OTA Update on U-Boot

Application Note for NUC970/NUC980 Series

Document Information

Abstract	This application note introduces how to do OTA firmware update on u-boot.
Apply to	NUC970/NUC980 series.

The information described in this document is the exclusive intellectual property of Nuvoton Technology Corporation and shall not be reproduced without permission from Nuvoton.

Nuvoton is providing this document only for reference purposes of NuMicro microcontroller based system design. Nuvoton assumes no responsibility for errors or omissions.

All data and specifications are subject to change without notice.

For additional information or questions, please contact: Nuvoton Technology Corporation.

www.nuvoton.com

Table of Contents

1 OVERVIEW 3

2 NUWRITER 5

2.1 Related Resources 5

3 FIRMWARE UPDATE APPLICATION - FWUPDATE 6

3.1 Compile and Program fwupdate 6

3.2 Use fwupdate to Write Image on Flash 6

4 OTA_UPDATE COMMAND ON U-BOOT 8

4.1 Create ota_update.c 8

4.2 Create *check_crc.h* and *check_crc.c* 11

4.2.1 Create *check_crc.h* 11

4.2.2 Create *check_crc.c* 11

4.3 Create *crc_checksum.h* and *crc_checksum.c* 12

4.3.1 Create *crc_checksum.h* 12

4.3.2 Create *crc_checksum.c* 12

4.4 Compile u-boot and Linux Kernel 13

4.5 Modify *env.txt* 13

5 CONCLUSION 15

1 Overview

This application note introduces how to do OTA firmware update on u-boot space. In the past, everyone usually updates firmware on user space. If a device is powered off during updating, it will cause damage to the device kernel. Therefore, Nuvoton provides a method to keep the original Linux kernel complete, and checks whether the new firmware update is completed or not.

This method separates Flash into four mtdpartitions: u-boot, original Linux kernel, new Linux kernel, and root filesystem, as shown in Figure 1-1. You need to pack the new Linux kernel into package by NuWriter and download the package to the root filesystem. Then use **fwupdate** to write the package to Flash and use **ota_update** to update the Linux kernel. After **ota_update** in u-boot environment variable is set up, it will be executed and do OTA firmware update.

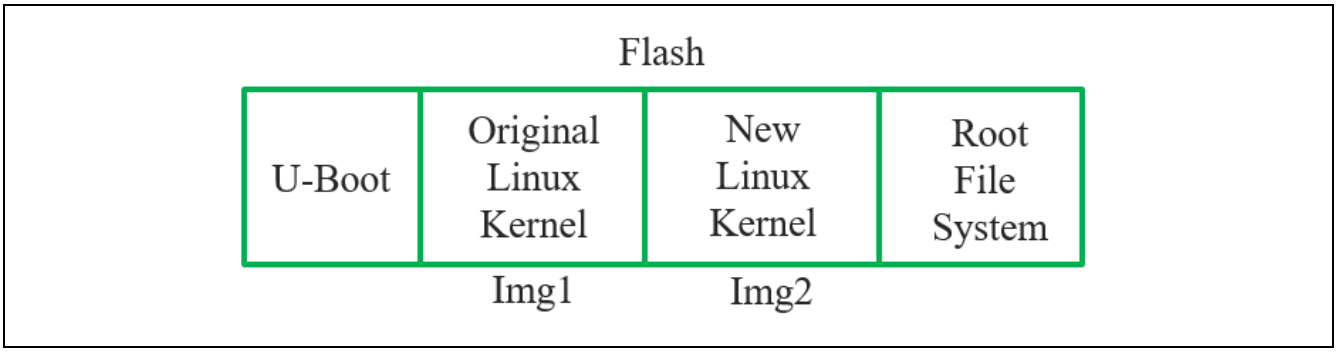


Figure 1-1 Flash Partition

The OTA update flowchart is shown in Figure1-2. First, you need to pack the new Linux kernel by NuWriter. When NuWriter packs the Linux kernel, it will attach a header file such that the **fwupdate** writes the package to a specified Flash address.

ota_update reads and compares Img1's and Img2's image header file to decide if firmware will be updated or not. If Img1 and Img2 are the same, system will not be updated and will boot from Img1. If Img1 is damaged or system needs to be updated, **ota_update** will check Img2 checksum to ensure Img2 is completed. If Img2 is completed, **ota_update** will copy Img2 to Img1's Flash address and reboot. However, if the Img1 and Img2 are all damaged, system will not boot up, then stop booting at u-boot to avoid rebooting repeatedly, and send a warning message.

- **fwupdate**: A Linux command to write package to Flash
- **ota_update**: A u-boot command to update the Linux kernel
- Img1: The original Linux kernel image
- Img2: The new Linux kernel image
- CRC checksum: Used to check the image completeness

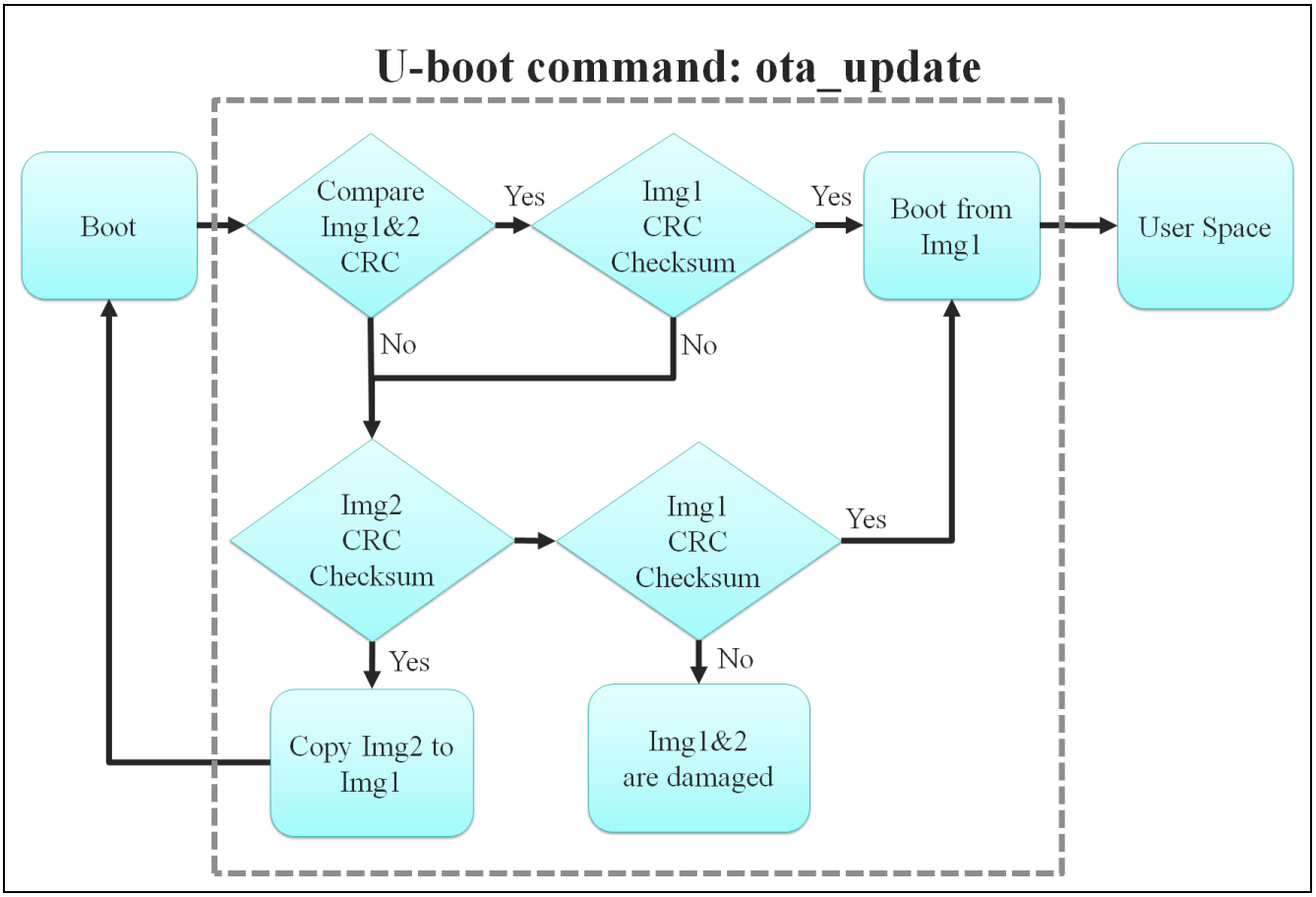


Figure1-2 **ota_update** Flowchart

2 NuWriter

The NuWriter is a programing tool provided by Nuvoton. The NuWriter application and firmware code are open sourced, and user can add new features or develop new user interfaces per user's application. The NuWriter tool uses chip's USB ISP mode with windows application on the PC by USB device for data transmission to program the image file to different storage devices. On-board ROM device includes NAND Flash, SPI Flash, eMMC/SD, and SPI NAND Flash.

2.1 Related Resources

Refer to Pack mode section in NuWriter User Manual for how to pack Linux kernel. You can download the NuWriter User Manual from Nuvoton's official website:

- NuWriter for NUC970 User Manual can be found in N9H30_emWin_Non-OS_BSP_v1.04: https://www.nuvoton.com/resource-download.jsp?tp_GUID=SW1820200910090527
  - NuWriter source code for NUC970 series: https://github.com/OpenNuvoton/NUC970_NuWriter
- NuWriter for NUC980 User Manual can be found in NUC980_Linux-4.4_BSP_v1.03.000.zip: https://www.nuvoton.com/resource-download.jsp?tp_GUID=SW1820200909165814
  - NuWriter source code for NUC980 series: https://github.com/OpenNuvoton/NUC980_NuWriter

3 Firmware Update Application - fwupdate

Nuvoton provides **fwupdate** to write package generated by NuWriter to Flash. The package can include uboot, Linux kernel, and *env.txt*. However, only Linux kernel is included here.

Refer to Nuvoton NUC970/NUC980 Application Note “Create Root Filesystem on Flash” for how to create root filesystem on Flash.

3.1 Compile and Program fwupdate

The **fwupdate** can be found in the nuc980bsp:

```
~/NUC970_Buildroot/nuc980bsp/application/demos/fwupdate
```

Compile **fwupdate** through make command.

```
~/NUC970_Buildroot/nuc980bsp/application/demos/fwupdate$ make
```

It will generate an **fwupdate** binary file, and copy it to the target folder.

```
~/NUC970_Buildroot/nuc980bsp/application/demos/fwupdate$ cp fwupdate
/home/user/Buildroot/NUC980_IIOT/NUC970_Buildroot/output/target/usr/bin/
```

Compile and program Linux kernel to device, and use **fwupdate** command.

3.2 Use fwupdate to Write Image on Flash

fwupdate application usage:

-p, --pack, path of pack file

-w, --whole, name of MTD partition and this partition must contain whole Flash address

-h, --help, help

Use **fwupdate** command in the device terminal to write “*New_Pack_Image*” package to Flash. The Flash address can be defined in NuWriter. “*New_Pack_Image”* package file is created by NuWriter pack mode. Refer to the “Modify env.txt” section. The name “*WHOLE*” is one of MTD partition can be defined in u-boot environment variable.

```
$fwupdate –p New_Pack_Image –w WHOLE
```

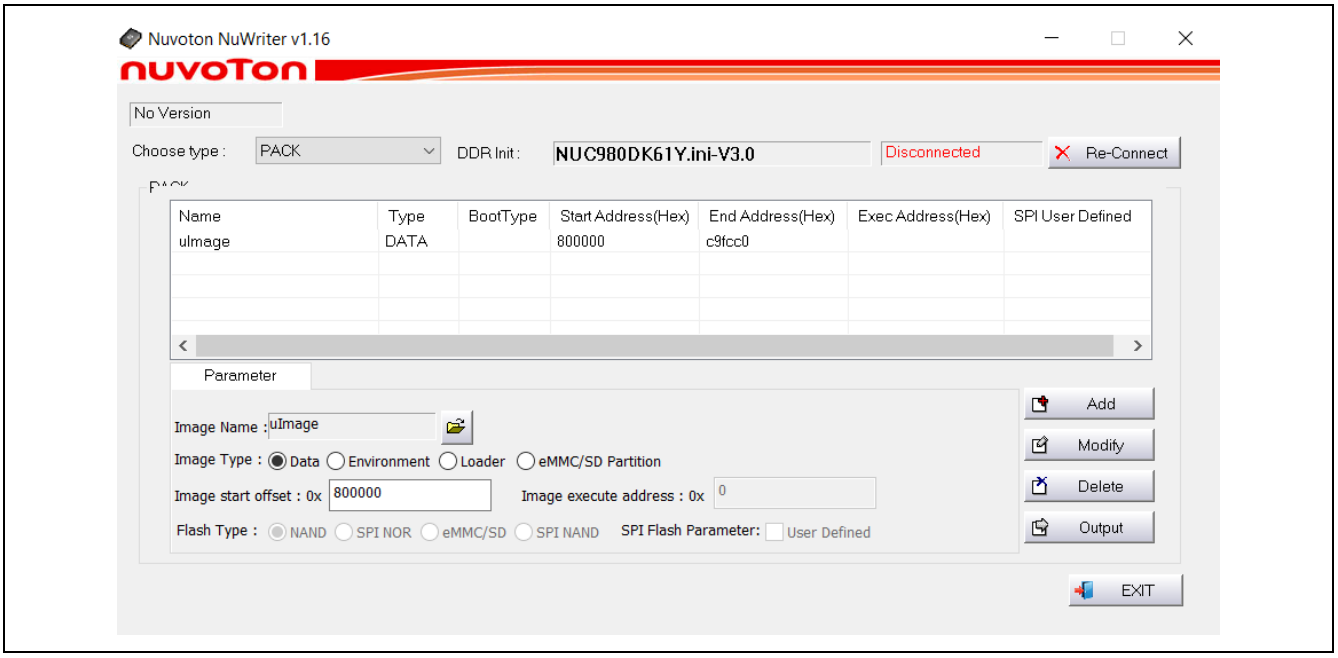


Figure 3-1 NuWriter Pack Mode

4 ota_update Command on U-Boot

ota_update checks the two image's CRC number. If the two CRC numbers are not the same, **ota_update** will check new Linux kernel completeness for firmware update. **ota_update** consists of five C language files in the u-boot.

- *ota_update.c*
- *check_crc.c*
- *check_crc.h*
- *crc_checksum.c*
- *crc_checksum.h*

Add the following text in Makefile to compiling function above.

Open the Makefile in common folder.

```
~/NUC970_Buildroot/output/build/uboot-master/common$ gedit Makefile
```

Add the following code:

```
obj-y += ota_update.o
obj-y += check_crc.o
obj-y += crc_checksum.o
```

4.1 Create ota_update.c

Go to the */Buildroot/NUC980_IIOT/NUC970_Buildroot/output/build/uboot-master/common* folder and create a file named *ota_update.c*. The code is shown below:

```
#include <common.h>
#include <command.h>
#include "check_crc.h"
#include "crc_checksum.h"
#include <spi.h>
#include <spi_flash.h>
#include <mapmem.h>
#include <div64.h>
#include <dm.h>
#include <malloc.h>
#include <asm/io.h>
#define SPI_MODE_0    (0|0)

extern int check_crc(char* addr1, char* addr2);
extern int crc_checksum(char* image_number, char* image_addr);
extern int do_reset(cmd_tbl_t *cmdtp, int flag, int argc, char * const argv[]);
```

```
extern int do_bootm(cmd_tbl_t *cmdtp, int flag, int argc, char * const argv[]);


//argv[0]=ota_update argv[1]=image1_ram_offset argv[2]=image2_ram_offset
argv[3]=image1_flash_offset argv[4]=image2_flash_offset argv[5]=image_size
static int ota_update(cmd_tbl_t *cmdtp, int flag, int argc, char * const argv[]){
    int check_crc_flag = 0;
    char* bootm[]={"bootm",argv[1]};
    int otaupdate_flag=0;//0 boot from image1; 1 image 1 and 2 are damaged;


    if(argc!=6)
        return CMD_RET_USAGE;


    check_crc_flag=check_crc(argv[1],argv[2]);
    if(check_crc_flag==1)
    {
        printf("CRC check is the same, check image 1 ...\n");
        if(crc_checksum("1",argv[1]))
        {
            printf("Boot from Image 1\n");
            do_bootm(cmdtp,0,2,bootm);
        }
        else
        {
            printf("Image 1 is damaged, check image 2 ...\n");
            if(crc_checksum("2",argv[2]))
            {
                printf("Prepare copy image 2 to 1\n");
                otaupdate_flag=0;
            }
            else
            {
                printf("Image 1 and 2 are damaged\n");
                otaupdate_flag=1;
            }
        }
    }
    else
    {
        printf("image 1's crc and 2's crc are different, check image 2 ...\n");
        if(crc_checksum("2",argv[2]))
        {
```

```
            printf("Prepare copy image 2 to 1\n");
            otaupdate_flag=0;
        }
        else
        {
            printf("Image 2 is damaged, check Image 1\n");
            if(crc_checksum("1",argv[1]))
            {
                printf("Boot from Image 1\n");
                do_bootm(cmdtp,0,2,bootm);
            }
            else
            {
                printf("Image 1 and 2 are damaged\n");
                otaupdate_flag=1;
            }
        }
    }
    return otaupdate_flag;
}


U_BOOT_CMD(
    ota_update,6,0,ota_update,"Check two image if update the kernel or not",
"ota_update [addr1] [addr2]\n"
"            addr1 is original kernel ram address\n"
"            addr2 is    new   kernel ram address\n"
);
```

4.2 Create *check_crc.h* and *check_crc.c*

4.2.1 Create *check_crc.h*

Go to the */Buildroot/NUC980_IIOT/NUC970_Buildroot/output/build/uboot-master/common* folder and create a file named as *check_crc.h*. The code is shown below:

```
int check_crc(char* addr1, char* addr2);
```

4.2.2 Create *check_crc.c*

check_crc will read two image's header to check CRC number if they are the same or not. This function need to input two RAM address, then it will return CRC number the same or not.

Go to the */Buildroot/NUC980_IIOT/NUC970_Buildroot/output/build/uboot-master/common* folder and create a file named as *check_crc.c*.

The code is shown below:

```
#include <common.h>
#include <command.h>
#include <image.h>
#include <mapmem.h>
#include "crc_checksum.h"
#include "check_crc.h"

int do_spi_flash_probe(int argc, char * const argv[]);
int do_spi_flash_read_write(int argc, char * const argv[]);

int check_crc(char* addr1, char* addr2)
{
    ulong addr;
    addr = simple_strtoul(addr1, NULL, 16);
    void *hdr = (void *)map_sysmem(addr, 0);
    ulong Image1crc=image_get_hcrc(hdr);
    printf("Image 1 crc=%ld\n",Image1crc);
    addr = simple_strtoul(addr2, NULL, 16);
    hdr = (void *)map_sysmem(addr, 0);
    ulong Image2crc=image_get_hcrc(hdr);
    printf("Image 2 crc=%ld\n",Image2crc);
    if(Image1crc==Image2crc)
        return 1;
    else
        return 0;
}
```

4.3 Create *crc_checksum.h* and *crc_checksum.c*

4.3.1 Create *crc_checksum.h*

Go to the */Buildroot/NUC980_IIOT/NUC970_Buildroot/output/build/uboot-master/common* folder and create a file named as *crc_checksum.h*. The code is shown below:

```
int crc_checksum(char* image_number, char* image_addr);
```

4.3.2 Create *crc_checksum.c*

crc_checksum will use CRC32 to check the whole image. *crc_checksum* needs to input image number and RAM address, and then it will return the image is complete or not.

Go to the */Buildroot/NUC980_IIOT/NUC970_Buildroot/output/build/uboot-master/common* folder and create a file named as *crc_checksum.c*. The code is shown below:

```
#include <common.h>
#include <command.h>
#include <image.h>
#include <mapmem.h>
#include "crc_checksum.h"

int crc_checksum(char* image_number, char* image_addr)
{
    ulong addr;
    addr = simple_strtoul(image_addr, NULL, 16);
    void *hdr = (void *)map_sysmem(addr, 0);
    int hcrc_flag=image_check_hcrc(hdr);
    if(hcrc_flag==0){
        printf("Image %s Check Fail\n",image_number);
        return 0;
    }
    int dcrc_flag=image_check_dcrc(hdr);
    printf("Image %s Header Check =%d\n",image_number,hcrc_flag);
    printf("Image %s Data Check =%d\n",image_number,dcrc_flag);
    if((hcrc_flag+dcrc_flag)==2){
        printf("Image %s Check Ok\n",image_number);
        return 1;
    }
    else{
        printf("Image %s Check Fail\n",image_number);
        return 0;
    }
}
```

4.4 Compile u-boot and Linux Kernel

Compile u-boot under *uboot-master* folder and program it to device.

```
~/NUC970_Buildroot/output/build/uboot-master$ make
```

Compile Linux kernel under Buildroot folder and program it to device.

```
~/NUC970_Buildroot$ make
```

4.5 Modify *env.txt*

Please refer to the Application Note “Create Root Filesystem on Flash” for how the u-boot environment variable is set up. You can modify *env.txt* shown below to execute **ota_update**. Besides, there is a difference between NOR Flash and NAND Flash. NAND Flash needs to use **nand** command and NOR Flash needs to use **sf** command.

- **image1_flash_offset**: Img1 Flash offset
- **image2_flash_offset**: Img2 Flash offset
- **image_size**: kernel image’s size
- **image1_ram_offset**: the RAM address offset which is put Img1
- **image2_ram_offset**: the RAM address offset which is put Img2
- **loadkernel1**, **loadkernel2**: read Flash to RAM
- **eraseflash**: erase Img1 Flash
- **copykernel**: copy Img2 to Img1 on Flash

NAND Flash *env.txt* is shown below:

```
baudrate=115200
bootdelay=1
stderr=serial
stdin=serial
stdout=serial
image1_flash_offset=0x200000
image2_flash_offset=0x800000
image_size=0x600000
image1_ram_offset=0x7fc0
image2_ram_offset=0x800000
loadkernel1=nand read ${image1_ram_offset} ${image1_flash_offset} ${image_size}
loadkernel2=nand read ${image2_ram_offset} ${image2_flash_offset} ${image_size}
eraseflash=nand erase ${image1_flash_offset} ${image_size}
copykernel=nand write ${image2_ram_offset} ${image1_flash_offset} ${image_size}
bootcmd=run loadkernel1;run loadkernel2;if ota_update ${image1_ram_offset}
${image2_ram_offset} ${image1_flash_offset} ${image2_flash_offset} ${image_size};then run
eraseflash;run copykernel;reset;fi;
```

```
bootargs=noinitrd root=/dev/mtdblock4 rootfstype=yaffs2 rootflags=inband-tags
console=ttyS0 rdinit=/sbin/init mem=64M
mtdparts=nand0:0x2000000@0x0(WHOLE),0x200000@0x0(u-
boot),0x600000@0x200000(kernel1),0x600000@0x800000(kernel2),-(user) ignore_loglevel
```

NOR Flash *env.txt* is shown below:

```
baudrate=115200
bootdelay=1
stderr=serial
stdin=serial
stdout=serial
image1_flash_offset=0x200000
image2_flash_offset=0x800000
image_size=0x600000
image1_ram_offset=0x7fc0
image2_ram_offset=0x800000
setspi=sf probe 0 30000000
loadkernel1=sf read ${image1_ram_offset} ${image1_flash_offset} ${image_size}
loadkernel2=sf read ${image2_ram_offset} ${image2_flash_offset} ${image_size}
eraseflash=sf erase ${image1_flash_offset} ${image_size}
copykernel=sf write ${image2_ram_offset} ${image1_flash_offset} ${image_size}
bootcmd=run setspi;run loadkernel1;run loadkernel2;if ota_update ${image1_ram_offset}
${image2_ram_offset} ${image1_flash_offset} ${image2_flash_offset} ${image_size};then run
eraseflash;run copykernel;reset;fi;
bootargs=noinitrd root=/dev/mtdblock4 rw rootfstype=jffs2 console=ttyS0 rdinit=/sbin/init
mem=64M mtdparts=m25p80:0x2000000@0x0(WHOLE),0x200000@0x0(u-
boot),0x600000@0x200000(kernel1),0x600000@0x800000(kernel2),-(user) ignore_loglevel
```

5 Conclusion

The **ota_update** function updates Linux kernel on u-boot to keep the original kernel work normally until the new kernel is verified completely. This can avoid copying the damaged kernel to original kernel. If the original kernel is damaged because of updating failed, **ota_update** will retry updating the new kernel until update is successful. If you want to update the whole image including u-boot, u-boot environment, Linux kernel, and root filesystem, you still can use NuWriter and **fwupdate** to update. You can pack u-boot, u-boot environment, Linux kernel, root filesystem as a package, and use **fwupdate** command to write the package into Flash. Updating the whole kernel must be performed carefully because it may damage the original image.

Revision History

Date	Revision	Description
2021.07.18	1.00	1. Initially issued.
2023.04.25	1.01	2. Add more descriptions in section 3

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.

Please note that all data and specifications are subject to change without notice.
All the trademarks of products and companies mentioned in this datasheet belong to their respective owners.