UG575: SiWx917 NCP Manufacturing Utility User Guide

Version 1.2

March 2026

This version of UG575 has been deprecated. For the latest version, see docs.silabs.com.

Note: This content may contain offensive terminology that is now obsolete. Silicon Labs is replacing these terms with inclusive language wherever possible. For more information, visit www.silabs.com/about-us/inclusive-lexicon-project.

Table of Contents

1 Introduction 4
1.1 Manufacturing Procedure 4

2 Manufacturing Demonstration Setup 5
2.1 Hardware 5
2.2 Software 5
2.3 Device Connections and Interfaces 5
2.4 Flash Manufacturing Application 6
2.5 Launching the Commander CLI 9
2.6 Manufacturing Utility Commands, Options and Arguments 10

3 Manufacturing Procedure without Device Security 12
3.1 Sample Manufacturing command flow for SiWN917M100LGTBA OPN 13

4 Security Features 15
4.1 Secure Boot 15
4.2 Secure Firmware Update 15
4.3 Security Key Programming 15
4.3.1 Step 1: Generate Activation Code and Flash 16
4.3.2 Step 2: Power Cycle the Device 16
4.3.3 Step 3: Generate Security Keys from Commander 16
4.3.4 Step 4: Load the Security Keys & Program the NWP MBR with Security 16
4.4 Add Security to NWP Connectivity Firmware 19
4.4.1 Secure NWP Connectivity Firmware 19
4.5 Debug Lock 19

5 Boot Configurations 20

6 Efuse Configurations 22

6 Manufacturing Procedure with Device Security 23
6.1 Sample Manufacturing commands flow for SiWN917M100LGTBA 24

7 Load SiWx917 Connectivity Firmware 26
7.1 Firmware loading via Commander CLI 26
7.2 Firmware loading via Commander GUI 26

8 Setting MAC Address 27

9 Performing RF Calibration 28
9.1 Setup 28
9.2 Transmission in Burst Mode 28
9.2.1 Steps for Frequency Offset Correction 28
9.2.2 Steps for Gain Offset Calibration 29
9.3 Transmission in Continuous Mode 29
9.3.1 Steps for Frequency Offset Correction 29
9.3.2 Steps for Gain Offset Calibration 29

10 Possible Error Codes 30

11 Appendix 31
11.1 Reading Manufacturing parameters 31
11.2 Programming efuse command 31
11.3 Sample boot configurations 31
11.4 Sample eFuse only configurations 32
11.5 MIC Protected Content Length Map 32
11.6 OTP write lock process 33
11.7 Security Key Management and Protection 34

12 Revision History 36

1 Introduction

Manufacturing means programming production-specific information into the device. This document outlines the manufacturing procedure for the SiWx917 NCP IC (SiWN917M) /module (SiWN917Y) using Silicon Labs' manufacturing utility.

The SiWx917 NCP has a multi-threaded Network Wireless Processor (NWP), which runs the network and Wireless stacks on independent threads. The SiWx917 NCP requires an external host for application processing.

1.1 Manufacturing Procedure

This section outlines the sequence of steps involved in the manufacturing procedure.
When the SiWx917 NCP IC/module is received for the first time, it comes with the following:

- A default boot configuration (includes default device security) in the NWP Master Boot Record (MBR).
- An empty eFuse.
- A default boot configuration (includes default device security) and default calibration parameters in the eFusecopy.
- No firmware included.
- Uncalibrated.

The manufacturing procedure for the SiWx917 NCP involves the following key steps:

1. Configuring the boot settings, which can either be done in the NWP MBR or eFuse (OTP).
    - The boot settings can be done either in MBR or eFuse. The MBR allows for multiple updates to the boot configurations, providing flexibility for changes. In contrast, eFuse or One-Time Programmable (OTP) memory permits the boot configurations to be updated only once, and once written, they cannot be erased.
2. Loading the SiWx917 connectivity firmware.
3. Setting the Wi-Fi and BLE MAC address (optional).
4. Performing RF calibration (Not required for SiWx917Y module OPNs).

This document demonstrates the manufacturing procedure for SiWx917 NCP using the EFR32MG24/EFR32MG21 host MCU and manufacturing utility or tool - Simplicity Commander CLI.

2 Manufacturing Demonstration Setup

Silicon Labs has developed a manufacturing application for the EFR32MG21 and EFR32MG24 host MCUs. Users input manufacturing commands through the Simplicity Commander CLI, which then translates these commands into a format that the manufacturing application can understand. The manufacturing application processes these commands and subsequently converts them into a format that the SiWx917 NCP can interpret and execute.

2.1 Hardware

- BRD4002A - Si-MB4002A Wireless Pro Kit Mainboard (hereafter referred to as WPK board)
- One of the following host radio boards (hereafter referred to as EFR32 radio boards)
  - BRD4186C - EFR32xG24 Wireless 2.4 GHz +10 dBm Radio Board
  - BRD4180B - EFR32xG21 Wireless 2.4 GHz +20 dBm Radio Board
- One of the following SiWx917 NCP Radio Boards
  - BRD4346A - SiWx917 Wi-Fi 6 and Bluetooth LE 4MB Flash Co-Processor Radio Board (hereafter referred to as SiWx917 radio board)
  - BRD4357A - SiWN917Y Module Wi-Fi 6 and Bluetooth LE 4 MB Flash RF-Pin Co-Processor Radio Board (hereafter referred to as SiWx917 radio board)
- BRD8045A - EXP Adapter Board for SiWx917 Co-Processors (hereafter referred to as adapter board)
- Windows/Linux/MacOS computer with a USB port
- Type C USB cable compatible with the computer's USB port
- Spectrum Analyzer (Required for RF Calibration only)

2.2 Software

- Manufacturing application binary
  - EFR32xG24 application
  - EFR32xG21 application
- Simplicity Studio IDE (Optional)
- Simplicity Commander CLI (hereafter referred to as Commander CLI)
  - The Commander CLI can be obtained in one of the following two ways:
    - **Simplicity Commander from Simplicity Studio**: Included with the Simplicity Studio IDE installation.
    - **Standalone Simplicity Commander**: Available for download and installation from the Silicon Labs website.

NOTE: We recommend using the latest version of Commander CLI.

2.3 Device Connections and Interfaces

- Plug the EFR32 radio board into the radio board connectors of the WPK board as shown below.
- Plug the SiWx917 radio board into the radio board connectors of the adapter board as shown below.
- Connect the adapter board to the EXP header on the EFR32 WPK board.
- Make sure the UART switch on the adapter board is in the USB position.
- Make sure the PWR MODE switch on the adapter board is in the BUF position.
- Connect the EFR32 WPK board to your computer using a type C USB cable.

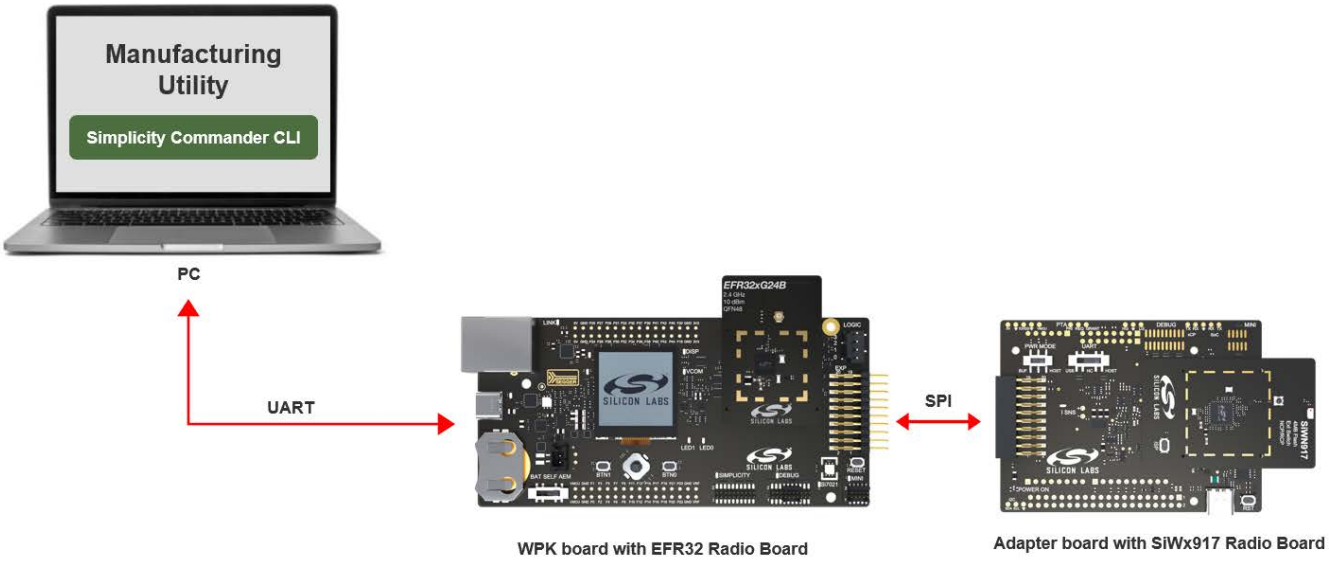
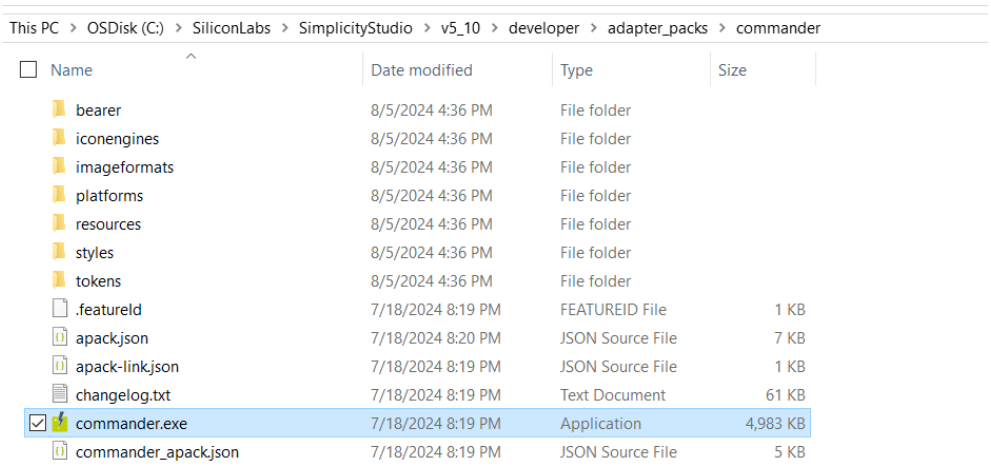


Fig.1.1 Device Connections and Interfaces

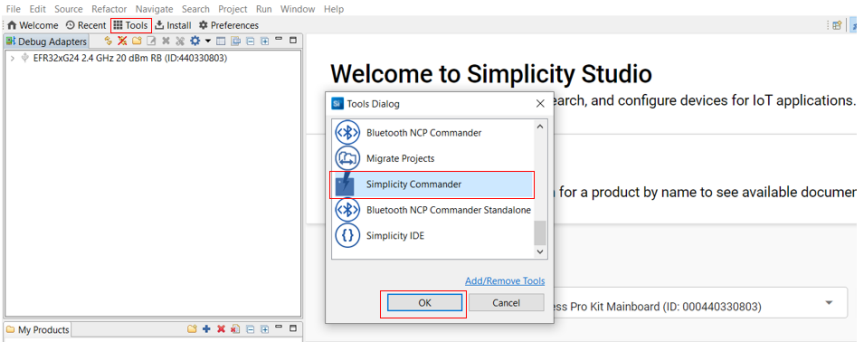
The EFR32MG24/EFR32MG21 host MCU is connected to the SiWx917 NCP via SPI and to the Commander CLI via UART. The subsequent step involves flashing the manufacturing application onto the EFR32 device using the Commander CLI.

2.4 Flash Manufacturing Application

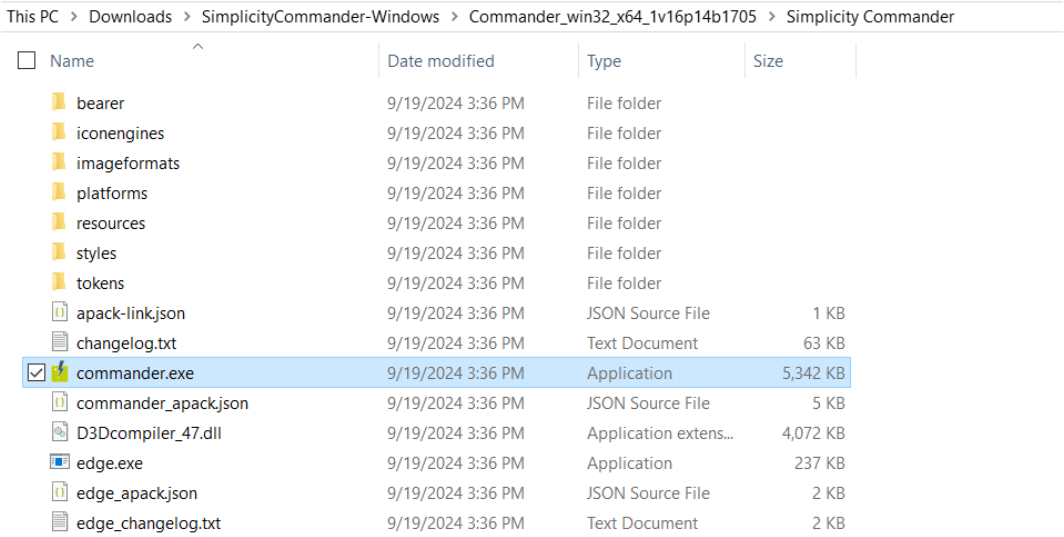
1. Open the Commander GUI
   - If you are using Simplicity Commander from the Simplicity Studio IDE installation, you can either
     - navigate to the following path within the Simplicity Studio installation directory and double click on "**commander.exe**": C:\..\..\SimplicityStudio\v5\developer\adapter_packs\commander or,



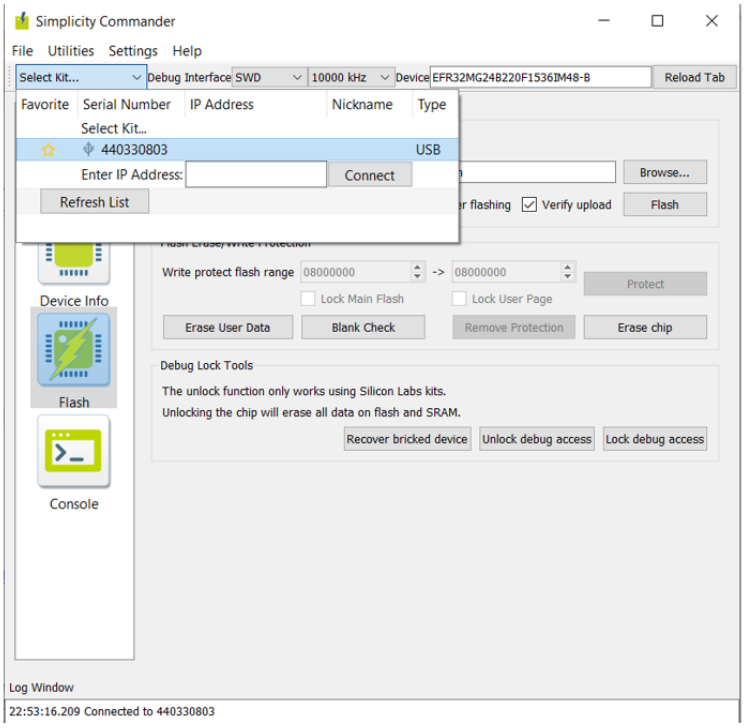
     - go to Simplicity Studio IDE home page, click "**Tools**" and select "**Simplicity Commander**" from the **Tools Dialog**.



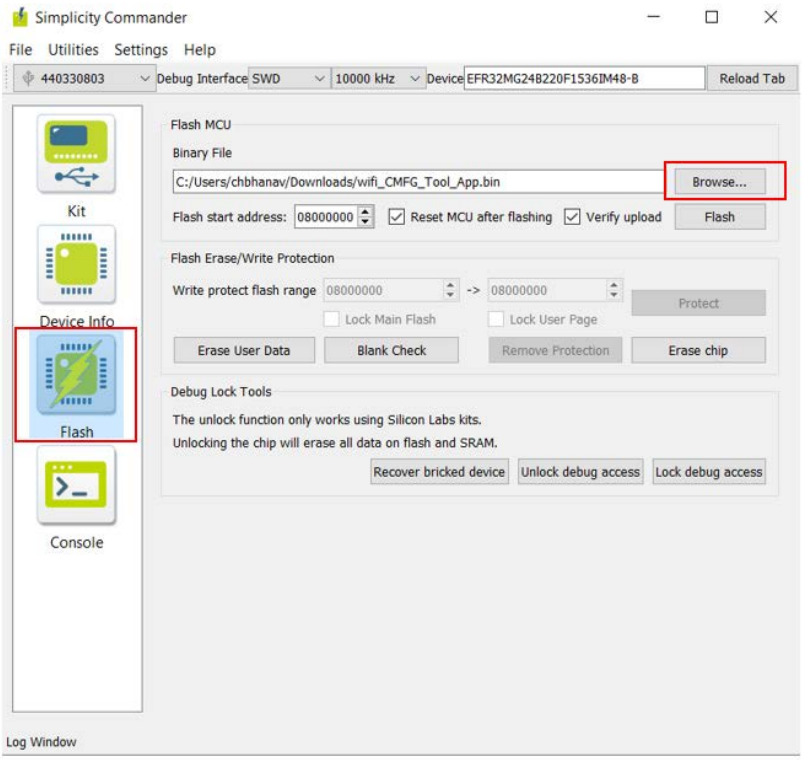
- If you are using Standalone Simplicity Commander, navigate to the following path within the Simplicity Commander installation directory and double click on "**commander.exe**": C:\..\..\SimplicityCommander-Windows\Commander_win32_x64_1v16p14b1705\Simplicity Commander.



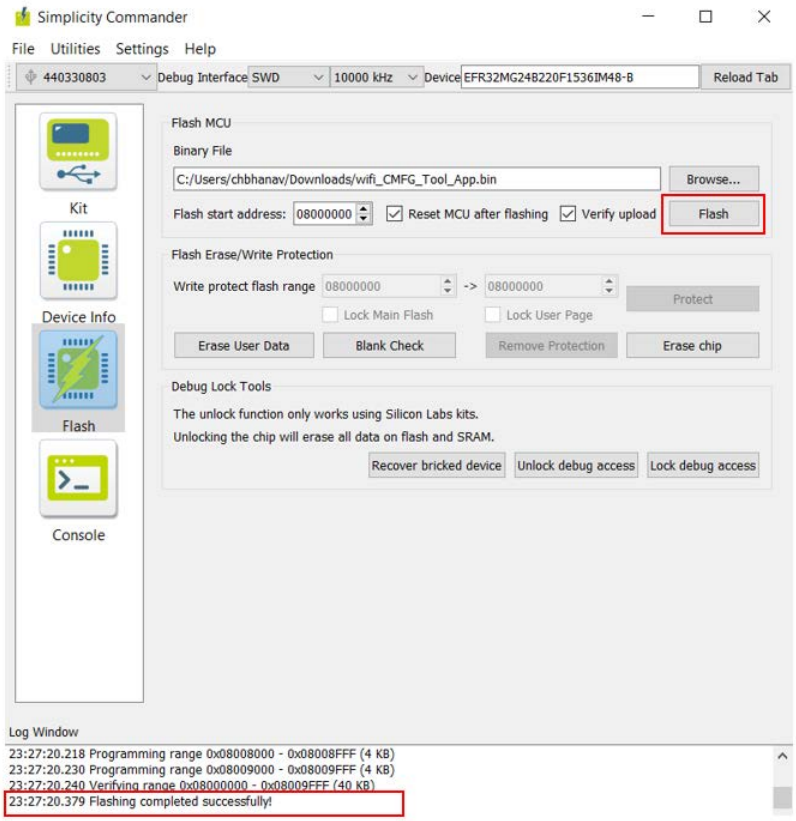
2. The Commander GUI window is displayed.
3. Go to "**Select Kit…**" and select the Serial number of the connected device.



4. Go to **Flash**, click **Browse…** and browse to the downloaded manufacturing application binary file (.bin): EFR32xG24 application or EFR32xG21 application depending on the EFR32 Radio board being used.

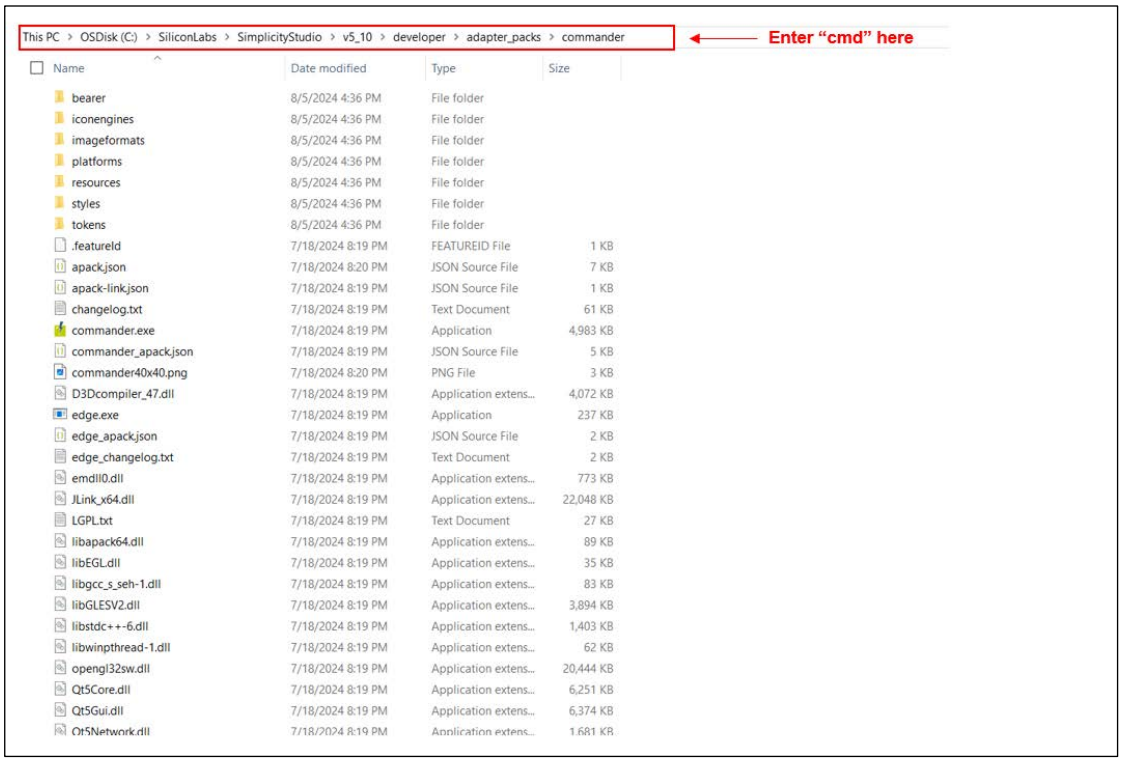


5. Click "**Flash**".

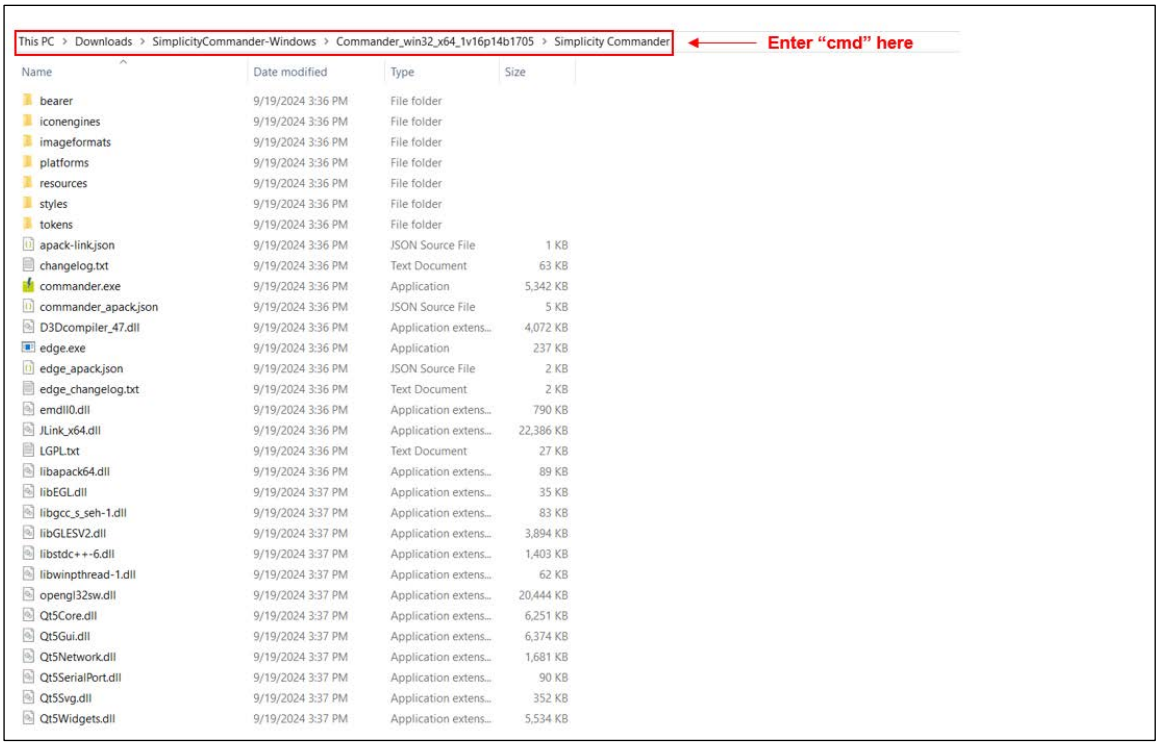


2.5 Launching the Commander CLI

- If you are using Simplicity Commander from the Simplicity Studio IDE installation, navigate to the following path within the Simplicity Studio installation directory and enter "cmd" to open the Commander CLI: C:\..\..\SimplicityStudio\v5\developer\adapter_packs\commander.



- If you are using Standalone Simplicity Commander, navigate to the following path within the Simplicity Commander installation directory and enter "cmd" to open the Commander CLI: C:\..\..\SimplicityCommander-Windows\Commander_win32_x64_1v16p14b1705\Simplicity Commander.



- For getting the SiWx917 NCP details, enter the following command:

  commander manufacturing info -d <full opn> --serialinterface

```
C:\Users\chbhanav\Desktop\SimplicityCommander-Windows\Commander_win32_x64_1v17p0b1771\Simplicity Commander>
commander manufacturing info -d SiWN917M100LGTBA --serialinterface
Using serial port 'COM68'.
Initializing target...
OPN                              : SiWN917M100LGTBA
WiFi MAC address                 : ECF64CA04684
BLE MAC address                  : 7CC6B6911BC1
Flash size                       : 4MB
Flash variant                    : 0
Flash type                       : 0
NWP firmware version             : 1711.2.12.3.3.0.3
Manufacturing SW version         : 2.4 (36)
Mode                             : Wiseconnect
Integrity protection active      : None
Safe upgrade from host           : Enabled
NWP roll-back prevention         : Disabled
NWP digital signature validation : Disabled
NWP firmware encryption          : Disabled
NWP secure boot                  : Disabled
DONE
```

2.6 Manufacturing Utility Commands, Options, and Arguments

The tables below provide a list of Manufacturing Utility commands, options, and arguments along with their descriptions, which are essential for the manufacturing of the SiWx917 NCPs.

Command	Description
manufacturing	Perform manufacturing operations
provision	Provision NWP (TA) MBR
tambr	Perform operation on NWP MBR
efuse	Perform operation on eFuse
efusecopy	Perform operation on eFusecopy
write	Compare the provided data with the existing data on the device and write the new changes into the respective region, whether it be flash, eFuse, or eFusecopy.
read	Read the contents in the respective region, whether it be flash, eFuse, or eFusecopy.
serial load	Load/flash the firmware
default	Use the default TA (NWP) MBR for your device, based on the provided device part number using the –device/-d option.

Option	Description
-d	Device
--mbr	NWP MBR
--out	Output into a file
--data	Data
--serialinterface	For configuring an NCP device, the Simplicity Commander communicates via a proprietary serial protocol to an interface device (host), which in turn communicates with the NCP device via serial peripheral interface (SPI) or secure digital input output (SDIO) commands. In this case, Simplicity Commander requires that you provide the --serialinterface option.
--skipinit	In case you are running multiple SiWx917 manufacturing commands in sequence and without resetting the target device in between, you may also provide the --skipinit option to skip initializing the target device's SPI/SDIO interface.
--skipload	For writing/erasing manufacturing data, or running the key provisioning processes (including its initialization), Simplicity Commander depends on a custom NWP firmware

	being loaded to the device. Loading this firmware can be skipped by providing the --skipload flag. This option must only be provided if you are completely certain that the NWP firmware is already loaded and running on your device, and the use of the --skipload flag is therefore generally discouraged.
--serialport	Serial Port of host MCU
--showprogress	Display a progress bar for the ongoing file transfer.
--keys	Provision device keys
--noprompt	Skip the confirmation prompt
--dryrun	Output the data to the terminal instead of writing it to the device
--taapp	Provide the TA (NWP) RPS firmware images

Argument	Description
<full opn>	Ordering Part Number (OPN) of SiWx917 NCP device
<filename>	The name of the file into which the contents will be read
[ ]	Optional parameter
<>	Required parameter

NOTE: For more information, refer to UG162: Simplicity Commander Reference guide.

3 Manufacturing Procedure without Device Security

The table below outlines the sequence of steps for manufacturing the SiWx917 NCP devices without security, along with the corresponding command syntax:

Step	Description	Command (Syntax)
1	Read or backup the default NWP MBR contents into .bin and .json format files. NOTE: This precaution allows you to revert to the original MBR in case you encounter issues while making changes to the MBR	commander manufacturing read tambr --out <read_mbr.bin> -d <full opn> --serialinterface commander manufacturing read tambr --out <read_mbr.json> -d <full opn> --serialinterface
2	Configure the boot configurations in NWP MBR or eFuse (optional).	• Refer to non-security fields in boot configurations section. • Create a sample .json file with the necessary updates to the boot configurations as required. Check the sample boot configurations here.
3	Provision the updated boot configuration in NWP MBR or eFuse (OTP) (Optional)	**MBR:** commander manufacturing provision --mbr <ta_mbr_SiWN917M100LGTBA.bin[a] |read_mbr.bin|'default'> --data <updatedbootconfigurationsfilename.json> -d <full opn> --serialinterface NOTE: In place of read_mbr.bin, you may also use the ta_mbr_SiWN917M100LGTBA.bin[a] for SiWN917M100LGTBA part or "default" option. **eFuse:** commander manufacturing write efuse --data <updatedbootconfigurationsfilename.json> -d <full opn> [--skipload] [--noprompt] [–dryrun] --serialinterface --skipinit
4	Flash NWP Connectivity Firmware Image (can also be done after step 6)	Refer to Load SiWx917 Connectivity Firmware section.
5	Set MAC Address in eFuse or eFusecopy (Optional)	Refer to Setting MAC address section.
6	RF (Frequency and Gain Offset) Calibration (Not required for SiWx917 module OPNs)	Refer to section Performing RF Calibration.

NOTE:

a. The **ta_mbr_SiWN917M100LGTBA.bin** file in the Simplicity Commander contains the default MBR for the part SiWN917M100LGTBA. By default, all the SiWN917M100LGTBA parts contain the MBR configuration present in this file.
   - The **ta_mbr_SiWN917M100LGTBA.bin** MBR file can be obtained in two ways:
     - **Simplicity Commander from Simplicity Studio -** Find the file in the path: C:\..\..\SimplicityCommander-Windows\Commander_win32_x64_1v16p14b1705\Simplicity Commander\resources\jlink\Si917
     - **Standalone Simplicity Commander** – Find the file in the path: C:\..\..\SimplicityStudio\v5\developer\adapter_packs\commander\resources\jlink\Si917
   - If the boot configurations are made in both the MBR and efuse (OTP), the bootloader will compare the configurations and consider the highest one.

b. Since eFuse is a One-Time Programmable (OTP) memory, it is crucial to ensure that all necessary fields are accurately configured before writing to it, as changes cannot be made to a field once enabled.

3.1 Sample Manufacturing Command Flow for SiWN917M100LGTBA OPN

This section provides the manufacturing command flow without device security for the SiWx917 NCP Ordering Part Number (OPN): SiWN917M100LGTBA.

1. Reset the host MCU.
2. Backup the default NWP MBR contents. Give the following command to read the MBR in .bin and .json formats.

.json format:

commander manufacturing read tambr --out default_nwp_mbr.json -d SiWN917M100LGTBA --serialinterface

```
C:\Users\chbhanav\Desktop\SimplicityCommander-Windows\Commander_win32_x64_1v17p0b1771\Simplicity Commander
>commander manufacturing read tambr --out default_nwp_mbr.json -d SiWN917M100LGTBA --serialinterface
Using serial port 'COM68'.
Initializing target...
Reading data from region: tambr
Reading 496 bytes from 0x04000000
Writing JSON...
Manufacturing data saved to file 'default_nwp_mbr.json'
DONE
```

.bin format:

commander manufacturing read tambr --out default_nwp_mbr.bin -d SiWN917M100LGTBA –serialinterface

```
C:\Users\chbhanav\Desktop\SimplicityCommander-Windows\Commander_win32_x64_1v17p0b1771\Simplicity Commander
>commander manufacturing read tambr --out default_nwp_mbr.bin -d SiWN917M100LGTBA --serialinterface
Using serial port 'COM68'.
Initializing target...
Reading data from region: tambr
Reading 496 bytes from 0x04000000
Manufacturing data saved to file 'default_nwp_mbr.bin'
DONE
```

NOTE: The MBR in .json format provides readable MBR fields, making it easier to understand and interpret the data. On the other hand, the .bin format is required when provisioning the updated MBR. This is why it is necessary to read the MBR in both formats.

3. Create a new file, say *updated_mbr_fields.json* file. Update boot configurations and save them in the file. For example:

MBR:

```
{
   "efuse_data": {
      "enable_autobaud_detection": 0
    }
}
```

eFuse (OTP):

```
{
   "ta_config": {
      "enable_autobaud_detection": 0

   }
}
```

4. Provision the updated boot settings in MBR or eFuse (OTP).

MBR:

commander manufacturing provision --mbr default --data updated_mbr_fields.json -d SiWN917M100LGTBA –serialinterface

```
C:\Users\chbhanav\Desktop\SimplicityCommander-Windows\Commander_win32_x64_1v17p0b1771\Simplicity Commander
>commander manufacturing provision --mbr default --data updated_mbr_fields.json -d SiWN917M100LGTBA --seri
alinterface
Using serial port 'COM68'.
Initializing target...
Loading RAM code (321776 bytes)...
Starting RAM code...
Using default MBR for your device: SiWN917M100LGTBA...
Updating MBR fields...
Reading JSON...
Programming ROM patch...
Data loaded successfully
Programming TA MBR...
Data loaded successfully
DONE
```

eFuse:

commander manufacturing write efuse --data updated_efuse_fields.json -d SiWN917M100LGTBA [--skipload] [--noprompt] [--dryrun] --serialinterface --skipinit

```
C:\Users\chbhanav\Desktop\SimplicityCommander-Windows\Commander_win32_x64_1v17p0b1771\Simplicity Commander
>commander manufacturing write efuse --data updated_efuse_fields.json -d SiWN917M100LGTBA --dryrun --seria
linterface
Using serial port 'COM68'.
Initializing target...
Reading JSON...
Binary data output:
{address:  0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F}
40012000: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
40012010: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
40012020: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
40012030: 00 00 00 00 00 00 00 00 00 00 41 39 11 1C 00 00
40012040: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
40012050: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
40012060: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
40012070: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

5. Optional – Check whether the updated configurations are reflected in MBR or eFuse. Read MBR or eFuse in .json format.

MBR:

commander manufacturing read tambr --out updated_nwp_mbr.json -d SiWN917M100LGTBA –serialinterface

```
C:\Users\chbhanav\Desktop\SimplicityCommander-Windows\Commander_win32_x64_1v17p0b1771\Simplicity Commander
>commander manufacturing read tambr --out updated_nwp_mbr.json -d SiWN917M100LGTBA --serialinterface
Using serial port 'COM68'.
Initializing target...
Loading RAM code (321776 bytes)...
Starting RAM code...
Reading data from region: tambr
Reading 496 bytes from 0x04000000
Writing JSON...
Manufacturing data saved to file 'updated_nwp_mbr.json'
DONE
```

eFuse:

commander manufacturing read efuse --out updated_nwp_efuse.json -d SiWN917M100LGTBA –serialinterface

```
C:\Users\chbhanav\Desktop\SimplicityCommander-Windows\Commander_win32_x64_1v17p0b1771\Simplicity Commander
>commander manufacturing read efuse --out updated_efuse.json -d SiWN917M100LGTBA --serialinterface
Using serial port 'COM68'.
Initializing target...
Reading data from region: efuse
Reading 1024 bytes from 0x40012000
Writing JSON...
Manufacturing data saved to file 'updated_efuse.json'
DONE
```

6. Flash the SiWx917 Connectivity Firmware Image as mentioned in Load SiWx917 Connectivity Firmware section.

7. Optional - Update the WLAN and BLE MAC addresses in the eFuse or eFusecopy following the steps mentioned in Setting MAC address section.

8. Perform RF Calibration following steps mentioned in section Performing RF Calibration.

4 Security Features

4.1 Secure Boot

To protect your software intellectual property (IP) and maintain your competitive edge in the market, you can configure or enable/disable a pre-flashed secure bootloader on the devices. Secure Boot ensures that only authenticated code can run on the device. If a device fails its security check, it is not allowed to run, and program control will typically stall in the validating module.

The Security Bootloader, which runs on the Network Wireless Processor (NWP), implements Secure Boot. Here is an overview of the Secure Boot process:

1. Upon reset, the Security Bootloader configures the module hardware based on the configurations present in the flash or eFuse.
2. The Security Bootloader authenticates the device configurations in the flash or efuse (OTP) using keys in OTP.
3. The Security Bootloader validates the integrity and authenticity of the NWP connectivity firmware in the flash and then invokes/loads the firmware.

To enable only Secure Boot or Secure Boot along with other security features, refer to the Security Levels section.

4.2 Secure Firmware Update

The secure firmware update feature of the bootloader ensures the authenticity and integrity of the connectivity firmware image. The bootloader updates the image only after successfully validating its authenticity and integrity. It also prevents downgrades to a lower version of firmware using the anti-rollback feature (ta_anti_roll_back) if enabled.

Additionally, the bootloader supports transparent migration to a wirelessly updated image and provides recovery mechanisms to protect against failures. The NWP Bootloader uses a proprietary format for its connectivity firmware images, known as RPS, with files having the extension .rps.

The connectivity firmware image supports the following security levels:

- **No Security**: CRC is calculated and verified against the default CRC of the image within the RPS header.
- **Secure Boot**: The Message Integrity Check (MIC) of the firmware is verified on every boot to check its integrity.
- **Message Integrity Check (MIC) + Signature**: The MIC is calculated for the firmware image and saved within the RPS header. The size of the image in the RPS header is incremented by the size of the signature. Finally, the signature of the entire RPS file (RPS Header + image) is calculated and appended to the end of the file.
- **MIC + Encryption + Signature**: The MIC is calculated for the firmware image and saved within the RPS header. The image is then encrypted and appended to the RPS header. The image can be encrypted using AES CTR or XTS mode. The size of the image in the RPS header is incremented by the size of the signature. Finally, the signature of the entire RPS file (RPS Header + image) is calculated and appended to the end of the file.

These security features are classified into different security levels. For guidance on selecting the appropriate security level, please refer to the Security Levels section.

.

Note: Once you are done backing up the files and Security Key Programming is done, then you only need to add security to the SiWx917 connectivity firmware image and flash the secure image.

4.3 Security Key Programming

To enable security features (encryption/MIC/signature) on your SiWx917 device, the device's Physically Unclonable Function (PUF) first needs to be initialized and an activation code must be generated on the device. This part of manufacturing includes generation of static keys and PUF generated keys and loading them into flash. While programming the static keys, the keys are saved in wrapped format in the flash. There are multiple keys that are generated inside and outside of device. Refer to Secure Key Management and Protection section for information on keys generated at commander and PUF Intrinsic keys.

The Security Key Programming sequence is as follows:

- Step 1: Generate Activation Code and Flash
- Step 2: Power cycle the device
- Step 3: Generate Security Keys

- Step 4: Load Keys & Program the MBR with Security Level

4.3.1 Step 1: Generate Activation Code and Flash

The activation code is generated by the Physical Unclonable Function (PUF) module embedded within the device. Once this code is generated, the firmware will securely write the activation code into the device's Flash memory.

Command syntax: *commander manufacturing init [--mbr <filename.bin|'default'> -d <full OPN>] --serialinterface*

Example: *commander manufacturing init --mbr default -d SiWN917M100LGTBA –serialinterface*

```
C:\Users\chbhanav\Desktop\SimplicityCommander-Windows\Commander_win32_x64_1v17p0b1771\Simplicity Commander>
commander manufacturing init --mbr default -d SiWN917M100LGTBA --serialinterface
Using serial port 'COM68'.
Initializing target...
Using default MBR for your device: SiWN917M100LGTBA...
Loading RAM code (321776 bytes)...
Starting RAM code...
Activation code generated successfully
DONE
```

Return Codes:

- **Success:** Activation Code gets generated
- **Failure:** Refer to Possible Error Codes and try again

4.3.2 Step 2: Power Cycle the Device

Users must power cycle the device after the Activation Code Generation and Flashing process. This mandatory step is required by the Hardware PUF block to set the security module into a ready state.

4.3.3 Step 3: Generate Security Keys from Commander

Generate the key configuration and save it as a JSON file for a SiWx917 device. This key configuration includes the OTA key, OTP AES key, OTP public key, OTP private key, TA (NWP) public key, and TA (NWP) private keys.

Command syntax: commander util genkeyconfig --outfile **<keys.json>** -d <'**si917'** | <fullopn>

Example: commander util genkeyconfig --outfile **commanderkeys.json** -d **si917**

Alternatively, users can generate keys themselves using their preferred tools and save them as a JSON file.

```
C:\Users\chbhanav\Desktop\SimplicityCommander-Windows\Commander_win32_x64_1v17p0b1771\Simplicity Commander>
commander util genkeyconfig --outfile commanderkeys.json -d si917
Generating symmetric key...
Generating symmetric key...
Generating symmetric key...
Generating ECC P256 key pair...
Generating ECC P256 key pair...
Generating ECC P256 key pair...
Generating ECC P256 key pair...
Key configuration written to commanderkeys.json
DONE
```

Return Codes:

- **Success:** Key configuration is saved as a JSON file.
- **Failure:** Refer to Possible Error Codes and try again.

4.3.4 Step 4: Load the Security Keys and Program the NWP MBR with Security

Load the Security Keys, configure the security level, PUF activation code address and key descriptor table, and program the NWP MBR. It is important to understand the security levels before proceeding with this step.

4.3.4.1 Security Levels

SiWx917 NCP offers various security features like Message Integrity Check (MIC), Signature, Encryption, etc., for NWP. There are three levels of security.

- **Security Level 1 (Low security level)**

  If the user wants to enable only secure boot, this level should be selected. This Security Level ensures that the bootloader performs MIC of the RPS during both firmware loading and firmware update.

  For Security Level 1, do the following configurations in the MBR fields:

  MBR:

```
{
   "efuse_data": {
      "ta_secure_boot_enable": 1
   }
}
```

  eFuse (OTP):

```
{
   "bootloader_config": {
      "ta_secure_boot_enable": 1
   }
}
```

- **Security Level 2 (Partial security level)**

  If the user wants to enable Secure boot along with Signature validation of the RPS (firmware) image, this level should be selected. This Security Level ensures the bootloader performs MIC and Signature validation of the RPS during both firmware loading and firmware update.

  For Security Level 2, do the following configurations in the MBR fields:

  MBR:

```
{
   "efuse_data": {
      "ta_secure_boot_enable": 1,
      "ta_digital_signature_validation": 1
   }
}
```

  eFuse (OTP):

```
{
   "bootloader_config": {
      "ta_secure_boot_enable": 1,
      "ta_digital_signature_validation": 1
   }
}
```

- **Security Level 3 (Full security level)**

  If the user wants to enable **Secure Boot, Signature Validation, and Encryption,** this level should be selected**.** This Security Level ensures the **MIC, Signature validation, and Encryption** of the RPS during Firmware loading and Firmware update and Inline Encryption while saving the firmware in the flash.

  For Security Level 3, do the following configurations in the MBR fields:

  MBR:

```
{
   "efuse_data": {
      "ta_secure_boot_enable": 1,
      "ta_digital_signature_validation": 1,
```

```
        "ta_encrypt_firmware": 1
      }
    }
```

eFuse (OTP):

```
    {
      "bootloader_config": {
        "ta_secure_boot_enable": 1,
        "ta_digital_signature_validation": 1
      },
      "ta_config": {
        "ta_encrypt_firmware": 1
      }
    }
```

With a clear understanding of the programmable security fields and security levels in MBR and eFuse, define the PUF activation code Address, Security level, and Key descriptor table address in a .json file. Below is the sample JSON file *enablesecurity.json*. Provision the keys contained in the JSON file (in step 3) to the SiWx917 device with the "–keys" option.

```
{
  "puf_activation_code_addr": 8192,
  "efuse_data": {
    "ta_secure_boot_enable": 1,
    "ta_digital_signature_validation": 1,
    "ta_encrypt_firmware": 1,
  },
  "key_desc_table_addr": 768
}
```

MBR:

Command syntax: commander manufacturing provision --mbr <ta_mbr_SiWN917M100LGTBA.bin | 'default'> --keys <keys.json> --data <securitylevelsfilename.json> -d <full opn> --serialinterface

Example: commander manufacturing provision --mbr default --keys commanderkeys.json --data enablesecurity.json -d SiWN917M100LGTBA --serialinterface

```
C:\Users\chbhanav\Desktop\SimplicityCommander-Windows\Commander_win32_x64_1v17p0b1771\Simplicity Commander
commander manufacturing provision --mbr default --keys commanderkeys.json --data enablesecurity.json -d Si
WN917M100LGTBA --serialinterface
Using serial port 'COM68'.
Initializing target...
Using default MBR for your device: SiWN917M100LGTBA...
Updating MBR fields...
Reading JSON...
Reading keys from provided JSON file...
Loading RAM code (321776 bytes)...
Starting RAM code...
Intrinsic keys generated successfully.
Programming TA OTA Key...
Key successfully stored
Programming M4 OTA Key...
Key successfully stored
Programming TA Public Key...
Key successfully stored
Programming M4 Public Key...
Key successfully stored
Programming attestation Key...
Key successfully stored
Programming ROM patch...
Data loaded successfully
Programming TA MBR...
Data loaded successfully
Programming key descriptor table...
Data loaded successfully
DONE
```

After the above command is processed, it implies that selected level of security is enabled for SiWx917 device.

4.4 Add Security to NWP Connectivity Firmware

When the Security is enabled in the device, you must load the secured NWP connectivity firmware. If you try to Flash the non-secured image, the system will throw an error, and the firmware image loading fails.

4.4.1 Secure NWP Connectivity Firmware

Give the following command to convert the already existing non-secure (no encryption, MIC, or signature) RPS NWP connectivity images into secure images by applying AES-ECB encryption, AES-CBC MIC integrity check, and ECDSA signatures.

Command Syntax:

For Security level 1:

commander rps convert <output_filename.rps> --taapp <original_unsecured_firmware.rps> --mic <commanderkeys.json>

For Security level 2:

commander rps convert <output_filename.rps> --taapp <original_unsecured_firmware.rps> --mic <commanderkeys.json> --sign <commanderkeys.json>

For Security level 3:

commander rps convert <output_filename.rps> --taapp <original_unsecured_firmware.rps> --mic <commanderkeys.json> --sign <commanderkeys.json> --encrypt <commanderkeys.json>

The above command creates a secure NWP connectivity firmware image.

```
C:\Users\chbhanav\Desktop\SimplicityCommander-Windows\Commander_win32_x64_1v17p0b1771\Simplicity Commander
>commander rps convert secure_firmware.rps --taapp SiWG917-B.2.13.3.0.0.11.rps --mic commanderkeys.json --
sign commanderkeys.json --encrypt commanderkeys.json
WARNING: No SHA type was provided, defaulting to SHA-256.
Parsing MIC key 'commanderkeys.json'...
Calculating MIC of image...
Parsing encryption key 'commanderkeys.json'...
Encrypting image...
Parsing signing key 'commanderkeys.json'...
Signing image...
Image SHA256: cdf5230eba9989b79c5f1bd7f121236bbc7ae28f5c6ac6db66825df01620a221
R = 100187E649D839A33791D4A61BE529411E03A74957AB16D6D58DABA6640656B8
S = C4596A78936BD6B240B75E8327A6868E510AAE7419D5CAB00091EA3A6686F30B
RPS file successfully created at 'secure_firmware.rps'.
DONE
```

4.5 Debug Lock

Configure the debug port securely before the ICs/modules leave the factory with the Debug Lock Feature (which can be unlocked with a token).

The SiWx917 NCP has a hardware debug solution. This provides high system visibility of the processor and memory through either a traditional JTAG port or a 2-pin Serial Wire Debug (SWD) port that is ideal for small package devices. The Secure Lock Feature locks the Debug Port. Any unauthorized access will be restricted.

This will be a one-time update of the MBR. This update should happen when the user has completed the development phase and entered the manufacturing phase for their products. Refer to the AN1428: SiWx917 Debug Lock for more information on enabling debug lock.

5 Boot Configurations

The following table shows the programmable boot configuration fields. The boot configurations can be done in either the eFuse (OTP) or NWP MBR.

S.no	Field	Description	Number of bits	Default Setting in MBR	Default setting in eFuse/OTP	Is a security feature?
1	enable_autobaud_detection	1: Auto baud rate detection for UART is enabled 0: Auto baud disabled • Default baud rate - 115200 bps	1	Disabled: 0	0	No
2	safe_upgrade_frm_host	1: upgrade the image from backup, instead of overwriting the primary location directly. 0: Overwrite the primary location directly.	1	Enabled: 1	0	Yes
3	disable_ta_jtag	0: Enable NWP JTAG interface. 1: Disable NWP JTAG Interface. Refer to debug lock section.	1	Enabled: 0	0	Yes
4	enable_glitch_mitigation	If glitch mitigation is enabled, the efuse bits related to JTAG interface are read again before enabling/disabling JTAG interface 1: Enable reading efuse bits related to JTAG interface again before enabling/disabling JTAG interface. 0: Disable reading efuse bits related to JTAG interface again before enabling/disabling JTAG interface.	1	Disabled: 0	0	Yes
4	ta_anti_roll_back	This feature prevents the NWP firmware from being downgraded to older versions. 1: Enable the Anti Rollback feature. 0: Disable the Anti Rollback feature.	1	Disabled: 0	0	Yes
5	ta_secure_boot_enable	1: Enable Secure Boot for NWP 0: Disable Secure Boot for NWP	1	Disabled: 0	0	Yes
6	ta_digital_signature_validation	1: Digital signature validation is enabled for NWP firmware 0: Digital signature validation is disabled for NWP firmware	1	Disabled: 0	0	Yes

7	ta_encrypt_firmware	00: Firmware is stored in an unencrypted form in the flash. 01: Firmware is stored in an encrypted form in the flash using CTR mode. 10: Firmware is stored in an encrypted form in the flash using XTS mode. 11: Reserved for future use.	2	Disabled : 00	0	Yes
8	mbr_mic_sign_enable	Enable MIC and Signature for the combined memory region, which includes the MBR, Boot Descriptor, and Key Descriptor Table. 00: CRC 01: MIC 10: reserved 11:Sign Applicable only when key is present in OTP	2	CRC check is enabled: 00	0	Yes
9	mic_protected_content_length	This field depicts how much space is MIC protected using OTP key and MIC stored in OTP See MIC Protected Content Length Map	4	Disabled:0	0	Yes
10	ta_otp_lock_r1	This is a programming lock for the NWP OTP, specifically for the R1 address region (0-127). When the OTP is locked, the user cannot modify the secure boot-related parameters present in the R1 address region. 1: NWP OTP programming is locked 0: NWP OTP programming is not locked	1	NA	0	Yes

6 Efuse Configurations

The following table shows the programmable eFuse (OTP) configuration fields. These configurations can only be done in eFuse (OTP).

S.no	Field	Description	Number of bits	Default Setting in OTP	Is a security feature?
1	ta_otp_lock_r1	This is a programming lock for the NWP OTP, specifically for the R1 address region (0-127). When the OTP is locked, the user cannot modify the secure boot-related parameters present in the R1 address region. 1: NWP OTP programming is locked 0: NWP OTP programming is not locked	1	0	Yes

2	"otp_lock_1": {
	"debug_port_opened": 0,
	"section_lock_128_255": 0
	}
		This is a programmable lock for the NWP OTP, specifically for the R2 address region (128-255). When the OTP is locked, users cannot modify the secure boot-related parameters present in the R2 address region.
		
		1: NWP OTP programming is locked
		0: NWP OTP programming is not locked
			1	0	Yes

3	"otp_lock_2": {
	"section_lock_256_767": 0
	}
		This is a programmable lock for the NWP OTP, specifically for the R3 address region (256-767). When the OTP is locked, users cannot modify the secure boot-related parameters present in the R3 address region.
		
		1: NWP OTP programming is locked
		0: NWP OTP programming is not locked
			1	0	Yes

4	"otp_lock_3": {
	"section_lock_768_1023": 0
	}
		This is a programmable lock for the NWP OTP, specifically for the R4 address region (768-1024). When the OTP is locked, users cannot modify the secure boot-related parameters present in the R4 address region.
		
		1: TA OTP programming is locked
		0: TA OTP programming is not locked
			1	0	Yes

6 Manufacturing Procedure with Device Security

The following table shows the sequence of steps for manufacturing the SiWx917 NCP devices with security:

Step	Description	Command (Syntax)
1	Backup the default NWP MBR contents. NOTE: This precaution allows you to revert to the original MBR in case you encounter issues while making changes to the MBR	commander manufacturing read tambr --out <filename.json> -d <full opn> --serialinterface commander manufacturing read tambr --out <filename.bin> -d <full opn> --serialinterface
2	Security Key Programming	Follow the steps mentioned in Security Key Programming section. After this step, the security level will be enabled in the MBR or Efuse as per security fields configurations made.
3	Configure the boot configurations in NWP MBR or eFuse (optional).	• Refer to all the boot configuration fields in boot configurations section. • Create a sample .json file with the necessary updates to the boot configurations as required. Check the sample boot configurations here.
4	Provision the updated boot configuration in NWP MBR or eFuse (OTP) (Optional)	**MBR:** commander manufacturing provision --mbr <ta_mbr_SiWN917M100LGTBA.bin[a] |read_mbr.bin|'default'> --data <updatedbootconfigurationsfilename.json> -d <full opn> --serialinterface NOTE: In place of read_mbr.bin, you may also use the ta_mbr_SiWN917M100LGTBA.bin[a] for SiWN917M100LGTBA part or "default" option. **eFuse:** commander manufacturing write efuse --data <updatedbootconfigurationsfilename.json> -d <full opn> [--skipload] [--noprompt] [–dryrun] --serialinterface --skipinit
5	Configure the eFuse-only configurations in eFuse (OTP) (optional).	• Refer to eFuse configurations section. • Create a sample .json file with the necessary updates to the boot configurations as required. Check the sample eFuse only configurations here.
6	Write the eFuse configurations to eFuse (OTP)	commander manufacturing write efuse --data <updatedbootconfigurationsfilename.json> -d <full opn> [--skipload] [--noprompt] [–dryrun] --serialinterface --skipinit
7	Secure NWP firmware	Follow the steps mentioned in Add Security to NWP firmware section.
8	Load the secure SiWx917 connectivity firmware	Refer to Load SiWx917 Connectivity Firmware section.
9	Set MAC Address (Optional)	Refer to Setting MAC address section.

10	RF (Frequency and Gain Offset) Calibration (Not required for)	Follow the steps mentioned in Performing RF Calibration section

6.1 Sample Manufacturing Commands Flow for SiWN917M100LGTBA

1. Reset the host MCU.
2. Backup the default NWP MBR contents. Give the following command to read the MBR in .bin and .json formats.

.json format:

commander manufacturing read tambr --out default_nwp_mbr.json -d SiWN917M100LGTBA --serialinterface

```
C:\Users\chbhanav\Desktop\SimplicityCommander-Windows\Commander_win32_x64_1v17p0b1771\Simplicity Commander
>commander manufacturing read tambr --out default_nwp_mbr.json -d SiWN917M100LGTBA --serialinterface
Using serial port 'COM68'.
Initializing target...
Reading data from region: tambr
Reading 496 bytes from 0x04000000
Writing JSON...
Manufacturing data saved to file 'default_nwp_mbr.json'
DONE
```

.bin format:

commander manufacturing read tambr --out default_nwp_mbr.bin -d SiWN917M100LGTBA –serialinterface

```
C:\Users\chbhanav\Desktop\SimplicityCommander-Windows\Commander_win32_x64_1v17p0b1771\Simplicity Commander
>commander manufacturing read tambr --out default_nwp_mbr.bin -d SiWN917M100LGTBA --serialinterface
Using serial port 'COM68'.
Initializing target...
Reading data from region: tambr
Reading 496 bytes from 0x04000000
Manufacturing data saved to file 'default_nwp_mbr.bin'
DONE
```

NOTE: The MBR in .json format provides readable MBR fields, making it easier to understand and interpret the data. On the other hand, the .bin format is required when provisioning the updated MBR. This is why it is necessary to read the MBR in both formats.

3. Follow the steps mentioned in Security Key Programming section to generate keys for SiWx917 device.

4. Create a new file, say *updated_mbr_fields.json* file. Update boot configurations and save them in the file. For example:

```
{
  "efuse_data": {
    "enable_autobaud_detection": 0,
    "ta_anti_roll_back": 0,
    "ta_digital_signature_validation": 0,
    "ta_encrypt_firmware": 0,
    "ta_otp_lock_r1": 0,

  }
}
```

5. Provision the updated boot settings in MBR.

commander manufacturing provision --mbr default --data updated_mbr_fields.json -d SiWN917M100LGTBA –serialinterface

6. Create a new file, say updated_efuse_fields.json file. Update eFuse-only (OTP) and boot configurations and save them in the file. For example:

```
{
    "bootloader_config": {
        "safe_upgrade_frm_host": 0,
        "ta_anti_roll_back": 0,
        "ta_digital_signature_validation": 0,
        "ta_secure_boot_enable": 0
    },

    "otp_lock_1": {
        "debug_port_opened": 0,
        "section_lock_128_255": 1
    },
    "otp_lock_2": {
        "section_lock_256_767": 1
    },
    "otp_lock_3": {
        "section_lock_768_1023": 1
    },

}
```

7. Provide the updated boot/efuse-only settings in eFuse (OTP).

   commander manufacturing write efuse --data updated_efuse_fields.json -d SiWN917M100LGTBA [--skipload] [--noprompt] [--dryrun] --serialinterface --skipinit

8. Optional – Check whether the updated configurations are reflected in MBR or eFuse. Read MBR or eFuse in .json format.

   MBR:

   commander manufacturing read tambr --out updated_nwp_mbr.json -d SiWN917M100LGTBA –serialinterface

```
C:\Users\chbhanav\Desktop\SimplicityCommander-Windows\Commander_win32_x64_1v17p0b1771\Simplicity Commander
>commander manufacturing read tambr --out updated_nwp_mbr.json -d SiWN917M100LGTBA --serialinterface
Using serial port 'COM68'.
Initializing target...
Loading RAM code (321776 bytes)...
Starting RAM code...
Reading data from region: tambr
Reading 496 bytes from 0x04000000
Writing JSON...
Manufacturing data saved to file 'updated_nwp_mbr.json'
DONE
```

   eFuse:

   commander manufacturing read efuse --out updated_nwp_efuse.json -d SiWN917M100LGTBA –serialinterface

```
C:\Users\chbhanav\Desktop\SimplicityCommander-Windows\Commander_win32_x64_1v17p0b1771\Simplicity Commander
>commander manufacturing read efuse --out updated_efuse.json -d SiWN917M100LGTBA --serialinterface
Using serial port 'COM68'.
Initializing target...
Reading data from region: efuse
Reading 1024 bytes from 0x40012000
Writing JSON...
Manufacturing data saved to file 'updated_efuse.json'
DONE
```

9. Secure the SiWx917 connectivity firmware following the steps mentioned in Add Security to NWP firmware section.

10. Load the SiWx917 Connectivity Firmware Image as mentioned in Load SiWx917 Connectivity Firmware section.

11. Optional - Update the WLAN and BLE MAC addresses in the eFuse or eFusecopy following the steps mentioned in Setting MAC address section.

12. Perform RF Calibration following steps mentioned in section Performing RF Calibration.

7 Load SiWx917 Connectivity Firmware

There are two ways to load the SiWx917 Connectivity firmware for SiWx917 NCP.

- Using Commander CLI
- Using Commander GUI

NOTE: Ensure to connect the USB port of the adapter board to your computer's USB port using a type C USB cable.

7.1 Firmware Loading via Commander CLI

The following command loads SiWx917 connectivity firmware image via serial port (UART).

commander serial load <filename.rps> --showprogress --serialport <COM Port No> -d si917

```
C:\Users\chbhanav\Desktop\SimplicityCommander-Windows\Commander_win32_x64_1v17p0b1771\Simplicity Commander
>commander serial load SiWG917-B.2.13.3.0.0.11.rps --showprogress --serialport COM66 -d si917
Setting target baud rate...
Target baud rate set to 921600 baud.
Initializing TA firmware upgrade...
Uploading file(s)...
Verifying file upload...
[==============================================================================] 100 %
TA firmware image was successfully uploaded.
DONE
```

7.2 Firmware Loading via Commander GUI

Refer to SiWx917 NCP connectivity firmware update.

8 Setting MAC Address

The default WLAN and BLE MAC addresses for a device are present in the following fields of eFusecopy:

- WLAN MAC Address - silabs_wlan_mac_address
- BLE MAC Address - ble_mac_address

Use the following fields in the eFusecopy/eFuse to customize the WLAN and BLE MAC Addresses:

- WLAN MAC Address - customer_wlan_mac_address
- BLE MAC Address - customer_ble_mac_address
- Additionally, update the following fields with value “90”.
    - customer_wlan_info_magic_byte
    - customer_ble_info_magic_byte

The Wi-Fi MAC address can be updated using the manufacturing utility using the below structure and available fields which are stored as a .json file. An example file is provided here:

```
{
    "customer_ble_info_magic_byte": 90,
    "customer_ble_mac_address": "123456789012"
    "customer_wlan_info_magic_byte": 90,
    "customer_wlan_mac_address": "112233445566"
}
```

The following command can be used to update the MAC address in the eFusecopy/eFuse.

eFusecopy:

*commander manufacturing write efusecopy --data MAC_Address_update_fields.json -d **SiWN917M100LGTBA***

eFuse:

*commander manufacturing write efusecopy --data MAC_Address_update_fields.json -d **SiWN917M100LGTBA***

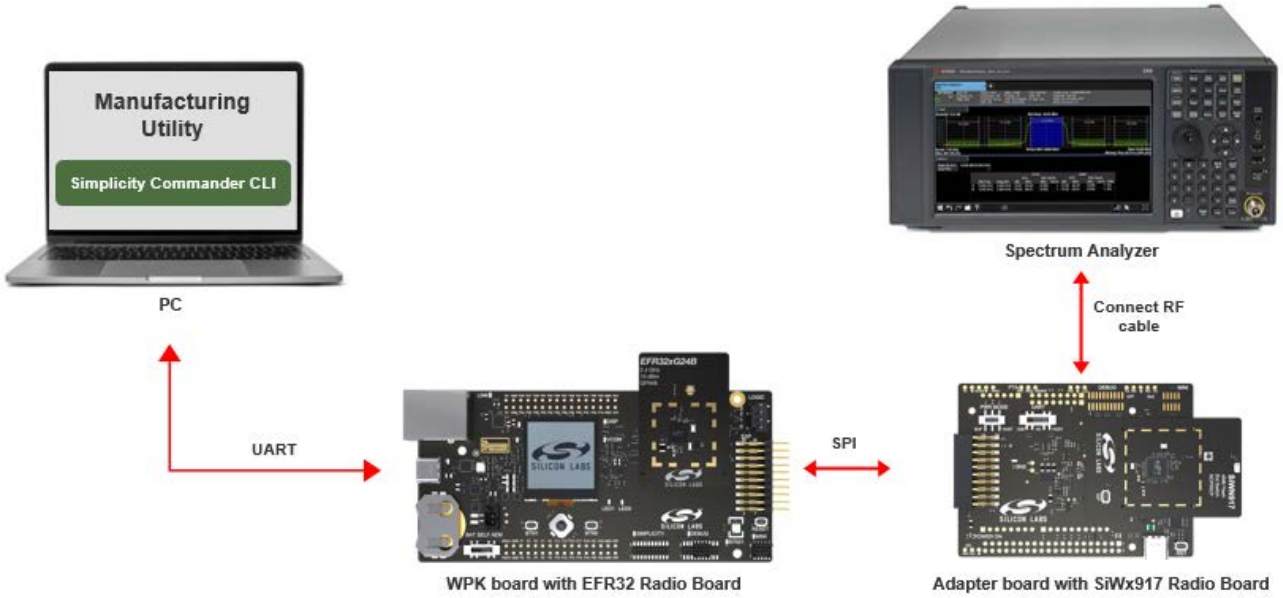
NOTE: Since eFuse is a One-Time Programmable (OTP) memory, it is crucial to ensure that all necessary fields are accurately configured before writing to it, as changes cannot be made once written.

9 Performing RF Calibration

The crystal calibration for SiWx917 NCP IC can either be done in Burst Mode or Continuous Wave (CW) mode. For more details, refer to SiWx917 QMS Crystal Calibration App Note.

9.1 Setup

Ensure the SiWx917's RF port is connected to Spectrum Analyzer as shown below:



9.2 Transmission in Burst Mode

This mode requires a test instrument that supports modulation analysis where frequency error can be reported by the instrument directly.

9.2.1 Steps for Frequency Offset Correction

1. Set up the radio and start transmission.
   commander manufacturing radio --power 16 --phy 6MBPS --channel 1 --start -d <full opn> --serialinterface
2. Check the frequency error on the instrument. Adjust the XO CTUNE values by providing the frequency error as input. Provide this frequency error as frequency offset. This value can be ranging from –255 to 255.

   commander manufacturing xocal --offset <Offset> -d <full opn> --serialinterface --skipinit --skipload

3. Observe the updated frequency error on the instrument. If the frequency error is not within +/-2 KHz, a new round of frequency error can be processed via step 2. This process may be repeated until the frequency error is brought within the tolerance limit (+/-2 KHz).
4. Store the XO CTUNE value in Efuse (OTP) or Flash. The radio transmission will stop.
   Efuse:
   commander manufacturing xocal --storeinefuse -d <full opn> --serialinterface --skipinit –skipload

   Flash:
   commander manufacturing xocal --store -d <full opn> --serialinterface --skipinit --skipload

9.2.2 Steps for Gain Offset Calibration

1. Set up the radio and start RF transmission in Burst mode.

   commander manufacturing radio --power 16 --phy 6MBPS --channel 1 --start -d <full opn> --serialinterface

1. Measure the output power with the instrument that has the capability of measuring burst power. For example, Keysight instrument with modulation analysis measurement setting capability shows the burst packet power.
2. Calculate the offset to meet the expected output power (16 dBm).
   *Offset = ceil ((output power measure (dBm) + cable loss -- 16) * 2)*
3. Store the gain values for channel 1
   commander manufacturing gain --ch1 <Offset> --skipload -d SiWG917M111MGTBA --serialinterface --skipinit
4. Repeat from step 1 for channel 6 and 11. In step 4 the replace –ch1 with --ch6 and --ch11 when measuring for channel 6 and 11 respectively.
5. Stop the radio.
   commander manufacturing radio --stop --skipload -d SiWG917M111MGTBA --serialinterface --skipinit

9.3 Transmission in Continuous Mode

9.3.1 Steps for Frequency Offset Correction

1. Set up the radio and start transmission.
   commander manufacturing radio --power 16 --phy CW --channel 1 --start --noburst -d <full opn> --serialinterface
2. Measure the frequency on the instrument using peak search (use the instrument which supports spectrum mode/modulation).
3. Adjust the CTUNE values by providing the frequency error as input. This is a frequency offset correction, with the value ranging from –255 to 255.
   Offset = measured frequency (in kHz) -- 2412000
   commander manufacturing xocal --offset <Offset> --skipload -d SiWG917M111MGTBA --serialinterface --skipinit
4. Check the instrument and verify that the channel frequency is within expectations. If not, repeat from step.
5. Store the CTUNE values. The radio transmission will stop.
   commander manufacturing xocal --store --skipload -d SiWG917M111MGTBA --serialinterface --skipinit

9.3.2 Steps for Gain Offset Calibration

1. Setup the radio and start transmission on channel 1
   commander manufacturing radio --power 16 --phy 6MBPS --channel 1 --noburst --start -d SiWG917M111MGTBA --serialinterface
2. Measure the output power (use the instrument which supports spectrum mode/modulation). Measure the integrated power in 20 MHz bandwidth.
3. Calculate the offset to meet the expected output power (16 dBm).
   *Offset = ceil ((output power measure (dBm) + cable loss -- 16) * 2)*
4. Store the gain values for channel 1
   commander manufacturing gain --ch1 <Offset> --skipload -d SiWG917M111MGTBA --serialinterface --skipinit
5. Repeat from step 1 for channel 6 and 11. In step 4 the replace –ch1 with --ch6 and --ch11 when measuring for channel 6 and 11 respectively.
6. Stop the radio.
   commander manufacturing radio --stop --skipload -d SiWG917M111MGTBA --serialinterface --skipinit

10 Possible Error Codes

Command	Error Code	Description	Cause of failure
MBR	0xa0ac	Verification failed from flash	There may be a wrong input from the user to select pinset/damaged MBR file
Activation code	0xa0b0	Activation code generated failed	security module might throw an error if PUF enroll fails, if the device is locked to use PUF
Intrinsic keys	0xa0b1	Intrinsic keys generation failed	This may occur if this command triggered before activation code command.
Static keys	0xa0b2	Static keys stored failed	This may occur if there are no intrinsic keys.
Update Firmware	108	Failed to load firmware	This may occur if the header part of rps file is not proper.

11 Appendix

11.1 Reading Manufacturing Parameters

- The following command is used to read the manufacturing parameters from a file.

Command*: commander manufacturing read <tambr|efuse|efusecopy> --out <filename.bin|filename.json> -d <full opn> serialinterface –skipinit*

Read	Region	Region to read the data from
Read <region>	tambr	Read the NWP flash MBR data available for memory region
	--out <filename>	Filename to store the read data in. "*.bin"
	efuse	Read the OTP data
	efusecopy	Read the efusecopy data in flash

11.2 Programming efuse Command

In the case of eFuse/OTP data, Commander will first check that the requested update is possible (since bits can only be set to 1 and never cleared). Then it will ask for confirmation from the user. It is possible to skip the confirmation via the --noprompt option although note that this is a one-time operation.

It is possible to check the results of the operation before physically going ahead with the one-time programming by using the --dryrun option.

This command is used to write efuse data into flash.

Syntax:

commander manufacturing write efuse --data file.json -d <full opn > [--skipload] [--noprompt] [–dryrun] --serialinterface --skipinit

11.3 Sample Boot Configurations

Make the boot configurations as required. For example:

Boot configurations in MBR:

```
{
   "efuse_data": {
      “safe_upgrade_frm_host: 1,
      "ta_secure_boot_enable": 1
    }
}
```

Boot configurations in Efuse:

```
"bootloader_config": {
      "ta_secure_boot_enable": 1,
      "safe_upgrade_frm_host": 1,
   },
```

Save the above as JSON file.

11.4 Sample eFuse Only Configurations

```
{
  "bootloader_config": {
    "safe_upgrade_frm_host": 0,
    "ta_anti_roll_back": 0,
    "ta_digital_signature_validation": 0,
    "ta_secure_boot_enable": 0
  },
  "otp_lock_1": {
    "debug_port_opened": 0,
    "section_lock_128_255": 0
  },
  "otp_lock_2": {
    "section_lock_256_767": 0
  },
  "otp_lock_3": {
    "section_lock_768_1023": 0
  },
  "ta_config": {
    "enable_autobaud_detection": 0,
    "ta_encrypt_firmware": 0
  },
  "wireless_chip_modes": {
    "disable_ta_jtag": 0,
    "ta_otp_lock_r1": 0,
    "ta_otp_lock_r2": 0,
    "ta_otp_lock_r3": 0,
    "ta_otp_lock_r4": 0
  }
}
```

11.5 MIC Protected Content Length Map

Bit Value	Length in KB
0	1
1	2
2	4
3	6

4	8
5	10
6	12
7	16
8	20
9	24
10	28
11	32
12	40
13	48
14	56
15	64

11.6 OTP Write Lock Process

SiWx917 has OTP (eFuse) memory of 1024 bytes. This memory is divided into 4 sections with control to enable write locks for each of these sections. This section explains the details of how the write locks for OTP are implemented.

OTP Lock Configuration Field	**Region**	**Lock Memory Range**
ta_otp_lock_r1	R1	0-127

"otp_lock_1": {
"debug_port_opened": 0,
"section_lock_128_255": 0
}
	R2	128-255

"otp_lock_2": {
"section_lock_256_767": 0
}
	R3	256-767

"otp_lock_3": {
"section_lock_768_1023": 0
}
	R4	768-1024

11.7 Security Key Management and Protection

Inject custom public and private keys, along with other secret keys, onto the chips during manufacturing to safeguard your products from the very beginning of their lifecycle. The Bootloader uses public and private key-based digital signatures to recognize authentic software. The Security Bootloader also provides the capability for inline execution (XIP) of encrypted firmware directly from Flash.

If this feature is enabled, the Physically Unclonable Function (PUF) engine will be activated. PUF Keys are generated internally by the PUF and stored in flash. These keys are used to encrypt and decrypt images, ensuring that only authorized firmware can run on the device. The PUF Intrinsic Keys are unique to each device and are generated randomly, making them highly secure.

Key	Description
Master Key	This key is used for MIC calculation and wrapping/unwrapping of NWP keys. Keys encrypted using Master Key is done using AES ECB Mode. MIC is calculated using AES CBC mode.
NWP FW key 1, NWP FW key 2	These two keys are used for encryption and inline decryption of NWP firmware in CTR or XTS mode based on configuration from security configurations

Keys generated using the commander are as follows.

Key	Description
OTP Symmetric Key	Used for flash content's (MIC) verification - AES CBC is used for calculation of the MIC value.
OTP Public Key	Used for flash content's (digital signature) verification - ECDSA-P256. This key is used to verify the signature of MBR.
NWP Public Key	Used for NWP firmware signature verification - ECDSA-P256
NWP OTA Key	Used for NWP firmware image encryption and MIC calculation of NWP Firmware. Mode of operation is AES CBC.

For information on how to generate the keys at commander, refer to Generate Keys from Commander. For PUF activation, refer to Activation Code Generation, and to generate intrinsic keys and load the keys (both the intrinsic keys and keys generated at commander), refer to Intrinsic Key Generation and Loading Keys.

The table below shows the Key type, Key length, and Storage type.

S.No	Key	Key Length	Algorithm	Operation	Modes	Storage	
						OTP	Flash
1	OTP Symmetric Key	16 bytes	AES	MIC Calculation	AES CBC	✔	
2	OTP Public Key	91 bytes - NIST Curve P-256	ECDSA	Sign verification	-	✔	
3	NWP Public Key	96 bytes* - NIST Curve P-256	ECDSA	Sign verification	-		✔
4	NWP OTA Key	32 bytes	AES	Encryption, MIC Calculation	Encryption: AESCBC MIC Calculation: AES CBC		✔
8	Master Key	52 bytes key code used to initialize key in Key Holder	AES	Encryption, MIC Calculation	Encryption: AES ECB MIC Calculation: AES CBC		✔

10	NWP FW Key (2 keys)	52 bytes key code used to initialize key in Key Holder	AES	Encryption	AES CTR, AES XTS		☑

***Note: The NWP Public key actual key size is 91 bytes, remaining 5 padding bytes added to make key size multiple of 16 for AES encryption.**

12 Revision History

Revision 1.1

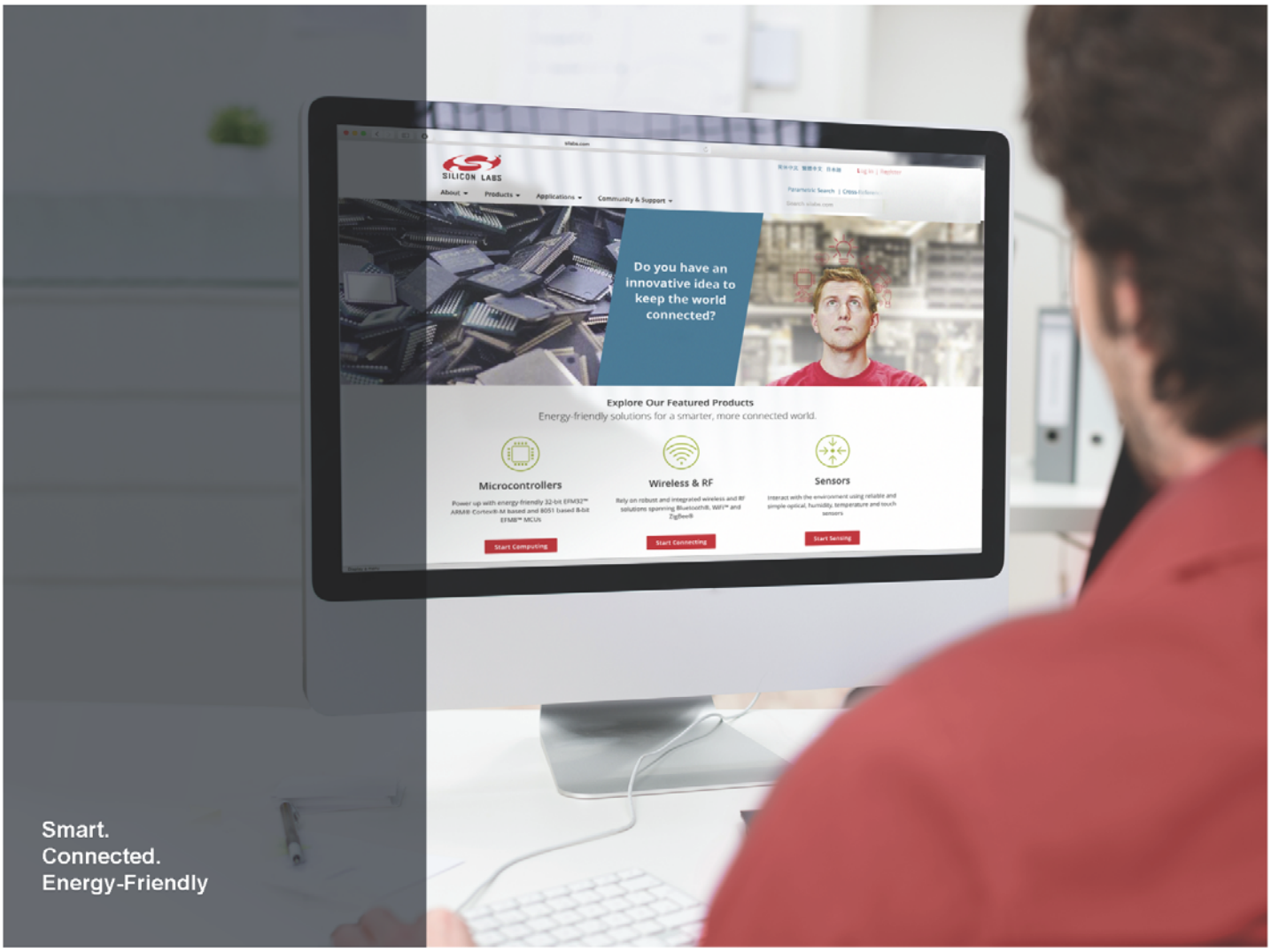
January 2025

- Updated the manufacturing steps to include both scenarios: with and without device security.
- Added a new section detailing Boot configurations and eFuse-only configurations.
- Re-organized the document sections to enhance the overall user experience.

Revision 1.0

February' 2024

- Initial Revision



Products
www.silabs.com/products

Quality
www.silabs.com/quality

Support and Community
community.silabs.com

Disclaimer
Silicon Laboratories 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 Laboratories 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 Laboratories reserves the right to make changes without further notice and limitation to product information, specifications, and descriptions herein, and does not give warranties as to the accuracy or completeness of the included information. Silicon Laboratories shall have no liability for the consequences of use of the information supplied herein. This document does not imply or express copyright licenses granted hereunder to design or fabricate any integrated circuits. The products must not be used within any Life Support System without the specific written consent of Silicon Laboratories. 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 Laboratories products are generally not intended for military applications. Silicon Laboratories 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.

Trademark Information
Silicon Laboratories Inc., Silicon Laboratories, Silicon Labs, SiLabs and the Silicon Labs logo, CMEMS®, EFM, EFM32, EFR, Energy Micro, Energy Micro logo and combinations thereof, "the world's most energy friendly microcontrollers", Ember®, EZLink®, EZMac®, EZRadio®, EZRadioPRO®, DSPLL®, ISOmodem ®, Precision32®, ProSLIC®, SiPHY®, USBXpress® and others are trademarks or registered trademarks of Silicon Laboratories Inc. ARM, CORTEX, Cortex-M3 and THUMB are trademarks or registered trademarks of ARM Holdings. Keil is a registered trademark of ARM Limited. All other products or brand names mentioned herein are trademarks of their respective holders.

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

http://www.silabs.com