be used to run EmberZNet PRO applications, 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 (MFB), as well as programming customer manufacturing tokens in the Customer Information Block (CIB).

CIB manufacturing tokens are values programmed into a special, non-volatile storage area of flash known as the CIB. The CIB contains data that manufacturers of EM35x-based devices can program. The Fixed Information Block (FIB) 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.

Table 2 identifies the CIB manufacturing tokens for EmberZNet PRO that OEMs and CMs may want to program at manufacturing time. Refer to `hal\micro\cortexm3\token-manufacturing.h` for the token definition, because it may differ from Table 2 below depending on the stack release version.

Use the `em3xx_load.exe` utility to set manufacturing tokens. `em3xx_load.exe` can be found in the Debug Adapter (ISA3) Utilities install directory. The default is `C:\Program Files\Ember\ISA3 Utilities\bin\`. `em3xx_load.exe` has many commands, but four commands are specifically designed for manipulating CIB manufacturing tokens, as shown in Table 1.

Table 2. em3xx_load.exe CIB Manufacturing Token Commands

Command line entry	Description
<code>--cibtokenspatch <file></code>	Patch the CIB tokens with the token set defined in the specified file. See <code>--cibtokenspatch-help</code> for additional help on this command.
<code>--cibtokensprint</code>	Print the contents of the CIB tokens from the target in a very easy to read form.
<code>--cibtokensdump</code>	Print the contents of the CIB tokens from the target in a format that can easily be imported using <code>--cibtokenspatch</code> .
<code>--cibtokenspatch-help</code>	Print out additional help about how to set the tokens using <code>--cibtokenspatch</code> , including detailed file syntax and a list of all supported tokens.

Executing **em3xx_load.exe --help** causes `em3xx_load.exe` to print its online help menu detailing all commands, modifiers, and arguments. In addition, document UG107, *EM35x Utilities Guide*, provides many examples detailing how the `em3xx_load.exe` utility can be used. This document also includes examples showing how to print and patch CIB manufacturing tokens.

Silicon Labs recommends that CIB manufacturing tokens be written with an external utility (`em3xx_load.exe`) at the same time as programming the MFB. Using an external utility allows for patching and reprogramming the CIB as many times as necessary. Some situations though, may require that a CIB manufacturing token be programmed at runtime from code running on the chip itself. The manufacturing token module of the HAL provides a token API to write CIB manufacturing tokens. However, this API only writes CIB tokens that are in a completely erased state. If a CIB token must be reprogrammed, you must use an external utility. The API is

`halCommonSetMfgToken(token, data)`. The parameters for this API are the same as the API `halCommonSetToken(token, data)`.

Note: CIB manufacturing tokens written with `halCommonSetMfgToken()` must be located on a 16-bit address boundary and the byte length must be a multiple of 16 bits.

Table 3. CIB manufacturing tokens for the EM35x

Address	Size (Bytes)	Name	Description
0x08040800	16	TOKEN_MFG_CIB_OBS	<p>Dedicated special flash storage called option bytes. Option bytes are special storage because they are directly linked to hardware functionality of the chip. There are 8 option bytes, each occupying 16 bits of flash as follows:</p> <ul style="list-style-type: none"> Option Byte 0: Configures flash read protection Option Byte 1: Reserved Option Byte 2: Available for customer use Option Byte 3: Available for customer use Option Byte 4: Configures flash write protection Option Byte 5: Configures flash write protection Option Byte 6: Configures flash write protection Option Byte 7: Reserved <p>Refer to the <i>EM35x Data Sheet</i> document for a detailed description of option bytes, what they mean, how they work, and what values should be used.</p> <p><i>Usage:</i> All option bytes must be set to a valid state.</p>
0x08040810	2	TOKEN_MFG_CUSTOM_VERSION	<p>Version number to signify which revision of CIB manufacturing tokens you are using. This value should match <code>CURRENT_MFG_CUSTOM_VERSION</code> which is currently set to 0x01FE.</p> <p><code>CURRENT_MFG_CUSTOM_VERSION</code> is defined in <code>\hal\micro\cortexm3\token-manufacturing.h</code>.</p> <p><i>Usage:</i> Recommended for all devices using CIB manufacturing tokens.</p>
0x08040812	8	TOKEN_MFG_CUSTOM_EUI_64	<p>IEEE 64-bit address, unique for each radio. Entered and stored in little-endian. Setting a value here overrides the pre-programmed Ember EUI64 stored in the FIB (TOKEN_MFG_EMBER_EUI_64). This is for customers who have purchased their own address block from IEEE.</p> <p><i>Usage:</i> Optionally set by device manufacturer if using custom EUI64 address block.</p>
0x0804081A	16	TOKEN_MFG_STRING	<p>Optional device-specific string, for example, the serial number.</p> <p><i>Usage:</i> Optionally set by device manufacturer to identify device.</p>
0x0804082A	16	TOKEN_MFG_BOARD_NAME	<p>Optional string identifying the board name or hardware model.</p> <p><i>Usage:</i> Optionally set by device manufacturer to identify device.</p>
0x0804083A	2	TOKEN_MFG_MANUF_ID	<p>16-bit ID denoting the manufacturer of this device. Silicon Labs recommends setting this value to match your ZigBee-assigned manufacturer code, such as in the stack's <code>emberSetManufacturerCode()</code> API call.</p> <p><i>Usage:</i> Recommended for devices utilizing the standalone bootloader.</p>

Address	Size (Bytes)	Name	Description												
0x0804083C	2	TOKEN_MFG_PHY_CONFIG	<p>Default configuration of the radio for power mode (boost/normal), transmit path selection (bi-directional/alternate), and boost transmit power level:</p> <ul style="list-style-type: none">Bit 0 (lsb): Set to 0 for boost mode; set to 1 for non-boost (normal) mode. Boost mode mainly increases rx sensitivity but also increases tx output power by a small amount. Larger increases in tx output power are available in bits 3-7 of this token.Bit 1: Set to 0 if an external PA is connected to the alternate TX path (RF_TX_ALT_P and RF_TX_ALT_N pins); set to 1 otherwise.Bit 2: Set to 0 if an external PA is connected to the bi-directional RF path (RF_P and RF_N pins); set to 1 otherwise.Bits 3-7: Set to the boost tx power offset from the first integer value above nominal maximum tx power (+3dBm). <table><tr><th>For tx power of...</th><th>Set this subfield to...</th></tr><tr><td>+4 dBm</td><td>0</td></tr><tr><td>+5 dBm</td><td>1</td></tr><tr><td>+6 dBm</td><td>2</td></tr><tr><td>+7 dBm</td><td>3</td></tr><tr><td>+8 dBm</td><td>4</td></tr></table> <p>Bit 7 is most significant; Bit 3 is least significant. All bits must be set to 1 if tx boost power level is not desired.</p> <ul style="list-style-type: none">Bits 8–15: Reserved. Must be set to 1 (the erased state). <p><i>Usage:</i> Required for devices that utilize boost mode or an external PA circuit.</p>	For tx power of...	Set this subfield to...	+4 dBm	0	+5 dBm	1	+6 dBm	2	+7 dBm	3	+8 dBm	4
For tx power of...	Set this subfield to...														
+4 dBm	0														
+5 dBm	1														
+6 dBm	2														
+7 dBm	3														
+8 dBm	4														
0x0804083E	16	TOKEN_MFG_BOOTLOAD_AES_KEY	<p>Sets the AES key used by the bootloader utility to authenticate bootloader launch requests of the standalone bootloader.</p> <p><i>Usage:</i> Required for devices that utilize the standalone bootloader. This is not used by the application bootloader.</p>												
0x0804084E	8	TOKEN_MFG_EZSP_STORAGE	<p>An 8-byte, general-purpose token that can be set at manufacturing time and read by the host microcontroller via EZSP's <code>getMfgToken</code> command frame.</p> <p><i>Usage:</i> Not required. Device manufacturer may populate or leave empty as desired.</p>												
0x0804087E	92	TOKEN_MFG_CBKE_DATA	<p>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.</p> <p><i>Usage:</i> Required by Smart Energy Profile certified devices.</p>												

AN710

Address	Size (Bytes)	Name	Description
0x080408DA	20	TOKEN_MFG_INSTALLATION_CODE	<p>Defines the installation code for Smart Energy devices. The installation code is used to create a pre-configured link key for initially joining a Smart Energy 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.</p> <p>Valid installation code sizes are 6, 8, 12, or 16 bytes in length. All unused bytes should be 0xFF. The flags field should be set as follows depending on the size of the install code:</p> <ul style="list-style-type: none"> 6 bytes = 0x0000 8 bytes = 0x0002 12 bytes = 0x0004 16 bytes = 0x0006 <p><i>Usage:</i> Required by Smart Energy Profile certified devices.</p>
0x080408EE	2	TOKEN_MFG_OSC24M_BIAS_TRIM	<p>A relative amount (between 0x0000 and 0x000F) that trims the 24 MHz crystal bias drive. A lower value reduces drive and current consumption, but increases instability. Although a value can be set here manually, Silicon Labs recommends leaving this value unpopulated (0xFFFF) so that the stack can perform auto-calibration at runtime to determine the optimal value for lowest current consumption while maintaining stability.</p> <p><i>Usage:</i> Not recommended (leave erased, 0xFFFF).</p>
0x080408F0	2	TOKEN_MFG_SYNTH_FREQ_OFFSET	<p>An offset applied to the radio synthesizer frequency. This offset can be used to compensate for variations in 24 MHz crystals to accurately center IEEE-802.15.4 channels.</p> <ul style="list-style-type: none"> Bits 0-7: Set to the signed offset to be applied to the synthesizer frequency. Each step provides approximately 11 kHz of adjustment. Bit 8: Set to 0 if the offset is valid and should be used. Set to 1 (the erased state) if the token is invalid or has not been set. Bits 9-15: Reserved. Must be set to 1 (the erased state).
0x080408F2	2	TOKEN_MFG_OSC24M_SETTLE_DELAY	<p>The delay in microseconds to allow the 24 MHz crystal to settle after a bias change when waking up from deep sleep. Silicon Labs recommends leaving this value unpopulated (0xFFFF) so that the stack uses its default conservative delay value.</p>
0x080408F4	2	TOKEN_MFG_SECURITY_CONFIG	<p>Defines the security policy for application calls into the stack to retrieve key values. The API calls <code>emberGetKey()</code> and <code>emberGetKeyTableEntry()</code> 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 <code>emberSetMfgSecurityConfig()</code> (see that API for more information). The stack utilizes the <code>emberGetMfgSecurityConfig()</code> to determine the current security policy for encryption keys. The token values are mapped to the <code>EmberKeySettings</code> stack data type (defined in <code>ember-types.h</code>). See Table 3 below for the mapping of token values to the stack values.</p>

Address	Size (Bytes)	Name	Description
0x080408F6	2	TOKEN_MFG_CCA_THRESHOLD	<p>Threshold used for energy detection clear channel assessment (CCA).</p> <ul style="list-style-type: none"> Bits 0-7: Set to the two's complement representation of the CCA threshold in dBm below which the 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 is valid and should be used. Set to 1 (the erased state) if the token is invalid or has not been set; in this case the default threshold will be used. Bits 9-15: Reserved. Must be set to 1 (the erased state). <p>The default CCA threshold for EM35x chips is -75 dBm. If bit 8 of this token is 1 then -75 dBm will be used.</p> <p>You may want to override the default CCA threshold by setting this token if your design uses an 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.</p> <p>Example: A design uses an LNA that provides gain of +12 dBm. Add the default -75 dBm gain to the LNA's gain to get the dBm value for the token: $-75 + 12 = -63$ dBm.</p> <p>The two's complement signed representation of -63 dBm is 0xC1, and so the complete token value to be programmed is 0xFEC1.</p>
0x080408F8	16	TOKEN_MFG_SECURE_BOOTLOADER_KEY	<p>This token holds the 128 bit key used by the secure bootloader to decrypt encrypted EBL files. A value of all F's is considered an invalid key and will not be used by the secure bootloader.</p>

Table 4. 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

For more information on the Smart Energy tokens, see document AN708, *Setting Manufacturing Certificates and Installation Codes*.

3 Bootloaders

For the EM35x, Silicon Labs provides two bootloader options:

- Application bootloader: Supports multi-hop bootloading from a gateway device. Enables an application to continue running and sending normal application packets while the new firmware image is being received.
- Stand-alone bootloader: Supports only one-hop bootloading from a gateway device. If some nodes are multiple hops away from a gateway, they must either be brought closer to the gateway for bootloading or you must create a portable device that is taken around the installation to bootload all of the nodes.

For a discussion on what the bootloaders are, their differences, their uses, and how to use the bootloaders, refer to document UG103.6, *Application Development Fundamentals: Bootloading*.

For a discussion on how to program applications and bootloaders to chips as well as convert applications to the bootloader compatible .ebl file format, refer to document UG107, *EM35x Utilities Guide*.

4 Running the nodetest Application

The nodetest application is included in the EmberZNet PRO stack installation directory, in the /app/nodetest subdirectory. This directory contains only .s37 file images that can only be installed onto a chip through the normal em3xx_load.exe procedure over an Ember Debug Adapter (ISA3).

4.1 Uploading and Running nodetest

nodetest can be installed using either Ember Desktop to upload the application or with em3xx_load.exe from a command line. The following is an example of loading nodetest from a command line:

```
$ em3xx_load.exe --ip myisahostname ./app/nodetest/nodetest-em357.s37
```

For a discussion on how to program chips using em3xx_load.exe from a command line, refer to document UG107, *EM35x Utilities Guide*.

Once the upload completes, you can interact with nodetest via the hardware UART or the Virtual UART. Using the Virtual UART requires telnetting to a Debug Adapter (ISA3), port 4900, with the Packet Trace Port cable connected to the chip. You can use the hardware UART in either of these methods:

- Using a direct UART connection (the Breakout Board instantiates an RS-232 connector)
- Using the passthrough UART mode and telnetting to the Debug Adapter (ISA3), port 4901, connected to a Breakout Board using the DEI Port.

Press **Enter** in the telnet window or com port window to start the nodetest application.

Note: If using the hardware UART without the Debug Adapter (ISA3), you must configure your comm port program to 115,200 bps, no parity bits, 1 stop bit. If using the hardware UART via the passthrough mode of the Debug Adapter (ISA3), port 4901, configure the Debug Adapter (ISA3) to 115,200 bps by executing the command ember> port 1 115200. Execute the Debug Adapter (ISA3) configuration command by telnetting to the Debug Adapter (ISA3) administration interface at TCP port 4902.

Note: Every time nodetest resets, it will not automatically print any characters. Instead, nodetest will listen on both the hardware UART and the Virtual UART looking for a Return key. Whichever port nodetest finds a Return key on will become the port that nodetest uses for all serial interaction until nodetest resets again.

4.2 nodetest Commands

A Return key initiates the nodetest application on reset. This displays the power-up prompt, ending in the > (greater than) symbol. Pressing the Return key at the prompt will always cause another prompt to be displayed. Executing the **help** command causes nodetest to print a list of all available commands with a brief description of the commands. The commands are listed by functional modules.

5 Performing Functional Testing

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

Note: When programming a device for test or retest, use the `—erase` option to ensure that all previous calibration data is erased. Silicon Labs recommends erasing the flash contents of a device prior to testing to ensure the calibration is executed at the time the device is tested.

1. Connect a device known to be in good operating order either to your computer or to a different computer through a serial port.
2. Upload `nodetest` to the known good device and then run it.
3. Make sure that `nodetest` is still running on the new device (you should see the `>` prompt).
4. Set both devices to a channel by typing `setchannel X`, where `X` is the channel in hex.
5. Optional: Set a power level on the test device by typing `settxpower X`, where `X` is the power level in hex.
6. On the known good device, type `rx`, which sets the device to receive and display statistics for each packet received. Type `e` to exit.
7. On the test device, type `tx X` to transmit `X` packets where `X` is in hex. (Note that `tx 0` sends infinite packets.) Type `e` to exit.
8. Reverse this procedure to test receiving on the test device.

Note: The fourth column in the display output, labeled “per”, shows the packet error rate. For this value to be accurate, the two devices being used must be configured per “Uploading and running `nodetest`” and the receiver should not hear any other devices. Exiting the test and restarting clears the values and reset the values being displayed.

Note: `nodetest` attempts to print packet data as fast as it can, but it is possible to receive packets faster than `nodetest` can print. Therefore, there may be gaps in the printed packets.

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.

6 Setting Stack Tokens

The EmberZNet PRO stack maintains several non-volatile settings (tokens) used for network operation. Additionally, certain applications may require the use of their own tokens for non-volatile data storage.

Note: `nodetest` cannot be rebuilt with application tokens.

To use the `nodetest` application for programming stack token values:

1. Install the `nodetest` application on the device.
2. Using a serial or telnet connection with port speed set to 115200 bps, press Enter to initiate the `nodetest` application.
3. Use the `tokmap` command to output the map of the token storage area.
4. Use the `loadtoks` command to initialize all stack tokens to their default values.
5. Use the `tokdump` command to confirm the values the tokens.
6. Use the `tokread` and `tokwrite` commands to view and manipulate individual tokens.

Note: For more information about tokens and the Simulated EEPROM, refer to the `token.h` File Reference in the EmberZNet API Reference for your Ember chip and document AN703, *Using the Simulated EEPROM*.

CONTACT INFORMATION

Silicon Laboratories Inc.

400 West Cesar Chavez

Austin, TX 78701

Tel: 1+(512) 416-8500

Fax: 1+(512) 416-9669

Toll Free: 1+(877) 444-3032

Please visit the Silicon Labs Technical Support web page for ZigBee products:

www.silabs.com/zigbee-support and register to submit a technical support request

Patent Notice

Silicon Labs invests in research and development to help our customers differentiate in the market with innovative low-power, small size, analog-intensive mixed-signal solutions. Silicon Labs' extensive patent portfolio is a testament to our unique approach and world-class engineering team.

The information in this document is believed to be accurate in all respects at the time of publication but is subject to change without notice. Silicon Laboratories assumes no responsibility for errors and omissions, and disclaims responsibility for any consequences resulting from the use of information included herein. Additionally, Silicon Laboratories assumes no responsibility for the functioning of undescribed features or parameters. Silicon Laboratories reserves the right to make changes without further notice. Silicon Laboratories makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Silicon Laboratories assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. Silicon Laboratories products are not designed, intended, or authorized for use in applications intended to support or sustain life, or for any other application in which the failure of the Silicon Laboratories product could create a situation where personal injury or death may occur. Should Buyer purchase or use Silicon Laboratories products for any such unintended or unauthorized application, Buyer shall indemnify and hold Silicon Laboratories harmless against all claims and damages.

Silicon Laboratories and Silicon Labs are trademarks of Silicon Laboratories Inc.

Other products or brandnames mentioned herein are trademarks or registered trademarks of their respective holders.