MA35D1 OTA Update by Uboot Stage

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

Document Information

Application	This example code shows how to do MA35D1 OTA update in uboot stage.
BSP Version	Linux-5.10.x
Hardware	NuMaker-HMI-MA35D1-S1

1. Overview

This sample code demonstrates how to implement over-the-air (OTA) firmware updates for the kernel, device tree, and root filesystem in uboot. Nuvoton provides a simple update method by using MTD (Memory Technology Device) and MMC (Multi Media Card) command to overwrite the corresponding hardware storage device to achieve the purpose of updating.

This application demonstrates the OTA update of MA35D1 series evaluation boards, supporting SD card, eMMC, NAND, SPI-NAND Flash. Users can design and update the software package of the corresponding device according to their own hardware configuration. For example, the MTD method overwrites the corresponding position of the hardware storage device in NAND/SPI-NAND Flash storage, allowing for partial updates of the Linux kernel or device-tree functions. Similarly, it supports partial updates of hardware devices such as SD card/eMMC. The memory partitioning is shown in Figure 1-1.

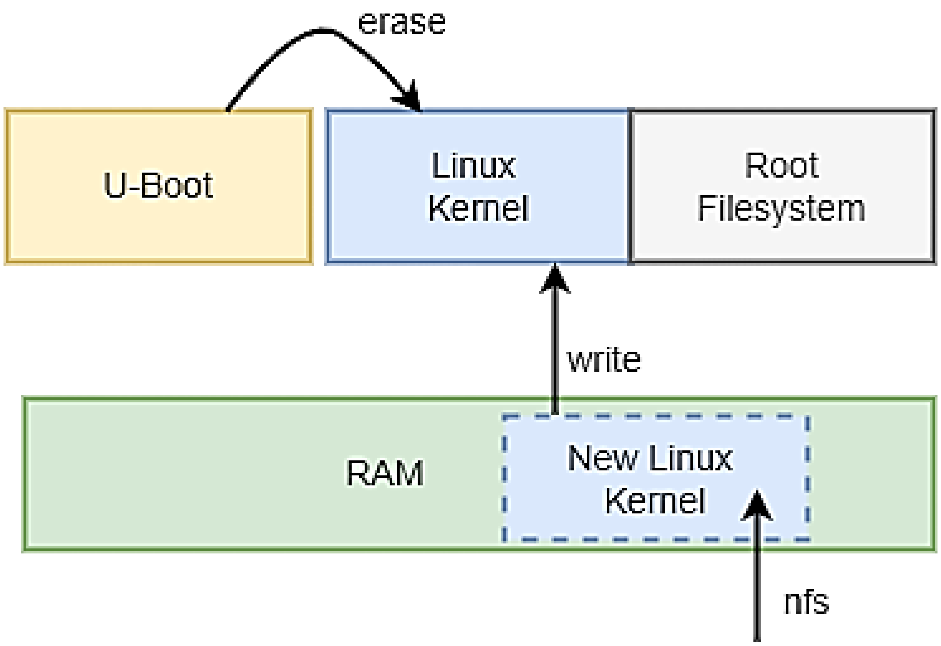


Figure 1-1 Memory Partition

The flowchart of OTA update steps is shown in Figure 1-2. The pre-updated files (linux kernel, device-tree, and root filesystem) are downloaded to the embedded device. To avoid update failure, please ensure the normal operation of the update files. Executing the OTA command in the command window will clear the old file package and overwrite the new file package to the original file location. The OTA command provides the function of erasing the original data group and writing the file in the RAM storage(0x80800000) to the original data group, and then restarting to exit u-boot space.

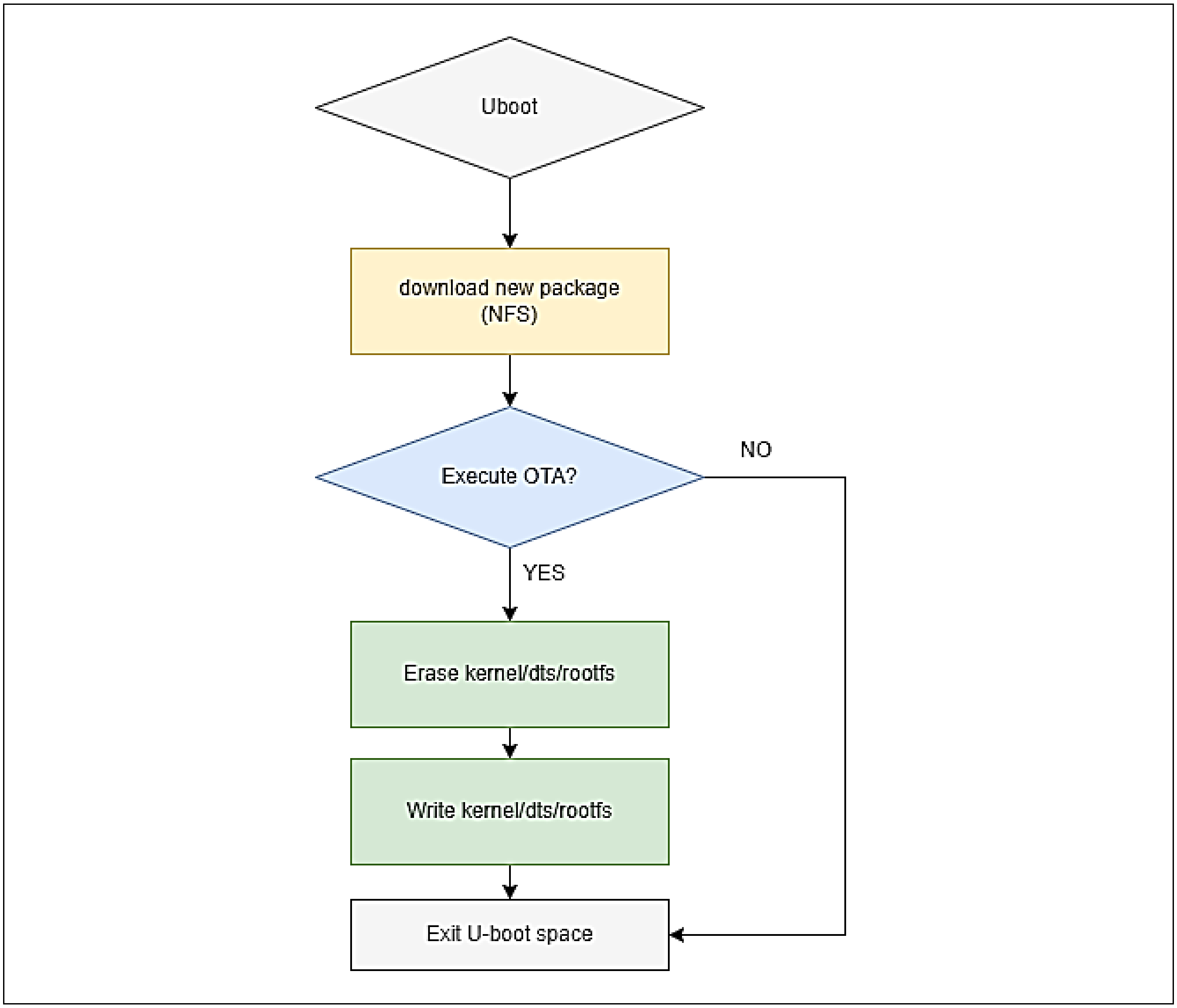


Figure 1-2 OTA Flowchart

1.1 Environment Settings

The OTA command ma35d1_ota_update.c can be downloaded from the Nuvoton MA35D1 series official website. That can be added to the uboot command and compiled it into U_BOOT_CMD as a command for use.

This example code will demonstrate the OTA update for device-tree, kernel, and root filesystem in uboot. For MA35D1 environment setup, please refer to the Nuvoton MA35D1 Evaluation Board Quick Start.

Before the update operation, the first step is to prepare the pre-updated files (such as kernel, device-tree, or root filesystem) and transfer them to the MA35D1 device. Users can choose the method of transfer, such as TFTP or NFS, to download the pre-updated files to the MA35D1 device.

This application uses network file system (NFS) to transfer the pre-updated kernel to the MA35D1 device evaluation board, and then boot into the uboot space to overwrite the kernel and restart the boot program to load the new kernel.

The following are the operating instructions for buildroot and Yocto Project:

- **Buildroot**

Move ma35d1_ota_update.c to MA35D1 Buildroot uboot-custom/cmd folder:

```
Buildroot u-boot path: ~MA35D1_Buildroot/output/build/uboot-custom/cmd
```

To modify cmd/Makefile and add the following directives:

```
obj-y += ma35d1_ota_update.o
```

Save and exit. Next, install NFS (network file system). The NFS is an RPC service that can share files. This example code uses NFS to download the pre-updated software package on the MA35D1 evaluation board. If you've used other methods to mount the pre-updated software package, you may skip this step.

Enable NFS function and MTD function in u-boot-menuconfig, which is shown below:

```
~/MA35D1_Buildroot$ make uboot-menuconfig
.
Command line interface --->
      [*] Network commands --->
      -*- dhcp
     ....
     [*] nfs

      Device access commands--->
      [*] mtd
```

Save and exit, and execute the make command to update the image:

```
~/MA35D1_Buildroot$ make uboot-rebuild arm-trusted-firmware-rebuild
~/MA35D1_Buildroot$ make
```

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

- **Yocto project**

Move ma35d1_ota_update.c to MA35D1 yocto u-boot/git/cmd folder:

```
yocto u-boot path:  ~yocto/build/tmp-glibc/work/numaker_som_ma35d16a81-poky-linux/u-boot-
ma35d1/2020.07-r0/git/cmd
```

To modify /git/cmd/Makefile and add the following directives:

```
obj-y += ma35d1_ota_update.o
```

Save and exit. Next, install NFS (network file system). The NFS is an RPC service that can share files. This example code uses NFS to download the pre-updated software package on the MA35D1 evaluation board. If you've used other methods to mount the pre-updated software package, you may skip this step.

MA35D1 supports SD card/NAND Flash/SPI-NAND Flash, etc. The yocto project provides the corresponding u-boot configuration files. Users can choose the corresponding configuration file according to their startup method:

- ma35d1_nand_defconfig
- ma35d1_sdcard0_defconfig
- ma35d1_spinand_defconfig

Enter the u-boot file of the yocto project, and select the corresponding u-boot folder: (take sdcard0 as an example):

```
~$ cd ~/shared/yocto/build/tmp-glibc/work/numaker_som_ma35d16a81-poky-linux/u-boot-
ma35d1/2020.07-r0/build/

~$ cd ma35d1_sdcard0_defconfig
```

Enable NFS function and MTD function in u-boot-menuconfig, which is shown below:

```
~ ma35d1_sdcard0_defconfig $ make menuconfig
.
Command line interface --->
      [*] Network commands --->
          -*- dhcp
     ....
      [*] nfs

      Device access commands--->
      [*] mtd
```

Note: The toolchain: source /usr/local/oecore-x86_64/environment-setup-aarch64-poky-linux will be used when compiling. Please refer to the Nuvoton MA35D1 Evaluation Board Quick Start.

Save and exit, and execute the bitbake command to update the image:

```
~/shared/yocto/build$ bitbake u-boot-ma35d1 -C compile
~/shared/yocto/build$ bitbake nvt-image-qt5 -c clean
~/shared/yocto/build$ bitbake nvt-image-qt5
```

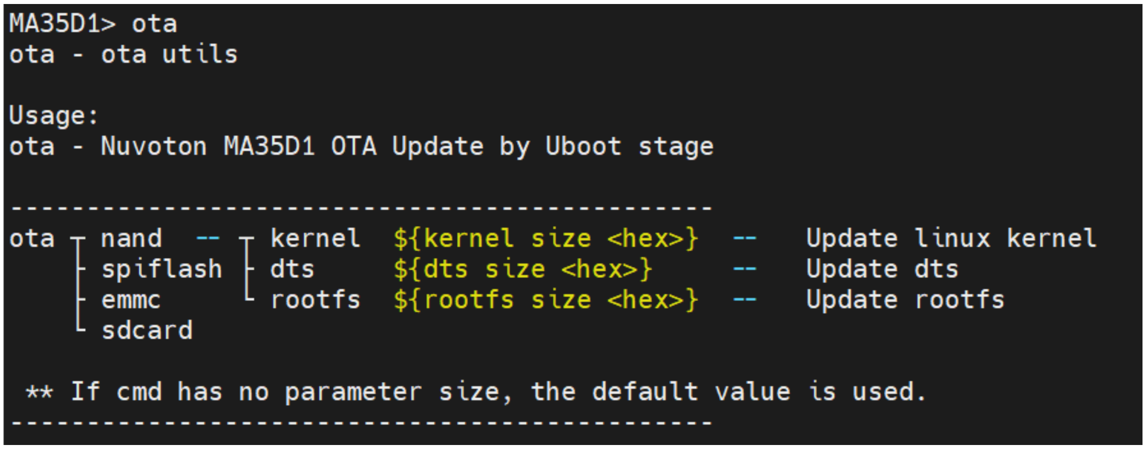
After compiling, program the image to the MA35D1 evaluation board through **NuWriter**.

1.2 OTA Command Structure

Start up the MA35D1 evaluation board. Before entering the kernel, press any key to enter the u-boot environment. Enter the following command to check whether the OTA CMS is installed successfully:

```
MA35D1> ota help
```

The OTA command structure is shown in Figure 1-3.



```
MA35D1> ota
ota - ota utils

Usage:
ota - Nuvoton MA35D1 OTA Update by Uboot stage

----------------------------------------------
ota ┬ nand    -- ┬ kernel  ${kernel size <hex>}  --   Update linux kernel
    ├ spiflash   ├ dts     ${dts size <hex>}     --   Update dts
    ├ emmc       └ rootfs  ${rootfs size <hex>}  --   Update rootfs
    └ sdcard

 ** If cmd has no parameter size, the default value is used.
----------------------------------------------
```

Figure 1-3 OTA Command Structure

This command writes to the corresponding block in RAM(0x80800000) with a fixed structure. The size is determined by the partition settings.

Command Format	Description
ota nand [part] [size]	Erase the corresponding position in NAND and write RAM (0x80800000) into the corresponding area in NAND. Part: kernel, dts, rootfs Size: Size of the updated files in hexadecimal format. Default: kernel=24M, dts=256K, rootfs=100M
ota spinand [part] [size]	Erase the corresponding position in SPI-NAND and write RAM (0x80800000) into the corresponding area in SPI-NAND. Part: kernel, dts, rootfs Size: Size of the updated files in hexadecimal format. Default: kernel=24M, dts=256K, rootfs=100M
ota sdcard [part] [size]	Erase the corresponding position in SD card and write RAM (0x80800000) into the corresponding area in SD card.

	Part: kernel, dts Size: Size of the updated files in hexadecimal format. Default: kernel=16M, dts=64K
ota emmc [part] [size]	Erase the corresponding position in eMMC and write RAM (0x80800000) into the corresponding area in eMMC. Part: kernel, dts Size: Size of the updated files in hexadecimal format. Default: kernel=16M, dts=64K

Table 1-1 OTA Command Table

1.3 Demo Result

Step 1: Download the pre-update package to the evaluation board.

This application uses network file system (NFS) to move the pre-update kernel into the MA35D1 evaluation board. **NFS service** should be installed on the PC host:

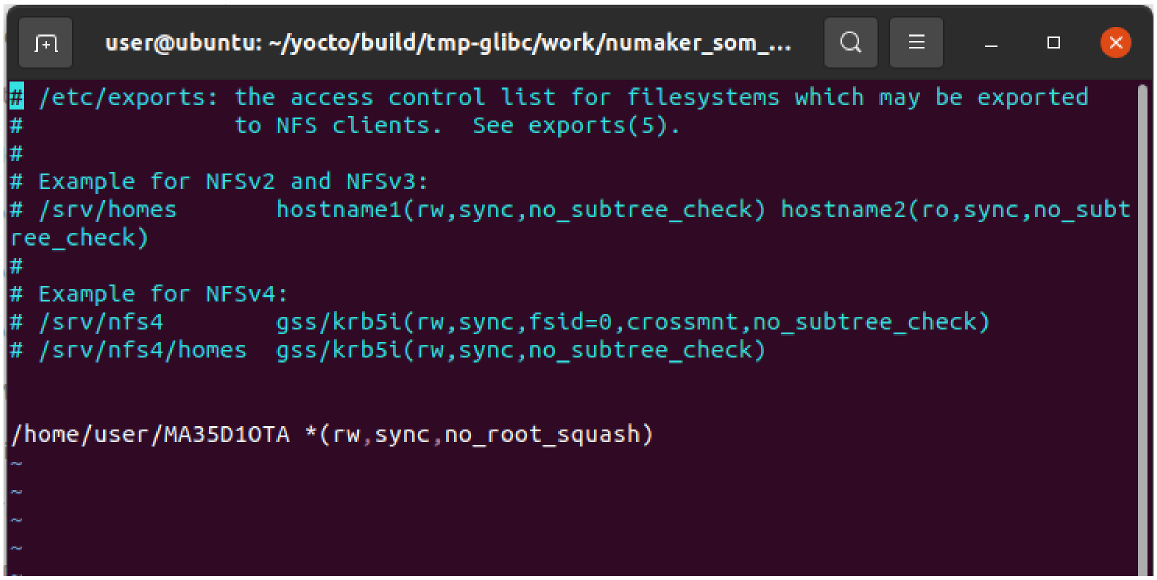
```
ubuntu:~$ sudo apt-get install nfs-kernel-server rpcbind
```

Set shared folders:

```
~$ sudo vi /etc/exports
```

Add ${NFS_FOLDER_PATH} path as the folder to share:

```
${NFS_FOLDER_PATH} *(rw,sync,no_root_squash)
```



```
# /etc/exports: the access control list for filesystems which may be exported
#               to NFS clients.  See exports(5).
#
# Example for NFSv2 and NFSv3:
# /srv/homes       hostname1(rw,sync,no_subtree_check) hostname2(ro,sync,no_subt
ree_check)
#
# Example for NFSv4:
# /srv/nfs4        gss/krb5i(rw,sync,fsid=0,crossmnt,no_subtree_check)
# /srv/nfs4/homes  gss/krb5i(rw,sync,no_subtree_check)


/home/user/MA35D1OTA *(rw,sync,no_root_squash)
~
~
~
~
```

Figure 1-4 NFS Settings

Restart the NFS service:

```
$sudo /etc/init.d/nfs-kernel-server restart
```

Before entering the kernel, press any key to enter the u-boot environment.

And get the IP address:

```
MA35D1> dhcp
```

Download the pre-update file by NFS command:

```
MA35D1>nfs [loadAddress] [[host IP addr]:[filename]
```

loadAddress: addr=0x80800000 is scratch space of MA35D1

host IP addr: PC host's network ip

filename: pre-download file path

Download the device-tree:

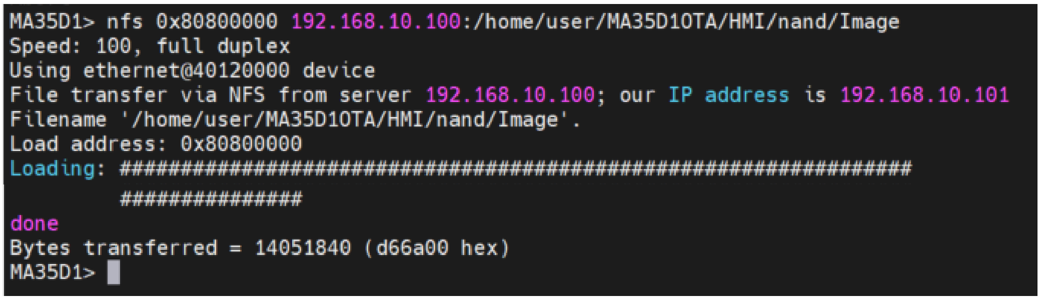
```
MA35D1> nfs 0x80800000 192.168.10.103:/home/user/buildroot/output/images/Image.dtb
```

Download the kernel:

```
MA35D1> nfs 0x80800000 192.168.10.103:/home/user/buildroot/output/images/Image
```

Dwnload the rootfs:

```
MA35D1> nfs 0x80800000 192.168.10.103:/home/user/buildroot/output/images/rootfs.ubi
```



```
MA35D1> nfs 0x80800000 192.168.10.100:/home/user/MA35D1OTA/HMI/nand/Image
Speed: 100, full duplex
Using ethernet@40120000 device
File transfer via NFS from server 192.168.10.100; our IP address is 192.168.10.101
Filename '/home/user/MA35D1OTA/HMI/nand/Image'.
Load address: 0x80800000
Loading: #################################################################
         ###############
done
Bytes transferred = 14051840 (d66a00 hex)
MA35D1> █
```

Figure 1-5 Execute NFS Execution Screen.

Step 2: Erase the corresponding area and overwrite to the file

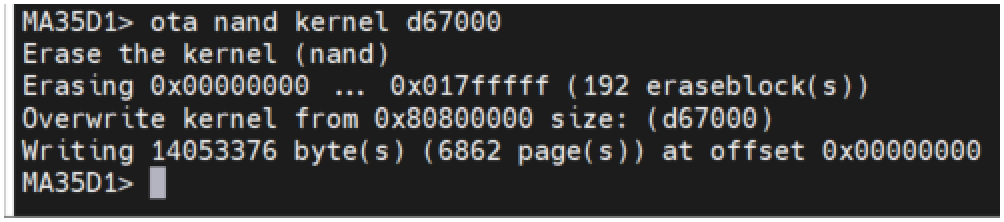
For example, update the kernel on the NAND:

```
MA35D1> ota nand kernel d67000
```

Note: The entered size should be in page units (0x800).

The program will be overwritten by the following command:

1. Erase the kernel (NAND):
   MA35D1> *mtd erase kernel*
2. Write the kernel from 0x80800000 (size: 0xd67000 bytes):
   MA35D1> *mtd write kernel 0x80800000 0x0 0xd67000*



```
MA35D1> ota nand kernel d67000
Erase the kernel (nand)
Erasing 0x00000000 ... 0x017fffff (192 eraseblock(s))
Overwrite kernel from 0x80800000 size: (d67000)
Writing 14053376 byte(s) (6862 page(s)) at offset 0x00000000
MA35D1>
```

Figure 1-6 Execute OTA Screen.

2. Code Description

Name the uboot command and the corresponding function:

```
/* Uboot CMD */
/* Define a U-Boot command "ota" with subcommands */
U_BOOT_CMD_WITH_SUBCMDS(ota, "ota utils", help_text,

    /* Subcommand list starts */
    /* Subcommand name, arguments, flags, Function to execute for this subcommand */
    U_BOOT_SUBCMD_MKENT(help,    2, 0, print_help_text),
    U_BOOT_SUBCMD_MKENT(nand,    4, 0, do_ma35d1_nand_ota),
    U_BOOT_SUBCMD_MKENT(spinand, 4, 0, do_ma35d1_spinand_ota),
    U_BOOT_SUBCMD_MKENT(emmc,    4, 0, do_ma35d1_emmc_ota),
    U_BOOT_SUBCMD_MKENT(sdcard,  4, 0, do_ma35d1_sdcard_ota)
    );
```

do_ma35d1_nand_ota: execute command processing on NAND.

```
static void do_ma35d1_nand_ota(struct cmd_tbl *cmdtp, int flag, int argc, char *const
argv[])
{
    if(argc == 2) {
         if ( !(strcmp(argv[1], "kernel")) ){
             /* Erase the kernel (nand) */
                printf("Erase the kernel (nand)");
             run_command("mtd erase kernel", 0);

             /* Overwrite to kernel from 0x80800000 size:(24M) */
                 printf("Overwrite to kernel from 0x80800000 size:(24M)");
             run_command("mtd write kernel 0x80800000 0x0 0x1800000", 0);
         }
         else if ( !(strcmp(argv[1], "dts")) ){
             /* Erase the device-tree (nand) */
                printf("Erase the device-tree (nand)");
             run_command("mtd erase device-tree", 0);

             /* Overwrite to device-tree from 0x80800000 size:(256k) */
                printf("Overwrite to device-tree from 0x80800000 size:(256k)");
             run_command("mtd write device-tree 0x80800000 0x0  0x40000", 0);
         }
         else if ( !(strcmp(argv[1], "rootfs")) ){
             /* Erase the rootfs (nand) */
                printf("Erase the rootfs (nand)");
             run_command("mtd erase rootfs", 0);

             /* Overwrite to rootfs from 0x80800000 size:(100m) */
                 printf("Overwrite to rootfs from 0x80800000 size:(100m)");
             run_command("mtd write rootfs 0x80800000 0x0 0x6400000", 0);
         }
         else
             printf("%s: parameters not found.\n", argv[1]);
    }
     else if (argc == 3) {
           char *size = argv[2];
         if (!(strcmp(argv[1], "kernel"))) {
             /* Erase the kernel (nand) */
             printf("Erase the kernel (nand)\r\n");
             run_command("mtd erase kernel", 0);

             /* Overwrite kernel from 0x80800000 with the specified size */
             printf("Overwrite kernel from 0x80800000 size: (%s)\r\n", size);
```

```
            char command[100];
            snprintf(command, sizeof(command), "mtd write kernel 0x80800000 0x0 %s",
size);
            run_command(command, 0);
        }
        else if (!(strcmp(argv[1], "dts"))) {
            /* Erase the device-tree (nand) */
            printf("Erase the device-tree (nand) \r\n");
            run_command("mtd erase device-tree", 0);

            /* Overwrite device-tree from 0x80800000 with the specified size */
            printf("Overwrite device-tree from 0x80800000 size: (%s) \r\n", size);
            char command[100];
            snprintf(command, sizeof(command), "mtd write device-tree 0x80800000 0x0 %s",
size);
            run_command(command, 0);
        }
        else if (!(strcmp(argv[1], "rootfs"))) {
            /* Erase the rootfs (nand) */
            printf("Erase the rootfs (nand)\r\n");
            run_command("mtd erase rootfs", 0);

            /* Overwrite rootfs from 0x80800000 with the specified size */
            printf("Overwrite rootfs from 0x80800000 size: (%s)\r\n", size);
            char command[100];
            snprintf(command, sizeof(command), "mtd write rootfs 0x80800000 0x0 %s",
size);
            run_command(command, 0);
        }
        else
            printf("%s: parameters not found.\n", argv[1]);
     }
    else
        printf("Invalid number of arguments.\n");
}
```

do_ma35d1_sdcard_ota: execute command processing on SD card.

```
static void do_ma35d1_sdcard_ota(struct cmd_tbl *cmdtp, int flag, int argc, char *const
argv[])
{
    if(argc == 2) {
        if ( !(strcmp(argv[1], "kernel")) ){
            /* "Erase the kernel (sdcard) */
            printf("Erase the kernel (sdcard)");
            run_command("mmc erase 0x1800 0x8000", 0);

            /* Overwrite to kernel from 0x80800000 size:(16M) */
                printf("Overwrite to kernel from 0x80800000 size:(16M)");
            run_command("mmc write 0x80800000 0x1800 0x8000", 0);
        }
        else if ( !(strcmp(argv[1], "dts")) ){
            /* Erase the device-tree (sdcard) */
               printf("Erase the device-tree (sdcard)");
            run_command("mmc erase 0x1600 0x80", 0);

            /* Overwrite to device-tree from 0x80800000 size:(64k) */
               printf("Overwrite to device-tree from 0x80800000 size:(64k)");
            run_command("mmc write 0x80800000 0x1600 0x80", 0);
        }
        else if ( !(strcmp(argv[1], "rootfs")) ){
            /* The rootfs.ext4 is too big. */
            printf("[E]:Non-function ! \n");
        }
        else
            printf("%s: parameters not found.\n", argv[1]);
    }
     else if (argc == 3) {
        char *size = argv[2];

        if (!(strcmp(argv[1], "kernel"))) {
            /* Erase the kernel (sdcard) */
            printf("Erase the kernel (sdcard)\r\n");
            run_command("mmc erase 0x1800 0x8000", 0);

            /* Overwrite kernel from 0x80800000 with the specified size */
            printf("Overwrite kernel from 0x80800000 size: (%s)\r\n", size);
            char command[100];
            snprintf(command, sizeof(command), "mtd write kernel 0x80800000 0x1800  %s",
size);
            run_command(command, 0);
        }
        else if (!(strcmp(argv[1], "dts"))) {
            /* Erase the device-tree (sdcard) */
            printf("Erase the device-tree (sdcard) \r\n");
            run_command("mtd erase device-tree", 0);

            /* Overwrite device-tree from 0x80800000 with the specified size */
            printf("Overwrite device-tree from 0x80800000 size: (%s) \r\n", size);
            char command[100];
            snprintf(command, sizeof(command), "mtd write device-tree 0x80800000
0x1600 %s", size);
            run_command(command, 0);
        }
        else if (!(strcmp(argv[1], "rootfs"))) {
            /* The rootfs.ext4 is too big. */
            printf("[E]: Non-function!\n");
        }
        else
            printf("%s: parameters not found.\n", argv[1]);
    }
    else
        printf("Invalid number of arguments.\n");

}
```

3. Software and Hardware Requirements

3.1 Software Requirements

- BSP version
  - Linux-5.10.x

3.2 Hardware Requirements

- Circuit components
  - NuMaker-HMI-MA35D1-S1

4. Directory Information

The directory structure is shown below.

🗁 EC_MA35D1_OTA_Update_by_Uboot_Stage_V1.00	
🗁 SampleCode	uboot cmd code

Figure 4-1 Directory Structure

5. Example Code Execution

1. Download the pre-update package to the evaluation board.
2. Erase the corresponding area and overwrite to the file.

6. Revision History

Date	Revision	Description
2023.03.13	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.