UG574: SiWx917 SoC Manufacturing Utility User Guide

Version 1.1

January 2024

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

SiWG917 (SiWx917 SoC) includes two processors: Silicon Labs' ThreadArch® (TA) and an ARM® Cortex® M4 Processor. All the networking and wireless stacks run on independent threads of the ThreadArch®.

In addition, in adherence to the Trusted Execution Environment architecture, the TA subsystem also acts as the secure processing domain and takes care of secure boot, secure firmware update, and provides access to security accelerators and secure peripherals through pre-defined APIs. The Cortex-M4 is dedicated to peripheral and application-related processing.

1.1 Manufacturing Utility

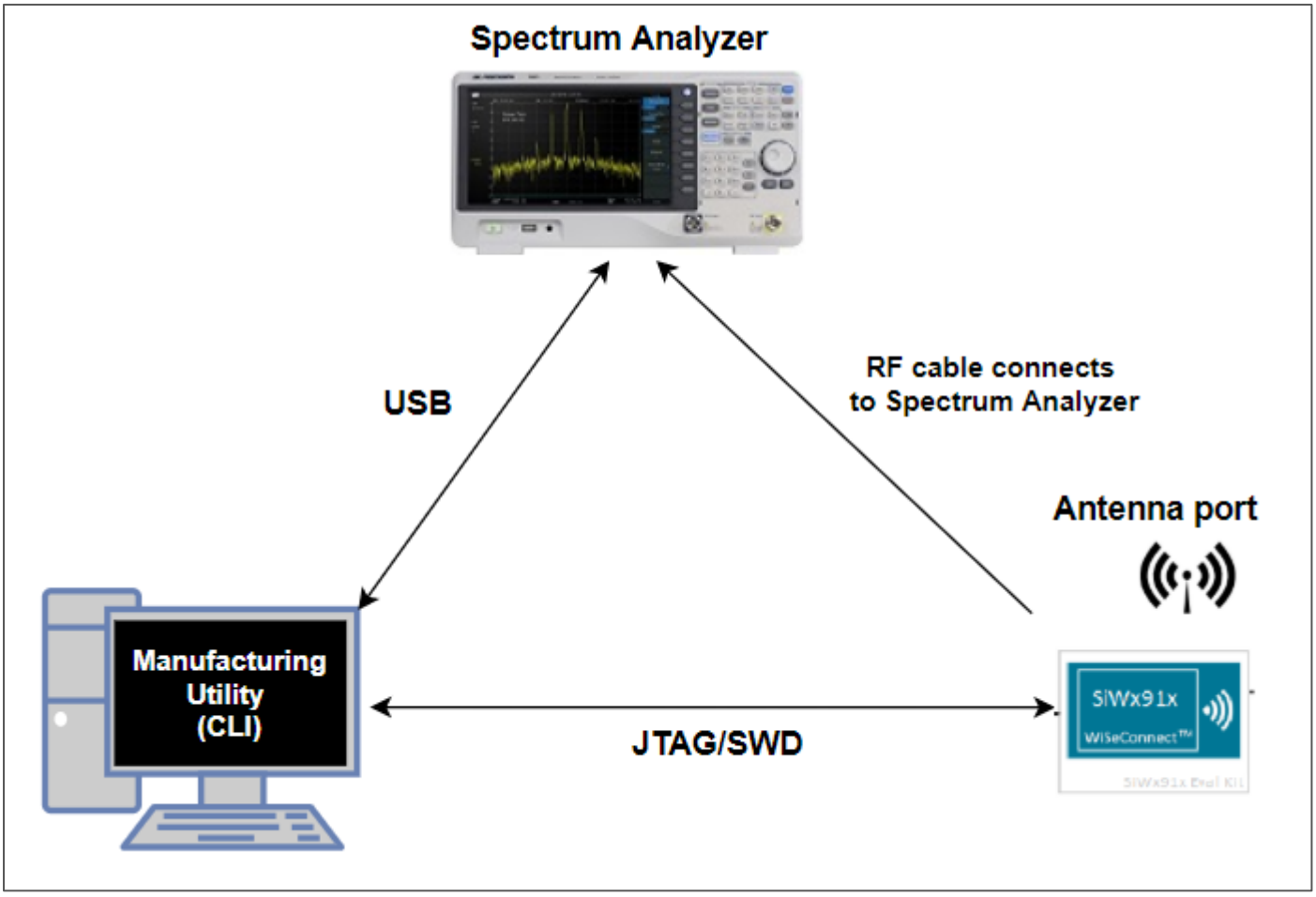
Manufacturing Utility or Simplicity Commander CLI (Command Line Interface) is used to update the required ***Master Boot Record (MBR)***, ***Enable the Security*** and ***Calibration*** in SiWG917.

This utility is enabled to work with Simplicity Commander through its Command Line Interface (CLI). Simplicity Commander CLI supports multiple commands for the user to program the production information into the device.

Master Boot Record (MBR): Stored in flash and contains information like clock frequencies, offsets of structures like eFuse copy, SPI configurations, External Flash details, etc. There are separate MBRs for TA and M4 at the beginning of their respective flash regions. Any SiWG917 IC that is shipped out of the factory will have a default MBR. Using the OPN of a particular device, the user can update it.

Enable the Security: The security fields in MBR, PUF Activation code, Key descriptors, and the Keys are to be programmed in the device while enabling the security features in the device.

Calibration: The CTUNE and gain offset adjustments can be done in burst and continuous mode in the SiWG917 using the manufacturing utility.



1.1.1 Simplicity Commander

Simplicity Commander is a single, all-purpose tool to be used in a production environment. It can be invoked using a simple Command Line Interface (CLI) that is also scriptable.

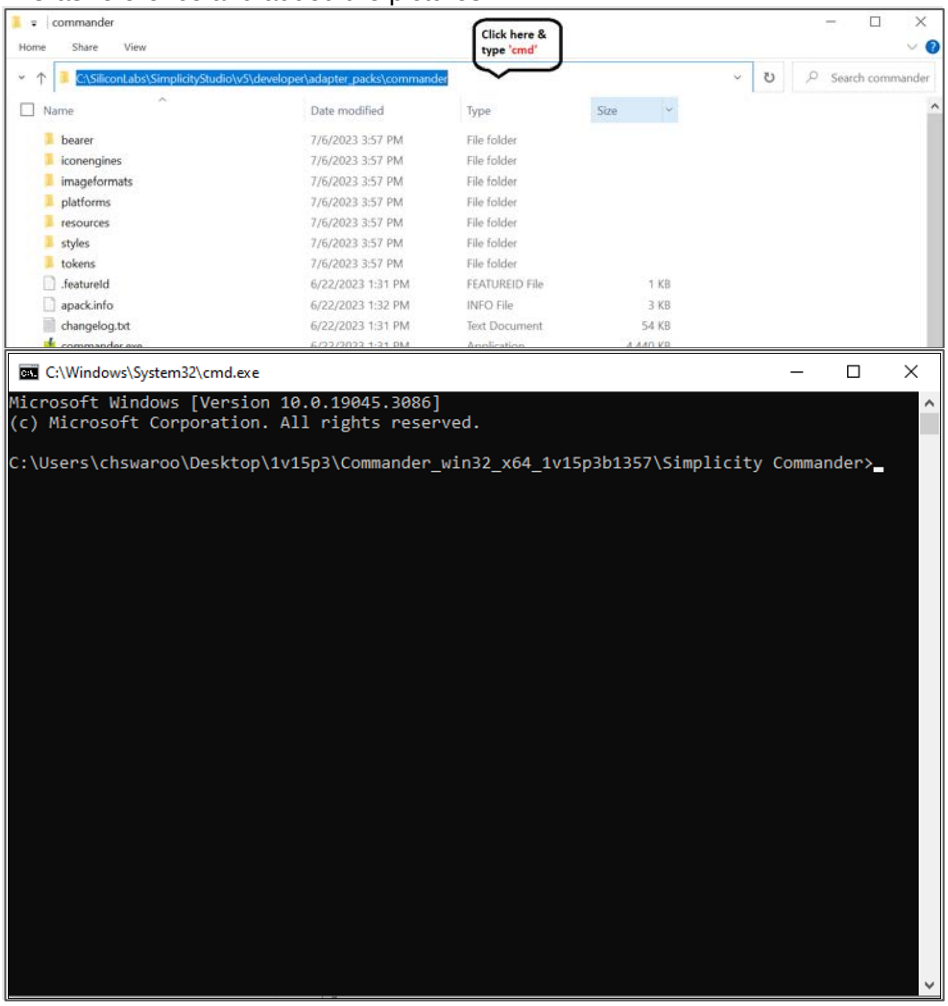
Simplicity Commander enables users to complete these essential tasks:

- Flash the application
- Configure the application
- Create binaries for production

1.1.2 Simplicity Commander CLI

To execute Simplicity Commander commands, start a Windows command window, and change to the Simplicity Commander directory (or) You can also open the commander as described below.

- Navigate to C:\SiliconLabs\SimplicityStudio\v5\developer\adapter_packs\commander folder.
- Type ‘cmd’ in the address bar and press Enter key. The Simplicity Commander CLI will be seen. We have taken Windows PC as reference and added the pictures.



2 Manufacturing Procedure – Non-Secure Device

MBR loading can be done based on the OPN number: a user can program the MBR by specifying the default option with OPN number. This will load the factory default MBR. If the users receive a flash-less device or OPNs which have external flash, MBR loading should not be done based on the OPN number.

MBR is to be loaded from a file given as input in **.bin** format.

Here is the sequence to follow for programming the devices in which security is disabled.

- Write TA MBR
- Write M4 MBR
- Write the Calibration data to the M4 flash.

2.1 Write TA MBR

Below is the Syntax for writing the TA MBR.

Syntax:

commander manufacturing provision --mbr <filename.bin|default> -d <full opn>

Note:

default → To load MBR based on the OPN number given as input,

filename.bin → To load MBR from a bin file. In this case OPN number input not required.

Example:

1. Load the default MBR.
   Command: commander manufacturing provision --mbr **default** -d **SiWG917M111MGTBA**
2. Load a specific MBR.
   Command: commander manufacturing provision --mbr **ta_mbr_SiWG917M111MGTBA.bin** -d **SiWG917M111MGTBA**

2.2 Write M4 MBR

Below is the Syntax for writing the M4 MBR.

Syntax:

commander manufacturing write <m4mbrcf> --data <filename.bin|filename.json> -d <full opn> [--skipload] [--pinset n]

Note:

filename.json → To load MBR from a json file.

filename.bin → To load MBR from a bin file.

m4mbrcf → Common Flash

Example:

commander manufacturing write m4mbrcf --data **m4_mbr_SiWG917M111MGTBA.bin** -d **SiWG917M111MGTBA** --skipload --pinset 0

3 Manufacturing Procedure – Secure Device

To do the production programming of security enabled parts, we need to update the security key programming in the SiWG917 and then program the MBR by selecting security levels.

The following steps are to be followed for securing your SiWG917 device.

1. Back-up the original TA, M4, and eFuse data/contents (Optional)
2. Security Key Programming
3. Add Security to TA and M4 firmware image files.
4. Flash the Secured images into flash.

3.1 Back-up the Original TA, M4, and eFuse Data/Contents (Optional)

Recommendation: Back-up the original TA, M4, and the eFuse data/contents before doing any updates. Backing up these is optional.

Field	Syntax	Example
TA MBR back-up	**commander manufacturing read <tambr> --out <filename.bin>**	commander manufacturing read tambr --out tambr.bin
M4 MBR back-up	**commander manufacturing read <m4mbrcf|m4mbrdf> --out <filename.bin>**	commander manufacturing read m4mbrcf --out m4mbr.bin
eFuse data copy (back-up) - optional	**commander manufacturing read <efusecopy> --out <filename.bin>**	commander manufacturing read efusecopy --out efusecopy.bin

Note: If no MBR is present in the device, you will see an error. Refer to Possible Error Codes section for more information on error-codes.

3.2 Security Key Programming

This part of manufacturing includes generation of static keys and Physically Unclonable Function (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 sequence to do security key programming is as follows,

1. Activation code generation for PUF Block.
2. Power cycle the device to boot again after step 1.
3. Generate keys from commander (generates keys at commander).
4. Intrinsic key generation (PUF generated keys) and loading keys (loads both device dependent keys and generated keys at commander)

3.2.1 Activation Code Generation for PUF Block

This command is used to generate activation code from the PUF module present in SiWG917 device. Once generated, the firmware will burn this activation code into flash.

Syntax:

commander manufacturing init --mbr <filename.bin|default> -d <full opn>

Example

commander manufacturing init --mbr **ta_mbr_SiWG917M111MGTBA.bin** -d **si917**

Return Code

- Success – 0xa05a
- Failure – Try again

3.2.2 Power Cycle the Device

The users need to power cycle the device after the Activation Code Generation and flash completion. This is a mandatory step.

3.2.3 Generate Keys from Commander

These keys are generated at commander. The key generation will produce a json file containing the OTA keys, Public keys, and Attestation key. For each of the public or private keys, the respective pair counterparts will be produced.

Syntax:

commander util genkeyconfig --outfile <keys.json> -d <full opn>

Example

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

Return Code

- Success – 0xa05a
- Failure – Try again

3.2.4 Intrinsic Key Generation, Update MBR and Loading Keys

This step performs the actual provisioning of the intrinsic keys (PUF generated keys), static keys, and final TA MBR and key descriptor table.

Along with keys loading and burning, the MBR will be updated with filename.bin or the "default" option (to select based on the OPN).

Syntax:

commander manufacturing provision --mbr <filename.bin|default> --keys <keys.json> --data <updated-mbr-fields.json> -d <full opn>

Note:

- **keys.json file contains the static keys i.e. TA OTA key, M4 OTA key, TA public key, M4 public key and attestation key.**

Example

commander manufacturing provision --mbr **ta_mbr_SiWG917M111MGTBA.bin** --keys **commanderkeys.json** --data **updatedmbrfields.json** -d **si917**

Note:

- The updated-mbr-fields.json file provided is to be updated with different security levels discussed in the MBR Programming with Security Section. Once we give this command, it updates default OPN-based MBR with .json file params and loads to the flash after keys loading.
- The default MBF I .bin format is present in the path: **$:\...\SimplicityStudio\v5\developer\adapter_packs\commander\resources\jlink**

3.3 Add Security to TA and M4 Firmware Image Files

When the security is enabled in the device, it expects the TA and the M4 files also to have the same level of security to be enabled. If the user tries to flash the non-secured image, the system will give an error.

3.3.1 Extract Keys

We need some of the keys generated in Section Generate Keys from Commander to secure the TA and M4 firmware images. We will have to extract these keys to a different folder.

Syntax:

commander util extractkeys <keys.json> --dir <foldername>

Example

commander util extractkeys **commanderkeys.json** --dir **extracted-keys**

- This command extracts all the Keys to individual files in a specified folder that is already created (Make sure to create the folder whose name is given in this command before giving this command). In the above example, commanderkeys.json is the key file that got generated from the Generate Keys from Commander section and extracted-keys is the folder name created.

3.3.2 Secure TA and M4 Firmware Images

TA Image

Syntax:

commander rps convert <filename.rps> --taapp <original non-encrypted TA rps> --mic "<OTA_KEY>" --encrypt "<OTA_KEY>" --sign <ta_private_key.pem>

Example

commander rps convert **ta_fw.rps** --taapp **SiWG917-B.2.9.0.0.15.rps** --mic "**01a46070c0a6def656beef50555f6c5a771b6775003bf16a7cd5aa6c8ef9cc0a**" --encrypt "**01a46070c0a6def656beef50555f6c5a771b6775003bf16a7cd5aa6c8ef9cc0a**" --sign **ta_private_key.pem**

M4 Image

Syntax:

commander rps convert <filename.rps> --app <original non-encrypted M4 rps> --mic "<M4_OTA_KEY>" --encrypt "<M4_OTA_KEY>" --sign <m4_private_key.pem>

Example

commander rps convert **Secured_sl_si91x_calendar.rps** --app **sl_si91x_calendar.rps** --mic "**ab8164a1f7053d96c5c47ecebc084afa5f3e9712e4ac3a7a589182305e8c5b78**" --encrypt "**ab8164a1f7053d96c5c47ecebc084afa5f3e9712e4ac3a7a589182305e8c5b78**" --sign **m4_private_key.pem**

3.4 Flash the Secured Images into Flash

The user can flash the secured TA and M4 firmware images using the Simplicity Commander CLI. You need to load the .rps file format for both TA and M4 firmware images.

Flash TA Image

Syntax:

commander rps load <filename.rps> -d <full opn>

Example

commander rps load **ta_fw.rps** -d **si917**

Flash M4 Image

Syntax:

commander rps load <filename.rps> -d <full opn>

Example

commander rps load **Secured_sl_si91x_calendar.rps** -d **si917**

For reference, a successful firmware upgrade is shown in the following image.

```
C:\917CCP\Commander_win32_x64_1v15p3b2\Simplicity Commander>commander rps load Secured_sl_si91x_calendar.rps -d si917
WARNING: No serial number or IP address given, cannot lock access to adapter.
Uploading flashloader...
Waiting for flashloader to become ready
Writing data...
Waiting for bootloader to perform upgrade...
Resetting
DONE
```

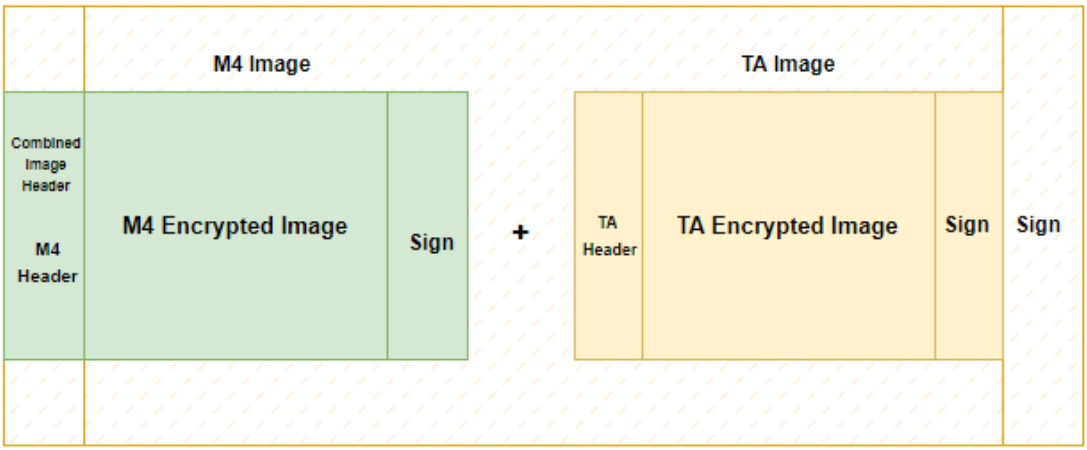
Note: If the security is enabled in your device, you will have to load the secured image; otherwise, the loading fails.

3.5 Combine Image

The combined image is a single image which is obtained by combining the TA and M4 images. The process of creating the combined image involves encrypting both the TA and M4 images, and then adding a RPS header and signature. In case of non-secure firmware, we will not be adding the signature.

- Combined Image RPS header format will be the same as the M4 RPS header format with few reserved bytes changed.
- Signature for complete combined image will be calculated and appended at the end of image.
- MIC computation and signature can be used to maintain integrity and confidentiality of the combined image.

Encryption of combined image is discarded as it will add overhead for firmware to decrypt and store into flash location. The TA and M4 images are individually encrypted.



To secure the combined image and flash it into the SiWG917 device, first complete the Security Key Programming and then follow the below steps.

Note: In case of non-secure device, you must still do the below steps, remove the MIC, encryption and signature parameters in the command.

Secure TA Image with combined flag

Syntax:

commander rps convert <filename.rps> --taapp <original non-encrypted TA rps> --mic "<OTA_KEY>" --encrypt "<OTA_KEY>" --sign <ta_private_key.pem> --combindimage

Example

commander rps convert **ta_fw.rps** --taapp **SiWG917-B.2.9.0.0.15.rps** --mic "**f6feea21e82e2f8fb69313f0b218ecc52110fc9756a3e8dd037f8e2e0dcb588d**" --encrypt "**f6feea21e82e2f8fb69313f0b218ecc52110fc9756a3e8dd037f8e2e0dcb588d**" --sign **ta_private_key.pem** –combinedimage

Secure M4 firmware Image with Combined Flag

Syntax:

commander rps convert <filename.rps> --app <original non-encrypted M4 rps> --mic "<M4_OTA_KEY>" --encrypt "<M4_OTA_KEY>" --sign <m4_private_key.pem> --combindimage

Example

commander rps convert **Secured_sl_si91x_calendar.rps** --app **sl_si91x_calendar.rps** --mic "**ab8164a1f7053d96c5c47ecebc084afa5f3e9712e4ac3a7a589182305e8c5b78**" --encrypt "**ab8164a1f7053d96c5c47ecebc084afa5f3e9712e4ac3a7a589182305e8c5b78**" --sign **m4_private_key.pem** –combinedimage

Combine TA and M4 Images

Syntax:

commander rps convert "combined_image.rps" --app "<m4_image_combinedflag.rps>" --app "<ta_image_combinedflag>" --sign "<m4_private_key.pem>"

Example

commander rps convert "**combined_image.rps**" --app " **Secured_sl_si91x_calendar.rps**" --app " **ta_fw.rps** " --sign "**m4_private_key.pem**"

Flash Combined Image

Syntax:

commander rps load <filename.rps> -d <full opn>

Example

commander rps load **combined_image.rps** -d si917

3.6 MBR Programming with Security

In the catalogue parts, the SiWG917 devices do not have security features enabled. The devices will have a default common flash configuration if the device's OPN has a flash.

In all the default cases, the device's MBR and EFUSE configurations have all the security features programmed to disabled state - the PUF is not enabled, the activation code for the PUF is not programmed, keys or key descriptors are not programmed.

- The security fields in MBR, PUF Activation code, Key descriptors, and the Keys are to be programmed in the device while enabling the security features in the device.
- The MBR includes the address of the sector where the PUF Activation code must be programmed as well as the address where the Key descriptors have to be saved.
- This way the user can change the location of the PUF Activation code, Key descriptor table, and the address of the Keys.
- The MBR contains a field – **valids,** which has bits to set the PUF_ACTIVATION_CODE_ADDRESS_VALID and KEY_DESCRIPTOR_ADDRESS_VALID
- The device uses these bits to determine the address of the PUF Activation code, Key Descriptor, and the Keys.

3.6.1 Programmable Fields in MBR and eFuse Data

The table below shows a few of the features and the MBR fields/eFuse data which can be programmed.

Feature	Structure	MBR Field
To enable TA MIC and encryption	efuse_data	ta_secure_boot_enable = 1;
To enableM4 MIC and encryption	efuse_data	m4_secure_boot_enable = 1;

To enable TA Signature	efuse_data	ta_digital_signature_validation = 1;
To enable M4 Signature	efuse_data	m4_digital_signature_validation = 1;
To enable TA inline encryption	efuse_data	ta_encrypt_firmware = 1; (1 for CTR, 2 for XTS mode)
To enable M4 inline encryption	efuse_data	m4_encrypt_firmware = 1;
To select M4 firmware encryption mode	efuse_data	m4_fw_encryption_mode = 1; (1 for CTR, 2 for XTS mode)
Disable the Key holder address to M4	efuse_data	disable_m4ss_kh_access = 0;
Start address of the sector where the PUF Activation code must be saved, 0x2000 is the default	mbr	puf_activation_code_addr = 0x2000;
Start address where the key descriptor table must be saved. (0x300 is the default)	mbr	key_desc_table_addr = 0x300;
If PUF Activation code's start address is being changed	mbr	valids |= PUF_ACTIVATION_CODE_ADDR_VALID;
If Key descriptor's start address is being changed	mbr	valids |= KEY_DESCRIPTOR_ADDRESS_VALID;

3.6.2 Security Levels

There are three different security Levels available. The levels are defined to make the security configurations easier for the user.

1. Security Level 1 or Low Security Level
2. Security Level 2 or Partial Security Level
3. Security Level 3 or Full Security Level

Security Level 1 (Low Security)

If the user wants to enable only a secure boot, this level shall be selected. This security level enables the bootloader to carry out the Message Integrity Check (MIC) of the firmware image during firmware loading and firmware update.

In this configuration, the firmware update process will be faster, the bootup time will be less, and the execution also will be faster.

This configuration is selected using the below fields:

```
"efuse_data": {
    "disable_m4ss_kh_access": 0,
    "ta_secure_boot_enable": 1,
    "m4_secure_boot_enable": 1
}
```

Security Level 2 (Partial Security)

If the user wants to enable secure boot along with signature check of firmware image, select this level. This security level enables the bootloader to carry out the Message Integrity Check (MIC) and signature validation of the firmware image during firmware loading and firmware update.

In this configuration, the firmware update process will be slower but faster than Security Level 3; the bootup time will be relatively less than the Security Level 3, and the execution will be same as Security Level 3.

This configuration is selected using the following fields:

```
"efuse_data": {
        "disable_m4ss_kh_access": 0,
        "ta_secure_boot_enable": 1,
        "ta_digital_signature_validation": 1,
        "m4_secure_boot_enable": 1,
        "m4_digital_signature_validation": 1
}
```

Security Level 3 (Full Security)

If the user wants to enable the secure boot, the signature check, and the inline encryption. This security level enables the bootloader to carry out the Message Integrity Check (MIC), encryption, signature check of the firmware image during firmware loading and firmware update and Inline Encryption while saving the firmware in the flash.

In this configuration, the firmware update process will be slower, the bootup time will be higher, and the execution also will be slower compared to security levels 1 and 2.

This configuration is selected using the following fields:

```
"efuse_data": {
        "disable_m4ss_kh_access": 0,
        "ta_secure_boot_enable": 1,
        "ta_digital_signature_validation": 1,
        "ta_encrypt_firmware": 1,         // 1 for CTR, 2 for XTS mode
        "m4_encrypt_firmware": 1,
        "m4_fw_encryption_mode ": 1, // 1 for CTR, 2 for XTS mode
        "m4_secure_boot_enable": 1,
        "m4_digital_signature_validation": 1
}
```

3.7 Example JSON file with Security Parameters of MBR

The following example shows the structure and the available fields:

```
{
  "puf_activation_code_addr": 8192,
    "efuse_data": {
        "disable_m4ss_kh_access": 0,
        "m4_digital_signature_validation": 1,
        "m4_encrypt_firmware": 1,
        "m4_fw_encryption_mode": 1,
        "m4_secure_boot_enable": 1,
        "ta_digital_signature_validation": 1,
        "ta_encrypt_firmware": 1,
        "ta_secure_boot_enable": 1
    },
    "key_desc_table_addr": 768
}
```

4 Manufacturing Procedure – eFuse data

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 users need to 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 –dry run option.

This command is used to write efuse data into flash.

Syntax:

commander manufacturing write efuse --data file.json -d <full opn > [--skipload] [--pinset n] [-s jlinkserialno] [--noprompt] [--dryrun]

5 MBR Programming – 1.6MB to 1.8MB

To program your MBR to 1.8MB (latest) from 1.6MB, follow the steps below.

Make sure the Simplicity Commander CLI is **1v16p1 and above version.**

For BRD4338A, download the 1.8MB MBR file from this link: ta_mbr_SiWG917M1xxMGTBA.bin

Copy the 1.8MB MBR file to the Simplicity Commander tool folder

- Go to the path where the Simplicity Commander is installed. For example: **C:\SiliconLabs\SimplicityStudio\v5\developer\adapter_packs\commander**
- Copy the 1.8MB MBR file to the above path.

Take Back-up of Calibration Data

Take back-up of the calibration data using the below command.

- commander manufacturing read efusecopy --out calib_data_Boardno.bin

Read the MBR

To confirm before moving forward, check the MBR format in the device, if it is 1.6MB or already the latest 1.8MB.

Use the following command to read the MBR.

- **commander manufacturing read m4mbrcf**
- If it reads the data from 0x41f0000 – it is 1.8MB MBR.
- If it reads the data from 0x41b0000 – it is 1.6MB MBR.

Write TA Firmware MBR

This command is used to flash the MBR content for TA region.

- commander manufacturing provision --mbr ta_mbr_SiWG917M1xxMGTBA.bin -d SiWG917M111MGTBA

Write M4 MBR

This command is used to flash the MBR content for M4 region.

- commander manufacturing write m4mbrcf --data ta_mbr_SiWG917M1xxMGTBA.bin -d SiWG917M111MGTBA

Take ipmu copy from TA region

- commander manufacturing read taipmu --out filename.bin

Load copied ipmu data it into M4 Region

- commander manufacturing write m4ipmucf --data filename.bin

Check if MBR is successfully loaded

The following command is used to check the MBR format.

- commander manufacturing read m4mbrcf
  - If it reads the data from 0x41f0000 – it is 1.8MB MBR – Successfully loaded.
  - If it reads the data from 0x41b0000 – it is 1.6MB MBR.

6 Security Features – Manufacturing Utility

The following sections provide information on how to enable SiWG917 security features using the manufacturing utility (Simplicity Commander CLI).

6.1 Secure Boot

Configure or enable/disable a pre-flashed secure bootloader on the chips to encrypt your software intellectual property (IP). Safeguard your competitive edge in the market.

Secure Boot is a feature for ensuring only an authenticated code can run on the chip. If any device fails its security check, it is not allowed to run, and program control will typically stall in the validating module.

SiWx917 SoC (SiWG917) possess two bootloaders.

- Security Bootloader
- Application Bootloader

The Security Bootloader runs on the ThreadArch processor, and the Application Bootloader runs on the Cortex M4 processor. **Secure Boot is implemented in the Security Bootloader.**

Secure Boot Process:

1. Upon reset, the Security Bootloader configures the module hardware based on the configuration present in the ROM and Flash.
2. Device configurations in Flash are authenticated by Security Bootloader
3. Security Bootloader validates the integrity and authenticity of the firmware (TA and M4) in the Flash and invokes the Application Bootloader.

To enable only secure boot or secure boot along with other security features refer to Security Levels section.

6.2 Secure Key Management and Protection

Inject custom public and private keys and other custom secret keys on the chips during manufacturing – safeguard your products right from the beginning of their lifecycle. The Bootloader uses public and private key-based digital signatures to recognize authentic software. The Security Bootloader provides provision for inline execution (XIP) of encrypted firmware from Flash.

If this feature is enabled, the PUF (Physically Unclonable Function) engine will be invoked. PUF Keys will be generated internally via PUF and stored in flash. Images are encrypted and decrypted using PUF intrinsic keys.

The PUF Intrinsic Keys are device-dependent keys. They are unique for each device.

PUF Intrinsic Keys are as below. These are generated randomly.

- Master Key – Used to encrypt the TA OTA Key and TA Public Key for storing in the Flash.
- Unwrap Key – Used to encrypt the M4 OTA Key and M4 Public Key for storing in the Flash.
- TA FW Keys (2) – Used for Inline decryption.
- M4 FW Keys (2) – Used for Inline decryption.

Keys generated at the commander are as follows.

- OTP Symmetric Key – Used for flash content (Message Integrity Check) verification.
- OTP Public Key – Used for flash content (digital signature) verification.
- TA Public Key – Used for TA firmware signature verification.
- TA OTA key - Used for TA Firmware upgrade.
- Attestation Private Key – Used for secure attestation.
- M4 OTA Key – Used for M4 firmware upgrade.
- M4 Public Key – Used for M4 firmware signature verification.

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

S.No	Key Type	Key Length	Storage	
			OTP	Flash
1	OTP Symmetric key	16 bytes	☑	☐
2	OTP Public key	91 bytes	☑	☐
3	TA public key	96 bytes*	☐	☑
4	TA OTA key	32 bytes	☐	☑
5	M4 OTA key	32 bytes	☐	☑
6	M4 public key	96 bytes*	☐	☑
7	Attestation Private key	240 Bytes*	☐	☑
8	Master Key	52 bytes key code	☐	☑
9	Unwrap Key	52 bytes key code	☐	☑
10	TA FW Key(2 keys)	52 bytes key code	☐	☑
11	M4 FW Key(2 keys)	52 bytes key code	☐	☑

Note: For TA and M4 Public key, actual key size is 91 bytes, remaining 5 padding bytes added to make key size multiple of 16 for AES encryption. For Attestation key also, actual key size is 227 bytes.

6.3 Secure Firmware Upgrade

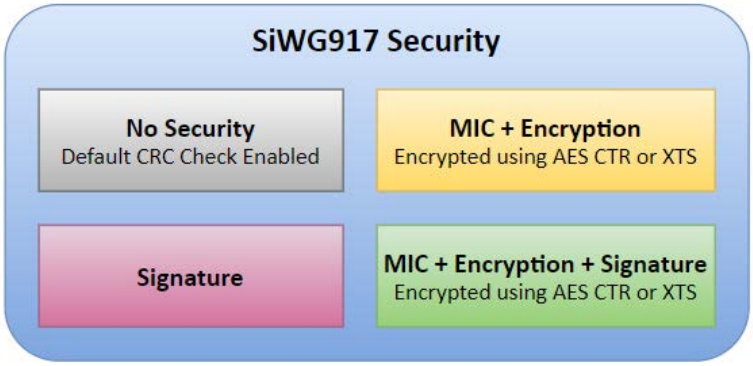
Pre-flash your firmware/application software in Silicon Labs chip securely in manufacturing without delaying your time to market at third parties.

The secure firmware update feature of the Bootloader checks the authenticity of the new firmware image along with its integrity. The Bootloader automatically detects the host interface in use and configures the host interface hardware accordingly. The Bootloader updates the image only after successfully validating the authenticity and integrity of the image. It prevents downgrade to a lower version of firmware using the anti-rollback feature if it is enabled.

The Bootloader also supports transparent migration to a wirelessly updated image and protection against failures by providing recovery mechanisms.

The TA Bootloader uses a proprietary format (called RPS) for its upgrade images. These files have extension “. rps”.

The firmware image for both TA and M4 supports the following:



- *No Security:* Default CRC of the image within the RPS header.
- *Message Integrity Check (MIC) + Encryption:* The MIC is calculated for the whole plain image and saved within the RPS header. The image is encrypted and appended to the RPS header. The image can be encrypted using AES CTR or XTS mode.
- *Signature:* The size of the image in the RPS header is incremented by the size of the signature. Finally, the whole RPS file's (RPS Header+ image) signature is calculated and appended to the end of the file.
- *MIC + Encryption + Signature:* The MIC is calculated for the whole plain image and saved within the RPS header. The image is 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 whole RPS file's (RPS Header+ image) signature is calculated and appended to the end of the file.

In each mode, the Control Flag of the RPS Header is updated with the correct configuration.

To understand how to select the image security of your choice refer to Security Levels section.

To enable security for TA and M4 images refer to Add Security to TA and M4 firmware Image Files.

To flash the secure TA and M4 images refer to Flash the Secured Images into Flash.

Note: Once you are done backing up the files and Security Key Programming is done, then you only need to add security to the TA/M4 firmware images and flash the secured images.

6.4 Secure Debug

Configure the debug port securely before the chips leave the factory with the Secure Lock Feature (which can be unlocked with a secure debug token)

The SiWG917 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 microcontrollers and other 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, however. This update should happen when the user has completed the development phase and entered the manufacturing phase for their products.

The MBR fields to update for JTAG locks are as follows:

```
"efuse_data": {
    "disable_ta_jtag": 1, // Disable TA JTAG
    " disable_m4_jtag ": 1, // Disable M4 JTAG
}
```

7 Calibration Using Manufacturing Utility

The following two flows are possible with the manufacturing utility (Simplicity Commander CLI) to calibrate the SiWG917 device.

7.1 Transmission in Burst Mode

7.1.1 Steps for CTUNE Adjustments

1. Setup the radio and start transmission.
   commander manufacturing radio --power 16 --phy 6MBPS --channel 1 --start -d SiWG917M111MGTBA
2. Measure the frequency on the instrument.
3. Adjust the CTUNE values.
   Offset = measured frequency (in kHz) – 2412000. This is a frequency offset correction value ranging from -255 to 255.
   commander manufacturing xocal --offset <Offset> --skipload -d SiWG917M111MGTBA
4. Check the instrument and verify that the channel frequency is within expectations. If not, repeat from step 3.
5. Store the CTUNE values. The radio transmission will stop.
   commander manufacturing xocal --store --skipload -d SiWG917M111MGTBA

7.1.2 Steps for Gain Offset Adjustments

1. Setup the radio and start transmission on channel 1
   commander manufacturing radio --power 16 --phy 6MBPS --channel 1 --start -d SiWG917M111MGTBA
2. Measure the output power.
   Calculate the offset to meet the expected output power (16 dBm).
   *Offset = ceil ((output power measure (dBm) + cable loss -- 16) * 2).* This is a frequency offset correction value ranging from -255 to 255.
3. Store the gain values for channel 1
   commander manufacturing gain --ch1 <Offset> --skipload -d SiWG917M111MGTBA
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

7.2 Transmission in Continuous Wave Mode

7.2.1 Steps for CTUNE Adjustments

1. Setup the radio and start transmission.
   commander manufacturing radio --power 16 --phy CW --channel 1 --start -d SiWG917M111MGTBA
2. Measure the frequency on the instrument using peak search.
3. Adjust the CTUNE values.
   Offset = measured frequency (in kHz) -- 2412000
   commander manufacturing xocal --offset <Offset> --skipload -d SiWG917M111MGTBA
4. Check the instrument and verify that the channel frequency is within expectations. If not, repeat from step 3.
5. Store the CTUNE values. The radio transmission will stop.
   commander manufacturing xocal --store --skipload -d SiWG917M111MGTBA

7.2.2 Steps for Gain Offset Adjustments

1. Setup the radio and start transmission on channel 1
   commander manufacturing radio --power 16 --phy 6MBPS --channel 1 --noburst --start -d SiWG917M111MGTBA
2. Measure the output power.

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
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

Note:

- **There can be a variation of up to +/- 2dB in power across parts at Typical/Room temperature.**
- **It is recommended that the user makes the "Customer Gain-offset" correction on customer-end products to correct for IC part-to-part variation and insertion-loss variations in the RF front-end on customer boards. This calibration process would ensure accurate Transmit power control for Regulatory compliance @ volume production.**

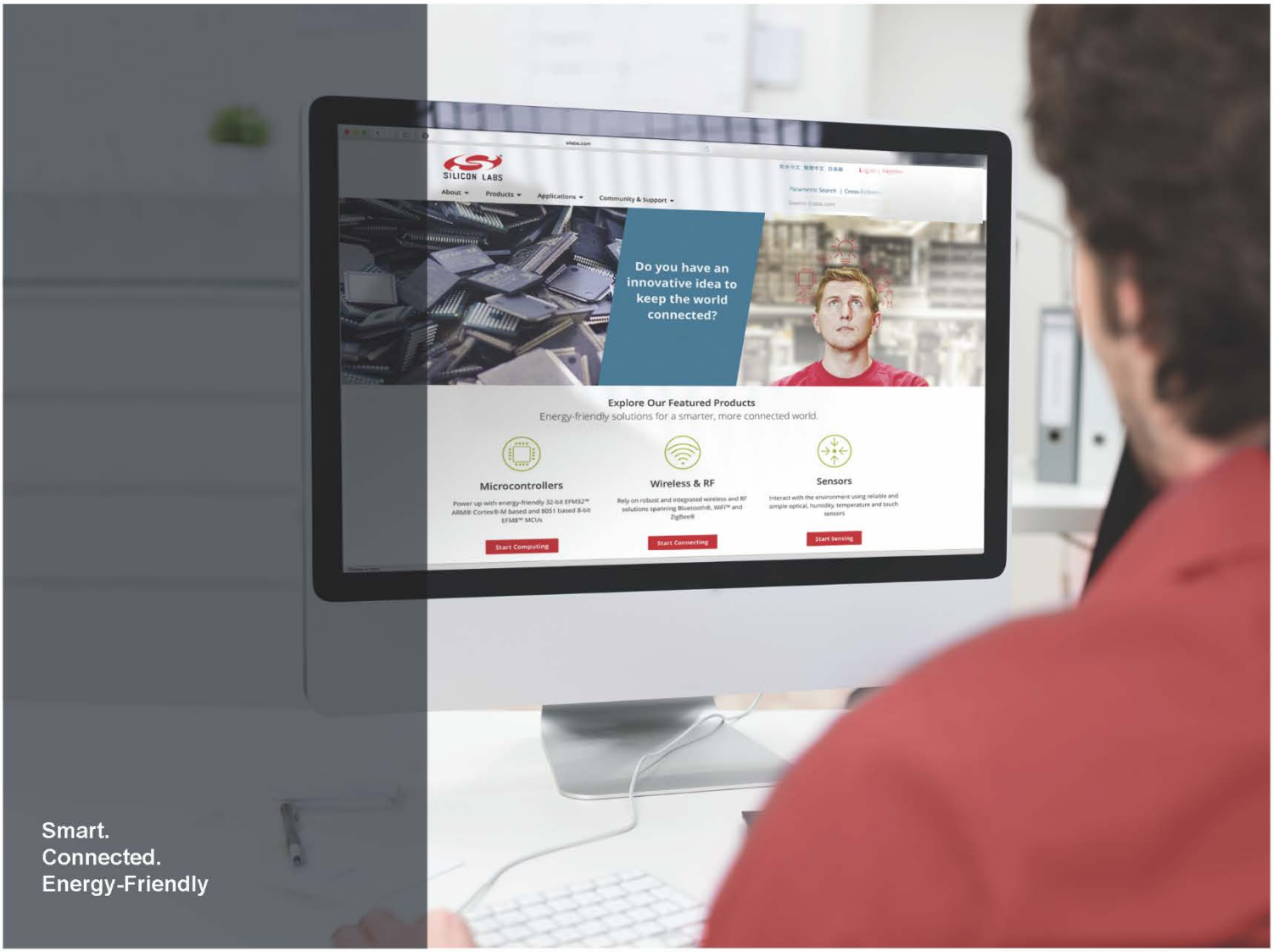
8 Possible Error Codes

Command	Error Code	Description	Cause of failure
MBR	0xa0ac	Verification failed from flash	There may be a wrong inputs 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 header part of rps file is not proper.

9 Appendix

9.1 Parameters - Manufacturing Utility

Field	Description
write	command for writing the MBR
read	read the contents
init	generate the activation code
--mbr	master boot record
--keys	security keys
m4mbrcf	Common flash M4 MBR
m4mbrdf	Dual flash M4 MBR
m4ipmucf	M4 IPMU data for common flash configuration
m4ipmudf	M4 ipmu data for dual flash configuration
efusecopy	updating eFuse data to flash for selected region
<full opn>	Provide the **OPN** number. **Example**: SiWG917M111MGTBA
<updated-mbr-fields.json>	File in which Security Level is programmed
-d	device
--skipload	Skip loading the TA provisioning firmware
--pinset	pin set(Self explanatory)
n	pin set number Internal Flash pin set: **0** External flash pin set: [**2/3**]



Products
www.silabs.com/products

Quality
www.silabs.com/quality

Support and Community
community.silabs.com

SILICON LABS

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

http://www.silabs.com