AN3188
Application note

Preparing custom devices for the STM32W108 platform

1 Introduction

The STM32W108 chips are delivered to customers with only the Fixed Information Block (FIB) programmed. The FIB contains a serial-link-only bootloader, chip identifiers, EUI64, and calibration values. The FIB cannot be modified. Before the STM32W108 chips can be used to run EmberZNet™ applications, the customer or a contract manufacturer/test house must prepared 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).

This application note describes how to initialize a piece of STMicroelectronics Enabled™ custom hardware (a “device”) based on the STM32W108 so that it interfaces correctly with the EmberZNet network stack. The same procedures can be used to restore STMicroelectronics Kit devices whose settings have been corrupted or erased.

Even though the STM32W108 SoC Flash memory is fully tested during production tests, the Flash memory contents in the MFB are not set to a known state prior to shipment. The CIB will be left erased with read protection disabled (read protection is an option byte and part of the CIB manufacturing tokens).

Contents

2 Setting CIB manufacturing tokens

CIB manufacturing tokens are values programmed into a special, non-volatile storage area of Flash memory known as the CIB. The CIB contains data that manufacturers of STM32W108-based devices can program. The Fixed Information Block (FIB) also contains manufacturing tokens, but these tokens are fixed values that cannot be modified.

Caution: **Unlike stack and application tokens, which reside in the Simulated EEPROM section of the chip and can be set using the process detailed in the "Setting Stack Tokens" section of this document, data in the CIB, including these tokens, cannot be written by code running on the chip itself (no writes while executing). Applications and the stack can read any manufacturing tokens at runtime.**

Table 1 identifies the CIB manufacturing tokens for EmberZNet 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 1 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 EmberZNet tools/em3xx directory. em3xx_load.exe has many commands, but four commands are specifically designed for manipulating CIB manufacturing tokens.

Table 1. Commands for handling CIB manufacturing tokens

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

Table 2. CIB manufacturing tokens for the STM32W108

Address	Size (Bytes)	Name	Description
0x08040800	16	TOKEN_MFG_CIB_OBS	Dedicated special Flash memory 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 memory as follows: – 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 Refer to the STM32W108 Datasheet for a detailed description of option bytes, what they mean, how they work, and what values should be used. Usage: All option bytes must be set to a valid state.
0x08040810	2	TOKEN_MFG_CUSTOM_VERSION	Version number to signify which revision of CIB 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\token-manufacturing.h. Usage: Recommended for all devices using CIB manufacturing tokens.
0x08040812	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 pre-programmed EUI64 stored in the FIB (TOKEN_MFG_EMBER_EUI_64). This is for customers who have purchased their own address block from IEEE. Usage: Optionally set by device manufacturer if using custom EUI64 address block.
0x0804081A	16	TOKEN_MFG_STRING	Optional device-specific string, for example, the serial number. Usage: Optionally set by device manufacturer to identify device.
0x0804082A	16	TOKEN_MFG_BOARD_NAME	Optional string identifying the board name or hardware model. Usage: Optionally set by device manufacturer to identify device.

Table 2. CIB manufacturing tokens for the STM32W108 (continued)

Address	Size (Bytes)	Name	Description
0x0804083A	2	TOKEN_MFG_MANUF_ID	16-bit ID denoting the manufacturer of this device. It is recommended setting this value to match your ZigBee-assigned manufacturer code, such as in the stack's `emberSetManufacturerCode()` API call. Usage: Recommended for devices utilizing the EmberZNet standalone bootloader.
0x0804083C	2	TOKEN_MFG_PHY_CONFIG	Default configuration of the radio for power mode (boost/normal) and power amplifier (PA) selection (internal/external): Bit 0 (LSB): Set to 0 for boost mode; set to 1 for non-boost (normal) mode. – 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); otherwise, set to 1. – Bits 3-15: Reserved. Must be set to 1 (the erased state). Usage: Required for devices that utilize boost mode or an external PA circuit.
0x0804083E	16	TOKEN_MFG_BOOTLOAD_AES_KEY	Sets the AES key used by the EmberZNet bootloader utility to authenticate bootloader launch requests. Usage: Required for devices that utilize the standalone bootloader.
0x0804084E	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. Usage: Not required. Device manufacturer may populate or leave empty as desired.
0x0804087E	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. Usage: Required by Smart Energy Profile certified devices.

Table 2. CIB manufacturing tokens for the STM32W108 (continued)

Address	Size (Bytes)	Name	Description
0x080408DA	20	TOKEN_MFG_INSTALLATION_CODE	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. 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: – 6 bytes = 0x0000 – 8 bytes = 0x0002 – 12 bytes = 0x0004 – 16 bytes = 0x0006 Usage: Required by Smart Energy Profile certified devices.
0x080408EE	2	TOKEN_MFG_OSC24M_BIAS_TRIM	A relative amount (between 0x0000 and 0x000F) that trims the 24MHz crystal bias drive. A lower value reduces drive and current consumption, but increases instability. Although a value can be set here manually, it is recommended 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. Usage: Not recommended (leave erased, 0xFFFF).

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

3 Bootloaders

EmberZNet 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 bootloaders are, their differences, their uses, and how to use them, refer to *UM0923: EmberZNet™ application developer guide*.

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 *UM0924: EmberZNet™ utilities*.

4 Running the Nodetest application

The nodetest application is included in the EmberZNet stack installation directory, in the /app/nodetest subdirectory. This directory contains only .s37 file images that can be installed onto a chip via the normal em3xx_load.exe procedure.

4.1 Uploading and running nodetest

The following is an example of loading nodetest from a command line:

```
$ em3xx_load.exe ./app/nodetest/nodetest.s37
```

For a discussion on how to program chips using em3xx_load.exe from a command line, refer to *UM0924: EmberZNet™ utilities*.

- Once the upload completes, you can interact with nodetest via the hardware serial port.

Press **Enter** in the HyperTerminal window to start the nodetest application.

Note: 1 You must configure your HyperTerminal program to 115,200 bps, no parity bits, 1 stop bit.

2 Every time nodetest resets, it will not automatically print any characters. Instead, nodetest will listen on the serial port looking for a Return key.

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.

1. Connect a device known to be in good operating order either to your computer or to a different computer via 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 hexadecimal format.
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 hexadecimal format. (Note that tx 0 sends infinite packets.) Type `e` to exit.
8. Reverse this procedure to test receiving on the test device.

Note: 1 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.

2 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 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 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 HAL API Reference for your STM32W108 chip and Using the Simulated EEPROM.

7 After reading this document

If you have questions or require assistance with the procedures described in this document, contact STMicroelectronics support at http://www.st.com/mcu, STM32W section.

8 Revision history

Table 3. Document revision history

Date	Revision	Changes
08-Apr-2010	1	Initial release.