SCM1612
Wi-Fi 6 and BLE 5 Low-Power SoC

Device Firmware Update Guide

Revision 1.1
Date 2023-08-11

Contact Information
Senscomm Semiconductor (www.senscomm.com)
Room 303, International Building, West 2 Suzhou Avenue,
SIP, Suzhou, China
For sales or technical support, please send email to
info@senscomm.com

Disclaimer and Notice

This document is provided on an “as-is” basis only. Senscomm reserves the right to make corrections, improvements and other changes to it or any specification contained herein without further notice.
All liability, including liability for infringement of any proprietary rights, relating to use of information in this document is disclaimed. No licenses express or implied, by estoppel or otherwise, to any intellectual property rights are granted herein.
All third party’s information in this document is provided as is with NO warranties to its authenticity and accuracy.
All trade names, trademarks and registered trademarks mentioned in this document are property of their respective owners and are hereby acknowledged.

Version History

Version	Date	Description
1.1	2023-08-11	Format Modification
1.0	2023-08-04	Version 1.0 Release
0.1	2023-07-11	Initial Draft

Table of Contents

1 Introduction

The SCM1612 SDK offers support for Device Firmware Updates Over The Air (DFU OTA). This guide provides detailed instructions on how to utilize network connectivity for firmware updates.

Within the SDK, we have integrated the widely-used open-source bootloader, MCUBoot. Additionally, a porting layer tailored to specific architectures is included to facilitate seamless integration.

It's essential to note that MCUBoot is designed to support full image updates exclusively, without the capability for incremental updates. For a comprehensive understanding of MCUBoot and its functionalities, users are encouraged to consult the official documentation available at https://docs.mcuboot.com/.

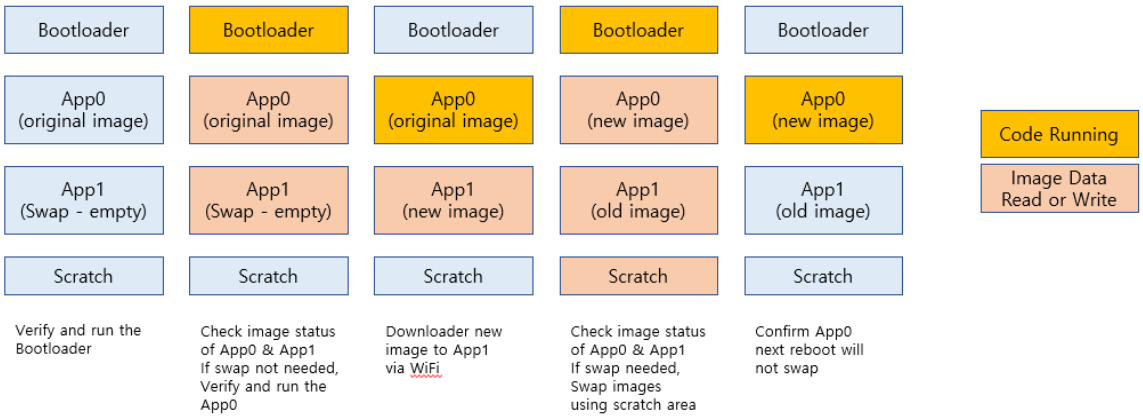
MCUBoot offers a variety of configuration options to cater to different application needs. While some of these options are accessible and can be modified via the `menuconfig`, others require manual adjustments in the `mcboot_config.h` file.

2 Work Flow

2.1 High level operation

The process for updating device firmware can be summarized in the following steps:

- Initial Setup: The device comes pre-loaded with a bootloader and the main application.
- Boot Process: Upon powering up, the bootloader initiates the application located in the primary slot.
- Update Preparation: Within the main application, an update agent is responsible for downloading the new application image. This new image is stored in a secondary slot.
- Image Download Completion: Once the new application image is successfully downloaded, the application logs this status and then triggers a system reboot.
- Bootloader Inspection: During the subsequent boot-up, the bootloader checks the recorded status. If an updated image is detected in the secondary slot, the bootloader swaps this image with the one in the primary slot.
- Application Boot-Up: The bootloader then initiates the application from the primary slot, which now contains the updated image.
- Status Clearance: If the new application image operates without issues, the application can proceed to clear the update status, signifying a successful update.



For the OTA firmware update process, it's crucial that the device allocates a secondary slot identical in size to the primary slot. Additionally, a designated scratch area is required, serving as a temporary space to facilitate the swapping between the primary and secondary images during updates.

2.2 Flash Layout

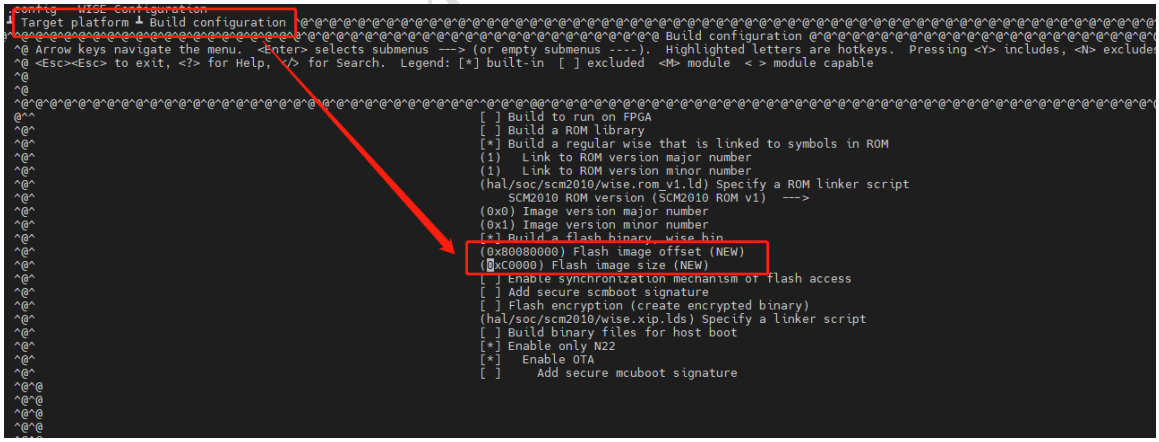
For a successful firmware update, the device requires specific regions in the flash memory. These areas include:

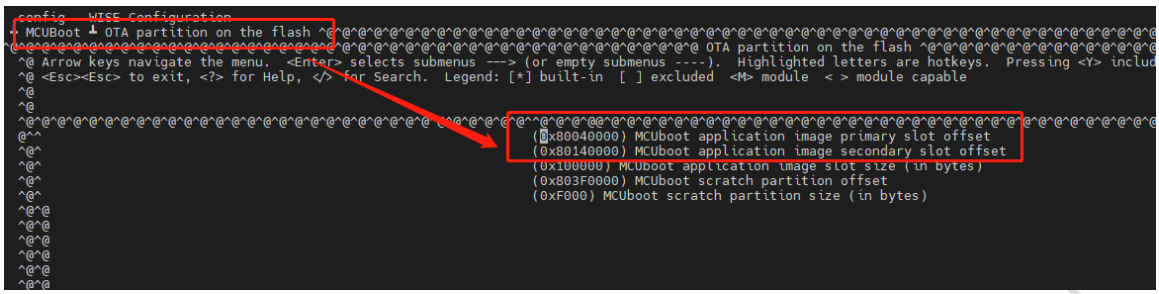
- Bootloader
- Primary Application Slot
- Secondary Application Slot
- Scratch

Below is a recommended flash memory layout for the SCM1612's internal 16Mbit flash:

Component	Address	Size (KB)
Bootloader	0x80000000	64
Scratch	0x80010000	16
Storage File System	0x80014000	16
Hibernation Backup	0x80018000	416
Primary Application Slot	0x80080000	768
Secondary Application Slot	0x80140000	768

You can modify the address and size of the flash memory layout using the menuconfig process. If any changes are made to the flash layout, both the bootloader and the application must be rebuilt to recognize these modifications.





Additionally, ensure that addresses align with 4KB boundaries, considering the flash erase block size.

3 Building the Firmware

3.1 Configuration and Build

The default configuration in the SDK already activates the firmware update feature. As a result, the generated application image will include the MCUBoot header, making it suitable for updates.

The menuconfig option to activate the firmware update feature is illustrated below:

```
[ ] Build to run on FPGA
[ ] Build a ROM library
[*] Build a regular wise that is linked to symbols in ROM
(1)   Link to ROM version major number
(1)   Link to ROM version minor number
(hal/soc/scm2010/wise.rom_v1.ld) Specify a ROM linker script
    SCM2010 ROM version (SCM2010 ROM v1)  --->
(0x0) Image version major number
(0x1) Image version minor number
[*] Build a flash binary, wise.bin
[ ]   Enable synchronization mechanism of flash access
[ ]   Add secure scmboot signature
[ ]   Flash encryption (create encrypted binary)
(hal/soc/scm2010/wise.xip.lds) Specify a linker script
[ ] Build binary files for host boot
[*] Enable only N22
[*]   Enable OTA
[ ]     Add secure mcuboot signature
```

When the device firmware update is enabled, the typical output is named ` wise.mcuboot.bin`.

3.2 Image Format

MCUboot mandates an image header that provides details about the image. A fixed-size header is always attached to the executable image's front, and variable-size TLV formatted data is appended if more information is required. The imgtool utility attaches this additional data during the build process.

The header comprises details like:

- Magic number
- Load memory address
- Header size
- TLV size
- Image size
- Flags
- Version

The trailer might include:

- Public key or key hash
- Image hash
- Signature

4 Source Code Overview

4.1 Key MCUboot Source Files

Below are some of the essential MCUboot source files:

Source Code	Description
lib/scm_mcuboot/mcuboot	original mcuboot library code
lib/scm_mcuboot/mcuboot/wise	Senscomm porting layer for the SDK
lib/scm_mcuboot/loader	mcuboot bootloader main function
lib/scm_mcuboot/update_agent	mcuboot update agent example

4.2 API Functions

The following functions are integral for both the bootloader and the application:

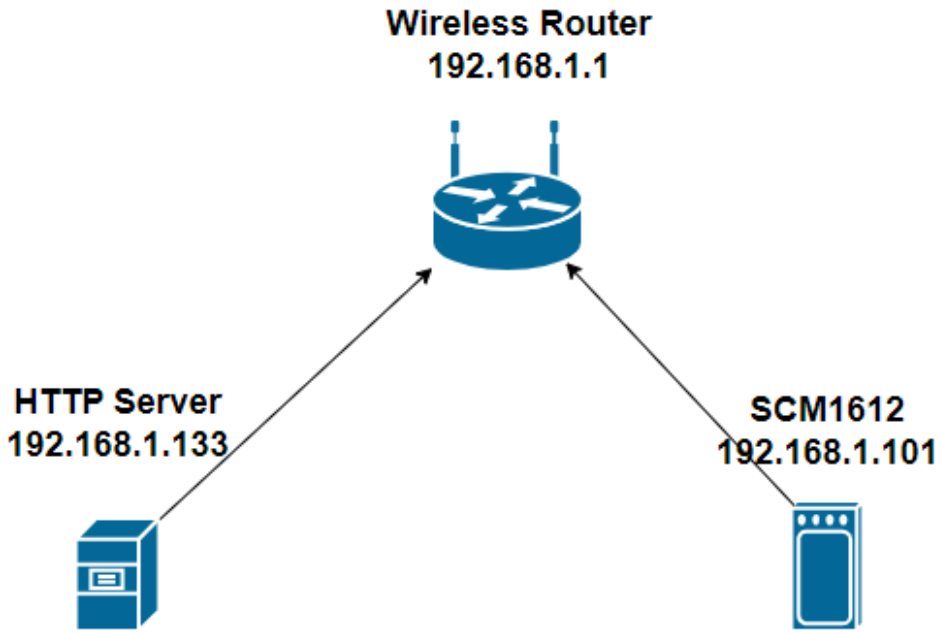
Name	Description
boot_go	Invoked by the bootloader to initiate the application boot process. This encompasses verification, swap, and booting based on status data.
boot_set_pending_multi	Used by the application to indicate a pending image, prompting the bootloader to swap and boot this image upon the next reboot.
boot_set_confirmed_multi	The application uses this to confirm the current image's validity and set it as permanent.
flash_area_open	Wrapper signaling the start of flash access.
flash_area_read	Wrapper for reading flash content into a buffer.
flash_area_write	Wrapper for writing buffer content to the flash.
flash_area_close	Wrapper to conclude flash operations.

5 Fireware Update Example

The device firmware can be updated using specific test commands. As an illustration, our sample update agent employs the HTTP protocol for image download. However, it's worth noting that other protocols can also be utilized for this purpose. When updating, applications must invoke the appropriate flash API functions to erase and write the updated image to the secondary slot.

5.1 Setting Up an HTTP Server

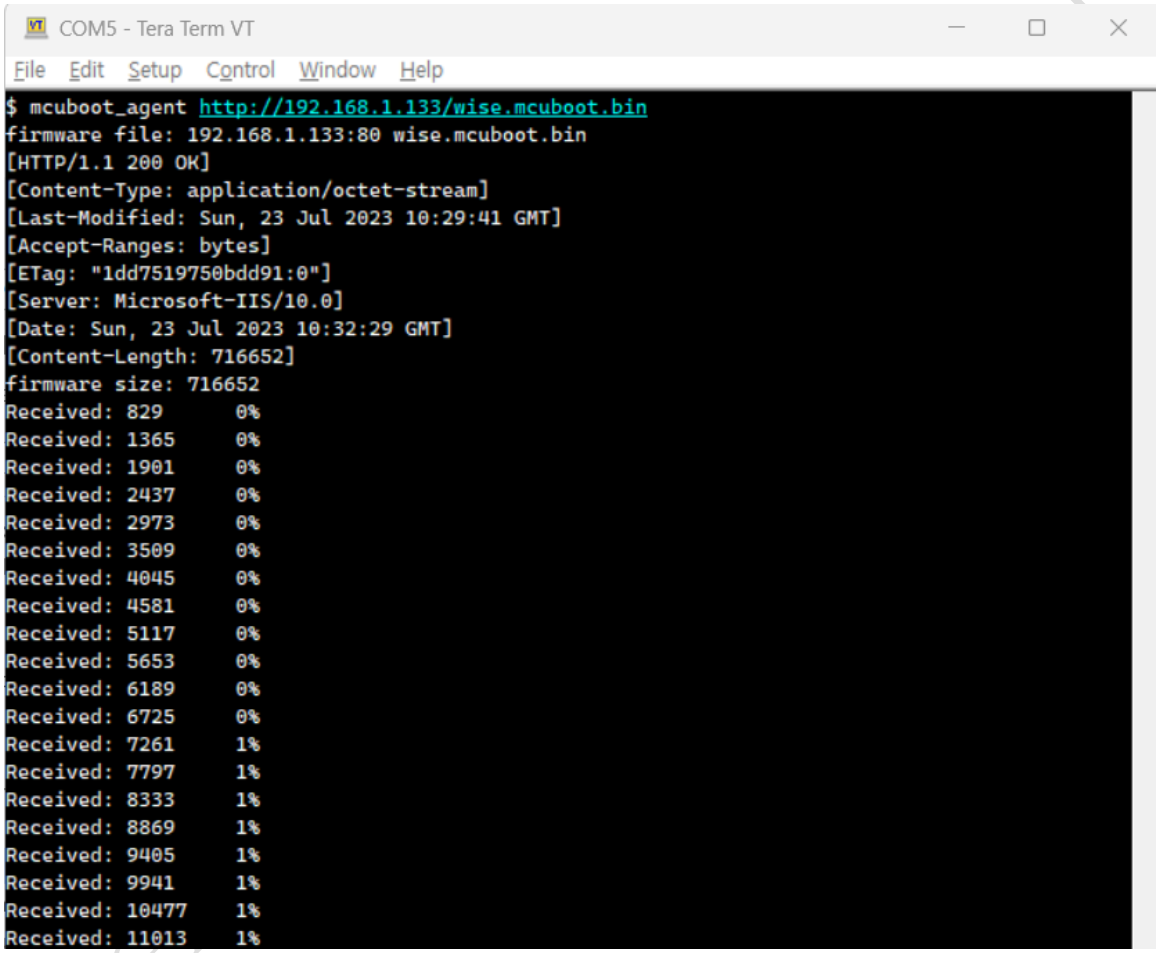
This section outlines the procedure to establish a Local Area Network (LAN) environment. Here, a wireless router acts as the gateway. Additionally, we'll delve into the steps to deploy an HTTP server, which will house the device firmware. The process to connect the SCM1612 to this wireless router via Wi-Fi is detailed in the "SCM1612 Wi-Fi Software Development Guide."



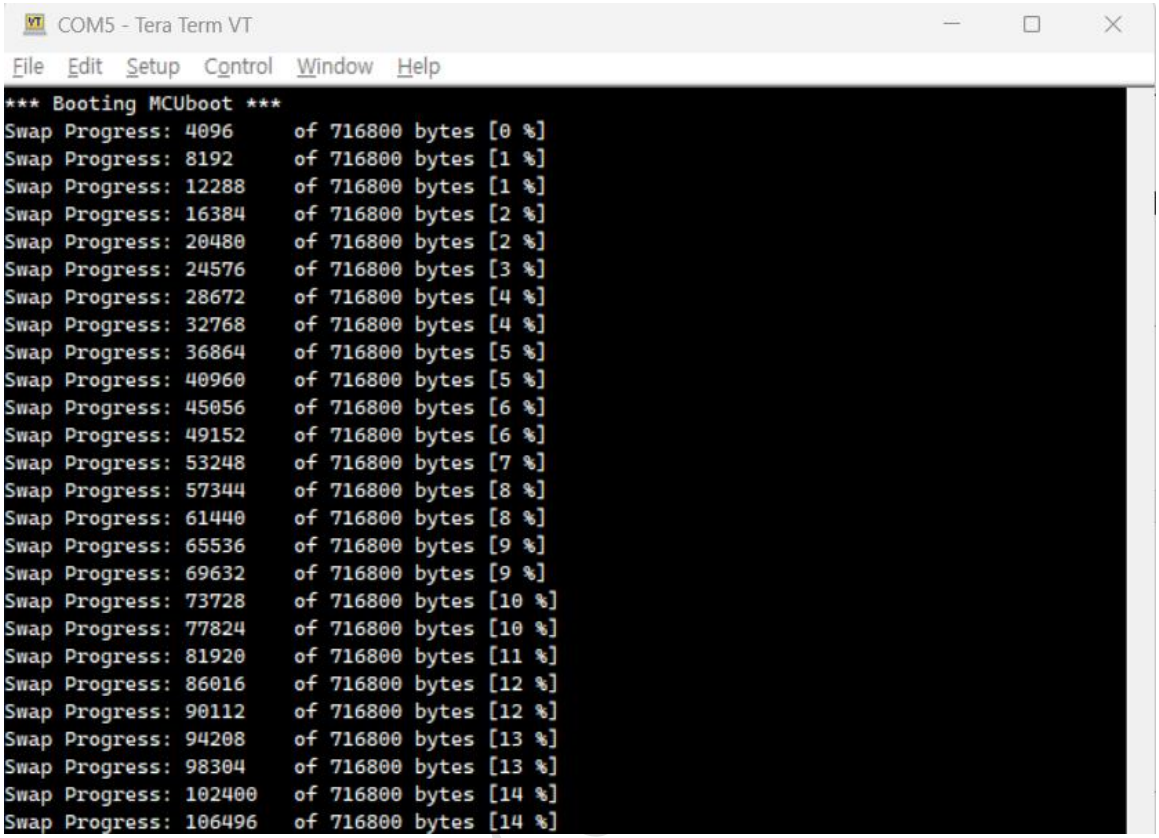
5.2 Initiating File Transfer

The diagram below depicts the commencement of the firmware update process. The new image is hosted on the HTTP server, accessible at the IP address 192.168.1.133.

```
# mcuboot_agent http://192.168.1.133/wise.mcuboot.bin
```



Upon successful transfer completion, the device will automatically reboot. Subsequently, the bootloader will interchange the primary and secondary images, as illustrated in the subsequent diagram.

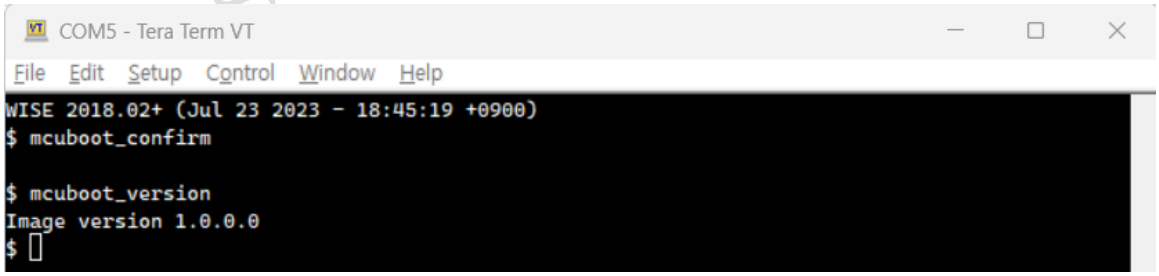


Once the swap is finalized, the bootloader will initiate the new application.

5.3 Verifying the Firmware

To designate the new image as permanent, use the following command:

```
# mcuboot_confirm
```



Important: If this confirmation step is skipped, the bootloader, upon the next device reboot, will revert the images in the primary and secondary slots,

effectively restoring the previous image. This is a safety measure, assuming the updated image might have encountered execution issues.

6 Secure Boot (TBD)

When Secure Boot is activated, the device will only boot images that have been signed using the user's security keys. The secure booting process is twofold:

1. BootROM authenticates the bootloader.
2. The bootloader, in turn, verifies the application.

6.1 Secure Boot by BootROM

The BootROM will activate Secure Boot if the corresponding eFuse bit is set. It uses the eFuse security keys to validate the bootloader's signature, or the image located at the flash's beginning. This image should possess a Senscomm-specific header with the necessary details.

6.2 Secure Boot by Bootloader

The bootloader will activate Secure Boot if it's constructed with the relevant menu option enabled. During the booting process, the bootloader, which is built with the security key, will validate the application firmware's signature. This application should have a MCUBoot-specific header and TLVs with the essential details.

7 Flash Encryption (TBD)

When Flash Encryption is active, the update procedure undergoes slight modifications. Given that raw firmware data (without encryption) shouldn't be exposed, the firmware intended for updates should also be encrypted before the downloading phase.

The build system is equipped to support flash encryption. When enabled, it produces both the encrypted image and the original image.