MA35D1 OTA Update by Linux Stage

Example Code Introduction for 64/32-bit NuMicro® Family

Document Information

Application	The MA35D1 series sample code shows how to do OTA update in Linux stage.
BSP Version	Linux-5.10.x
Hardware	NuMaker-IoT-MA35D1

1. Overview

The purpose of this example code is to provide instructions for using the "**swupdate**" application to update the Linux Kernel, device tree, and root filesystem on several storage devices of MA35D1, including NAND, SPI NAND, SD card, and eMMC. The update process involves generating a pack file with the firmware updates and a header file that contains information about the storage addresses for each firmware update.

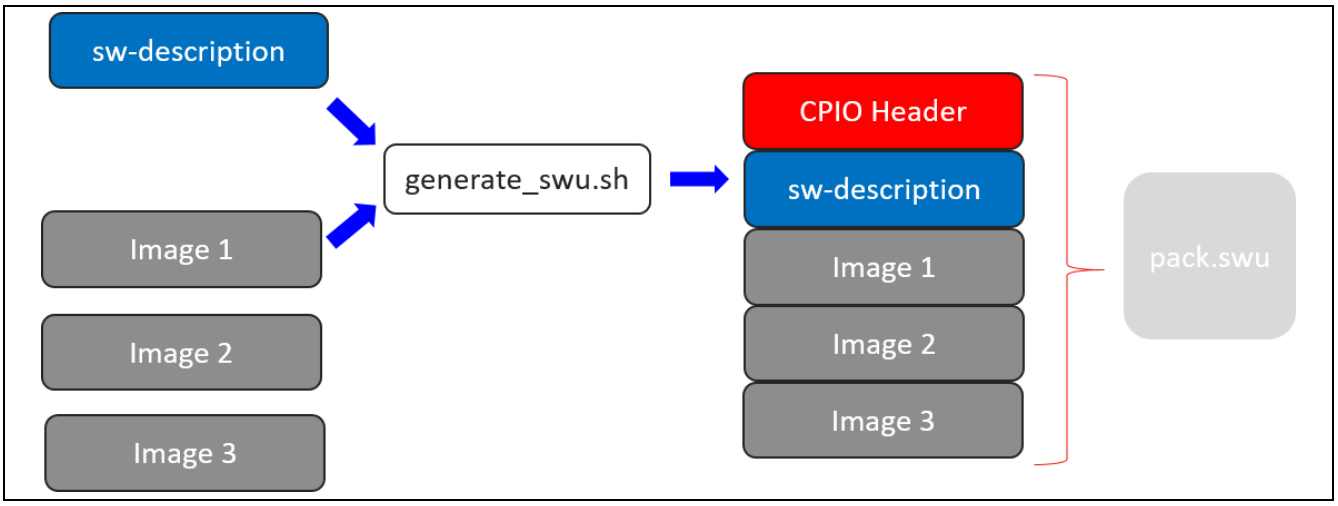

Figure 1-1 **Swupdate** Package Generation

To update the firmware, the first step is to generate the pack file, which contains the updated firmware for the Kernel, device tree, and root filesystem. it can be done using a script (**generate_swu.sh**) to combine updated files (Image1, Image2, Image3…) into a single file. Once the pack file is generated, the next step is to create a header file, which contains information about the storage addresses for each firmware update. This header file is typically named "**sw-description**" and contains the following information:

- Version of the firmware
- Hardware compatibility information
- File information for each firmware update, including the filename, path, and storage location

Once the pack.swu file has been generated on the host side, the next step is to open a web browser and connect to the swupdate webserver running on the MA35D1 board. From there, simply drag the pack.swu file onto the swupdate webpage, and swupdate will begin receiving the file and parsing sw-description to update the appropriate files to their corresponding storage locations. The streamlined process makes it easy to update firmware across multiple devices quickly and efficiently.

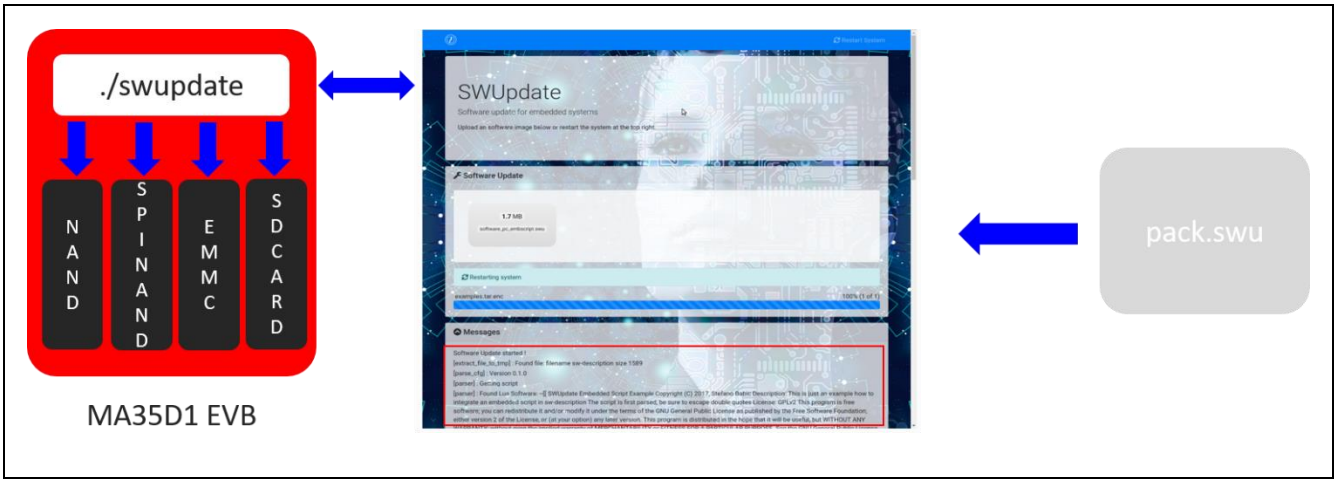

Figure 1-2 **swupdate** Webserver Updating Framework

Overall, the process of updating the firmware on multiple storage devices using the "swupdate" application involves generating a pack file with the firmware updates and creating a header file with information about the storage addresses for each update. With these files in hand, the "swupdate" application can be used to update the firmware on the target devices.

1.1 Environment Settings

For MA35D1 environment setup, please refer to the Nuvoton MA35D1 Evaluation Board Quick Start.

- **Buildroot**

To enable the required packages for MA35 support in Buildroot menuconfig, please follow these steps:

1. Open the Buildroot configuration menu by running the command make menuconfig.
2. Search for each package by selecting the corresponding option in the package selection menu and pressing the Spacebar to enable it. It is available to use the '/' key to search for packages by name.
   a. Enable **BR2_PACKAGE_UBOOT_TOOLS_FWPRINTENV** to include the U-Boot Tools fw_printenv package.
   b. Enable **BR2_PACKAGE_MTD** to include the MTD package.
   c. Enable **BR2_PACKAGE_SWUPDATE** to include the swupdate package.
   d. Enable **BR2_PACKAGE_MMC_UTILS** to include the mmc-utils package.
3. Once you have enabled all the required packages, exit the menuconfig and save the configuration.
4. Build the Buildroot system using the updated configuration by running make.

After compiling, program the image to the MA35D1 evaluation board by **NuWriter**.

1.2 Creating Firmware Update Packages with sw-description for Different Storage Types

To create a firmware update package for swupdate, the sw-description file is crucial as it instructs swupdate on the location to update the files. The following section will introduce the process of generating the sw-description file, taking into consideration of the specific storage devices (NAND/SPI NAND/SDcard/eMMC). Furthermore, an example is provided for packaging the sw-description file together with the firmware image, enabling the creation of a comprehensive firmware update package.

1.2.1 MA35D1 Storages Partition Layout

1.2.1.1 NAND

The NAND partition layout for MA35D1 can be found in ma35d1.dtsi

```
nand:nand@401A0000 {
    …
    partitions {
     compatible = "fixed-partitions";
     #address-cells = <1>;
     #size-cells = <1>;

     uboot@0 {
              label = "nand-uboot";
          reg = <0x0000000 0x300000>;
          read-only;
     };
     uboot-env@300000 {
          label = "nand-uboot-env";
          reg = <0x300000 0xC0000>;
     };
     device-tree@3c0000 {
          label = "nand-device-tree";
          reg = <0x3C0000 0x40000>;
     };
     kernel@400000 {
          label = "nand-kernel";
          reg = <0x400000 0x1800000>;
     };
     rootfs@1c00000 {
          label = "nand-rootfs";
          reg = <0x1C00000 0x6400000>;
     };
    };
};
```

To update the kernel in the sw-description file for the MA35D1 NAND partition, please follow the instructions provided below:

```
software =
{
        version = "0.1.0";
      description = "Linux kernel update in NAND";
        images: (
         {
```

```
                    version ="5.10";
          name = "nand-kernel";
          filename = "Image";
          device = "/dev/mtdblock3";
          type = "raw";
          }
         );
}
```

In the example, "Image" is the filename of the kernel image file, "/dev/mtdblock3" is the mtdblock to the actual kernel image file.

1.2.1.2 SPI NAND

The SPI NAND partition layout for MA35D1 can be found in [ma35d1.dtsi]

```
qspi0: spi@40680000 {
    …
    flash: flash@0 {
     compatible = "spi-nand";
     reg = <0 0x0 0x2000000>;
     spi-ax-frequency = <50000000>;
     #address-cells = <1>;
     #size-cells = <1>;

     uboot@0 {
          label = "spinand-uboot";
          reg = <0x0 0x300000>;
     };
     uboot-env@300000 {
          label = "spinand-uboot-env";
          reg = <0x300000 0xc0000>;
     };
     device-tree@3c0000 {
          label = "spinand-device-tree";
          reg = <0x3c0000 0x40000>;
     };
     kernel@400000 {
          label = "spinand-kernel";
          reg = <0x400000 0x1800000>;
     };
     rootfs@1c00000 {
          label = "spinand-rootfs";
          reg = <0x1c00000 0x6400000>;
     };
    };
…
};
```

To update the kernel in the sw-description file for the MA35D1 SPI NAND partition, please follow the instructions provided below:

```
software =
{
version = "0.1.0";
description = " Linux kernel update in SPI NAND";
       images: (
         {
                  version ="5.10";
                  name = "nand-kernel";
                  filename = "Image";
                  device = "/dev/mtdblock8";
                  type = "raw";
          }
         );
}
```

In the example, "Image" is the filename of the kernel image file, "/dev/mtdblock8" is the mtdblock to the actual kernel image file.

Please note that the MTD (Memory Technology Device) block numbers may vary based on the device tree configuration. In the case of NuMaker-IoT-MA35D1, which has both NAND and SPI NAND connected, the MTD block numbering is as follows:

For NAND: The MTD blocks are numbered as mtdblock0 to mtdblock4.

For SPI NAND: The MTD blocks are numbered as mtdblock5 to mtdblock9.

It's important to refer to the specific device tree configuration or the target device to determine the correct MTD block numbering for each storage type. Above information will be used in the sw-description file to specify the correct partition for firmware updates.

1.2.1.3 eMMC/SDcard

The eMMC/SDcard partition layout for MA35D1 can be found in pack-sdcard.json

```
{
  "image": [
    …
    {
      "offset": "0x2c0000",
      "file": "Image.dtb",
      "type": 0
    },
    {
      "offset": "0x300000",
      "file": "Image",
      "type": 0
    },
    {
      "offset": "37748736",
      "file": "rootfs.ext4",
      "type": 0
    }
  ]
}
```

To update the kernel in the sw-description file for the MA35D1 eMMC/SDcard partition, please follow the instructions provided below:

```
software =
{
        version = "0.1.0";
      description = " Linux kernel update in eMMC/SDcard";
        images: (
         {
              filename = "Image";
              device = "/dev/mmcblk0";
              offset = "0x300000";

         }
        );
}
```

1.2.2 Firmware Images Packaging

To pack the sw-description file and other firmware update files together into a single package, follow these steps:

1. Prepare the necessary files: Ensure the sw-description file, along with other firmware files such as the Kernel image, device tree binary, and root filesystem image, are placed in the same folder.

2. Create a packaging script: Develop a script that will handle the packaging process. The script will gather the files from the specified folder and package them into a single file. Here is an example:

```
#!/bin/sh
CONTAINER_VER="1.0"
PRODUCT_NAME="pack"
FILES="sw-description Image"
for i in $FILES;do
        echo $i;done | cpio -ov -H crc >  ${PRODUCT_NAME}_${CONTAINER_VER}.swu
```

The script will pack sw-description and image into a cpio format file named as pack.swu.

3. Run the packaging script: Execute the packaging script to create the pack.swu file, which contains the compressed firmware update files and the sw-description file.

By putting the sw-description file and other firmware update files into the same folder and running the packaging script, the pack.swu file will be generated. The file serves as a comprehensive firmware update package that can be easily transferred and applied using the swupdate application.

2. Code Description

2.1 Updating Files/User Applications of Rootfs

```
/* sw-description for updating files/upser applicaitons */

/*denotes the beginning of the software section of the sw-description file*/
software =
{
/*specifies the version number of the software to be updated*/
        version = "0.1.0";
/*specifies the hardware versions that are compatible with this software version*/
        hardware-compatibility = [ "1.0", "1.2", "1.3" ];

        files: (
        /* single copy mode */
         {
/*the name of the file/user application to be updated*/
                filename = "target.txt";
/*specifies the path where the file should be installed on the target system*/
                path = "/home/target.txt“
         },
        );
}
```

2.2 Updating Kernel and Device Tree

2.2.1 NAND

```
/* sw-description for updating Kernel and device tree in NAND */

/* Denotes the beginning of the software section of the sw-description file */
software =
{
    /* Specifies the version number of the software to be updated */
    version = "0.1.0";
    /* Provides a brief description of the software update */
    description = "Linux kernel and dtb update for MA35 NAND";
    /* Specifies the hardware versions that are compatible with this software version */
    hardware-compatibility = [ "1.0", "1.2", "1.3" ];
    /* Begins the block for specifying the firmware update images */
    images: (
        {
            /* Specifies the version of the image. */
            version ="5.10";
            /* Provides a name or description for the image */
            name = "nand-kernel device tree";
            /* Specifies the filename of the image */
            filename = "Image.dtb";
                /* Specifies the device or storage location where the image will be
updated */
            device = "/dev/mtdblock2";
            /* Defines the type of the image file */
            type = "raw";
        },
        {
            /* Specifies the version of the image. */
            version ="5.10";
            /* Provides a name or description for the image */
            name = "nand-kernel";
            /* Specifies the filename of the image */
```

```
            filename = "Image";
            /* Specifies the device or storage location where the image will be updated
*/
            device = "/dev/mtdblock3";
            /* Defines the type of the image file */
            type = "raw";
        }
    );
}
```

2.2.2 SPI NAND

```
/* sw-description for updating Kernel and device tree in SPI NAND */

/* Denotes the beginning of the software section of the sw-description file */
software =
{
    /* Specifies the version number of the software to be updated */
    version = "0.1.0";
    /* Provides a brief description of the software update */
    description = "Linux kernel and dtb update for MA35 SPI NAND";
    /* Specifies the hardware versions that are compatible with this software version */
    hardware-compatibility = [ "1.0", "1.2", "1.3" ];
    /* Begins the block for specifying the firmware update images */
    images: (
        {
            /* Specifies the version of the image. */
            version ="5.10";
            /* Provides a name or description for the image */
            name = "spinand-kernel device tree";
            /* Specifies the filename of the image */
            filename = "Image.dtb";
            /* Specifies the device or storage location where the image will be updated
*/
            device = "/dev/mtdblock7";
            /* Defines the type of the image file */
            type = "raw";
        },
        {
            /* Specifies the version of the image. */
            version ="5.10";
            /* Provides a name or description for the image */
            name = "spinand-kernel";
            /* Specifies the filename of the image */
            filename = "Image";
            /* Specifies the device or storage location where the image will be updated
*/
            device = "/dev/mtdblock8";
            /* Defines the type of the image file */
            type = "raw";
        }
    );
}
```

2.2.3 eMMC/SDcard

```
/* sw-description for updating Kernel and device tree in eMMC/SDcard */

/* Denotes the beginning of the software section of the sw-description file */
software =
{
    /* Specifies the version number of the software to be updated */
    version = "0.1.0";
    /* Provides a brief description of the software update */
```

```
    description = "Linux kernel and dtb update for MA35 eMMC/SDcard";
    /* Specifies the hardware versions that are compatible with this software version */
    hardware-compatibility = [ "1.0", "1.2", "1.3" ];
    /* Begins the block for specifying the firmware update images */
    images: (
        {
            /* Specifies the filename of the image */
                filename = "Image.dtb";
            /* Specifies the device or storage location where the image will be updated
*/
                device = "/dev/mmcblk0";
            /* Specifies the offset at which the image will be written in the storage
device */
                offset = "0x2c0000";
        },
        {
                /* Specifies the filename of the image */
                filename = "Image";
                /* Specifies the device or storage location where the image will be
updated */
                device = "/dev/mmcblk0";
                /* Specifies the offset at which the image will be written in the storage
device */
                offset = "0x300000"
        }
    );
}
```

2.3 Script for Firmware Packaging

A script will handle the packaging process. The script will gather the files from the specified folder and package them into a single file.

```
# generate_swu.sh
# Indicating that the script should be executed using the shell interpreter specified
#!/bin/sh
# Assigns the value "1.0" to the variable CONTAINER_VER, which represents the version of
the container being created
CONTAINER_VER="1.0"
# Assigns the value "pack" to the variable PRODUCT_NAME, which represents the name of the
product or container being created
PRODUCT_NAME="pack"
# Assigns the value "sw-description Image" to the variable FILES, which contains a space-
separated list of filenames to be included in the container
FILES="sw-description Image Image.dtb"
# Starts a loop that iterates over each element in the FILES variable
# | cpio -ov -H crc: Pipes the output of the loop to the cpio command, which creates a
cpio archive. The -ov options specify that the output should be in verbose mode and
overwrite existing files. The -H crc option specifies the header format to use, in this
case, CRC.
for i in $FILES;do
        echo $i;done | cpio -ov -H crc >  ${PRODUCT_NAME}_${CONTAINER_VER}.swu
```

3. Software and Hardware Requirements

3.1 Software Requirements

- BSP version
  - Linux-5.10.x

3.2 Hardware Requirements

- Circuit components
  - NuMaker-IoT-MA35D1

4. Directory Information

The directory structure is shown below.

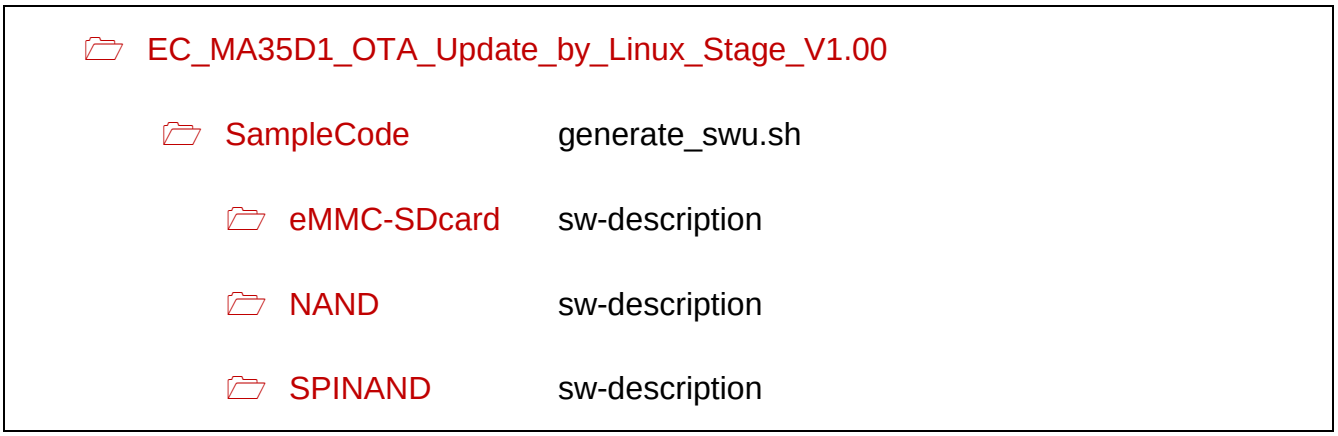

Figure 4-1 Directory Structure

5. Example Code Execution

Packaging images on Host side:

To package images on the host side, ensure the sw-description Image, Image.dtb, and generate.sh files in the same folder. It ensures that all the required files are accessible for the packaging process on the PC host:

```
      @ubuntu:              swupdate       $ ls -l
total 10044
-rwxrw-rw- 1                    196 May  9 14:18 generate_swu.sh
-rwxrw-rw- 1               10232320 Oct 31  2022 Image
-rwxrw-rw- 1                  37525 Oct 31  2022 Image.dtb
-rwxrw-rw- 1                    607 Oct 31  2022 sw-description
```

Figure 5-1 Image Folder(1)

By executing the generate_swu.sh script, the final image package will be created in the current folder. The script is responsible for performing the necessary steps to generate the package, which may include file manipulation, compression, or other operations. After the script is executed successfully, the resulting image package will be found in the current directory.

```
ubuntu:~$ ./generate_swu.sh
```

```
      @ubuntu:              swupdate/      $ ./generate_swu.sh
sw-description
Image
Image.dtb
20061 blocks
      @ubuntu:              swupdate/      $ ls -l
total 20076
-rwxrw-rw- 1                    196 May  9 14:18 generate_swu.sh
-rwxrw-rw- 1               10232320 Oct 31  2022 Image
-rwxrw-rw- 1                  37525 Oct 31  2022 Image.dtb
-rw-rw-r-- 1               10271232 May  9 14:22 Linux_kernel_dtb_update_1.0.swu
-rwxrw-rw- 1                    607 Oct 31  2022 sw-description
```

Figure 5-2 Image Folder(2)

On the MA35D1 device, obtain an IP address from the router by configuring the device to use DHCP (Dynamic Host Configuration Protocol).

```
MA35D1# ifconfig eth0 up
MA35D1# udhcpc -i eth0
udhcpc: started, v1.33.1
udhcpc: sending discover
udhcpc: sending discover
udhcpc: sending select for 192.168.0.175
udhcpc: lease of 192.168.0.175 obtained, lease time 7200
deleting routers
adding dns 192.168.0.1
```

The swupdate process will run in the background after system initialization. To list the running swupdate process, use the ps command which displays information about active processes on the system. Here's an example of how to list the swupdate process:

```
MA35D1# ps aux | grep swupdate
  212 root     /usr/bin/swupdate -v -w -r /var/www/swupdate -p 8080
  219 root     /usr/bin/swupdate -v -w -r /var/www/swupdate -p 8080
  258 root     grep swupdate
```

To access the swupdate web server running on the MA35D1 device from your PC, open a web browser and enter the IP address of the MA35D1 device followed by ":8080". Here are the steps to follow:

1. Ensure that both the PC and the MA35D1 device are connected to the same network.
2. On your PC, open a web browser (such as Edge, Chrome, Firefox…).
3. In the address bar of the web browser, type the IP address of the MA35D1 device followed by ":8080". For example, if the IP address of the MA35D1 device is 192.168.0.175, please enter "192.168.0.175:8080".
4. The web browser will establish a connection to the swupdate web server running on the MA35D1 device.

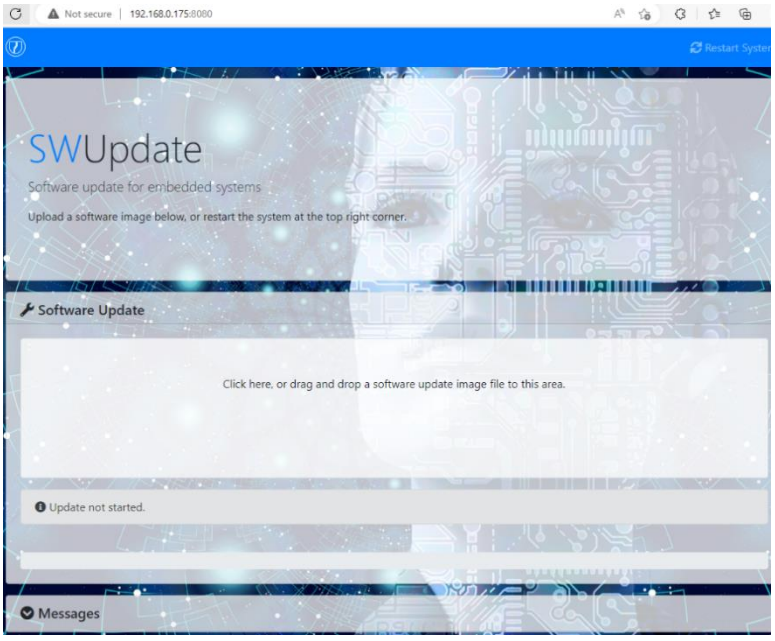

Figure 5-3 Swupdate Webpage

After accessing the swupdate web server, there are actions such as uploading the pack.swu firmware package for updating, initiating the update process, and monitoring the progress or status of the update.

Transferring the pack.swu firmware package to the MA35D1 device can be done by dragging and dropping the pack.swu file onto the swupdate web page in a web browser.

The following are the example screenshots:

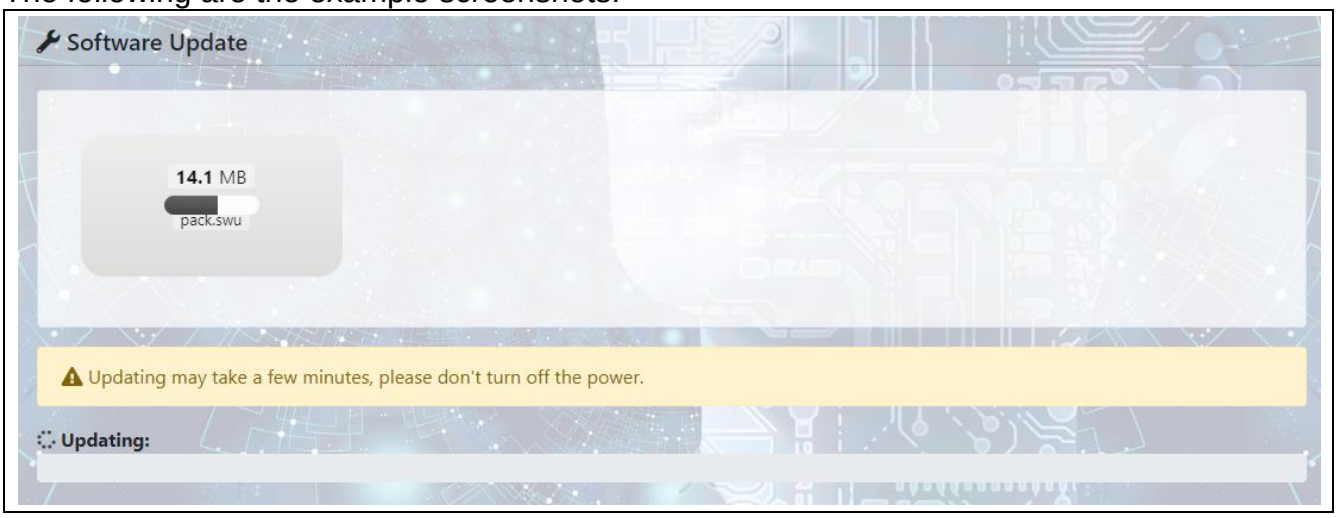

Figure 5-4 Webpage Updating in Progress

Figure 5-5 Webpage Updating Completed

Figure 5-6 Webpage Updating Message Log

6. Revision History

Date	Revision	Description
2023.05.09	1.00	Initial version.

Important Notice

Nuvoton Products are neither intended nor warranted for usage in systems or equipment, any malfunction or failure of which may cause loss of human life, bodily injury or severe property damage. Such applications are deemed, "Insecure Usage".

Insecure usage includes, but is not limited to: equipment for surgical implementation, atomic energy control instruments, airplane or spaceship instruments, the control or operation of dynamic, brake or safety systems designed for vehicular use, traffic signal instruments, all types of safety devices, and other applications intended to support or sustain life.

All Insecure Usage shall be made at customer's risk, and in the event that third parties lay claims to Nuvoton as a result of customer's Insecure Usage, customer shall indemnify the damages and liabilities thus incurred by Nuvoton.