AN1428: SiWx917 Debug Lock

This application note describes the architectural overview of Silicon Labs SiWx917 device and summarizes debug lock and unlock functionality with example command references.

SiWx917 is a low-cost, dual-core Wi-Fi device with hardware isolation and best-in-class security features. This document details and provides example code for the debug lock functionality of SiWx917 products.

KEY POINTS

- Architectural Overview of Debug Lock
- Debug Token and challenge format
- Using the Debug Lock Command API
- Examples of Locking and Unlocking commands

Copyright © 2024 by Silicon Laboratories

1. Introduction

The Silicon Labs SiWx917 is a dual-core, ultra-low-power Wi-Fi 6 and Bluetooth Low Energy (BLE) SoC ideal for developing battery-operated devices that need long battery life. The cryptographic and wireless protocol subsystem consists of a multi-threaded processor (Network Wireless Processor), integrated baseband digital signal processing and analog front-end with integrated power amplifier. The Applications Micro-processor (MCU) is an ARM Cortex-M4F with embedded SRAM, flash, an AI/ML accelerator and a security engine designed for high device and protocol security. Advanced security features such as Secure Boot, Anti-Rollback protection, Debug Lock, and hardware isolation of the Network Wireless Processor Subsystem (NWPSS) from the MCU Subsystem (MCSS) make the Silicon Labs SiWx917 a best-in-class, security-conscious platform for ultra-low-power, low-cost Wireless IoT devices.

2. Debug Lock

To aid in the development and debug process, both cores in the SiWx917 have standard Joint Test Action Group (JTAG) debug interfaces providing access to memory, registers, and halt mode debugging of firmware operating on each device. For security purposes, this functionality is disabled by default, preventing malicious entities from accessing sensitive data or tampering with normal device operation. However, in the event that a device must be debugged in the field, the SiWx917 bootloader implements a cryptographically-secured debug activation protocol allowing the holder of a firmware's private sign key to generate a debug token that can be used to unlock the debug port upon reboot. This debug lock functionality allows developers to quickly triage device and firmware failures without compromising security during normal operation. This document will describe the hardware and firmware architecture of the debug lock functionality of the SiWx917, detail the debug lock/unlock process and lock/unlock API, and provide example commands to lock and unlock both debug ports on the NWPSS and MCSS, respectively.

3. Architectural Overview

The SiWx917 is a low-cost Wi-Fi device with advanced security features. In the following sections, hardware isolation, secure boot, reset behavior, and debug token storage will be outlined with the intent of establishing a background for understanding further sections on the Debug Lock behavior of this device.

3.1 eFuse

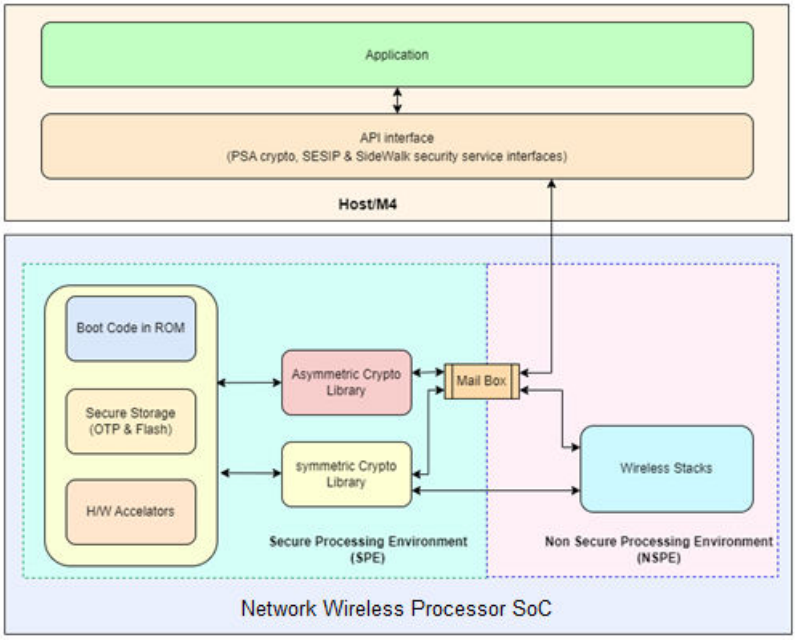
eFuses are a set of OTP configurations that determine the behavior of the SiWx917. The eFuse settings relevant to debug lock are described in the following table.

Table 3.1. eFuse Settings

eFuse Name	Description	Default
disable_ta_jtag	Read/write bit to enable/disable the NWPSS JTAG port on startup	Disabled (1)
disable_ta_jtag_status	Read only bit indicating the accesibility of the NWPSS JTAG port	Disabled (1)
disable_m4_jtag	Read/write bit to enable/disable the MCSS JTAG port on startup	Disabled (1)
disable_m4_jtag_status	Read only bit indicating the accesibility of the MCSS JTAG port	Disabled (1)

3.2 NWPSS and Secure Zone

To provide isolation of mission-critical communication protocol and security functionality, the SiWx917 has two cores: the MCSS, tasked with execution of application-level code, and the NWPSS – which handles communication protocol implementations and sensitive cryptographic functionality. By restricting code on the NWPSS to a minimum subset of protocol and cryptographically sensitive operations, the NWPSS firmware base can remain small and easily auditable, with broad automation test coverage. The MCSS can then consist of a larger codebase and attack surface while still ensuring that mission critical hardware and cryptographic operation remains secure in the event of an exploit in the broader operational codebase.



To protect device registers, memory access, and program flow control from malicious entities, the debug ports of both the NWPSS and MCSS cores are disabled by default on reboot. Debug access can only be enabled by the NWPSS to prevent vulnerabilities in application code from allowing unauthorized debug access. NWPSS memory and peripherals are isolated in hardware from the MCSS by the SecureZone interface, ensuring that all transactions take place through the secure Mailbox and associated APIs. By restricting interaction between the MCSS and NWPSS, vulnerabilities can be prevented from spreading across the SecureZone boundary. Because of these architectural considerations, the debug unlock process requires a debug challenge be provided to the NWPSS.

3.3 Secure Boot

The secure bootloader of the SiWx917 consists of two mandatory stages and one optional second-stage bootloader patch, effectively acting as a third stage. The first stage is present in ROM and is not updatable or patchable. This First Stage Bootloader (FSB) forms the immutable root of trust upon which SiWx917's secure code platform relies. The immutable FSB is responsible for the following operations:

- Sensing and applying EFUSE settings
- Performing hardware-specific device setup (calibration, reset value modifications, etc.)
- Validating and executing SecureBoot and ImageLoad of external flash, if present
- Validating and executing SecureBoot and ImageLoad from external primary devices if flash is not present
- Applying patches to the Second Stage Bootloader (SSB)

The FSB is also responsible for writing to write-once registers that configure hardware peripherals, and in particular is responsible for locking the debug ports. On reset, the FSB will write to the debug disable registers for both the NWPSS and MCSS unless a valid debug token is stored in flash.

The bootloader communicates with the host over UART as follows:

Table 3.2. Bootloader UART Settings

Baud rate	115200
parity	none
Data bits	8
Stop bits	1
Tx pin	GPIO_9
Rx pin	GPIO_8

3.4 Debug Token Data Storage

Because the immutable FSB is responsible for writing the write-once debug lock register, which is only writable once after reset, any debug unlock token must be passed to the FSB and the device must be rebooted for a change in the state of either JTAG port to take effect. The nonce from this debug unlock token is stored AES-CBC encrypted by the NWPSS using a Physically Unclonable Function (PUF) wrapped key in flash to persist it for verification by the FSB between reboots. Note that while the nonce can be dumped from flash, because of its PUF wrapped format it can only be successfully decrypted on the device it was originally generated for. The debug token itself consists of one 16-byte nonce generated by the NWPSS after a debug request is issued, a one-byte command denoting which port shall be unlocked, seven bytes of data padding, and a 72-byte ECDSA p256k host signature – which is used to verify the identity of the host device requesting debug access. One debug token may be stored per-core in addition to one unlock token – which consists of a nonce, command, and data pad that must be equivalent to the stored debug token when decrypted. Upon reboot, the NWPSS will check the nonce, command, and signature per-core, writing the write-once debug lock register if the two tokens are not equal.

4. Debug Unlock Process

The following section details the step-by-step process for unlocking and locking the debug interface. Note that for both cores, the process is the same except for the write-once locking register that is written during FSB SecureBoot.

4.1 Provisioning for Debug Lock/Unlock

4.1.1 Firmware Public Keys

Each core in the SiWx917, the NWPSS and MCSS, has a firmware public key used for verifying the authenticity of the firmware intended to run on that core. The same key is also used to verify the authenticity of debug lock/unlock tokens. To lock and unlock the NWPSS and MCSS cores of the SiWx917, the corresponding firmware public keys must be provisioned to the device.

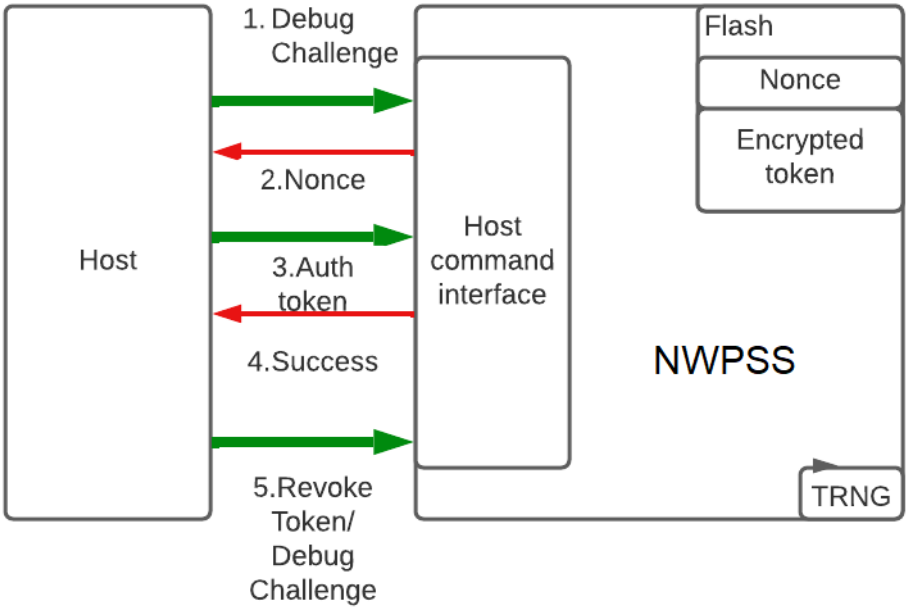
4.1.2 Secure Boot

Secure boot must be enabled in order for the ISP bootloader to be able to unlock either JTAG port. Locking a JTAG port with secure boot disabled results in that port being permanently locked.

4.2 Debug Unlock

To unlock either of the JTAG ports, the host first sends a command to start the debug challenge. This command arrives over an enabled host interface such as UART and will contain an "m" to initiate MCSS unlock or a "t" to initiate NWPSS unlock. In response to this command, the host will generate a 16-byte nonce for the requested device, which it encrypts using a PUF-wrapped key and saves in flash as well as sending the nonce in plaintext back to the host. This token is also known as the Debug Challenge Token.

Note: The M4 JTAG port must be unlocked to unlock the TA JTAG port. Debugging on the TA JTAG port requires a custom JTAG debugger.



Upon receiving the Debug Challenge Token, the host then constructs a payload containing the following:

- 128 bit nonce
- device information defined in the following table

Table 4.1. Device Information

Byte 0	Bytes 1 – 7
t - NWPSS m - M4SS	Reserved

- serial number
- debug command

and signs the token with the firmware update private key for that device. This "authorization" token is then transmitted to the NWPSS, which uses its corresponding firmware public key to verify the token. If the authorization token is valid, its corresponding nonce is stored encrypted in flash. For both nonces and their commands, 7 bytes of random padding are inserted to ensure the two instances of same plaintext do not result in identical ciphertexts.

Table 4.2. Debug Authorization Token

Nonce	Command	User Data	Signature (ASN.1 Format)
16 bytes	1 byte	7 bytes	72 – 74 bytes

Table 4.3. Bootloader Commands for Debug Lock/Unlock

Command	Description
s	Instructs bootloader to expect a debug authorization token
l	Change debug interface
m	MCSS core
t	NWPSS core

4.3 Debug Lock

When a debug session is revoked, the host device simply sends a new Debug Challenge Request that causes the NWPSS to update its stored Debug Challenge Token's nonce, which the host then intentionally does not return, leading to the authorization token to remain at its old value. Note that due to the persistent nature of storing nonces in flash, once a device is unlocked, it will remain unlocked until the nonces are no longer equal in memory either by a revocation or by flash modification. A nonce can be exfiltrated from a device and used to unlock that device at a later time, but since the tokens are encrypted with PUF wrapped keys, it is not possible to use one device's valid debug token to unlock another device.

5. Examples

5.1 Provisioning for Secure Boot

1. Initialize security features:

   commander manufacturing init –mbr default

```
C:\Users\stegerte\Documents\Silabs\Application notes\AN1428>commander manufacturing init --mbr default
WARNING: No serial number or IP address given, cannot lock access to adapter.
Using default MBR for SiWG917M111MGTBA...
Loading TA firmware. Writing 321504 bytes to 0x00400000...
Setting TA PC to 0...
Writing 0xA0AC to 0x41050034...
Found PC set to 0x00000000... (RELEAS_REG=0x00000000, HOLD_REG=0x00000000)
Loading TA firmware. Writing 321504 bytes to 0x00400000...
Setting TA PC to 0...
Writing 0xA0AC to 0x41050034...
Command: 0xa022, Address: 0x00000000, Length: 0x00000000, Flags: 0x00000010
Writing 12 bytes to 0x0042d000...
Interrupting TA...
Command completed with code: 0xa05a
Command: 0xa00c, Address: 0x00002000, Length: 0x00000000, Flags: 0x00000020
Writing 12 bytes to 0x0042d000...
Interrupting TA...
Command completed with code: 0xa05a
Activation code generated successfully
DONE
```

2. Create a JSON file with the keys data:

   commander util genkeyconfig –outfile keys\keys.json –device Si917

```
C:\Users\stegerte\Documents\Silabs\Application notes\AN1428>commander util genkeyconfig --outfile keys\keys.json --device
 Si917
Generating symmetric key...
Generating symmetric key...
Generating ECC P256 key pair...
Generating ECC P256 key pair...
Generating ECC P256 key pair...
Key configuration written to keys/keys.json
DONE
```

3. Save the keys to PEM format as follows:

   commander util extractkeys keys\keys.json –dir keys

```
C:\Users\stegerte\Documents\Silabs\Application notes\AN1428>commander util extractkeys keys\keys.json --dir keys
Writing ATTESTATION_PRIVATE_KEY to 'keys/attestation_private_key.pem'
Writing ATTESTATION_PUBLIC_KEY to 'keys/attestation_public_key.der'
Writing M4_OTA_KEY to 'keys/m4_ota_key.txt'
Writing M4_PRIVATE_KEY to 'keys/m4_private_key.pem'
Writing M4_PUBLIC_KEY to 'keys/m4_public_key.der'
Writing OTA_KEY to 'keys/ota_key.txt'
Writing TA_PRIVATE_KEY to 'keys/ta_private_key.pem'
Writing TA_PUBLIC_KEY to 'keys/ta_public_key.der'
DONE
```

4. Provision keys to the device:

   commander manufacturing provision –keys keys\keys.json

```
C:\Users\stegerte\Documents\Silabs\Application notes\AN1428>commander manufacturing provision --keys keys\keys.json
WARNING: No serial number or IP address given, cannot lock access to adapter.
Reading MBR from the connected device...
Reading 496 bytes from 0x04000000
Reading keys from provided JSON file...
Loading TA firmware. Writing 321504 bytes to 0x00400000...
Setting TA PC to 0...
Writing 0xA0AC to 0x41050034...
Command: 0xa022, Address: 0x00000000, Length: 0x00000000, Flags: 0x00000010
Writing 12 bytes to 0x0042d000...
Interrupting TA...
Command completed with code: 0xa05a
Command: 0xa00d, Address: 0x00002000, Length: 0x00000000, Flags: 0x00000000
Writing 12 bytes to 0x0042d000...
Interrupting TA...
Command completed with code: 0xa05a
Intrinsic keys generated successfully.
Programming TA OTA Key...
Command: 0xa00e, Address: 0x00005000, Length: 0x00000024, Flags: 0x00000020
Writing 12 bytes to 0x0042d000...
Writing 36 bytes to 0x0042d00c...
Interrupting TA...
Command completed with code: 0xa05a
Key successfully stored
Programming M4 OTA Key...
Command: 0xa00e, Address: 0x00005000, Length: 0x00000024, Flags: 0x00000020
Writing 12 bytes to 0x0042d000...
Writing 36 bytes to 0x0042d00c...
Interrupting TA...
Command completed with code: 0xa05a
Key successfully stored
Programming TA Public Key...
Command: 0xa00e, Address: 0x00005000, Length: 0x00000064, Flags: 0x00000020
Writing 12 bytes to 0x0042d000...
Writing 100 bytes to 0x0042d00c...
Interrupting TA...
Command completed with code: 0xa05a
Key successfully stored
Programming M4 Public Key...
Command: 0xa00e, Address: 0x00005000, Length: 0x00000064, Flags: 0x00000020
Writing 12 bytes to 0x0042d000...
Writing 100 bytes to 0x0042d00c...
Interrupting TA...
Command completed with code: 0xa05a
Key successfully stored
Programming attestation Key...
Command: 0xa00e, Address: 0x00005000, Length: 0x000000f4, Flags: 0x00000020
Writing 12 bytes to 0x0042d000...
Writing 244 bytes to 0x0042d00c...
Interrupting TA...
Command completed with code: 0xa05a
Key successfully stored
Programming TA MBR...
Command: 0xa00a, Address: 0x00000000, Length: 0x000001f0, Flags: 0x00000020
```

5. Provision secure boot and security settings:

   commander manufacturing provision –data mbr_security.json

```
C:\Users\stegerte\Documents\Silabs\Application notes\AN1428>commander manufacturing provision --data mbr_security.json
WARNING: No serial number or IP address given, cannot lock access to adapter.
Reading MBR from the connected device...
Reading 496 bytes from 0x04000000
Updating MBR fields...
Reading JSON...
Loading TA firmware. Writing 321504 bytes to 0x00400000...
Setting TA PC to 0...
Writing 0xA0AC to 0x41050034...
Command: 0xa022, Address: 0x00000000, Length: 0x00000000, Flags: 0x00000010
Writing 12 bytes to 0x0042d000...
Interrupting TA...
Command completed with code: 0xa05a
Programming TA MBR...
Command: 0xa00a, Address: 0x00000000, Length: 0x000001f0, Flags: 0x00000020
Writing 12 bytes to 0x0042d000...
Writing 496 bytes to 0x0042d00c...
Interrupting TA...
Command completed with code: 0xa05a
Data loaded successfully
DONE
```

An example mbr_security.json file is provided below:

```
{
    “valids”: 0,
    “puf_activation_code_addr”: 8192,
    “valids”: 0,
 “ ffuse_data”: {
        “disable_m4ss_kh_access”: 0,
        “m4_digital_signature_validation”: 1,
        “m4_encrypt_firmware”: 0,
        “m4_fw_encryption_mode”: 0,
        “m4_secure_boot_enable”: 0,
        “ta_digital_signature_validation”: 1,
        “ta_encrypt_firmware”: 0,
        “disable_m4_access_frm_tass_sec”: 1,
        “ta_secure_boot_enable”: 1,
        “disable_ta_jtag”:0,
        “disable_m4_jtag”:0
    },
    “key_desc_table_addr”: 768
}
```

Figure 5.1. Example eFuse Settings JSON File

5.2 Using Simplicity Commander to Lock/Unlock Debug Access

Operations such as locking or unlocking JTAG port access require access to the ISP bootloader through the UART. On the BRD4002A baseboard (WPK), access is available through the virtual com port (VCOM). WPK firmware version 1v12p4 or greater is required. To enable access to the ISP bootloader through the VCOM port, the following steps are required:

1. Connect the WPK board by USB to your computer.
2. Open Simplicity Studio.
3. Right click the WPK in the "debug adapters" panel and select the "Launch Console" menu item.
4. Select the Admin tab.
5. Enter the following in the text entry box and press enter "serial vcom config handshake aux".
6. Close Simplicity Studio.

Accessing the ISP bootloader with the BRD4001A (WSTK) requires an external USB/UART adapter such as Silicon Labs CP2102. The connections are specified in Table 3.2 Bootloader UART Settings on page 6.

5.2.1 Locking Debug Access

This section demonstrates how to lock the JTAG port of the SiWx917. Saving a copy of the lock token is optional, but it is recommended as the token can be used to unlock the JTAG port by users who do not have access to the private signing key.

1. Put the SiWx917 target device in ISP mode. For instance, hold the ISP button on the radioboard (BRD4338A), and press the reset button on the WSTK/WPK board.
2. Lock the part and save the token as follows:

   commander serial lock M4 --token unlock.token --key signkey.pem --serialport COM14

```
C:\Users\stegerte\Documents\Silabs\Application notes\AN1428>commander serial lock m4 --token lock.token --key ta_private_key.pem
WARNING: No serial number or IP address given, cannot lock access to adapter.
Using serial port 'COM17' for file transfers.
Initializing debug lock...
Nonce generated by the device: D02F21A6AC1CFA2525989543E8BC04B3
Debug access will be locked after the device is reset.
Parsing signing key 'ta_private_key.pem'...
Debuglock token raw data:
d02f21a6ac1cfa2525989543e8bc04b36d000000000000000304402203d044355f78527a0714b774431f74bdfdc2021c0d7282600bf949a18b7c2182f022050d1d0229343e61a2
50bbf6d9e915149cb7c2d031a11ca18237230eaa1ff7d330000
Debuglock token written to 'lock.token'.
DONE
```

3. Reset the device to lock the JTAG port.

5.2.2 Unlocking Debug Access with a Token

1. Put the SiWx917 target device in ISP mode. For instance, hold the ISP button on the radioboard (BRD4338A), and press the reset button on the WSTK/WPK board.
2. Unlock the part as follows: commander serial unlock –token lock.token

```
C:\Users\stegerte\Documents\Silabs\Application notes\AN1428>commander serial unlock m4 --token lock.token
Using the provided token 'lock.token' to unlock debug access...
WARNING: No serial number or IP address given, cannot lock access to adapter.
Using serial port 'COM17' for file transfers.
Verifying debuglock token 'lock.token'...
Initializing debug unlock...
Debug access will be unlocked after the device is reset.
DONE
```

3. Reset the part to unlock the JTAG port.
4. Once the debugging session is complete, update the debug challenge nonce by locking the port as described in 5.2.1 Locking Debug Access.

5.2.3 Unlocking Debug Access with the Private Signing Key

1. Put the SiWx917 target device in ISP mode. For instance, hold the ISP button on the radioboard (BRD4338A), and press the reset button on the WSTK/WPK board.
2. Unlock the JTAG port as follows:

   commander serial unlock m4 --key keys\m4_private_key.pem

```
C:\Users\stegerte\Documents\Silabs\Application notes\AN1428>commander serial unlock m4 --key keys\m4_private_key.pem
WARNING: No serial number or IP address given, cannot lock access to adapter.
Using serial port 'COM17' for file transfers.
Initializing debug unlock...
Nonce generated by the device: 6D6DA3100AB5AD6D2B5BA1B624786528
Parsing signing key 'keys\m4_private_key.pem'...
Debuglock token raw data:
6d6da3100ab5ad6d2b5ba1b62478652b6d000000000000003046022100d8038b7633f74a019d7e3656f9d72a7e5d7b4fbb2c691c85b67bffd79e35556
0022100ed2b57569f2101f3a6006ea119bf1f53577e72b8374b31cce79a480be34de6b3
Debuglock token written to 'C:/Users/stegerte/Documents/Silabs/Application notes/AN1428/token_fDcOXb'.
Debug access will be unlocked after the device is reset.
DONE
```

3. Reset the part to unlock the JTAG port.
4. Once the debugging session is complete, update the debug challenge nonce by locking the port as described in 5.2.1 Locking Debug Access.

6. Revision History

Revision 0.2

May 2024

- Updated name to "SiWx917 Debug Lock"
- Replace "ThreadArch" with "Network Wireless Processor"
- Replaced "TSS" with "NWPSS"

Revision 0.1

February 2024

- 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

Disclaimer
Silicon Labs intends to provide customers with the latest, accurate, and in-depth documentation of all peripherals and modules available for system and software implementers using or intending to use the Silicon Labs products. Characterization data, available modules and peripherals, memory sizes and memory addresses refer to each specific device, and "Typical" parameters provided can and do vary in different applications. Application examples described herein are for illustrative purposes only. Silicon Labs reserves the right to make changes without further notice to the product information, specifications, and descriptions herein, and does not give warranties as to the accuracy or completeness of the included information. Without prior notification, Silicon Labs may update product firmware during the manufacturing process for security or reliability reasons. Such changes will not alter the specifications or the performance of the product. Silicon Labs shall have no liability for the consequences of use of the information supplied in this document. This document does not imply or expressly grant any license to design or fabricate any integrated circuits. The products are not designed or authorized to be used within any FDA Class III devices, applications for which FDA premarket approval is required or Life Support Systems without the specific written consent of Silicon Labs. A "Life Support System" is any product or system intended to support or sustain life and/or health, which, if it fails, can be reasonably expected to result in significant personal injury or death. Silicon Labs products are not designed or authorized for military applications. Silicon Labs products shall under no circumstances be used in weapons of mass destruction including (but not limited to) nuclear, biological or chemical weapons, or missiles capable of delivering such weapons. Silicon Labs disclaims all express and implied warranties and shall not be responsible or liable for any injuries or damages related to use of a Silicon Labs product in such unauthorized applications.

Trademark Information
Silicon Laboratories Inc.®, Silicon Laboratories®, Silicon Labs®, SiLabs® and the Silicon Labs logo®, Bluegiga®, Bluegiga Logo®, EFM®, EFM32®, EFR, Ember®, Energy Micro, Energy Micro logo and combinations thereof, "the world's most energy friendly microcontrollers", Redpine Signals®, WiSeConnect , n-Link, EZLink®, EZRadio®, EZRadioPRO®, Gecko®, Gecko OS, Gecko OS Studio, Precision32®, Simplicity Studio®, Telegesis, the Telegesis Logo®, USBXpress® , Zentri, the Zentri logo and Zentri DMS, Z-Wave®, and others are trademarks or registered trademarks of Silicon Labs. ARM, CORTEX, Cortex-M3 and THUMB are trademarks or registered trademarks of ARM Holdings. Keil is a registered trademark of ARM Limited. Wi-Fi is a registered trademark of the Wi-Fi Alliance. All other products or brand names mentioned herein are trademarks of their respective holders.

SILICON LABS

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

www.silabs.com