



Airoha IoT SDK for RTOS Firmware Update Developer's Guide

Version: 1.8

Release date: 30 June 2017

© 2015 - 2018 Airoha Technology Corp.

This document contains information that is proprietary to Airoha Technology Corp. ("Airoha") and/or its licensor(s). Airoha cannot grant you permission for any material that is owned by third parties. You may only use or reproduce this document if you have agreed to and been bound by the applicable license agreement with Airoha ("License Agreement") and been granted explicit permission within the License Agreement ("Permitted User"). If you are not a Permitted User, please cease any access or use of this document immediately. Any unauthorized use, reproduction or disclosure of this document in whole or in part is strictly prohibited. THIS DOCUMENT IS PROVIDED ON AN "AS-IS" BASIS ONLY. AIROHA EXPRESSLY DISCLAIMS ANY AND ALL WARRANTIES OF ANY KIND AND SHALL IN NO EVENT BE LIABLE FOR ANY CLAIMS RELATING TO OR ARISING OUT OF THIS DOCUMENT OR ANY USE OR INABILITY TO USE THEREOF. Specifications contained herein are subject to change without notice.

Document Revision History

Revision	Date	Description
1.0	7 March 2016	Initial release.
1.1	31 March 2016	Added MT2523x FOTA.
1.2	2 May 2016	Modified the output binary name to the current naming.
1.3	30 June 2016	Refined the chipset information and full FOTA update.
1.4	2 September 2016	Refined architecture diagram and flash default layout setting diagram on MT2523x.
1.5	4 November 2016	<ul style="list-style-type: none"> Adjusted the default flash layout on MT2523x and MT7687. Added default flash layout setting description on MT2533 and MT7697.
1.6	13 January 2017	Correct the FOTA tool download path.
1.7	5 May 2017	<ul style="list-style-type: none"> Refined section structure to describe FOTA by sub-features. Provide summary for FOTA features support status on all chipsets.
1.8	30 June 2017	<ul style="list-style-type: none"> Added default flash layout setting description on MT7682 and MT7686.

Table of Contents

1.	Overview	4
1.1.	Architecture layout of the SDK FOTA feature	4
1.1.1.	FOTA architecture layout for updating the firmware through Bluetooth	4
1.1.2.	FOTA architecture layout for updating the firmware through Wi-Fi	5
1.2.	Feature support status	6
2.	Using the FOTA Update.....	7
2.1.	FOTA packaging tool	7
2.1.1.	Using the Microsoft Windows version of the FOTA packaging tool	7
2.1.2.	Using the Linux version of the FOTA packaging tool	8
2.2.	FOTA update methods.....	9
2.2.1.	Full binary FOTA update	9
2.2.2.	Dual binary FOTA update.....	10
2.3.	Suggested flash layout setting for enabled FOTA.....	11
2.3.1.	MT2523 FOTA buffer partition configuration.....	11
2.3.2.	MT2533 FOTA buffer partition configuration.....	12
2.3.3.	MT7687 FOTA buffer partition configuration.....	12
2.3.4.	MT7697 FOTA buffer partition configuration.....	14
2.3.5.	MT7682 FOTA buffer partition configuration.....	14
2.3.6.	MT7686 FOTA buffer partition configuration.....	15
3.	FOTA Update Workflow	16
3.1.	Update through Bluetooth	16
3.1.1.	Download Manager Application	16
3.1.2.	Update Agent operation	17
3.1.3.	Custom configuration of FOTA	19
3.2.	Update through Wi-Fi.....	21
3.2.1.	Connect to network through Wi-Fi.....	21
3.2.2.	FOTA CLI command.....	21
3.2.3.	Download workflow.....	22
3.2.4.	Update Agent's workflow	26
4.	Appendix A: Acronyms and Abbreviations.....	28

Lists of Tables and Figures

Table 1. FOTA feature support status on chipsets	6
Table 2. The FOTA packaging tool's folder content on Windows PC	7
Table 3. The FOTA packaging tool's folder content on Linux PC.....	8
Table 4. Configuration attributes	8
Table 5. FOTA related compile options, when using Wi-Fi connectivity	9
Table 6. Dual binary FOTA update related compile options	11
Table 7. MT76x7 FOTA CLI command	21
Table 8. Acronyms and abbreviations	28
Figure 1. FOTA architecture layout for updating the firmware through Bluetooth.....	5
Figure 2. FOTA architecture layout for updating the firmware through Wi-Fi	6
Figure 3. FOTARomPacker.ini FOTA configuration file content	7
Figure 4. The output of build flow when dual binary FOTA update is enabled	10
Figure 5. MT2523 flash layout default settings.....	11
Figure 6. MT2533 flash layout default settings.....	12
Figure 7. MT7687 flash layout default settings.....	13
Figure 8. MT7697 flash layout default settings.....	14
Figure 9. MT7682 flash layout default settings.....	15
Figure 10. MT7686 flash layout default settings.....	15
Figure 11. Download manager's workflow	17
Figure 12. Update Agent operation flow	18
Figure 13. FOTA binary reserved memory	19
Figure 14. Update Agent's general flow and customization	20
Figure 15. HDK in Device Manager.....	23
Figure 16. PuTTY configuration panel	24
Figure 17. HDK connected to the Wi-Fi router.....	25
Figure 18. Update file successfully downloaded.....	26
Figure 19. MT76x7 FOTA update agent workflow	27

1. Overview

Airoha IoT SDK v4 enables firmware over the air (FOTA) update, a widely adopted cost and time efficient solution to update the firmware on connected devices. The purpose of the developer's guide is to provide complete description on how to deploy FOTA on the NOR flash.

This document guides you through:

- FOTA architecture layout.
- FOTA update on Airoha IoT development platform for RTOS.
- FOTA update workflow using the **Download Manager** and **Update Agent** tools.

A complete list of FOTA features for Airoha IoT development platform for RTOS is provided below.

- Full binary update
- Dual image update
- FOTA packaging tool
- FOTA update through Bluetooth (example)
- FOTA update through Wi-Fi
- Command line operation (example)

By default, the SDK provides full binary update mechanism. The SDK also enables third party solution integration to provide incremental update mechanism.

1.1. Architecture layout of the SDK FOTA feature

1.1.1. FOTA architecture layout for updating the firmware through Bluetooth

FOTA architecture for updating the firmware through Bluetooth is shown in Figure 1.

- The FOTA **Update Agent** is part of the **Bootloader** and it uses the flash APIs to process the update on the target device.
- The **Download Manager** in the **Application** is responsible for retrieving FOTA package file.
- The **Bluetooth notification service** is a proprietary protocol to communicate with Android smart devices to exchange data through Bluetooth. On the smartphone side, the smart device application enables to push the FOTA package file.
- The **FOTA packaging tool** (FOTARomPacker.exe located under the "FOTA" package of [labs download link](#).) is used on the PC to apply the Lempel–Ziv–Markov chain algorithm (LZMA) compression and sign signature in the new firmware file and to generate a FOTA update package.

The numbered items on Figure 1 are explained below:

- 1) **FOTA packaging tool** — generates an update package.
- 2) **HTTP server** — the update package storage location.

- 3) **Smart device application** — Bluetooth-enabled mobile device downloads the update package from the server.
- 4) **Bluetooth notification service** — Mobile device pushes the update package to the LinkIt HDK using Bluetooth communication.
- 5) **Download Manager** — While **Download Manager** receives the complete package, it sets the triggered flag and reboots the device.
- 6) **Update Agent** — checks the flag status and proceeds with the update.

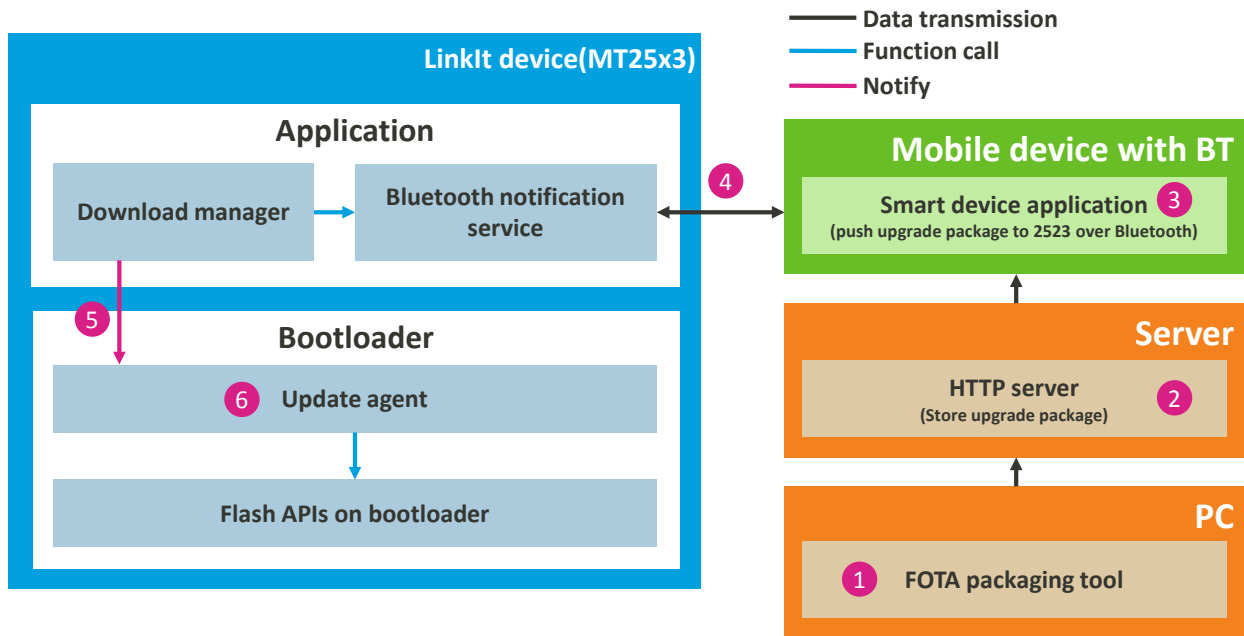


Figure 1. FOTA architecture layout for updating the firmware through Bluetooth

1.1.2. FOTA architecture layout for updating the firmware through Wi-Fi

FOTA architecture layout for updating through Wi-Fi is shown in Figure 2.

- The **FOTA Update Agent** is a part of the **Bootloader** and it uses flash APIs to process the update on the target device. While the device is connected to the Wi-Fi network, FOTA command line interface (CLI) commands can be used to download the FOTA package files from Trivial File Transfer Protocol (TFTP) server.
- The **FOTA packaging tool** (FOTARomPacker.exe located under root folder of the package) is used on the PC to apply the Lempel–Ziv–Markov chain algorithm (LZMA) compression and signature in the new firmware file and to generate a FOTA update package.

A TFTP server is required to download the files using the TFTP protocol, such as a third party open source software [tftp32](#).

The numbered items on Figure 2 are explained below:

- 1) **FOTA packaging tool** — generates an update package.
- 2) **HTTP or TFTP server** — puts the update package to the server.
- 3) **Wi-Fi** — the device connects to the network.

- 4) **FOTA CLI command** — triggers update package download using HTTP or TFTP client.
- 5) Sets the triggered flag and reboots the device.
- 6) **Update Agent** — checks the flag and proceeds with update.

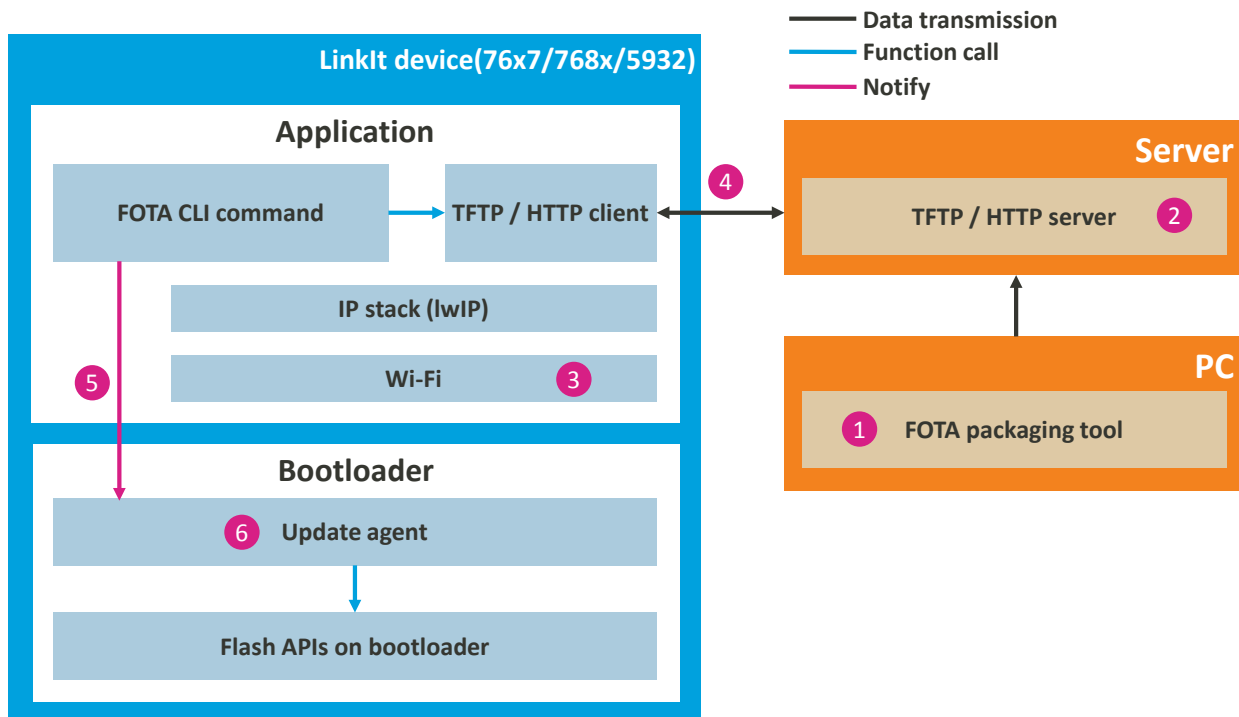


Figure 2. FOTA architecture layout for updating the firmware through Wi-Fi

1.2. Feature support status

Airoha IoT SDK for RTOS development platform is an integrated solution with limited availability of FOTA features on each chipset. The feature support summary on each chipset is shown in Table 1.

Table 1. FOTA feature support status on chipsets

Chipset	Full binary FOTA update	Dual binary FOTA update	Update through Bluetooth	Update through Wi-Fi
MT2523x	✓		✓	
MT2533	✓		✓	
MT7687	✓	✓		✓
MT7697	✓	✓		✓
MT7682	✓			✓
MT7686	✓			✓
MT5932	✓			✓

2. Using the FOTA Update

2.1. FOTA packaging tool

Airoha provides a FOTA packaging tool that runs on Microsoft Windows and Linux OS to compress data, generate a checksum and prefix header for the new binary FOTA package file. During the update, the **Update Agent** parses the header, validates the package file using the checksum and identifies where the data should be written.

2.1.1. Using the Microsoft Windows version of the FOTA packaging tool

Windows version of the FOTA packaging tool consists of the following items shown in Table 2.

Table 2. The FOTA packaging tool's folder content on Windows PC

Folders and Files	Description
_ini	Contains configuration and initialization files
_Load	A folder for a new image
_Output	A folder with output package files.
FOTARomPacker.exe	The main executable file to parse configuration files, header prefix, compress the data and generate a checksum.
gen_image.bat	Batch file with a complete configuration.

To use the FOTA packaging tool:

- 1) Copy the new binary to the _Load folder.
- 2) Configure the FOTARomPacker.ini file located under _ini folder.
 - Set the load path to the line below [General Setting] (see Figure 3).
 - Set the binary file's name (File) and the start address (Start_Address) of this binary in the flash.
 - Set the size of the partition where the binary will be written.
 - Configure the setting to compress the binary if needed. The LZMA compression is applied, if the value of Is_Compressed is true.
- 3) Execute the gen_image.bat batch file to generate a FOTA package file under _Output folder.

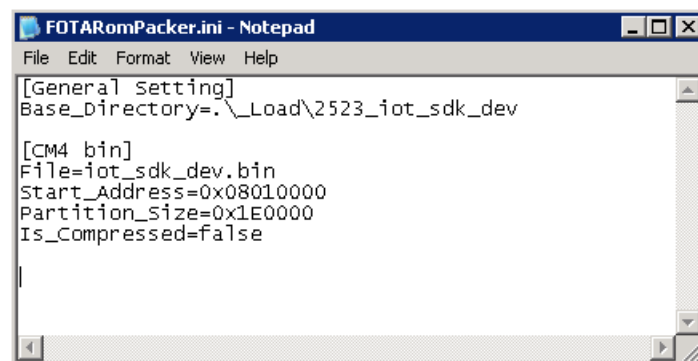


Figure 3. FOTARomPacker.ini FOTA configuration file content

2.1.2. Using the Linux version of the FOTA packaging tool

The Linux version of the FOTA packaging tool consists of the items shown in Table 3.

Table 3. The FOTA packaging tool's folder content on Linux PC

Folders and Files	Description
config_template	Contains configuration files in JSON format for Airoha IoT SDK for RTOS development platform.
example	An example with configuration, binary, executable and other files to generate the FOTA update file. The files should be replaced for other HDK configuration.
src	The source code of the tool.
build.linux.sh	The source should be compiled by cmake compiler on Linux OS to generate an executable named "BinBuilder" for the FOTA packaging tool. (If cmake compiler is installed on Windows OS, use build.windows.bat.)
CMakeLists.txt	Makefile for cmake compiler.
README.md	Describes how to use the tool.

To use the FOTA packaging tool, configure the JSON configuration file with the following attributes:

Table 4. Configuration attributes

Attribute	Type	Description
output	string	Defines the output configuration file path.
bins (An array of dictionary with 1 to 4 items)	file_path	Path to the binary file.
	start_addr	Offset address of the binary file in the flash, such as "0x7B00".
	partition_size	Size of the binary file on the flash, such as "0x8000".
	is_compressed	"true", if the binary file was compressed before packing.

Each item in "bins" represents a binary file with attributes listed in Table 4. For example, there are two binary files "mt7687_iot_sdk.bin" and "WIFI_RAM_CODE_MT76x7_in_flash.bin" with file path, start address, partition size and compression status, as shown below.

Note, that "start_addr" and "partition_size" are strings instead of numbers to accept C-style hexadecimal literals, such as "0xFFEF". An example configuration file looks like this:

```
{
  "output" : "image.bin"
  "bins": [
    {
      "file_path": "mt7687_iot_sdk.bin",
      "start_addr": "0x79000",
      "partition_size": "0xBF000",
      "is_compressed": true
    },
    {
      "file_path": "WIFI_RAM_CODE_MT76x7_in_flash.bin",
      "start_addr": "0x10000",

```

```

        "partition_size": "0x69000",
        "is_compressed": true
    }
]
}

```

By default the program loads “config.json” in the same directory of the executable file (“BinBuilder”). Alternatively, JSON configuration file path can be assigned through command parameter, as shown below:

```
./BinBuilder my_config.json
```

2.2. FOTA update methods

2.2.1. Full binary FOTA update

This update method is easy to use and doesn't occupy much flash memory space if a compression algorithm is applied to reduce update package size.

In the full binary update, the project is completely rebuilt and a new binary file is generated including all linked object files. There is a reserved space on the NOR Flash to store the new binary. Once the new binary file is successfully transferred and stored in the specified address of the reserved space, the system will reboot automatically. Normally, the system reboots to enter bootloader phase.

Once the bootloader detects there is a new binary file on the flash, it replaces the old binary file with the new one in the ARM Cortex-M4 firmware partition. After successful replacement, the system reboots with the new binary.

2.2.1.1. Compile options for the full binary FOTA update

The compile options for the FOTA update are configured in the main project and bootloader project makefiles (feature.mk). One of the makefiles is in the main application folder, the other is in the bootloader project folder that builds the bootloader binary. Only when FOTA options in both of them are enabled, the FOTA feature is enabled.

Configure the following option in PROJECT_DIRECTORY/GCC/feature.mk of the main application.

```
MTK_FOTA_ENABLE = y
```

The same option also needs to be configured for the bootloader's makefile under project/mt2523_hdk/apps/bootloader/GCC/feature.mk in bootloader project folder, as shown below:

```
MTK_FOTA_ENABLE = y
```

If updating the firmware through Wi-Fi, other compile options can be enabled or disabled depending on the required modules for FOTA update on the HDK in the PROJECT_DIRECTORY/GCC/feature.mk file, as shown in Table 5.

Table 5. FOTA related compile options, when using Wi-Fi connectivity

Option	Description	Build the ARM Cortex-M4	Build the bootloader
MTK_TFTP_ENABLE	TFTP module on/off	=y	=n

Option	Description	Build the ARM Cortex-M4	Build the bootloader
MTK_FOTA_CLI_ENABLE	Command line on/off	=y	=n
MTK_BL_FOTA_LOG_ENABLE	Bootloader logging on/off	=n	=y
MTK_HAL_PLAIN_LOG_ENABLE	HAL debug	=n	=y

2.2.1.2. FOTA buffer partition size for full binary update

The FOTA update package data is saved in the FOTA reserved partition. To customize the flash layout, consider saving the storage size for the FOTA reserved partition. Usually FOTA packaging tool will compress the binary file to reduce the reserved quota size by default, so the reserved storage is 65% of the total size of the target partitions ready to update.

2.2.2. Dual binary FOTA update

In this FOTA design, there are two executable partitions on flash layout, called partition A and partition B. Both of them could run firmware. However, at any time, only one of these two partitions is used to run firmware called active partition, the other one is used to store the update file during FOTA update process called inactive partition. After updating new firmware, swap the role of these two partitions so that the original inactive partition becomes active and run the new firmware, the original active partition becomes inactive and retains the former version of the firmware version.

If dual binary FOTA is enabled, build flow will always output two binary files to run on partition A and partition B respectively, and they aren't swappable due to firmware data is associated with ROM flash physical address in compiler link phase. The output of build flow is usually shown as Figure 4.

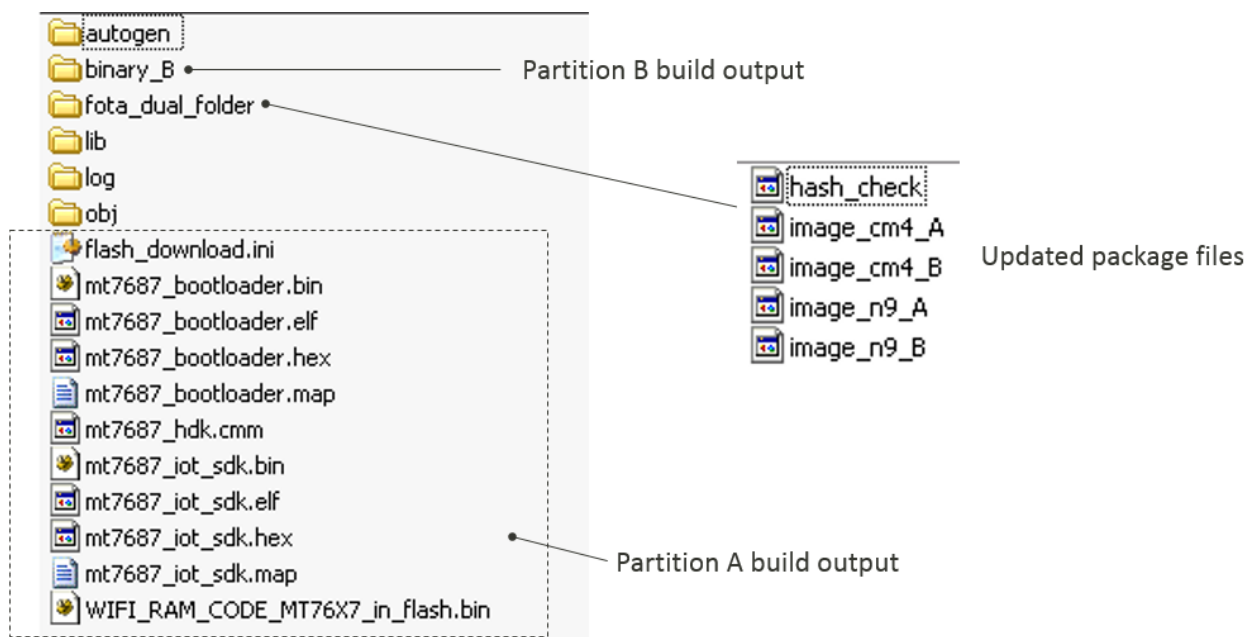


Figure 4. The output of build flow when dual binary FOTA update is enabled

2.2.2.1. Compile options for the dual binary FOTA update

The compile options for the FOTA update are configured in the main project and bootloader project makefiles (feature.mk). One of the makefiles is in the main application folder, the other is in the bootloader project folder that builds the bootloader binary. Only when FOTA options in both of them are enabled, the dual binary FOTA update feature is enabled.

The compile option settings to enable dual binary FOTA update are shown in Table 6.

Table 6. Dual binary FOTA update related compile options

Option	Description	Build the ARM Cortex-M4	Build the bootloader
MTK_FOTA_ENABLE	FOTA module on/off	=y	=y
MTK_FOTA_DUAL_IMAGE_ENABLE	Dual binary feature option on/off	=y	=y
MTK_FOTA_DUAL_IMAGE_ONE_PACKET_ENABLE	Automatically generates a single update package file.	=y	N/A
MTK_FOTA_CLI_ENABLE	Dual binary FOTA update is triggered by CLI command, should enable this option.	=y	=n

2.3. Suggested flash layout setting for enabled FOTA

2.3.1. MT2523 FOTA buffer partition configuration

Default flash layout with FOTA enabled on MT2523x is shown in Figure 5.

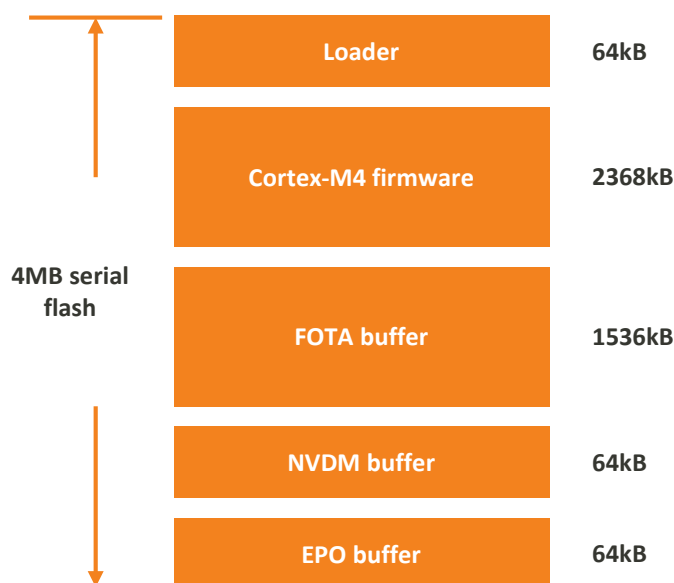


Figure 5. MT2523 flash layout default settings

To change the FOTA reserved partition size, adjust the flash layout settings.

- Find more details in the Memory Layout Developer's Guide under <sdk_root>/doc folder.
- Change the FOTA buffer's start address and length.

In `driver/board/25x3_hdk/bootloader/core/src/custom_bl_config.c` the function `bl_custom_fota_start_address()` provides the FOTA buffer partition's start address and the function `bl_custom_fota_size()` provides the FOTA buffer partition's length. Bootloader loads the FOTA binary from the FOTA buffer partition's start address and writes the FOTA update result to the tail of the FOTA buffer partition.

Bootloader identifies the tail of the FOTA buffer partition by using the start address and length of the FOTA buffer partition.

Modify the return value of functions `bl_custom_fota_start_address()` and `bl_custom_fota_size()` to adjust with custom modifications of the flash layout to provide matching start address and length of the FOTA buffer partition.

2.3.2. MT2533 FOTA buffer partition configuration

MT2533 FOTA update is almost the same as for MT2523x; the differences are in default flash layout settings. The default flash layout settings of MT2533 are shown in Figure 6.

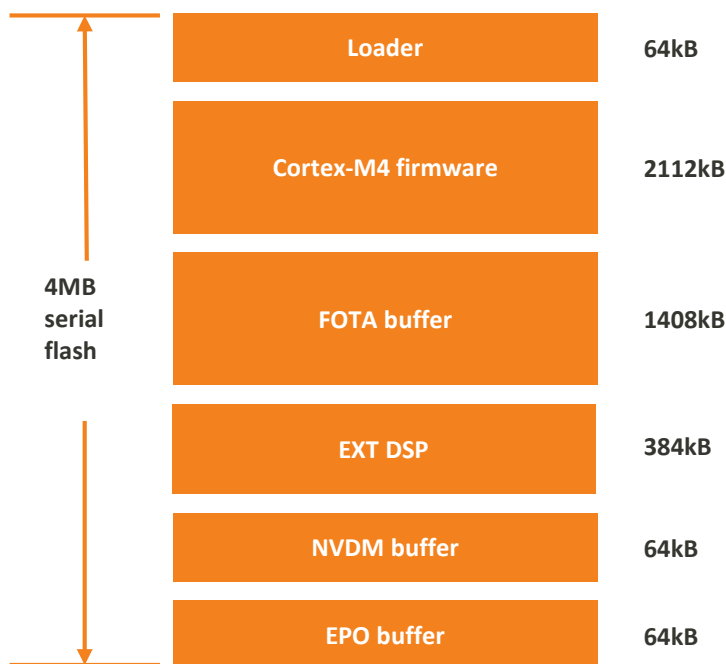


Figure 6. MT2533 flash layout default settings

To change the FOTA reserved partition size, see section 2.3.1, “MT2523 FOTA buffer partition configuration”.

2.3.3. MT7687 FOTA buffer partition configuration

Update package data should be saved on FOTA buffer partition first. Consider reserving space for FOTA buffer partition when customizing the FOTA layout. MT7687 FOTA feature can update the ARM Cortex-M4 and N9 firmware together, thus, the reserved quota size should be the sum of the maximum of ARM Cortex-M4 and N9 partition sizes. In addition, Airoha provides LZMA compression to reduce the update package size. LZMA compression ratio is about 60%. It's suggested to apply 65% to safely calculate sufficient reserved quota.

Reserved size $\approx ((\text{max ARM Cortex-M4}) + (\text{max N9})) * 65\%$

Default settings of the MT76x7 flash layout are shown in Figure 7.

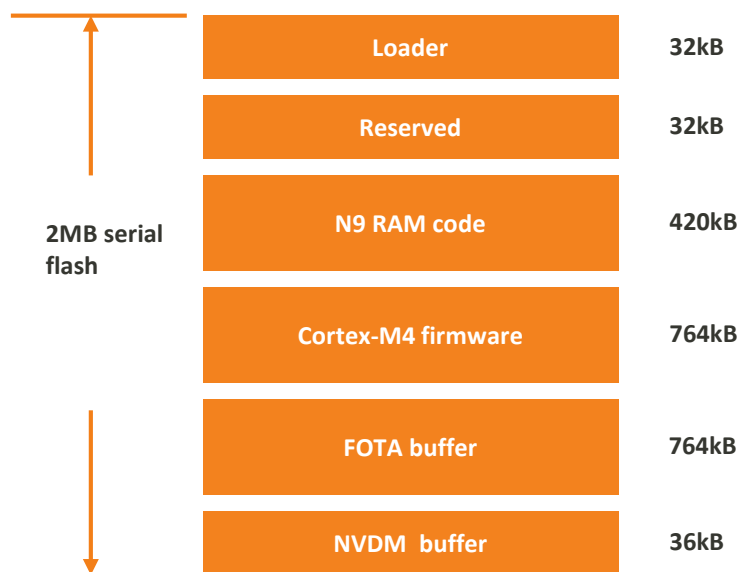


Figure 7. MT7687 flash layout default settings

To change the FOTA buffer partition size, refer to the following.

- Find more details in the Memory Layout Developer's Guide under <sdk_root>/doc folder.
- Change the start address and length of the FOTA buffer.

The bootloader reads the FOTA package from the FOTA buffer. The start address is defined as CM4_FLASH_TMP_ADDR and the length is defined as FLASH_TMP_SIZE in PROJECT_DIRECTORY/inc/flash_map.h header file. If the FOTA buffer is modified, the values of CM4_FLASH_TMP_ADDR and FLASH_TMP_SIZE also need to be modified in flash_map.h, as shown below.

```
#define FLASH_LOADER_SIZE          0x8000          /* 32kB */
#define FLASH_COMM_CONF_SIZE       0x1000          /* 4kB */
#define FLASH_STA_CONF_SIZE        0x1000          /* 4kB */
#define FLASH_AP_CONF_SIZE         0x1000          /* 4kB */
#define FLASH_N9_RAM_CODE_SIZE     0x71000         /* 452kB */
#define FLASH_CM4_XIP_CODE_SIZE    0xBF000         /* 764kB */
#define FLASH_TMP_SIZE              0xBF000         /* 764kB */
#define FLASH_USR_CONF_SIZE        0x5000          /* 20kB */
#define FLASH_EEPROM_SIZE          0x1000          /* 4kB */

#define CM4_FLASH_LOADER_ADDR       0x0
#define CM4_FLASH_COMM_CONF_ADDR   (CM4_FLASH_LOADER_ADDR +
FLASH_LOADER_SIZE)
#define CM4_FLASH_STA_CONF_ADDR    (CM4_FLASH_COMM_CONF_ADDR +
FLASH_COMM_CONF_SIZE)
#define CM4_FLASH_AP_CONF_ADDR     (CM4_FLASH_STA_CONF_ADDR +
FLASH_STA_CONF_SIZE)
#define CM4_FLASH_N9_RAMCODE_ADDR  (CM4_FLASH_AP_CONF_ADDR +
FLASH_AP_CONF_SIZE)
#define CM4_FLASH_CM4_ADDR          (CM4_FLASH_N9_RAMCODE_ADDR +
FLASH_N9_RAM_CODE_SIZE)
#define CM4_FLASH_TMP_ADDR          (CM4_FLASH_CM4_ADDR +
FLASH_CM4_XIP_CODE_SIZE)
#define CM4_FLASH_USR_CONF_ADDR    (CM4_FLASH_TMP_ADDR +
FLASH_USR_CONF_SIZE)
```

```
#define CM4_FLASH_EEPROM_ADDR      (CM4_FLASH_USR_CONF_ADDR +  
FLASH_EEPROM_SIZE)
```

2.3.4. MT7697 FOTA buffer partition configuration

MT7697 FOTA feature is almost the same as for MT7687, the differences are in default flash layout settings. The default flash layout settings on MT7697 are shown in Figure 8:

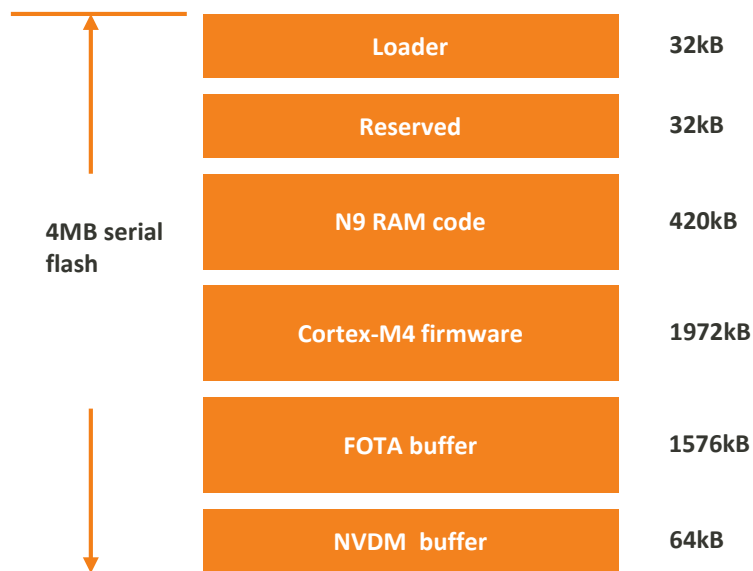


Figure 8. MT7697 flash layout default settings

To change the FOTA buffer partition size, refer to the method on MT7687, described in section 2.3.3, “MT7687 FOTA buffer partition configuration”.

2.3.5. MT7682 FOTA buffer partition configuration

Default flash layout with FOTA enabled on MT7682 is shown in Figure 9.

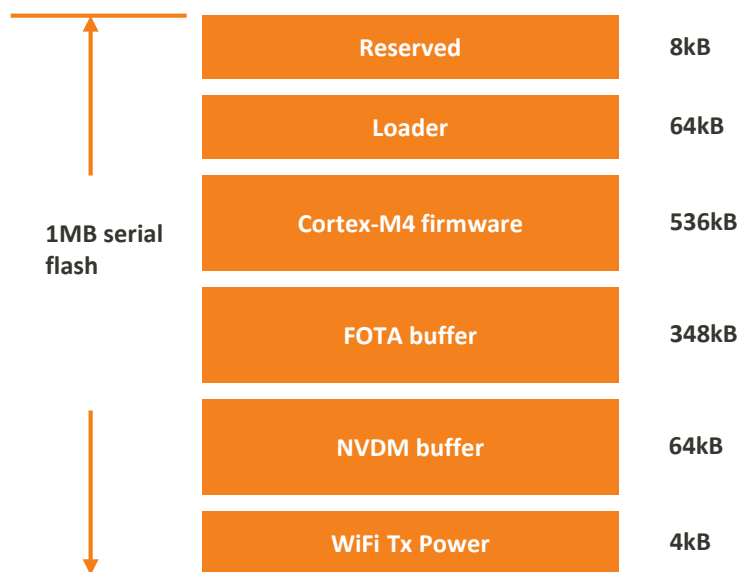


Figure 9. MT7682 flash layout default settings

To change the FOTA buffer partition size, refer to the method on MT7687, described in section 2.3.3, “MT7687 FOTA buffer partition configuration”.

2.3.6. MT7686 FOTA buffer partition configuration

Default flash layout with FOTA enabled on MT7682 is shown in Figure 10.

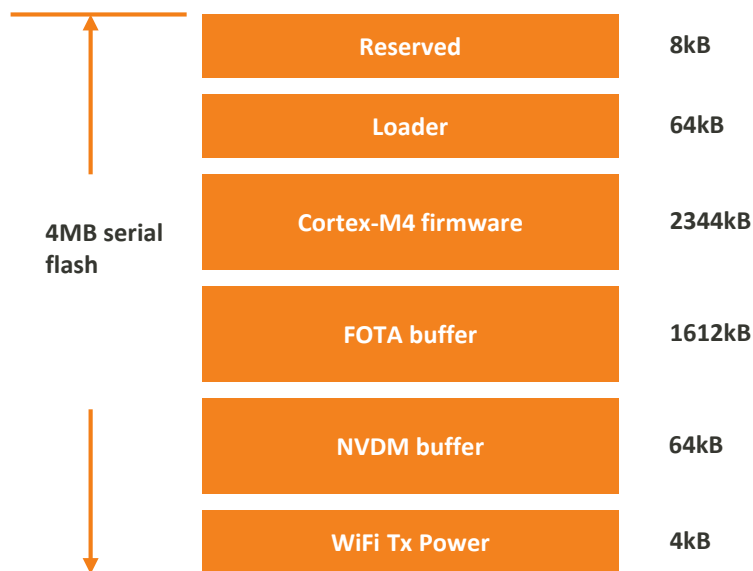


Figure 10. MT7686 flash layout default settings

To change the FOTA buffer partition size, refer to the method on MT7687, described in section 2.3.3, “MT7687 FOTA buffer partition configuration”.

3. FOTA Update Workflow

This section provides details on how to download the FOTA package and update the firmware in bootloader on HDK, including two update methods through Bluetooth and Wi-Fi. The update process for HDKs included in the Airoha IoT SDK for RTOS development platform is almost the same.

3.1. Update through Bluetooth

3.1.1. Download Manager Application

Download Manager is an application that receives the update binary from the APK on the smart device through Bluetooth, stores it on the flash while doing an integrity check, triggers a reboot process to update the binary file and returns the result to the smart device. The **Download Manager** application operates on Bluetooth service. It registers a callback on the Bluetooth event, notifies about a Bluetooth connection, data transfer, Bluetooth disconnection and other Bluetooth related events.

As there is no file system available on the flash, the **Download Manager** interacts with the flash directly using the flash APIs to read, write, erase the flash that stores the binary file and access the update result provided by the bootloader after the system reboot. The **Download Manager** also provides UI for the smart device to get information, such as binary file's version number. The workflow of **Download Manager** is shown in Figure 11.

A reference example project for **Download Manager** is provided in the SDK release package as the example project, located at <sdk_root>/project/mt2523_hdk/apps/fota_download_manager. This example project includes middleware modules, such as Bluetooth, FOTA, bt_callback_manager, nvdm and bt_notify under <sdk_root>/middleware/MTK, along with the fundamental BSP module, and provides the capability of transmitting update file over SPP profile through Bluetooth or over GATT profile through Bluetooth LE.

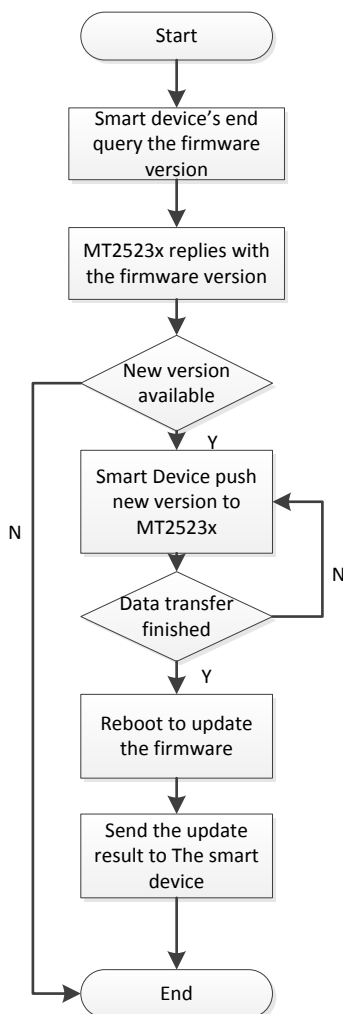


Figure 11. Download manager's workflow

3.1.2. Update Agent operation

The purpose of the **Update Agent** is to read the FOTA binary that contains the OS binary and then write the OS binary to the OS partition. The workflow of **Update Agent** is shown in Figure 12.

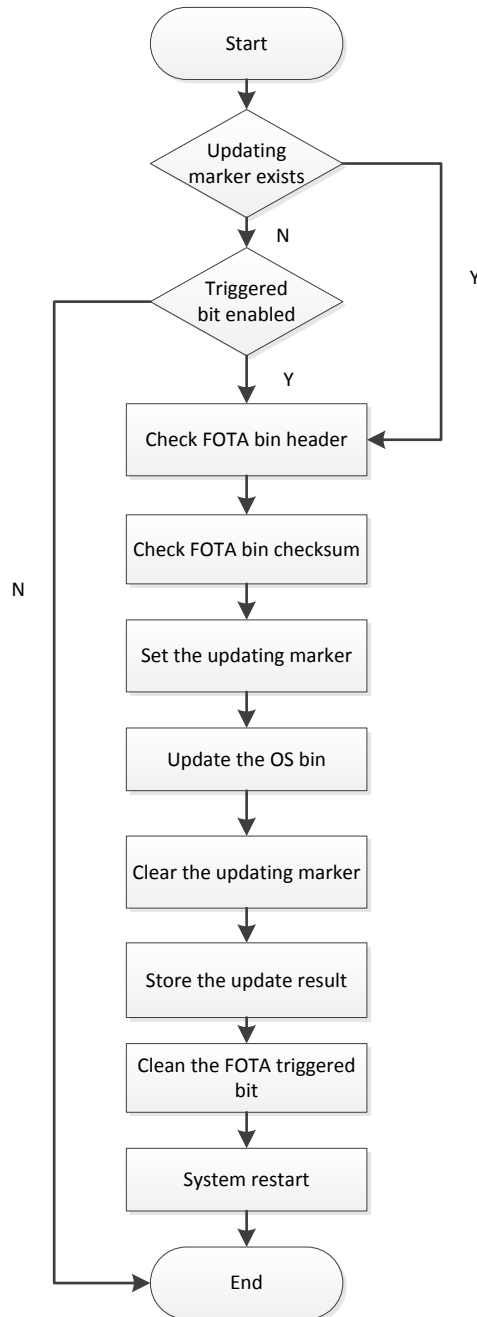


Figure 12. Update Agent operation flow

The FOTA binary includes a header, the OS binary and a checksum. The **Update Agent** reads the header to identify the start address and size of the OS partition. The header also includes the checksum position and a magic number. The **Update Agent** uses the magic number to check if the FOTA binary is available and uses the checksum to confirm the FOTA binary contains no errors. The FOTA binary header is provided in the IOT_FOTA_HEADER structure in the header file located at `driver/board/mt25x3_hdk/bootloader/core/inc/bl_fota.h`, as shown below.

```
typedef struct
{
    U32 m_bin_offset;
    U32 m_bin_start_addr;
    U32 m_bin_length;
    U32 m_partition_size;
    U32 m_sig_offset;
    U32 m_sig_length;
    U8 m_bin_reserved[4];
} IOT_FOTA_BIN_INFO;
typedef struct
{
    U32 m_magic_ver;
    IOT_FOTA_BIN_INFO m_bin_info;
} IOT_FOTA_HEADER;
```

The **Update Agent** decides whether to start the FOTA update flow based on the value of the FOTA triggered bit in the retention register, once the bootloader starts.

There is a power failure protection for the OS binary (main binary) update. Before the OS binary update, the **Update Agent** writes a marker to the flash, as shown in Figure 13. After the OS binary update is complete, the marker is cleared from the flash. If power failure occurred during the OS binary update, the **Update Agent** checks marker to restart the process.

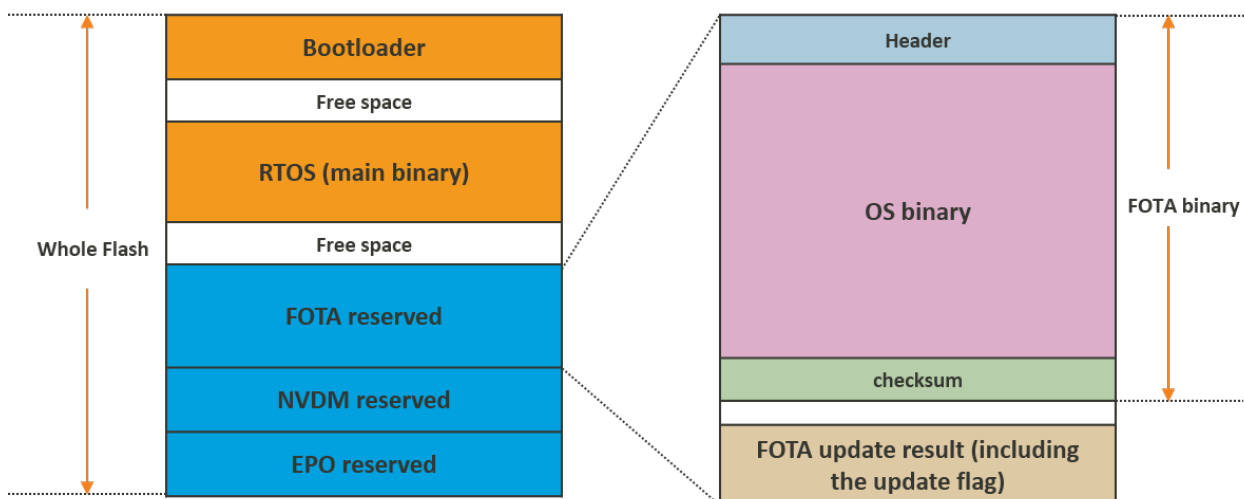


Figure 13. FOTA binary reserved memory

The FOTA binary is stored in the FOTA buffer partition. The FOTA buffer partition contains the binary and the final update result. The FOTA binary is written using the **Download Manager** and the update result is written by the **Update Agent** in bootloader. The **Download Manager** then sends the update result to the smart device application. Then the application displays the corresponding string to notify the end user.

The FOTA binary is placed at the tail of the FOTA buffer partition, as it only occupies a block size and is smaller compared to the FOTA buffer partition size. FOTA update result is stored in the tail of the FOTA Buffer partition. Once the update is finished, the **Download Manager** reads the result and then erases the tail of FOTA buffer partition.

3.1.3. Custom configuration of FOTA

To apply a custom flow for the **Update Agent**'s workflow see Figure 14. According to Figure 14, the **Update Agent** verifies if an update is required, applies the update operation and follows the post processing after the update is complete.

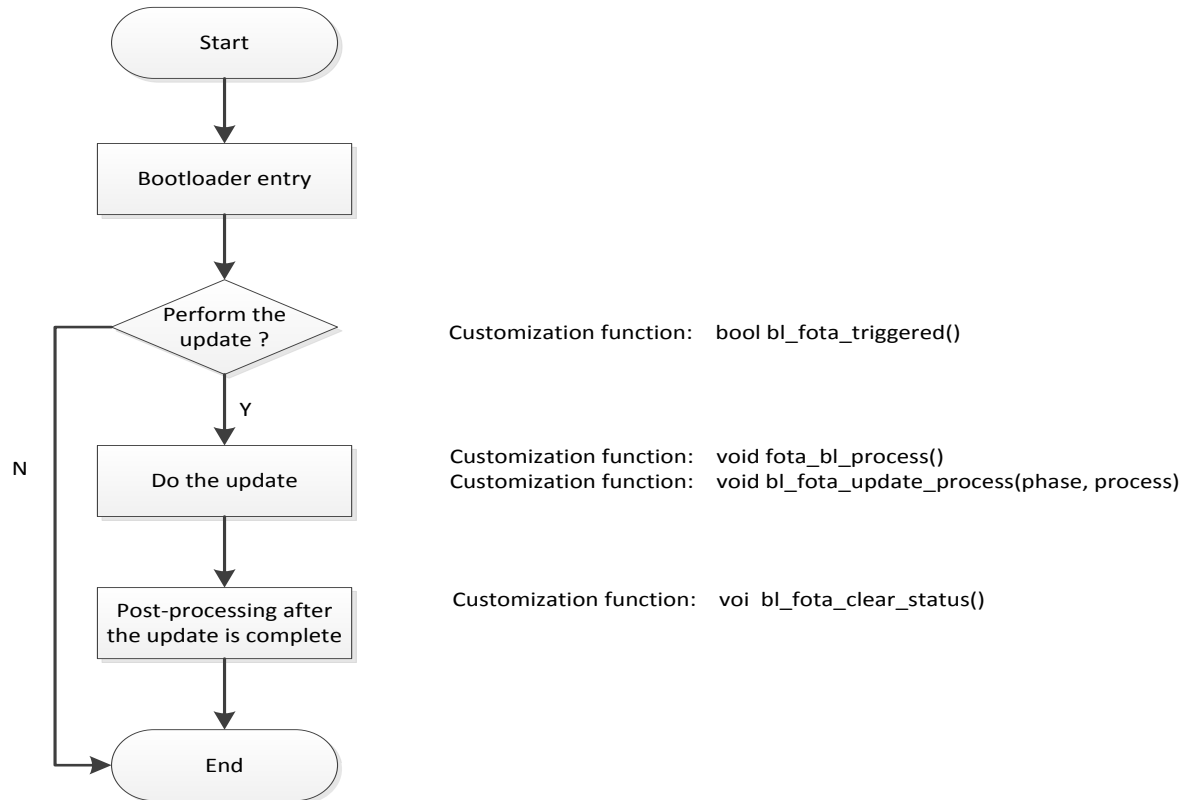


Figure 14. Update Agent's general flow and customization

Apply the customizations in `custom_fota.c` source file located under `/project/mt2523_hdk/apps/bootloader/src/`. Implement the functions as follows.

```

bool bl_fota_trigger()
{
    //TODO: check if perform update action or not in this function
    //e.g. check certain retention flgh for trigger update, check
    maker for
    //power loss or unexpected failure case
}

int32_t bl_fota_update_progress(uint32_t phase, kal_uint32 progress)
{
    //TODO: indicate update progress in this function.
    // e.g. display update progress by TRACE, or call graphic API to
    show
    //percent number in LCD.
}

void bl_fota_process(void)
{
    // TODO: add "do update" process in this function.
    //user can call HAL flash SDK APIs to access flash operation.
}
  
```

Change the bootloader makefile located at `/project/mt2523_hdk/apps/bootloader/GCC/Makefile`, as follows.

```

Ifeg $(FOTA_CUST), 1)
BOOTLOADER_FILES += $(APP_PATH)/src/custom_fota.c
else
  
```

```
BOOTLOADER_FILES += $(BOOTLOADER_SRC)/core/bl_funet.c
endif
```

If the internal option FOTA_CUST is set to 1 the bootloader will include custom_fota.c and build custom defined **Update Agent**, otherwise the default **Update Agent** is built using bl_funet.c.

3.2. Update through Wi-Fi

3.2.1. Connect to network through Wi-Fi

The HTTP/TFTP server provides the update file to the HTTP/TFTP client located on the MT76x7. The HTTP/TFTP client must connect to a router first and then update file can be transferred from the server to the client.

To connect the HDK to the router automatically after boot up, store the router SSID and password on the flash memory of HDK. The parameters could be passed to the flash through a command line from any console.

```
wifi config set ssid 0 bingo
wifi config set psk 0 12345678
wifi config set opmode 1
wifi config set reload
```

The command starting with “wifi config” sets the designated parameter similar to passing a parameter to a function call. OpMode defines the operation mode, if it's set to 1, device automatically connects to a router according to the parameters set after the reboot. HDK should run in Wi-Fi station mode (STA).

Run the following command to verify that the data is successfully written to the flash.

```
wifi config get ssid 0
wifi config get psk 0
wifi config get opmode
```

3.2.2. FOTA CLI command

The main interface to interact with MT76x7 is through CLI commands. There are five commands starting with “fota” to mark the FOTA commands, followed by the operation that can take one or more parameters, as shown in Table 7.

Table 7. MT76x7 FOTA CLI command

Operation	Format	Example
Read bytes of data	fota r <addr> <len>	fota r 0x1f9000 32
Write a character to the flash	fota w <addr> <len> <char>	fota w 0x1f9000 4 a
Download the update file by TFTP client	fota dl <ip> <file> [w]	fota dl 192.168.0.20 mt7687_iot_sdk.bin fota dl 192.168.0.25 mt7687_iot_sdk.bin w
Download the update file by HTTP client	fota httpdl <url>	fota httpdl http://192.168.0.199 /image.bin
Erase a memory block	fota e <addr>	fota e 0x1f9000
Trigger an update flow	fota go	fota go

Pay attention to the download command, the “w” means download and write the file to the flash, without the “w” it means only to download.

3.2.3. Download workflow

To download the FOTA update, use terminal emulation software, such as [PuTTY](#), [secureCRT](#) or HyperTerminal on your PC. The tool used in this section is [PuTTY](#). Before running the tool, identify the device port on your PC.

- 1) Open Windows Control Panel then click **System** and:
 - On Windows 7 and 8, click **Device Manager**.
 - On Windows XP, click the **Hardware** tab and then **Device Manager**.
- 2) In Device Manager, navigate to **Ports (COM & LPT)** (see Figure 13).
- 3) Connect the HDK to your computer.

- 4) A new COM device should appear under **Ports (COM & LPT)** in Device Manager, as shown in Figure 13. Note the COMx port number of the **USB Serial Port**, this information is needed to complete configuration of PuTTY.

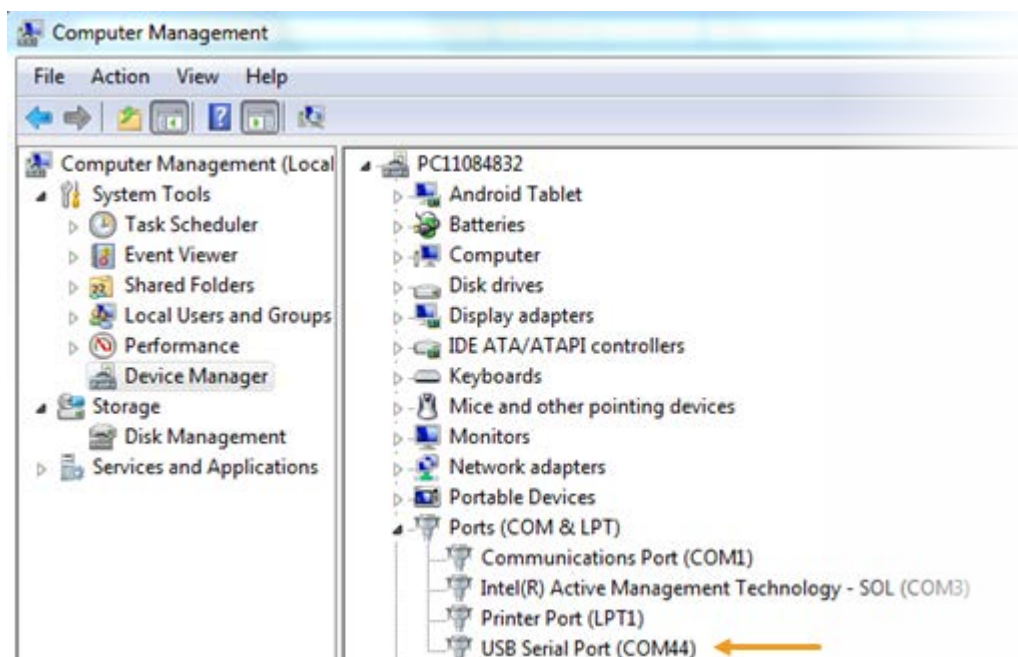


Figure 15. HDK in Device Manager

To configure PuTTY:

- 1) Launch PuTTY.
- 2) Provide the COM Port of the device and set the **Speed** to 115200 under **Specify the destination you want to connect to**.
- 3) Set the **Connection type** to **Serial**, as shown in Figure 16.
- 4) Click **Open** to run the PuTTY.

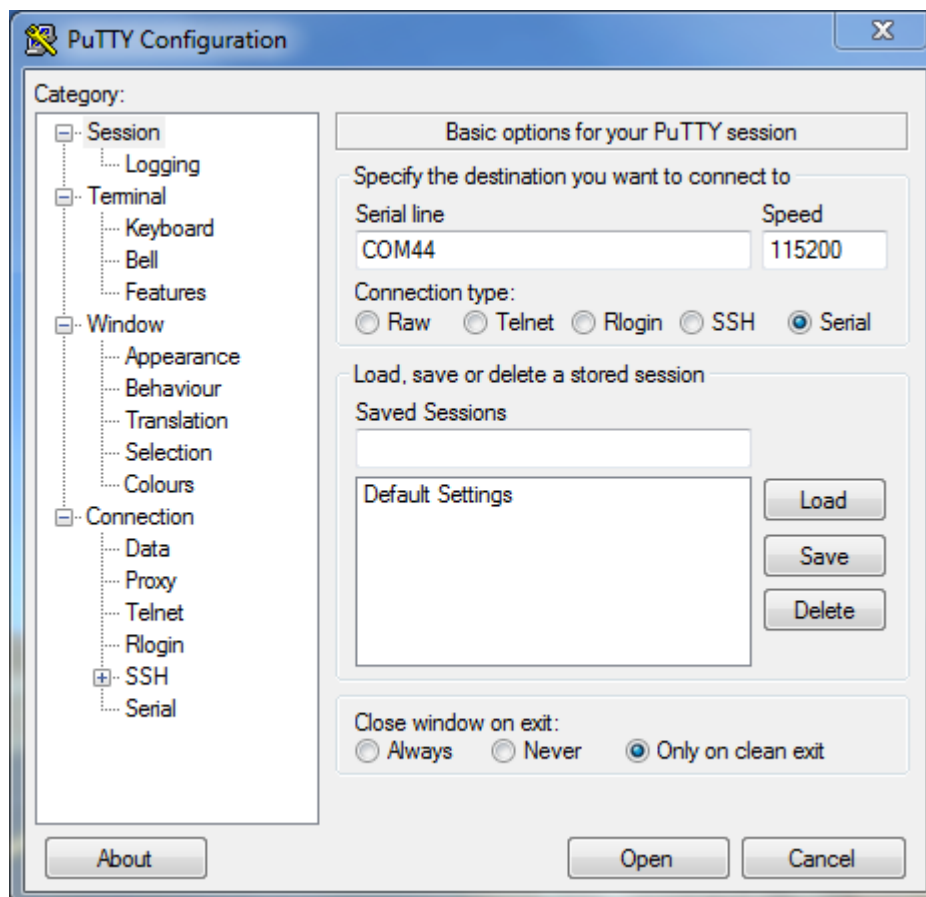
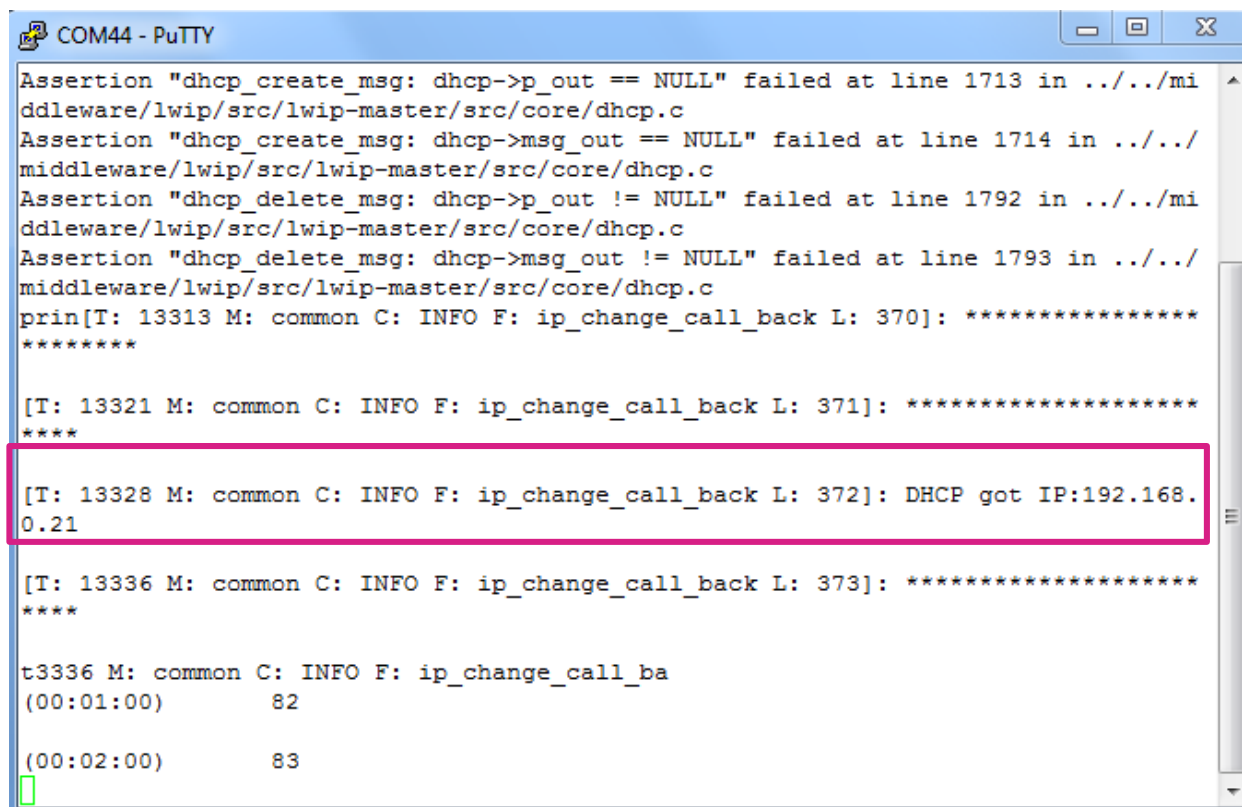


Figure 16. PuTTY configuration panel

If the parameters are set correctly as in section, then the HDK will successfully connect to the router and a random LAN IP address will be allocated to the HDK similar to “DHCP got IP: 192.168.0.21”, as shown in Figure 17.



```
COM44 - PuTTY
Assertion "dhcp_create_msg: dhcp->p_out == NULL" failed at line 1713 in ../../middleware/lwip/src/lwip-master/src/core/dhcp.c
Assertion "dhcp_create_msg: dhcp->msg_out == NULL" failed at line 1714 in ../../middleware/lwip/src/lwip-master/src/core/dhcp.c
Assertion "dhcp_delete_msg: dhcp->p_out != NULL" failed at line 1792 in ../../middleware/lwip/src/lwip-master/src/core/dhcp.c
Assertion "dhcp_delete_msg: dhcp->msg_out != NULL" failed at line 1793 in ../../middleware/lwip/src/lwip-master/src/core/dhcp.c
prin[T: 13313 M: common C: INFO F: ip_change_call_back L: 370]: *****
*****

[T: 13321 M: common C: INFO F: ip_change_call_back L: 371]: *****
*****

[T: 13328 M: common C: INFO F: ip_change_call_back L: 372]: DHCP got IP:192.168.0.21

[T: 13336 M: common C: INFO F: ip_change_call_back L: 373]: *****
*****

t3336 M: common C: INFO F: ip_change_call_ba
(00:01:00)      82
(00:02:00)      83
```

Figure 17. HDK connected to the Wi-Fi router

If the output on your console is not similar to the one in Figure 18, it means there is something wrong with your settings about the parameters. Please verify the configuration according to section 3.2.1..

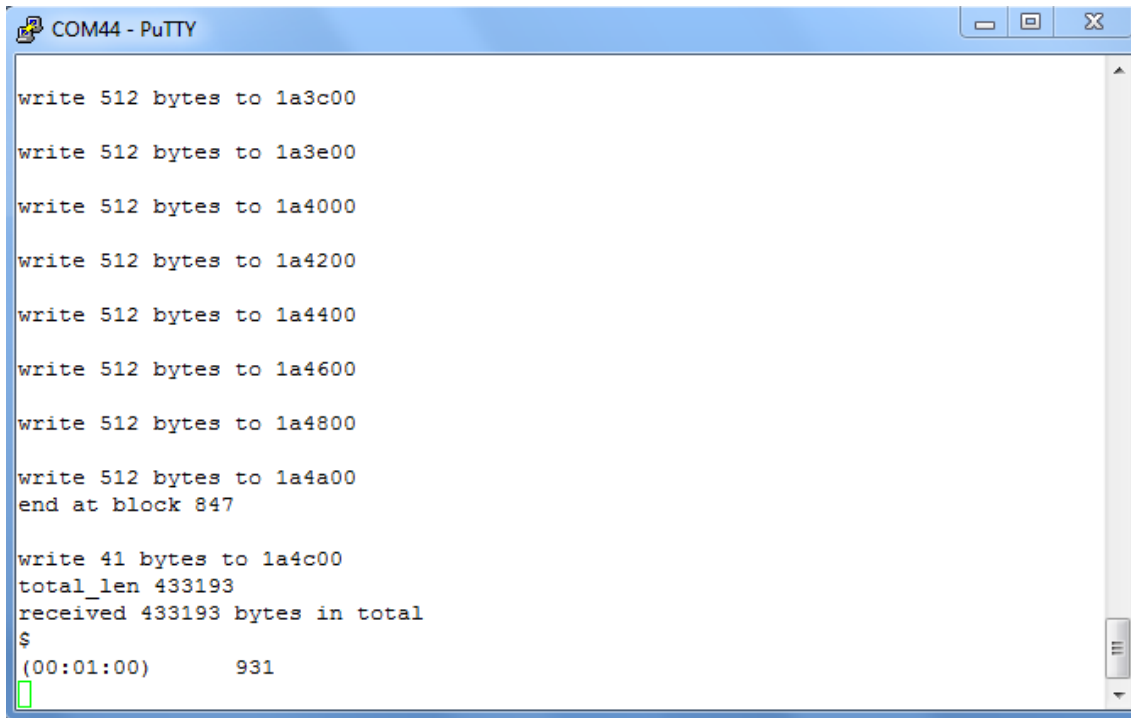
To complete the download process, setup the TFTP server.

- 1) Launch [tftp32](#). For more information on how to setup the server configuration, please refer to [tftp32](#) specifications.
- 2) The server setup is fully customizable; simply make sure the server can be accessed through the router. For example, if the TFTP server has an IP address of 218.30.13.36, run the following command to download the FOTA binary file named `mt7687_iot_sdk.bin`.

```
fota dl 218.30.13.36 mt7687_iot_sdk.bin w
```

The TFTP access port is set to Port 69 by default in [tftp32](#).

After the download is triggered, the data packet is transferred to HDK and stored on the flash. The total received file size will be printed out on the console once the transfer and save is complete. You'll also be notified if any network or flash access errors occurred. The error information is printed out to help the user with debugging.



```
COM44 - PuTTY

write 512 bytes to 1a3c00

write 512 bytes to 1a3e00

write 512 bytes to 1a4000

write 512 bytes to 1a4200

write 512 bytes to 1a4400

write 512 bytes to 1a4600

write 512 bytes to 1a4800

write 512 bytes to 1a4a00
end at block 847

write 41 bytes to 1a4c00
total_len 433193
received 433193 bytes in total
$
(00:01:00)      931
```

Figure 18. Update file successfully downloaded

3.2.4. Update Agent's workflow

After download is completed successfully, use reboot command in terminal emulation software on your PC or press the reset key on your board to trigger a reboot. Once the reboot starts, the bootloader checks to perform the firmware update or not. The workflow of **Update Agent** is shown in Figure 19.

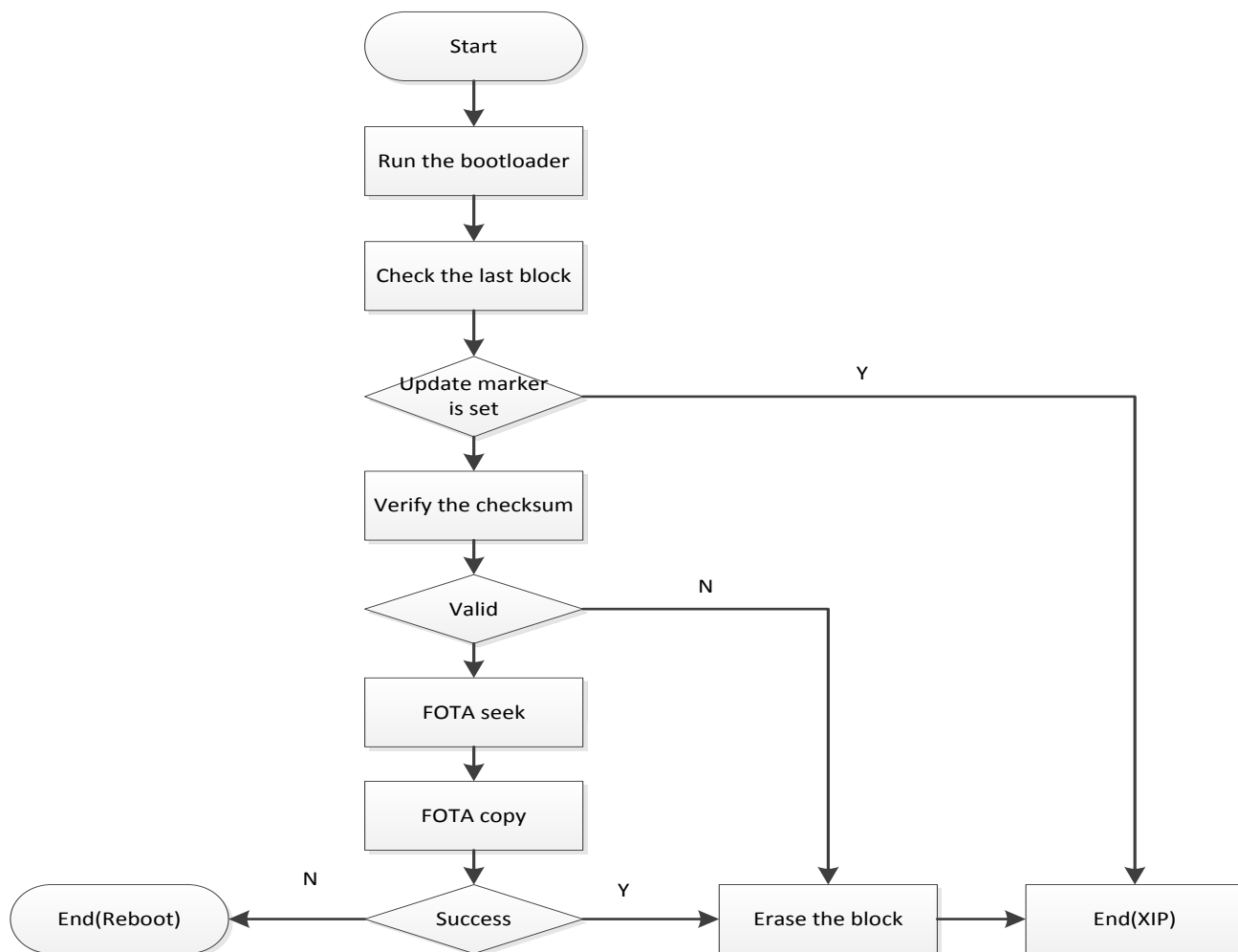


Figure 19. MT76x7 FOTA update agent workflow



Note: The SDK FOTA update on MT76x7 mainly focuses on N9 and ARM Cortex-M4 firmware, bootloader partition cannot be updated.

4. Appendix A: Acronyms and Abbreviations

The acronyms and abbreviations used in this developer's guide are listed in Table 8.

Table 8. Acronyms and abbreviations

Abbreviation/Term	Expansion/Definition
FOTA	Firmware over the air, a way to update firmware wireless
Full Binary	A type of FOTA to update the whole firmware.
TFTP	Trivial File Transfer Protocol
NOR flash	NOR flash is a type of non-volatile storage that does not require power to retain data.