

NUC970 Non-OS BSP User Manual

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	Introduction to NUC970 Non-OS BSP.....	3
1.1	Develop Environment	3
1.2	DEV Board Setting	3
2	BSP Content.....	4
2.1	BSP directory structure	4
2.2	Non-OS BSP content.....	4
3	Nu-Writer Usage Guide	5
3.1	Introduction	5
3.2	Driver Installation	5
3.3	USB ISP Setting.....	9
3.4	Chip Selection.....	10
3.5	DDR/SRAM Mode	10
3.6	eMMC Mode.....	11
3.7	SPI Mode	13
3.8	NAND Mode.....	14
3.9	MTP Mode	16
3.10	PACK Mode	Error! Bookmark not defined.
3.11	Program U-Boot	22
3.12	Nu-Writer Trouble Shooting	23
4	U-Boot user manual	24
4.1	Configuration.....	24
4.2	Directory structure	31
4.3	Compile U-Boot.....	33
4.4	NAND AES secure boot example.....	34
4.5	U-Boot Command.....	43
4.6	Environment variables	69
4.7	mkimage tool	71
4.8	Security issue	74
4.9	Watchdog timer	75
4.10	U-Boot LCD.....	76
4.11	GPIO.....	76
4.12	Network test environment	78
4.13	Notice	84
5	Revision History.....	86

1 Introduction to NUC970 Non-OS BSP

This BSP supports Nuvoton NUC970 family processors. The NUC970 series targeted for general purpose 32-bit microcontroller embeds an outstanding CPU core ARM926EJ-S, a RISC processor designed by Advanced RISC Machines Ltd., runs up to 300 MHz, with 16 KB I-cache, 16 KB D-cache and MMU, 56KB embedded SRAM and 16 KB IBR (Internal Boot ROM) for booting from USB, NAND and SPI FLASH.

The NUC970 series integrates two 10/100 Mb Ethernet MAC controllers, USB 2.0 HS HOST/Device controller with HS transceiver embedded, TFT type LCD controller, CMOS sensor I/F controller, 2D graphics engine, DES/3DES/AES crypto engine, I2S I/F controller, SD/MMC/NAND FLASH controller, GDMA and 8 channels 12-bit ADC controller with resistance touch screen functionality. It also integrates UART, SPI/MICROWIRE, I2C, CAN, LIN, PWM, Timer, WDT/Windowed-WDT, GPIO, Keypad, Smart Card I/F, 32.768 KHz XTL and RTC (Real Time Clock)

This Non-OS BSP includes following contents:

- NUC970 Non-OS drivers.
- Precompiled U-Boot images for different boot mode.
- Flash programming tool Nu-Writer, and its Windows driver.
- User manuals.

1.1 Develop Environment

Keil IDE is used as Non-OS BSP develop environment, and use ULINK2 ICE for debug. The IDE does not belong to the content of this document. Please refer to official Keil website <http://www.keil.com/> for the user manual of Keil IDE.

NUC970 supports J-TAG debug interface. Users could use this interface to download programs to DDR and debug. After power up, the J-TAG interface is disabled by default. To use the J-TAG interface, either of the criteria must be met. 1) NUC970 boot up in USB ISP mode and successfully set up a connection with NuWriter tool, or 2) NUC970 does not boot up in secure boot mode and successfully execute the loader (or application) store in SPI flash, NAND flash, or eMMC.

NUC970 Non-OS BSP uses the same open source loader as Linux BSP, U-Boot. U-Boot uses Linux as develop environment. Users could either the precompiled U-Boot images in this BSP. If it is necessary to modify and re-build U-Boot, please download NUC970 Linux BSP and refer to NUC970 Linux BSP user manual to set up the develop environment. If the system boot from SPI flash or eMMC, it is not required to use a loader and can execute main program directly after system booting up. But while booting from NAND flash, it is recommend using a loader while booting from NAND flash and let it handles the bad block during system boot up.

1.2 DEV Board Setting

NUC970 family supports different modes, it can boot from SPI, NAND, eMMC, or enter USB ISP mode. The booting mode is selected by PA[1:0] jumper. Because most I/O pins support multiple functions, the jumpers on DEV board must be set according to the enabled peripherals. Please refer to “NUC972 Development Board User Manual” for the usage of DEV board.

2 BSP Content

2.1 BSP directory structure

Non-OS BSP contains four directories. The content of each directory listed in the table below.

Directory Name	Content
BSP	A compressed file including Non-OS driver, third party software and sample applications.
Documents	BSP related documents
Loader	Pre-compiled U-Boot image files.
Tools	Tool for programming NAND, SPI, eMMC or download image to DDR. And its Windows driver.

2.2 Non-OS BSP content

The file under BSP directory shows following content after decompressed.

Directory Name	Content
Driver	NUC970 peripheral drivers. Please refer to NUC970 Non-OS BSP Driver Reference Guide.chm under Documents directory for the usage of driver APIs.
Library	NUC970 libraries, including USB Host and smartcard.
SampleCode	Driver sample application.
Script	Link script and debug initialization file for Keil.
ThirdParty	Third party software. Including FATFS, yaffs2 file system and LwIP network protocol stack.

3 Nu-Writer Usage Guide

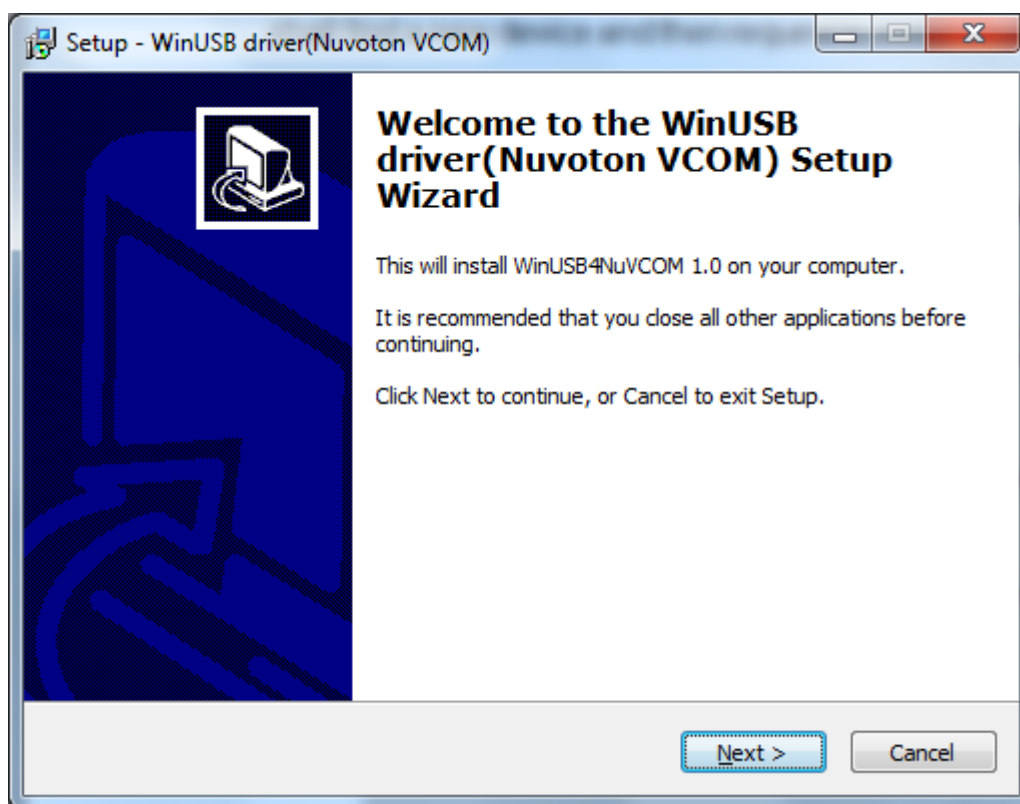
3.1 Introduction

Nu-Writer can download images to NAND flash, SPI flash, eMMC, or DDR while NUC970 is in USB ISP mode.

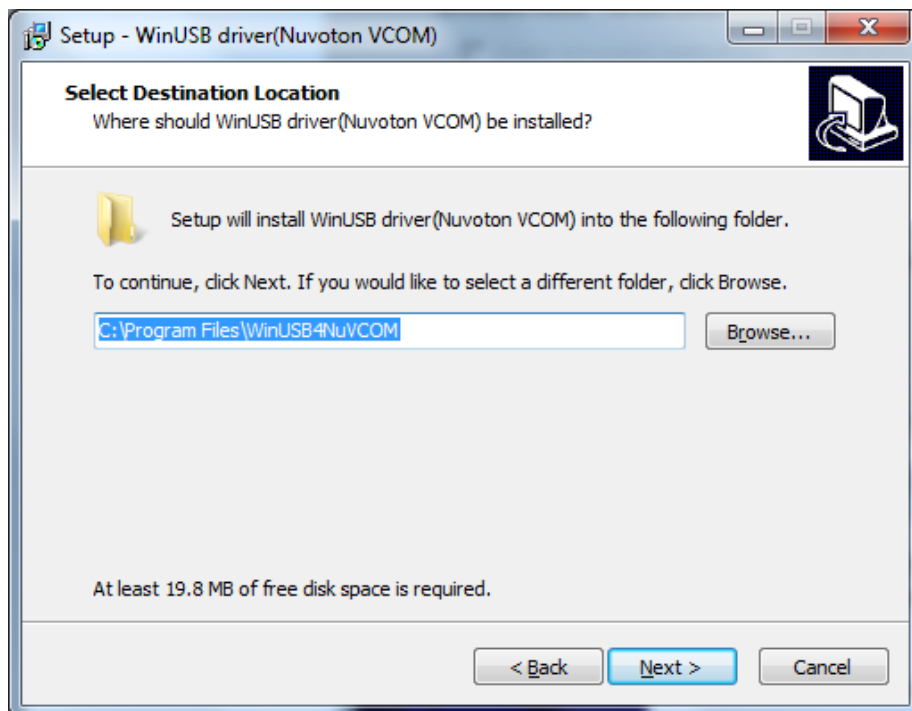
3.2 Driver Installation

Nu-Writer can only communicate with NUC970 after VCOM driver properly installed. Please follow the steps below to install VCOM driver:

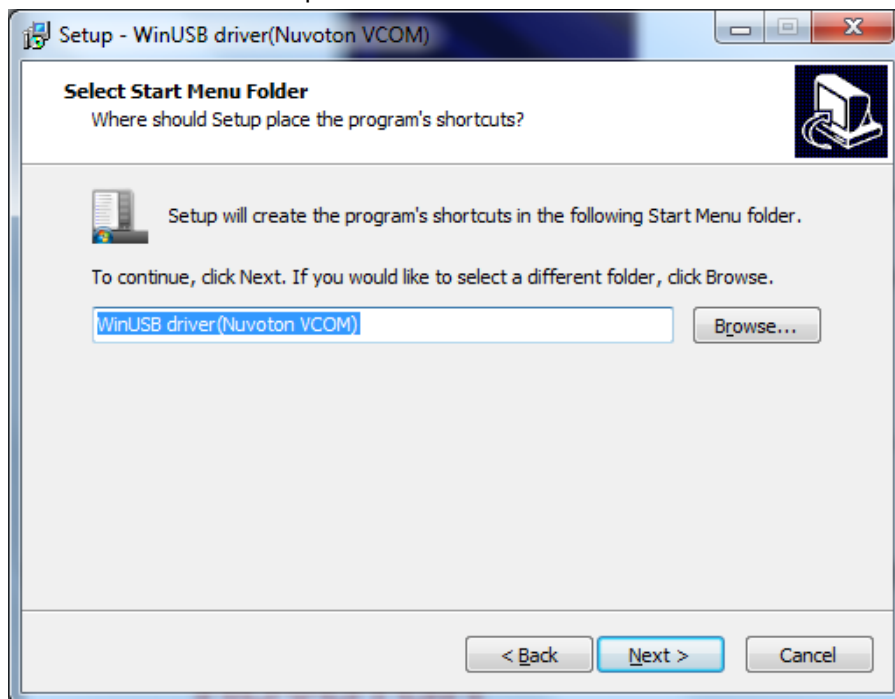
1. Connect PC with NUC970 evaluation board using USB cable. Execute WinUSB4NuVCOM.exe to install driver.
2. Turn on the power of NUC970 evaluation board, Windows will detect new device and request to install driver.



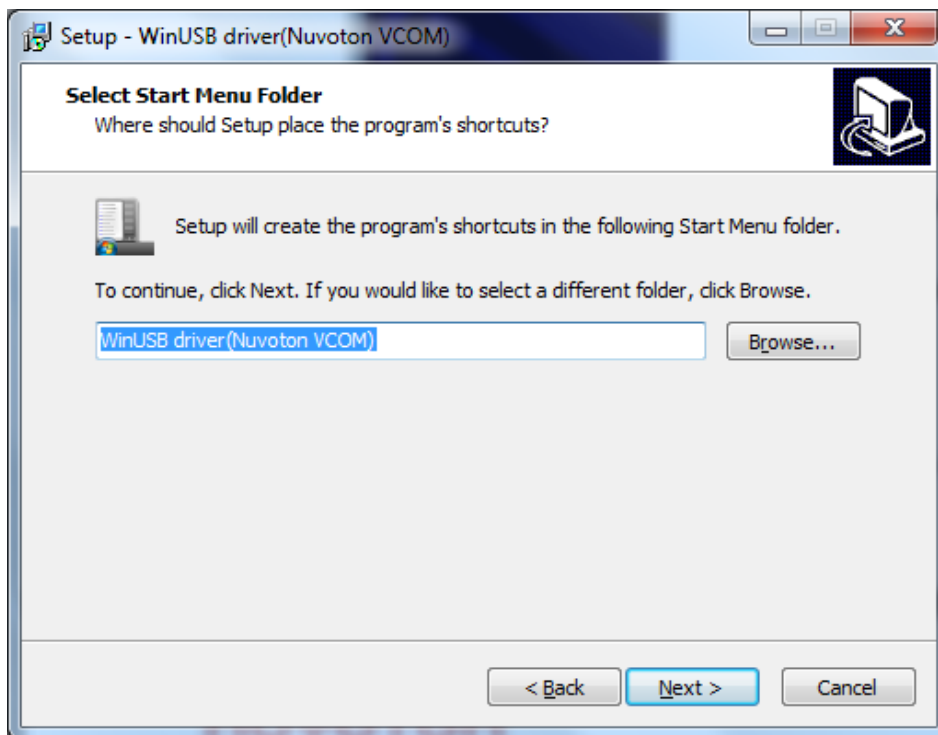
3. Press “Next” button. This window tells you the system is going to install WinUSB4NuVCOM 1.0 driver:



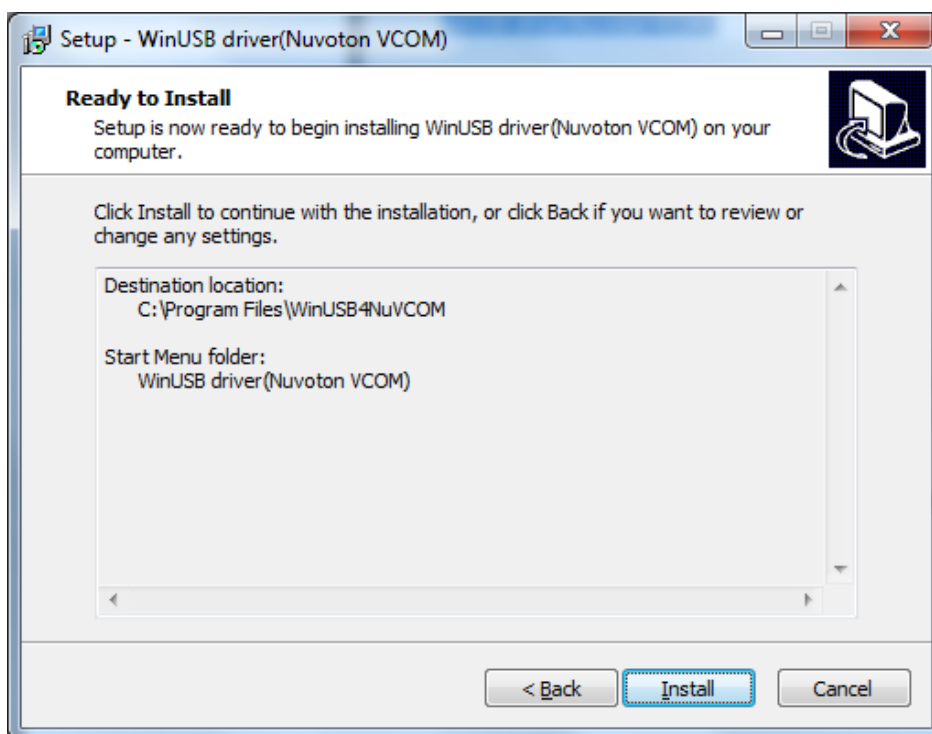
4. Select the driver installation path and then click "Next" button:



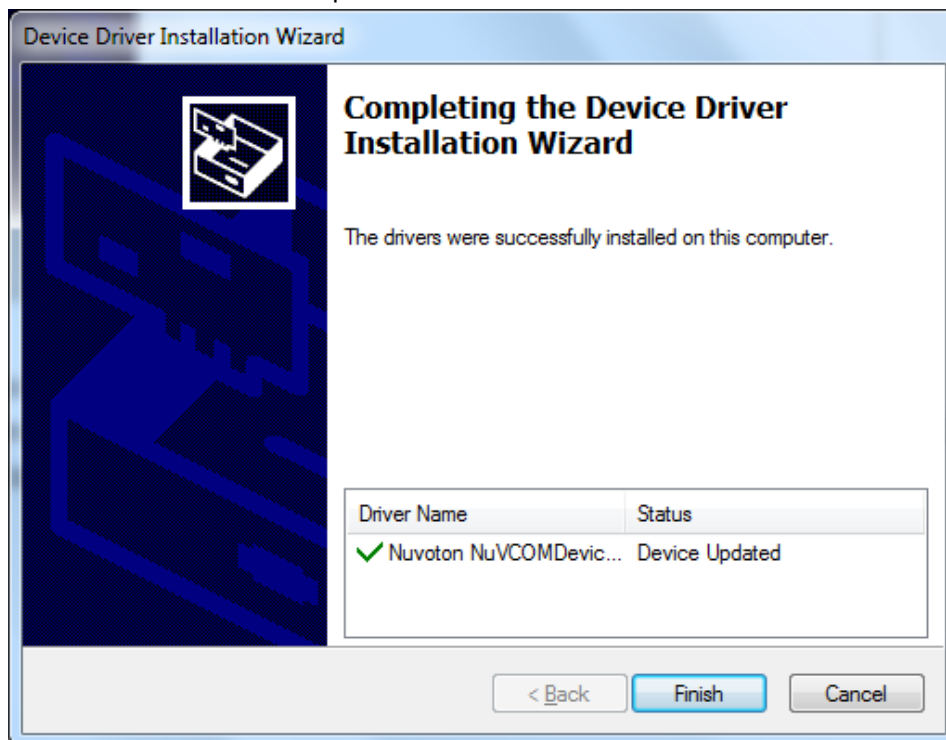
5. Press "Next" button:



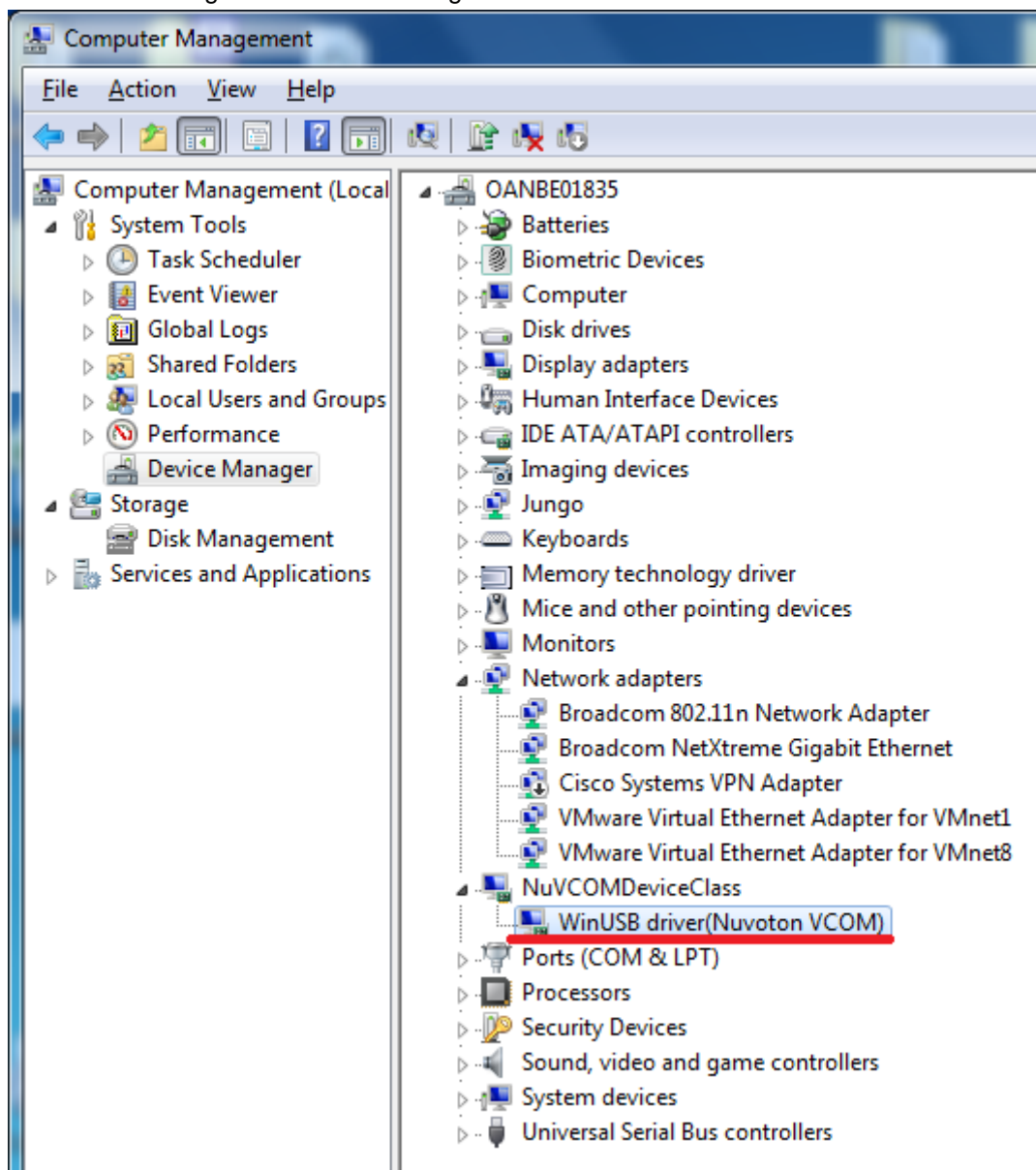
6. Press "Install" button:



7. Press "Finish" button to complete driver installation:



8. After driver successfully installed, “WinUSB driver (Nuvoton VCOM)” could be found in device manager as shown in the figure below.



3.3 USB ISP Setting

NUC970 series use jumpers to select boot mode. PA0 and PA1 must both stay at low level during system power up to force NUC970 enter USB ISP mode. Following table list the boot selection of NUC970:

Power-On Setting	PA1	PA0
USB ISP Mode	Low	Low
eMMC Boot	Low	High
NAND Boot	High	Low

SPI Boot	High	High
----------	------	------

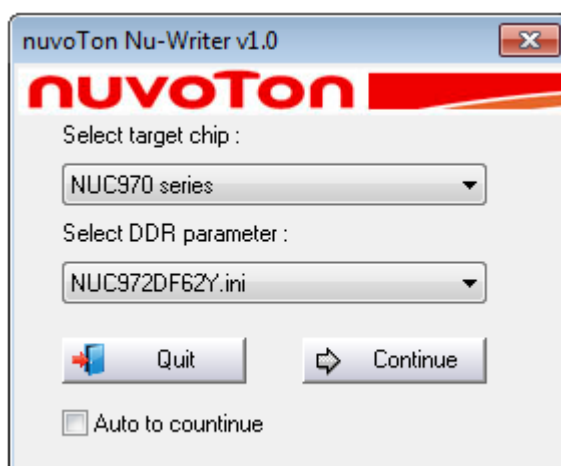
After turn on the NUC970 evaluation board power in USB ISP mode, then Nu-Writer can communicate with NUC970.

Note: Driver must be installed in order to let Nu-Writer communicate with NUC970.

3.4 Chip Selection

Decompress NuWriter-xxxxxxx.7z(in BSP/Tools directory) and execute “nuwriter.exe”, the first window shown as below

The target chip selection must chose NUC970 series (NUC972, NUC976...). And then select the DDR parameter based on the comment NUC970 device. The ID is printed on the top of NUC970 IC. After select correct DDR parameter, press “Continue” button to start using Nu-Writer.



3.5 DDR/SRAM Mode

DDR/SRAM Mode supports to load Image to DDR or SRAM. The operation flow listed below:

1. Select “DDR/SRAM” Mode.
2. Select Image file.
3. Input the target address Image file is going to load to in DDR/SRAM. **Note:** Target address must not exceed 31MB if the destination is in DDR.
4. Select “Download only” or “Download and run”
5. Press “Download.”



The screenshot shows the nuvoTon Nu-Writer v1.0 application window. The title bar reads "nuvoTon Nu-Writer v1.0". Below the title bar is a red banner with the "nuvoTon" logo. The main interface has a light blue background. At the top left, there's a text box labeled "No Version". Below it, a "Choose type:" label is followed by a dropdown menu set to "eMMC". To the right of the dropdown is a text field for "DDR Init:" containing "NUC972DF6ZY.ini". Further right are two buttons: "Disconnected" (in red) and "Re-Connect" (with a red X icon). On the left side, under the "eMMC" heading, is a large table with four columns: "Name", "Type", "Start offset", and "End offset". The table is currently empty. Below the table is a horizontal scrollbar. Underneath the scrollbar, it says "Alignment : 0x200, Reserved Size : 0x53edc800". On the right side, there's a "Parameters" section. It contains several fields and controls: "Image Name:" with a text box showing "u-boot_0x100000" and a folder icon; "Image Type:" with four radio button options: "Data" (selected), "Environment", "uBoot", and "Pack"; "Image encrypt:" with a dropdown menu and an "Enable" checkbox; "Image execute address: 0x" with a text box showing "200"; and "Image start offset: 0x" with a text box showing "100000". At the bottom of the Parameters section are four buttons: "Burn" (with a red arrow icon), "Verify" (with a magnifying glass icon), "Read" (with a blue up arrow icon), and "Format" (with a circular arrow icon). Below these buttons is a small USB drive icon and a long progress bar. In the bottom right corner of the entire window is an "EXIT" button with a red square icon.

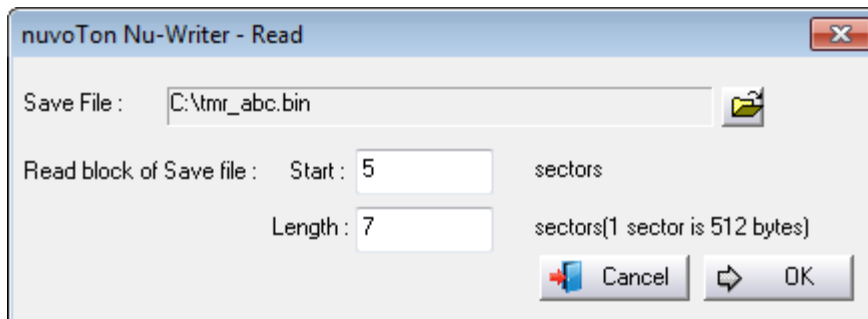
3.6.1 Add New Image

1. Select "eMMC" mode. The table in the left only record the image programmed in during this connection. It does not show the images that are programmed in eMMC before.
2. Set image attributes:
 - Image Name
 - Image Type
 - Image encrypt
 - Image execute address
 - Image start offset
3. Press "Burn" button.
4. Wait for the progress bar reach 100%. After complete, users could press "Verify" button to verify the programmed image.

3.6.2 Read from eMMC

Follow the steps below to read the content in eMMC:

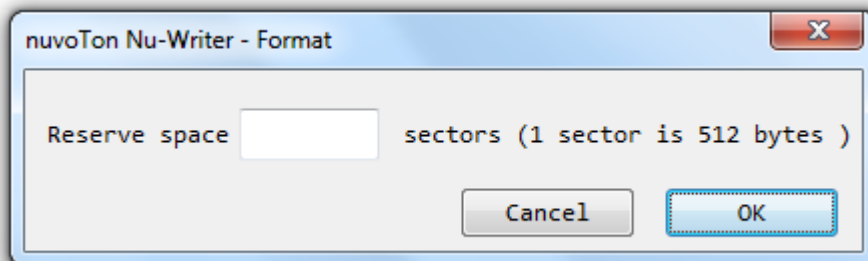
1. Select "eMMC" mode.
2. Press "Read".
3. Set the file to store read back data.
4. Set the read back sectors (1 sector is 512 bytes).
 - Start : Starting sector number
 - Length : Sector count
5. Press "OK".



3.6.3 Format (FAT32)

Follow the steps below to format eMMC to FAT32 file system:

1. Select "eMMC" Mode.
2. Press "Format".
3. Input reserved sector count. **Note:** Modify this value could damage the images in reserved space or the content in file system.
4. Press "OK".

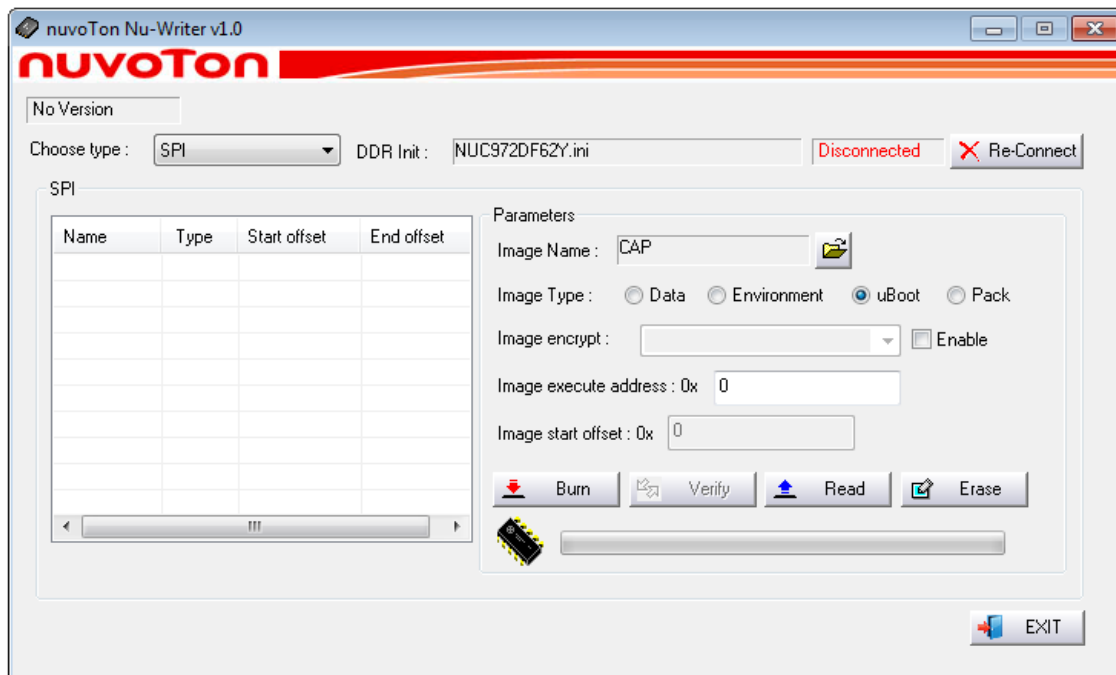


3.7 SPI Mode

SPI Mode can load image files into SPI flash, and the image could be one of the types between Data, Environment, uBoot, and Pack.

3.7.1 Add New Image

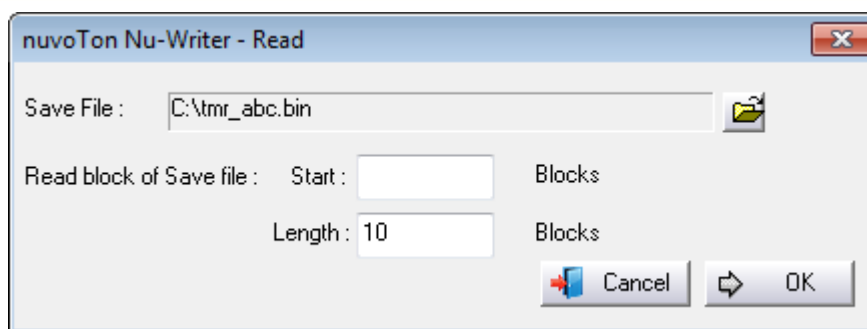
1. Select "eMMC" mode. The table in the left only record the image programmed in during this connection. It does not show the images that are programmed in eMMC before.
2. Set image attributes:
 - Image Name
 - Image Type
 - Image encrypt
 - Image execute address
 - Image start offset
3. Press "Burn" button.
4. Wait for the progress bar reach 100%. After complete, users could press "Verify" button to verify the programmed image.



3.7.2 Read Image

Follow the steps below to read the content in SPI flash

1. Select "SPI" Mode.
2. Press "Read" button.
3. Set the file to store read back data.
4. Set the read back blocks. Block size depends on the SPI flash in used.
 - Start : Starting block
 - Length : Block count
5. Press "OK" to read image.



3.7.3 Erase Image

Follow the steps below to remove images in SPI flash:

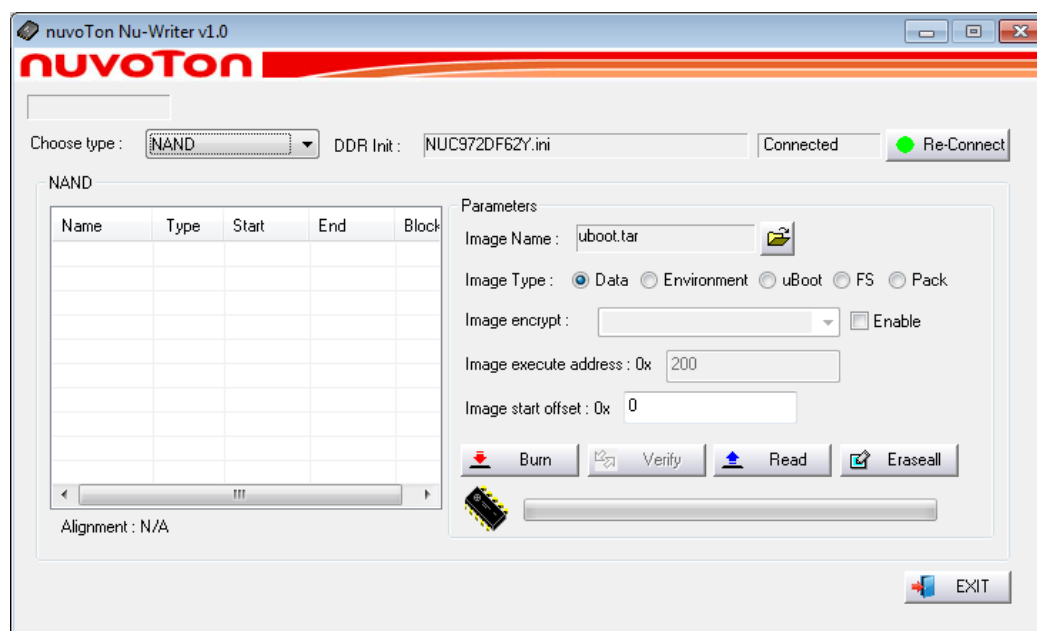
1. Select "SPI" mode.
2. Press "Erase all" to erase all images.

3.8 NAND Mode

NAND Mode can load image files into NAND flash, and the image could be one of the types between Data, Environment, uBoot, and Pack.

3.8.1 Add Image

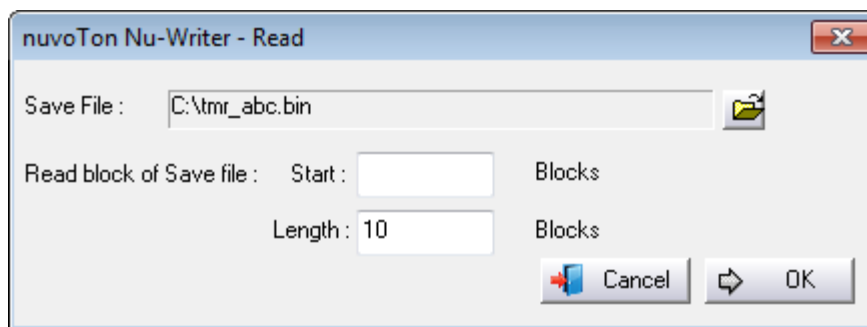
1. Select "NAND" mode. The table in the left only record the image programmed in during this connection. It does not show the images that are programmed in NAND before.
2. Set image attributes:
 - Image Name
 - Image Type
 - Image encrypt
 - Image execute address
 - Image start offset
3. Press "Burn" button.
4. Wait for the progress bar reach 100%. After complete, users could press "Verify" button to verify the programmed image.



3.8.2 Read Image

Follow the steps below to read back image:

1. Select "NAND" Mode.
2. Press "Read" button.
3. Set the file to store read back data.
4. Set the read back blocks. Block size depends on the NAND flash in used.
 - Start : Starting block
 - Length : Block count
5. Press "OK" to read image.



3.8.3 Erase Image

Follow the steps below to remove image:

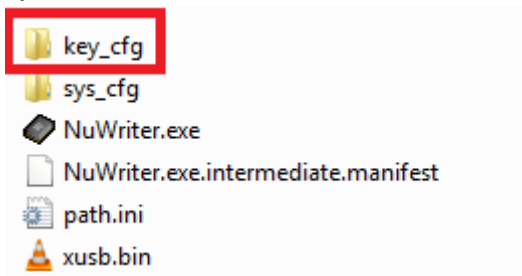
1. Select "NAND" mode.
2. Press "Erase all" to erase all images.

3.9 MTP Mode

MTP mode can program secure key in to NUC970's MTP to protect the images stores in eMMC, NAND flash, and SPI flash.

3.9.1 Add new Key file

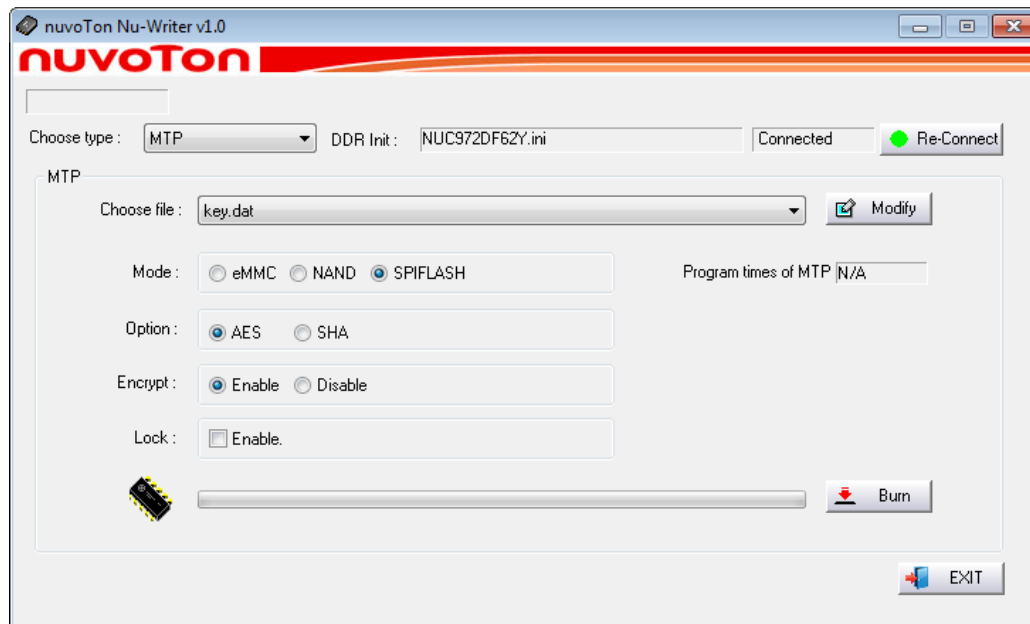
1. Enter key_cfg directory shown below.



2. Create a file to store the key. The format must be the same as below. First line start with 256, follow by 8 lines of hexadecimal key stored in big endian mode.

```
256
0x12345678
0x23456789
0x3456789a
0x456789ab
0x56789abc
0x6789abcd
0x789abcde
0x89abcdef
```

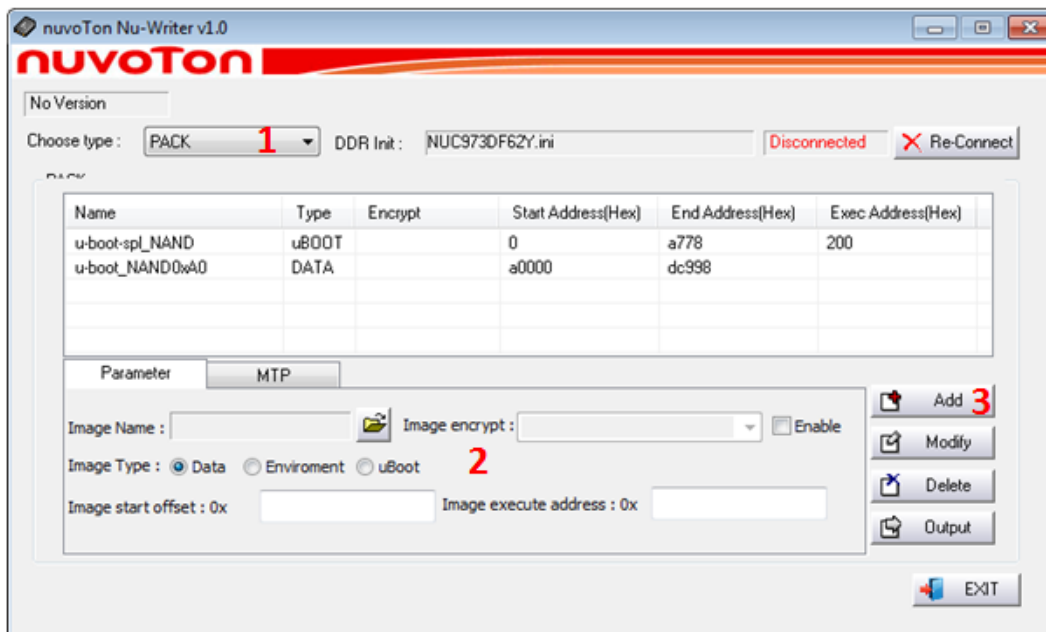
3. Re-launch Nu-Writer tool and select MTP mode.
4. Select newly created key file.
5. Configure MTP mode
 - Select boot mode: eMMC, NAND, or SPIFLASH
 - Select secure mode: SHA or AES
6. Press "Burn" to program the key



3.10 PACK Mode

Pack mode can merge many image into a pack image, user can use Nu-Writer to burn pack image into the device (NAND flash, SPI flash, eMMC).

3.10.1 Add a New Image

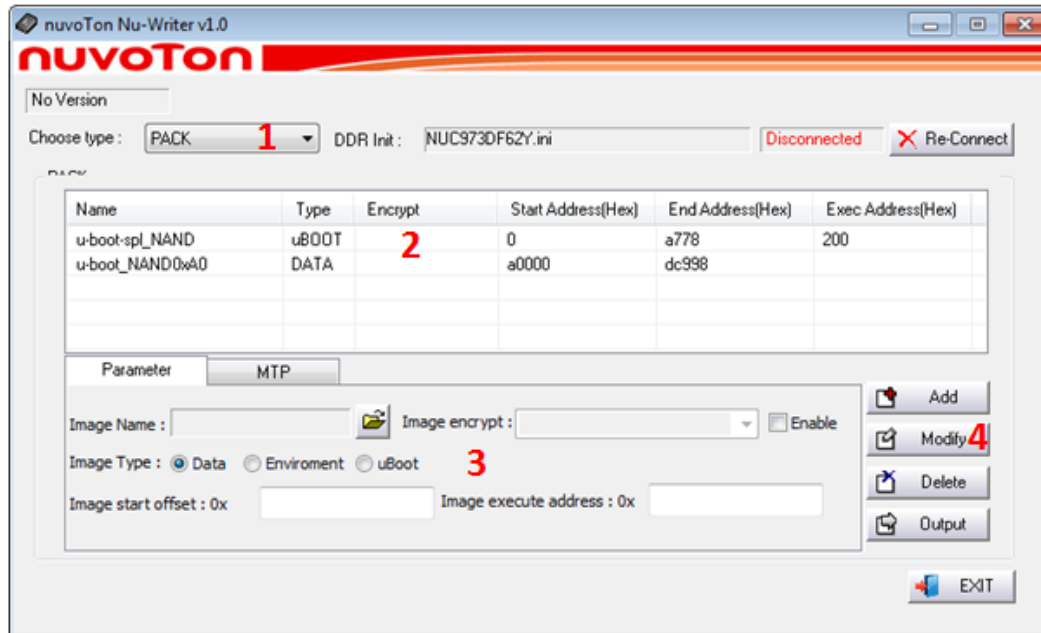


Follow the steps below to add a new image to pack list:

1. Select the "Pack".
2. Fill in the image information :
 - Image Name: Browse the image file.
 - Image Type Select the image type. (only one type can be selected)
 - Image encrypt: Select encrypt file and Set enable or disable.
 - Image execute address: Enter image execute address. This setting is only valid for

- u-boot.
- Image start offset: Enter image start offset.
- 3. Click "Add".
- 4. Waiting for finishing progress bar.
- 5. After "Burn" the image, click the "Verify" button to read back the image data to make sure the burning status.

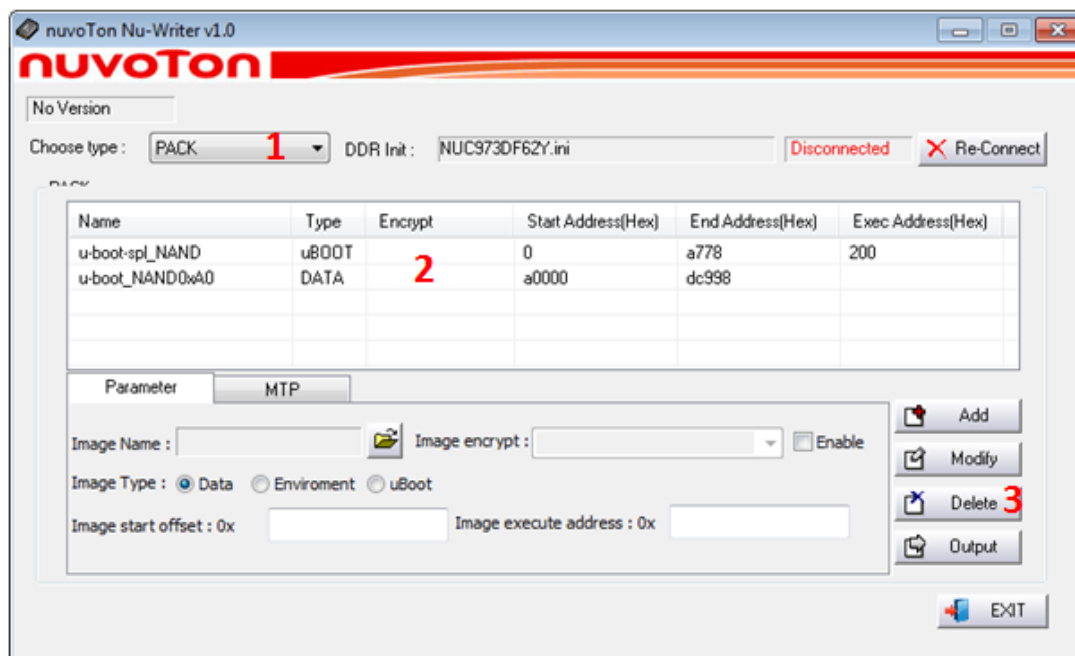
3.10.2 Modify an Image



Follow the steps below to modify image from pack list:

1. Select the "PACK".
2. Double click Image name in the pack list to modify.
3. Fill in the image information:
 - Image Name: Browse the image file.
 - Image Type Select the image type. (only one type can be selected)
 - Image encrypt: Select encrypt file and Set enable or disable.
 - Image execute address: Enter image execute address. This setting is only valid for u-boot.
 - Image start offset: Enter image start offset.
4. Click "Modify"

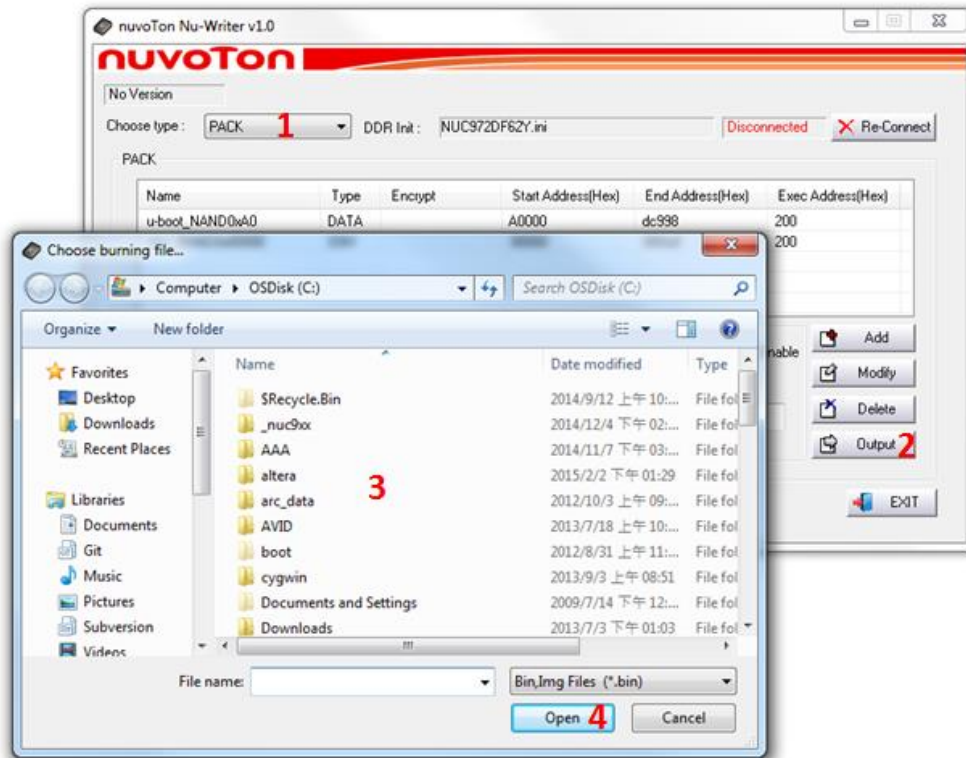
3.10.3 Delete an Image



Follow the steps below to delete image from pack list:

1. Select "PACK".
2. Click Image name on the pack list.
3. Click "Delete"

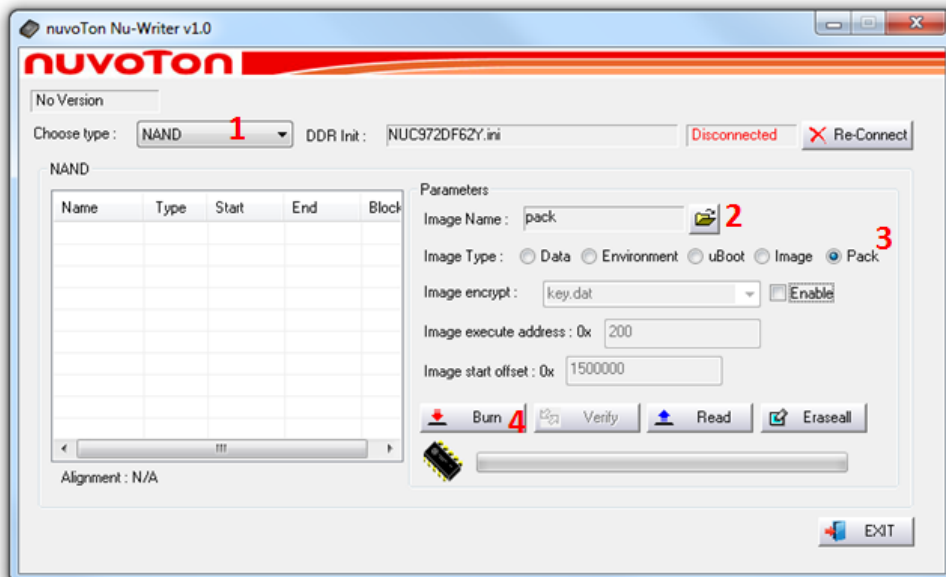
3.10.4 Create a Pack Image



Follow the steps below to output pack image:

1. Select "PACK".
2. Click "Output".
3. Browse save file.
4. Click "Open" to output pack image.

3.10.5 Program a Pack Image



Follow the steps below to burn pack image into NAND flash:

1. Select "NAND".
2. Browse pack image.
3. Select image type to "Pack".
4. Click "Burn".

3.10.6 Create and Program a Pack Image

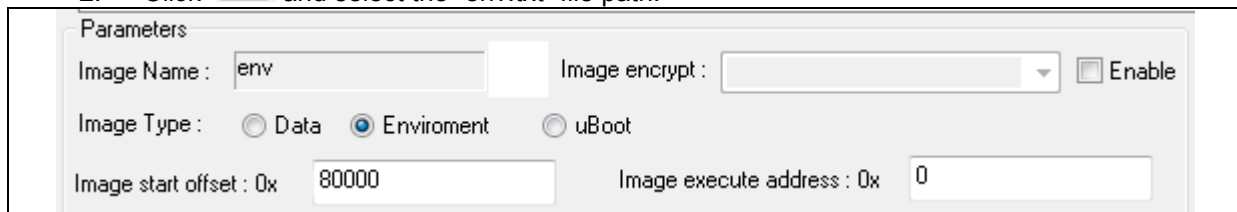
Preparation of related files:

1. u-boot.bin (Default: offset address is set to 0xA0000, execute address is set to 0xE00000) .
2. u-boot-spl.bin (Default: DDR execute address is set to 0x200) .
3. env.txt (Default: offset address is set to 0x80000).

Suppose user wants to build a pack image includes env.txt, u-boot.bin and u-boot-spl.bin. And program to NAND flash. Follow the steps below to build pack image:

1. Select "PACK".

2. Click  and select the "env.txt" file path:




Parameters


Image Name : env Image encrypt : ☐ Enable

Image Type : ☐ Data ☒ Enviroment ☐ uBoot

Image start offset : 0x 80000 Image execute address : 0x 0

3. Click  Add .

4. Click  and select the "u-boot-spl.bin" file path:





Parameters

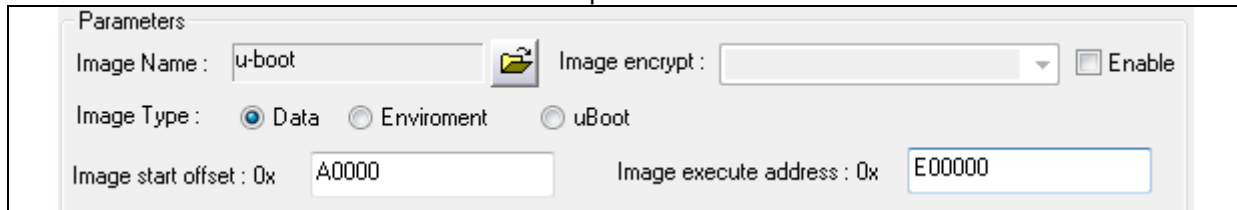
Image Name : u-boot-spl  Image encrypt : ☐ Enable

Image Type : ☐ Data ☐ Enviroment ☒ uBoot

Image start offset : 0x 0 Image execute address : 0x 200

5. Click  Add .

6. Click  and select "u-boot.bin" file path:



Parameters


Image Name : u-boot  Image encrypt : ☐ Enable

Image Type : ☒ Data ☐ Enviroment ☐ uBoot

Image start offset : 0x A0000 Image execute address : 0x E00000

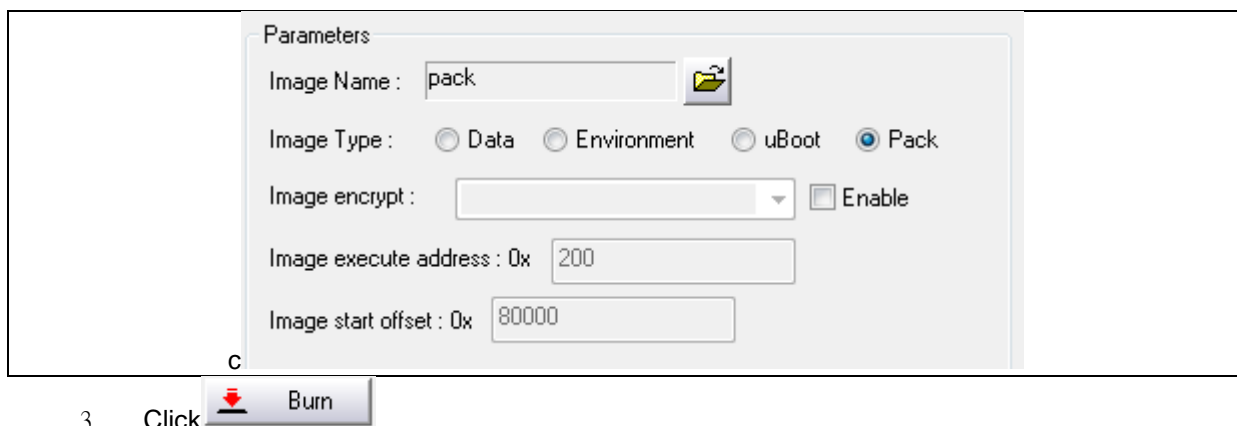
7. Click  Add .

8. Click  Output , and save the pack image.

Follow the steps below to burn pack image into NAND flash:

1. Select "NAND".

2. Click  and select pack image file path:



3. Click  Burn.

3.11 Program U-Boot

Follow the steps below to program U-Boot to NAND Flash/SPI Flash/eMMC.

Please refer to section 4.3 for the configuration and compilation of U-Boot.

3.11.1 Prepare required image.

1. u-boot.bin (Offset 0xA0000 in NAND flash, execution address 0xE00000)
2. u-boot-spl.bin (DDR execution address 0x200)
3. env.txt (Offset 0x80000 in NAND flash)

3.11.2 U-Boot Environment Variable (env.txt)

env.txt stores the environment variable used by U-Boot, here is an example:

```
baudrate=115200
bootdelay=3
ethact=emac
ethaddr=00:00:00:11:66:88
stderr=serial
stdin=serial
stdout=serial
```

Each line stores one environment variable, the format is:

<variable>=<value>

There should left no space between <variable>, =, and <value>, and use 0x0d, 0x0a to represent new line.

Please refer to 4.6.2 for the meaning of each environment variable.

3.11.3 Program to NAND Flash

1. Select "NAND" mode/

2. Select u-boot-spl.bin, set **image type** as **uBoot**, set **image execute address** as 0x200, press **burn** button.
3. Select u-boot.bin set **image type** as **Data**, set **image start offset** as 0x100000, press **burn** button.
4. Select env.txt, set **image type** as **Environment**, set **image start offset** as 0x80000, press **burn** button.

3.11.4 Program to SPI Flash

1. Select “SPI” mode.
2. Select u-boot.bin, set **image type** as **uBoot**, set **image execute address** as 0xE00000, press **burn** button
(**image execute address** is adjustable, refer to section 4.3.3)
3. Select env.txt, set **image type** as **Environment** , set **image start offset** as 0x80000 , press **burn** button

3.11.5 Program to eMMC

1. Select “eMMC” mode.
2. Select u-boot.bin, set **image type** as **uBoot**, set **image execute address** as 0xE00000, press **burn** button
(**image execute address** is adjustable, refer to section 4.3.3)
3. Select env.txt, set **image type** as **Environment** , set **image start offset** as 0x80000 , press **burn** button

3.12 Nu-Writer Trouble Shooting

Nu-Writer is developed using Microsoft visual C++ 2008. So if the NuWriter cannot execute, it is likely the operating system lack a component named “Microsoft Visual C++ 2008 Redistributable”. You can download and install this component from Microsoft’s website.

4 U-Boot user manual

The U-Boot utility is a multi-platform, open-source, universal boot-loader with comprehensive support for loading and managing boot images, such as the Linux kernel. It supports the following features:

- Network download: TFTP, BOOTP, DHCP
- Serial download: s-record, binary (via Kermit)
- Flash management: erase, read, update, yaffs2
- Flash types: SPI flash, NAND flash
- Memory utilities: dump, compare, copy, write
- Interactive shell: commands with scripting features

NUC970 U-Boot version is 201304RC2. It is downloaded from <http://www.denx.de/wiki/U-Boot/SourceCode>

To know more detailed description of U-Boot can visit U-Boot official website <http://www.denx.de/wiki/view/DULG/UBoot>

4.1 Configuration

U-Boot is configurable by modifying the definitions in configuration file.

NUC970 configuration file is located in include/configs/nuc970_evb.h

Below are the definitions in nuc970_evb.h.

```
#define CONFIG_SYS_LOAD_ADDR      0x8000
#define CONFIG_EXT_CLK            12000000 /* 12 MHz crystal */
#define CONFIG_TMR_DIV            120      /* timer prescaler */
#define CONFIG_SYS_HZ             1000
#define CONFIG_SYS_MEMTEST_START  0xA00000
#define CONFIG_SYS_MEMTEST_END    0xB00000

#define CONFIG_ARCH_CPU_INIT
#undef CONFIG_USE_IRQ

#define CONFIG_CMDLINE_TAG        1          /* enable passing of ATAGs */
#define CONFIG_SETUP_MEMORY_TAGS  1
#define CONFIG_INITRD_TAG         1
#define CONFIG_SETUP_MEMORY_TAGS  1
#define CONFIG_NUC970_HW_CHECKSUM
#define CONFIG_CMD_TIMER
```

- CONFIG_SYS_LOAD_ADDR: the load address for downloading image
- CONFIG_EXT_CLK: external crystal clock rate
- CONFIG_TMR_DIV: timer timer pre-scale
- CONFIG_SYS_HZ: timer frequency
- CONFIG_SYS_MEMTEST_START: start address of memory test
- CONFIG_SYS_MEMTEST_END: end address of memory test

- CONFIG_NUC970_HW_CHECKSUM: Use SHA-1 to calculate the checksum of Linux kernel (otherwise, use crc32 to calculate checksum) , It should cooperate with mkimage tool , Please reference chapter 0 .
- CONFIG_CMD_TIMER: Use timer relative command

```
#define CONFIG_SYS_USE_SPIFLASH
#define CONFIG_SYS_USE_NANDFLASH
#define CONFIG_ENV_IS_IN_NAND
//#define CONFIG_ENV_IS_IN_SPI_FLASH
//#define CONFIG_ENV_IS_IN_MMC

#define CONFIG_BOARD_EARLY_INIT_F
#define CONFIG_BOARD_LATE_INIT

#define CONFIG_NUC970_WATCHDOG
#define CONFIG_HW_WATCHDOG

#define CONFIG_DISPLAY_CPUINFO
#define CONFIG_BOOTDELAY          3
#define CONFIG_SYS_SDRAM_BASE    0
#define CONFIG_NR_DRAM_BANKS     2
#define CONFIG_SYS_INIT_SP_ADDR  0xBC008000
#define CONFIG_BAUDRATE          115200
#define CONFIG_SYS_BAUDRATE_TABLE {115200, 57600, 38400}

#define CONFIG_NUC970_ETH0
//#define CONFIG_NUC970_ETH1
#define CONFIG_CMD_NET
#define CONFIG_NUC970_ETH
#define CONFIG_NUC970_PHY_ADDR    1
#define CONFIG_ETHADDR           00:00:00:11:66:88
#define CONFIG_SYS_RX_ETH_BUFFER 16 // default is 4, set to 16 here.
#define CONFIG_NUC970_CONSOLE

#define CONFIG_SYS_ICACHE_OFF
#define CONFIG_SYS_DCACHE_OFF
```

- CONFIG_SYS_USE_SPIFLASH: Use SPI flash
- CONFIG_SYS_USE_NANDFLASH: Use NAND flash
- CONFIG_ENV_IS_IN_NAND: Environment variables are stored in NAND flash
- CONFIG_ENV_IS_IN_SPI_FLASH: Environment variables are stored in SPI flash
- CONFIG_ENV_IS_IN_MMC: Environment variables are stored in eMMC
- CONFIG_NUC970_WATCHDOG: Compile NUC970 watchdog timer driver

- CONFIG_HW_WATCHDOG: Enable hardware watchdog timer function (Enable CONFIG_NUC970_WATCHDOG at the same time)
- CONFIG_DISPLAY_CPUINFO: Display CPU relative information
- CONFIG_BOOTDELAY: default boot delay time
- CONFIG_SYS_INIT_SP_ADDR: the stack pointer during system initialization
- CONFIG_BAUDRATE: UART baud rate
- CONFIG_NUC970_EMAC0: Use NUC970 EMAC0
- CONFIG_NUC970_EMAC1: Use NUC970 EMAC1
- CONFIG_NUC970_ETH: Support NUC970 Ethernet
- CONFIG_NUC970_PHY_ADDR: PHY address
- CONFIG_CMD_NET: support network relative commands
- CONFIG_ETHADDR: MAC address
- CONFIG_SYS_RX_ETH_BUFFER: the number of Rx Frame Descriptors

```

/*
 * BOOTP options
 */
#define CONFIG_BOOTP_BOOTFILESIZE      1
#define CONFIG_BOOTP_BOOTPATH          1
#define CONFIG_BOOTP_GATEWAY           1
#define CONFIG_BOOTP_HOSTNAME          1
#define CONFIG_BOOTP_SERVERIP /* tftp serverip not overruled by dhcp server */
/*
 * Command line configuration.
 */
#include <config_cmd_default.h>

#undef CONFIG_CMD_LOADS
#undef CONFIG_CMD_SOURCE

#define CONFIG_CMD_PING                1
#define CONFIG_CMD_DHCP                1
#define CONFIG_CMD_JFFS2               1

```

- CONFIG_BOOTP_SERVERIP: TFTP server IP not overruled by DHCP server.
- CONFIG_CMD_PING: Use ping command
- CONFIG_CMD_DHCP: Use DHCP command
- CONFIG_CMD_JFFS2: Support JFFS2 command

```

#ifdef CONFIG_SYS_USE_NANDFLASH
#define CONFIG_NAND_NUC970
#define CONFIG_CMD_NAND                1
#define CONFIG_CMD_UBI                  1
#define CONFIG_CMD_UBIFS                1
#define CONFIG_CMD_MTDPARTS             1
#define CONFIG_MTD_DEVICE               1
#define CONFIG_MTD_PARTITIONS           1

```

```
#define CONFIG_RBTREE          1
#define CONFIG_LZO             1
#define MTDIDS_DEFAULT "nand0=nand0"
#define MTDPARTS_DEFAULT "mtdparts=nand0:0x200000@0x0(u-
boot),0x1400000@0x200000(kernel),-(user)"
#define MTD_ACTIVE_PART "nand0,2"
#define CONFIG_SYS_MAX_NAND_DEVICE    1
#define CONFIG_SYS_NAND_BASE          0xB000D000
#ifdef CONFIG_ENV_IS_IN_NAND
#define CONFIG_ENV_OFFSET              0x80000
#define CONFIG_ENV_SIZE                0x10000
#define CONFIG_ENV_SECT_SIZE          0x20000
#define CONFIG_ENV_RANGE              (4 * CONFIG_ENV_SECT_SIZE) /* Env range : 0x80000 ~
0x100000 */
#define CONFIG_ENV_OVERWRITE
#endif
#endif
```

```
#define CONFIG_SYS_NAND_U_BOOT_OFFS  (0x100000) /* Offset to RAM U-Boot image */
/* total memory available to uboot */
#define CONFIG_SYS_UBOOT_SIZE        (1024 * 1024)

#ifdef CONFIG_NAND_SPL
/* base address for uboot */
#define CONFIG_SYS_PHY_UBOOT_BASE    (CONFIG_SYS_SDRAM_BASE + 0xE00000)

#define CONFIG_SYS_NAND_U_BOOT_DST   CONFIG_SYS_PHY_UBOOT_BASE /*
NUB load-addr */
#define CONFIG_SYS_NAND_U_BOOT_START CONFIG_SYS_NAND_U_BOOT_DST /*
NUB start-addr */

#define CONFIG_SYS_NAND_U_BOOT_SIZE  (500 * 1024) /* Size of RAM U-Boot image */

/* NAND chip page size */
#define CONFIG_SYS_NAND_PAGE_SIZE    2048
/* NAND chip block size */
#define CONFIG_SYS_NAND_BLOCK_SIZE   (128 * 1024)
/* NAND chip page per block count */
#define CONFIG_SYS_NAND_PAGE_COUNT   64

#endif //CONFIG_NAND_SPL
```

- CONFIG_NAND_NUC970: Enable NUC970 NAND function
- CONFIG_CMD_NAND: Use nand command
- CONFIG_MTD_DEVICE: Enable MTD device
- CONFIG_MTD_PARTITIONS: Enable MTD partition
- CONFIG_CMD_UBI: Enable UBI
- CONFIG_CMD_UBIFS: Enable UBIFS file system
- CONFIG_CMD_MTDPARTS: Use MTD partition command.
- CONFIG_RBTREE: Enable the configuration UBI need
- CONFIG_LZO: Enable the configuration UBI need
- MTDIDS_DEFAULT: Set MTD ID name, it need to be the same as Linux kernel.
- MTDPARTS_DEFAULT: MTD partition configuration
- CONFIG_SYS_MAX_NAND_DEVICE: Maximum NAND device
- CONFIG_SYS_NAND_BASE: NAND controller base address
- CONFIG_ENV_OFFSET: flash offset address that environment variables are stored.
- CONFIG_ENV_SIZE: The space reserved for environment variables
- CONFIG_ENV_SECT_SIZE: The sector size of flash that environment variables are stored.
- CONFIG_ENV_RANGE: The range of environment variables, from CONFIG_ENV_OFFSET to CONFIG_ENV_OFFSET + CONFIG_ENV_RANGE. (When the block is a bad block, U-Boot will store environment variables to next block.)
- CONFIG_SYS_NAND_U_BOOT_OFFS: The NAND flash offset address that U-Boot stores.
- CONFIG_SYS_UBOOT_SIZE: U-Boot total space (code + data + heap)
- CONFIG_SYS_PHY_UBOOT_BASE: U-Boot execution address
- CONFIG_SYS_NAND_U_BOOT_SIZE: U-Boot image size
- CONFIG_SYS_NAND_PAGE_SIZE: NAND flash page size
- CONFIG_SYS_NAND_BLOCK_SIZE: NAND flash block size
- CONFIG_SYS_NAND_PAGE_COUNT: The page count per NAND flash block

```

/* SPI flash test code */
#ifdef CONFIG_SYS_USE_SPIFLASH
#define CONFIG_SYS_NO_FLASH 1
// #define CONFIG_SYS_MAX_FLASH_SECT 256
// #define CONFIG_SYS_MAX_FLASH_BANKS 1
#define CONFIG_NUC970_SPI 1
#define CONFIG_CMD_SPI 1
#define CONFIG_CMD_SF 1
#define CONFIG_SPI 1
#define CONFIG_SPI_FLASH 1
// #define CONFIG_SPI_FLASH_MACRONIX 1
#define CONFIG_SPI_FLASH_WINBOND 1
#define CONFIG_SPI_FLASH_EON 1
#ifdef CONFIG_ENV_IS_IN_SPI_FLASH
#define CONFIG_ENV_OFFSET 0x80000
#define CONFIG_ENV_SIZE 0x10000
#define CONFIG_ENV_SECT_SIZE 0x10000
#define CONFIG_ENV_OVERWRITE
#endif
#endif

```

- CONFIG_CMD_SF: Use SPI flash sf command.
- CONFIG_SPI_FLASH_MACRONIX: Use MACRONIX SPI flash
- CONFIG_SPI_FLASH_WINBOND: Use Winbond SPI flash
- CONFIG_SPI_FLASH_EON: Use EON SPI flash
- CONFIG_ENV_OFFSET: The offset of flash that environment variables are stored
- CONFIG_ENV_SIZE: The space reserved for environment variables

```
#define CONFIG_SYS_PROMPT      "U-Boot> "
#define CONFIG_SYS_CBSIZE      256
#define CONFIG_SYS_MAXARGS     16
#define CONFIG_SYS_PBSIZE      (CONFIG_SYS_CBSIZE +
sizeof(CONFIG_SYS_PROMPT) + 16)
#define CONFIG_SYS_LONGHELP    1
#define CONFIG_CMDLINE_EDITING 1
#define CONFIG_AUTO_COMPLETE
#define CONFIG_SYS_HUSH_PARSER
#define CONFIG_SYS_PROMPT_HUSH_PS2 "> "
```

- CONFIG_SYS_PROMPT: Show prompt message
- CONFIG_SYS_LONGHELP: Display detailed help message.
- CONFIG_CMDLINE_EDITING: Permit command line editing.

```
/* Following block is for LCD support */
#define CONFIG_LCD
#define CONFIG_NUC970_LCD
#define LCD_BPP                LCD_COLOR16
#define CONFIG_LCD_LOGO
#define CONFIG_LCD_INFO
#define CONFIG_LCD_INFO_BELOW_LOGO
#define CONFIG_SYS_CONSOLE_IS_IN_ENV
#define CONFIG_SYS_CONSOLE_OVERWRITE_ROUTINE
```

- CONFIG_LCD: Enable LCD
- CONFIG_NUC970_LCD: Compile NUC970 driver
- LCD_BPP: The number of bits per pixel output to LCD.
- CONFIG_LCD_LOGO: Show the LOGO to LCD
- CONFIG_LCD_INFO: Show U-Boot version and NUC970 relative information to LCD.
- CONFIG_LCD_INFO_BELOW_LOGO: Show NUC970 relative information below the LOGO.
- CONFIG_SYS_CONSOLE_IS_IN_ENV: stdin/stdout/stderr use the setting of environment variables
- CONFIG_SYS_CONSOLE_OVERWRITE_ROUTINE: stdin/stdout/stderr switch to serial port

```
/* Following block is for MMC support */
#define CONFIG_NUC970_MMC
#define CONFIG_CMD_MMC
```

```
#define CONFIG_CMD_FAT
#define CONFIG_MMC
#define CONFIG_GENERIC_MMC
#define CONFIG_DOS_PARTITION
#define CONFIG_NUC970_SD_PORT0
#define CONFIG_NUC970_SD_PORT1
// #define CONFIG_NUC970_EMMC /* Don't enable eMMC(CONFIG_NUC970_EMMC) and
NAND(CONFIG_NAND_NUC970) at the same time! */
#ifndef CONFIG_ENV_IS_IN_MMC
#define CONFIG_SYS_MMC_ENV_DEV 2
#define CONFIG_ENV_OFFSET 0x80000
#define CONFIG_ENV_SIZE 512
#define CONFIG_ENV_SECT_SIZE 512
#define CONFIG_ENV_OVERWRITE
#endif
```

- CONFIG_NUC970_MMC: Compile NUC970 driver
- CONFIG_CMD_MMC: Support MMC command
- CONFIG_CMD_FAT: Support FAT command
- CONFIG_MMC: Support MMC
- CONFIG_GENERIC_MMC: Support generic MMC
- CONFIG_DOS_PARTITION: Support DOS partition
- CONFIG_NUC970_SD_PORT0: Support SD port 0
- CONFIG_NUC970_SD_PORT1: Support SD port 1
- CONFIG_NUC970_EMMC: Support eMMC
- CONFIG_SYS_MMC_ENV_DEV: The MMC device number that environment variables stored
- CONFIG_ENV_OFFSET: Environment variables offset
- CONFIG_ENV_SIZE: Environment variables size
- CONFIG_ENV_SECT_SIZE: Environment variables sector size

```
/* Following block is for EHCI support*/
#if 1
#define CONFIG_CMD_USB
#define CONFIG_CMD_FAT
#define CONFIG_USB_STORAGE
#define CONFIG_USB_EHCI
#define CONFIG_USB_EHCI_NUC970
#define CONFIG_EHCI_HCD_INIT_AFTER_RESET
#define CONFIG_DOS_PARTITION
#endif
```

- CONFIG_CMD_USB: Support USB command
- CONFIG_CMD_FAT: Support FAT command
- CONFIG_USB_STORAGE: Support USB storage

- CONFIG_USB_EHCI: Support USB 2.0
- CONFIG_USB_EHCI_NUC970: Support NUC970 USB 2.0
- CONFIG_DOS_PARTITION: Support DOS partition

```
#define CONFIG_NUC970_GPIO

/*
 * Size of malloc() pool
 */
#define CONFIG_SYS_MALLOC_LEN    (1024*1024)

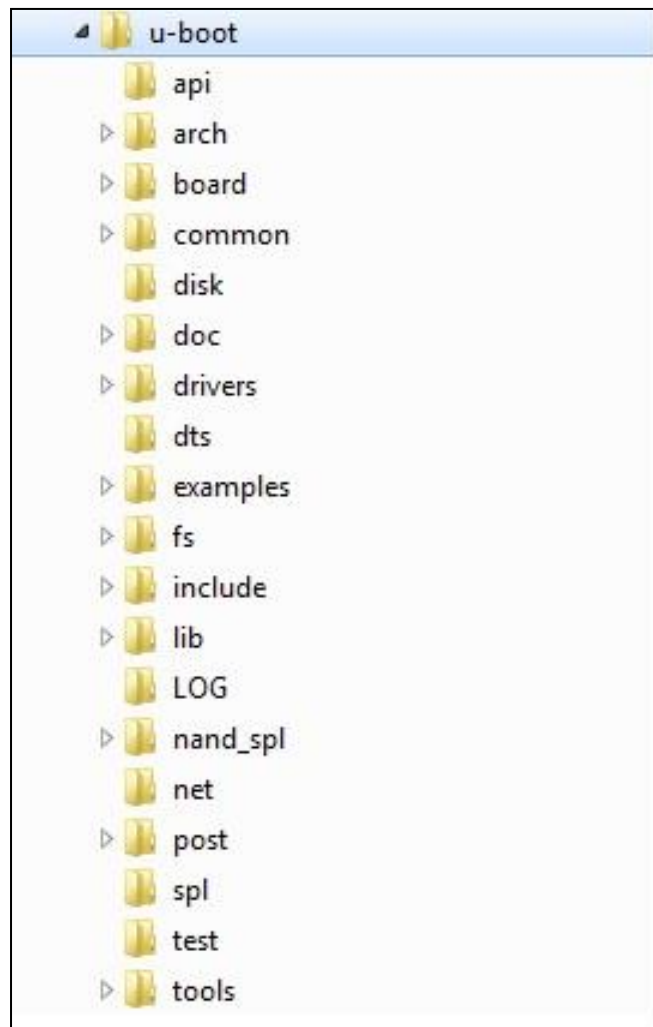
#define CONFIG_STACKSIZE        (32*1024)    /* regular stack */

#endif
```

- CONFIG_NUC970_GPIO: Enable GPIO function
- CONFIG_SYS_MALLOC_LEN: The space reserved for malloc
- CONFIG_STACKSIZE: Stack size.

4.2 Directory structure

The directory structure of U-Boot source code is as below.



- arch: This directory contains CPU relative source code. The CPU relative source code of NUC970 are under arch/arm/cpu/arm926ejs/nuc970.
- board: This directory contains board relative source code. The board relative source code of NUC970 are under board/nuvoton/nuc970_evb.
- common: This directory contains U-Boot command and other common source code.
- doc: This directory contains miscellaneous README document.
- drivers: This directory contains miscellaneous driver source code. The driver relative source code of NUC970 are under directory drivers. For instance the Ethernet driver is under drivers/net/nuc970_eth.c
- examples: This directory contains some examples. For instance, mips.lds is the linker script file for MIPS.
- fs: This directory contains miscellaneous file systems. For instance, FAT, yaffs2.
- include: This directory contains header file and configuration file. NUC970 configuration file is under include/configs/nuc970_evb.h
- lib: This directory contains miscellaneous library.
- nand_spl: This directory contains NAND boot source code.
- net: This directory contains network relative source code. For instance, tftp.c, ping.c,
- tools: This directory contains some tools. For instance, mkimage is the tool to make a image.

4.3 Compile U-Boot

4.3.1 Compile command

Clean all object code.

```
# make O=../build/nuc970_uboot/ distclean
```

Compile U-Boot

```
# make O=../build/nuc970_uboot/ nuc970_config
# make O=../build/nuc970_uboot/ all
```

(make option "O" designates object code will be generated to which directory)

If you don't need SPL U-Boot (for NAND boot) , the compile command are as below.

```
# make O=../build/nuc970_uboot/ distclean
# make O=../build/nuc970_uboot/ nuc970_nonand_config
# make O=../build/nuc970_uboot/ all
```

(make option "O" designates object code will be generated to which directory)

4.3.2 Output file after compilation

If you compile successfully, you can get Main U-Boot and SPL U-Boot:

Main U-Boot : Full function U-Boot

SPL U-Boot : Move Main U-Boot from NAND flash to DDR and boot Main U-Boot

SPL U-Boot : It's only for NAND boot ; SPI boot and eMMC boot need Main U-Boot only.

Main U-Boot and SPL U-Boot are generated in root directory and sub-directory nand_spl:

Main U-Boot files are generated in root directory.

- u-boot - Elf executable file (for download with GDB or IDE)
- u-boot.bin - binary file (You can use Nu-Writer to burn it to NAND/SPI flash 、eMMC , Please reference 3.11)
- u-boot.map –Linker memory map file

SPL U-Boot files are generated in sub-directory nand_spl

- u-boot-spl - Elf executable file (for download with GDB or IDE)
- u-boot-spl.bin - binary file (You can use Nu-Writer to burn it to NAND/SPI flash 、eMMC , Please reference 3.11.3)
- u-boot-spl.map –Linker memory map file

4.3.3 Main U-Boot link address

Main U-Boot link address is defined in Makefile.

Please find the following code segment

```
nuc970_config:  unconfig
    @mkdir -p $(obj)include $(obj)board/nuvoton/nuc970evb
    @mkdir -p $(obj)nand_spl/board/nuvoton/nuc970evb
    @echo "#define CONFIG_NAND_U_BOOT" > $(obj)include/config.h
    @echo "CONFIG_NAND_U_BOOT = y" >> $(obj)include/config.mk
    @echo "RAM_TEXT = 0xE00000" >> $(obj)board/nuvoton/nuc970evb/config.tmp
```

RAM_TEXT is U-Boot link address ,

The example , "RAM_TEXT = 0xE00000" , U-Boot link address is 0xE00000

If boot mode is NAND Boot , please also modify the definition in include/configs/nuc970_evb.h

```
#define CONFIG_SYS_PHY_UBOOT_BASE    (CONFIG_SYS_SDRAM_BASE + 0xE00000)
```

CONFIG_SYS_PHY_UBOOT_BASE Must be the same as RAM_TEXT in Makefile.

4.3.4 SPL U-Boot link address

SPL U-Boot link address is defined in board/nuvoton/nuc970evb/config.mk

Default address is 0x200 , if you want to modify it to other address , please find the following code segment , and replace 0x200 with new address.

```
ifndef CONFIG_NAND_SPL
CONFIG_SYS_TEXT_BASE = $(RAM_TEXT)
else
CONFIG_SYS_TEXT_BASE = 0x200
```

4.4 NAND AES secure boot example

NAND AES secure boot need Main U-Boot and SPL U-Boot , and use Nu-Writer to encrypt SPL U-Boot by AES and burn to NAND flash.

4.4.1 Compile Main U-Boot 以及 SPL U-Boot

```
# make O=../build/nuc970_uboot/ distclean
# make O=../build/nuc970_uboot/ nuc970_config
# make O=../build/nuc970_uboot/ all
```

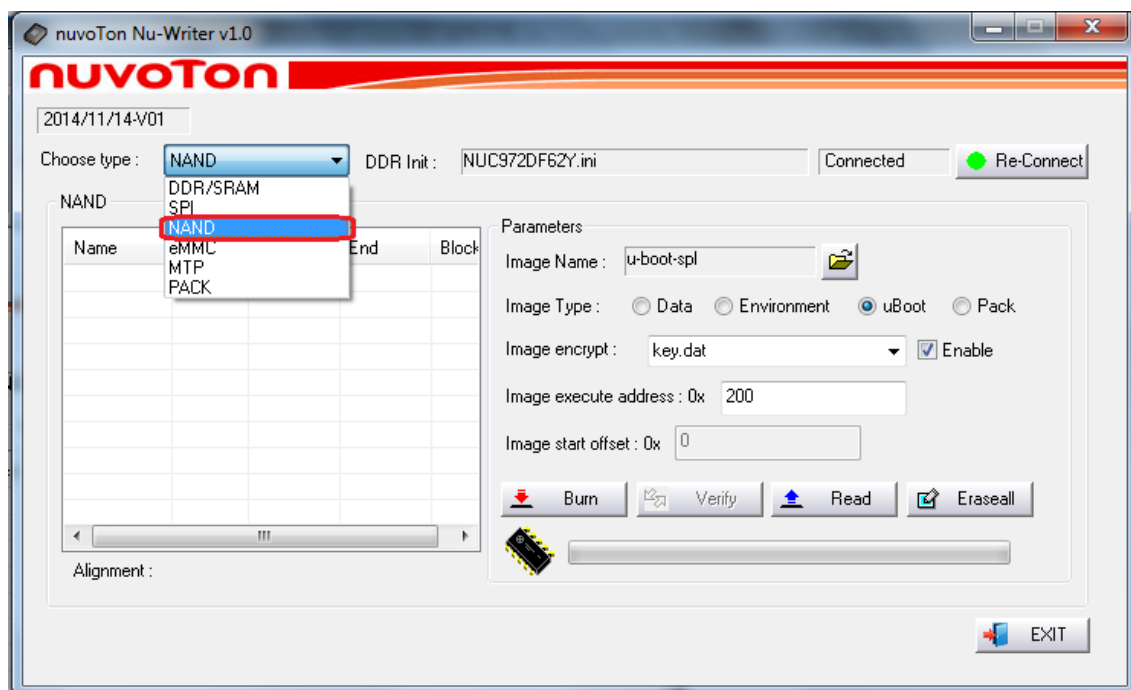
(make option "O" designates object code will be generated to which directory)

After compilation success , find out the binary file of Main U-Boot and SPL U-Boot.

- Main U-Boot binary file is generated in root directory , file name is u-boot.bin
- SPL U-Boot binary file is generated in sub-directory nand_spl , file name is u-boot-spl.bin

4.4.2 Burn SPL U-Boot

Choose type : select "NAND".



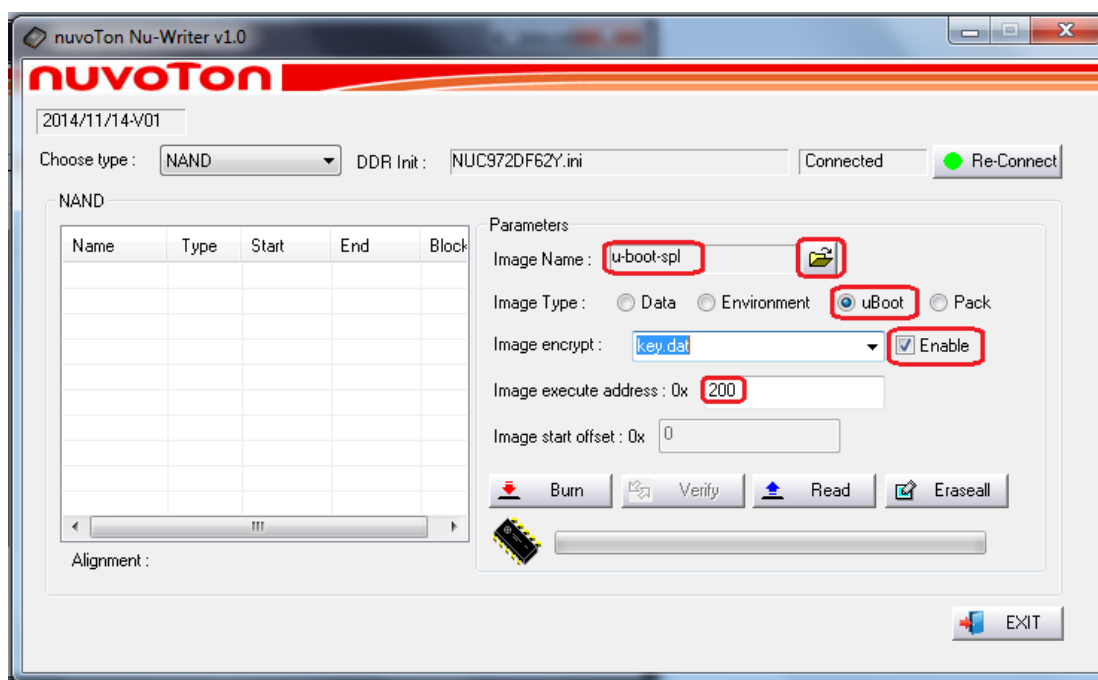
Then set Parameters as below picture ,

Image Name: select u-boot-spl.bin ,

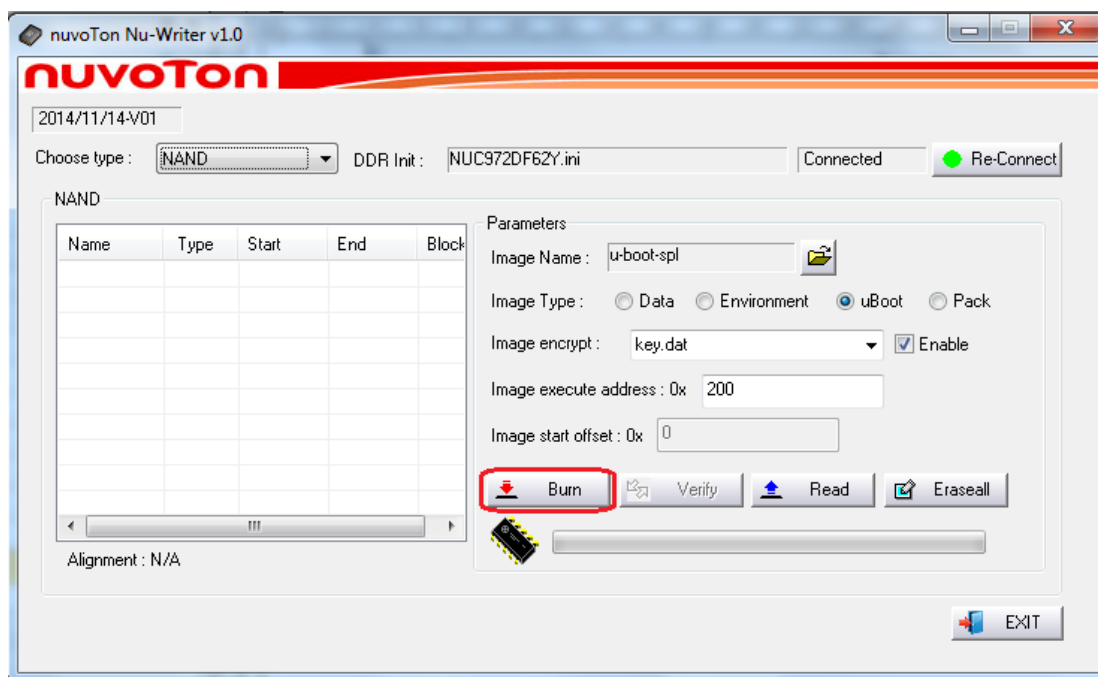
Image Type: select uBoot ,

Image encrypt: select Enable

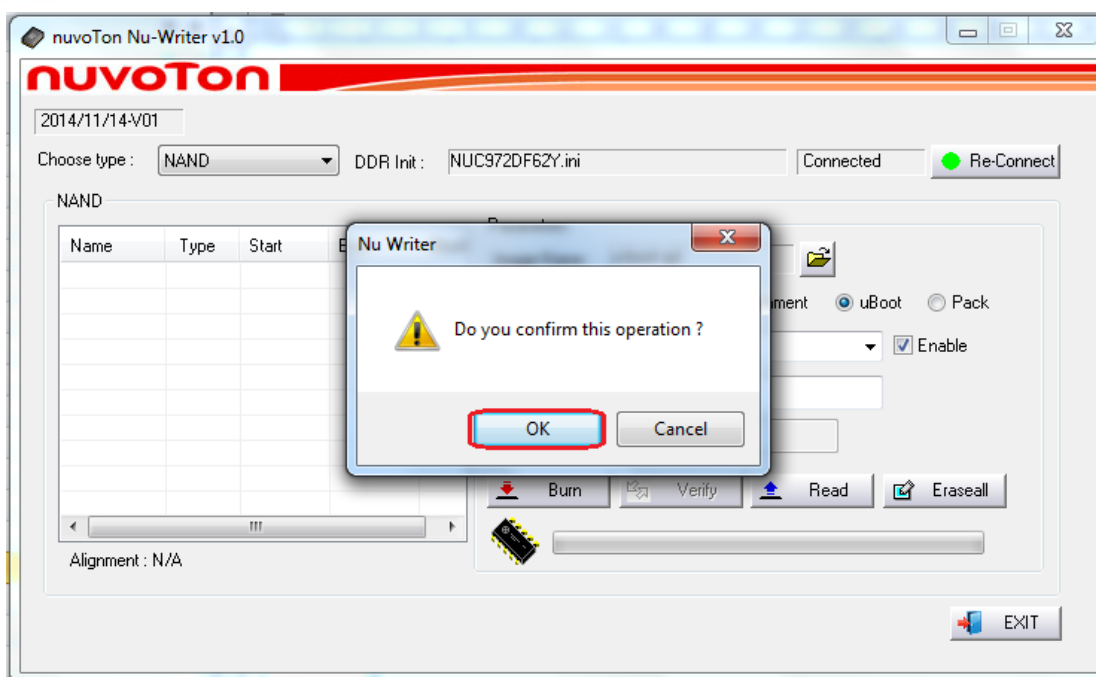
Image execute address:0x Fill in 200



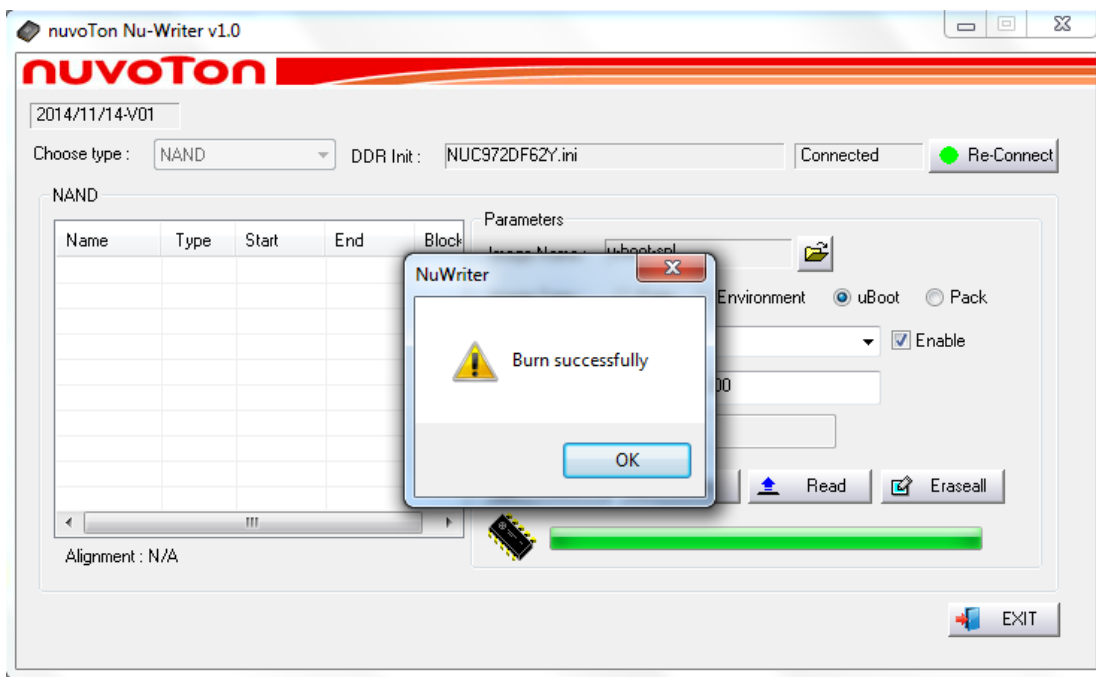
Then press “Burn” button.



There is a dialog box , choose “OK”.

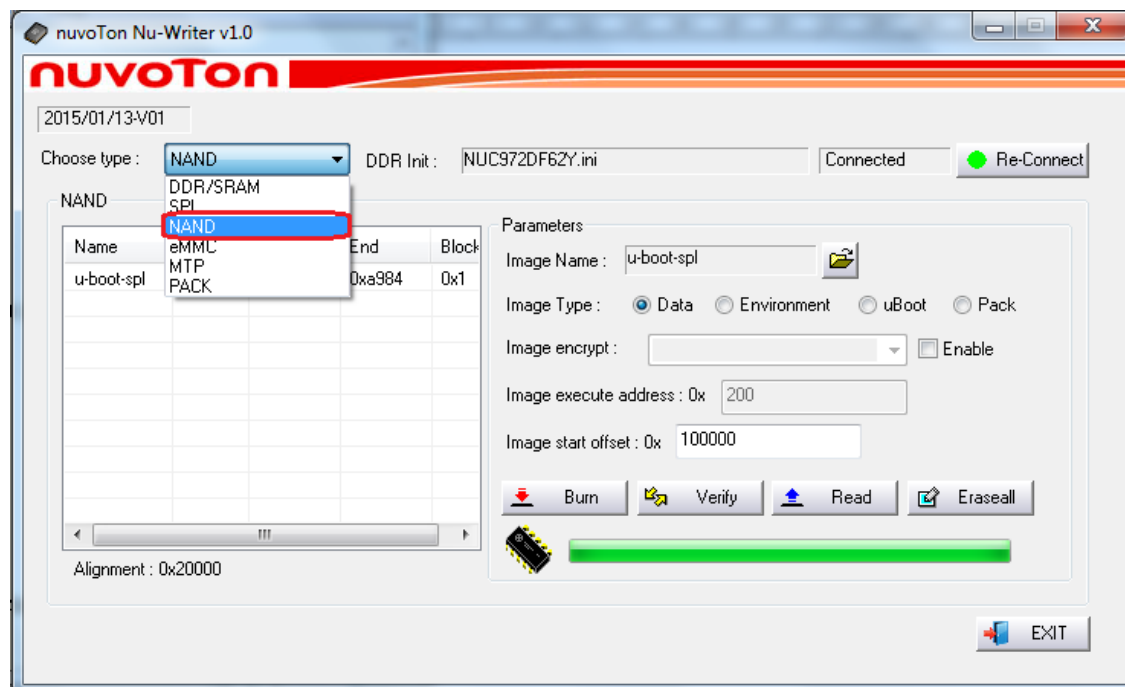


A dialog box shows burn successfully message.



4.4.3 Burn Main U-Boot

Choose type: select "NAND".

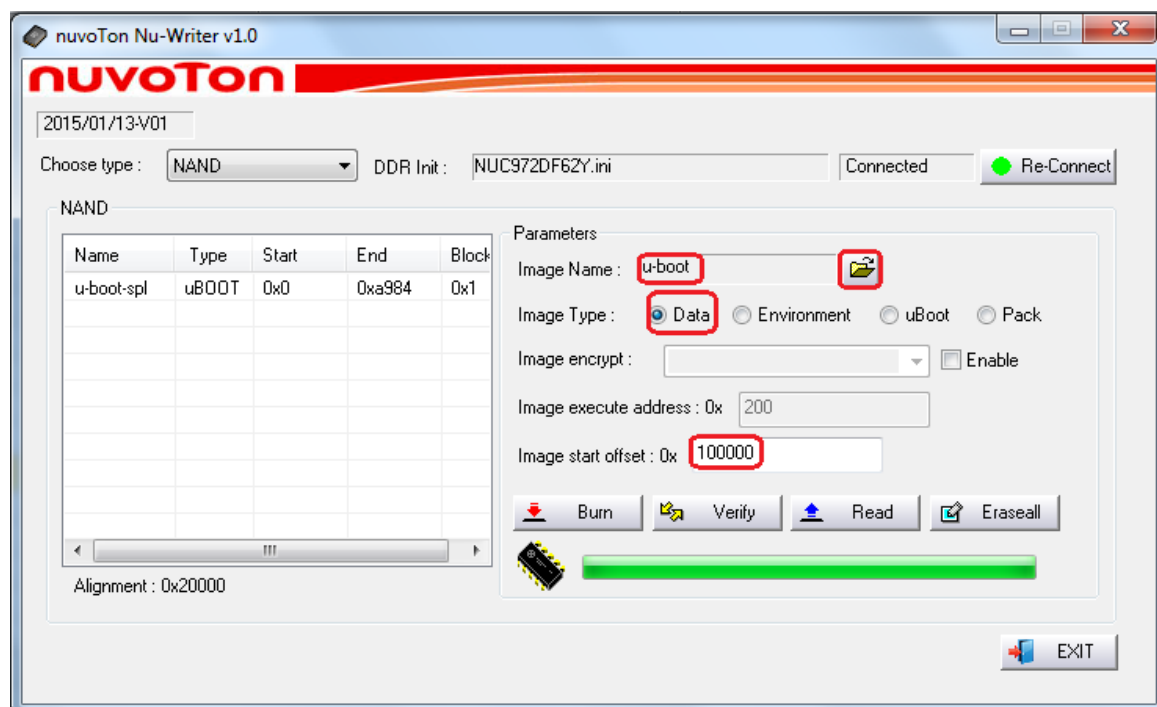


Set burning parameters.

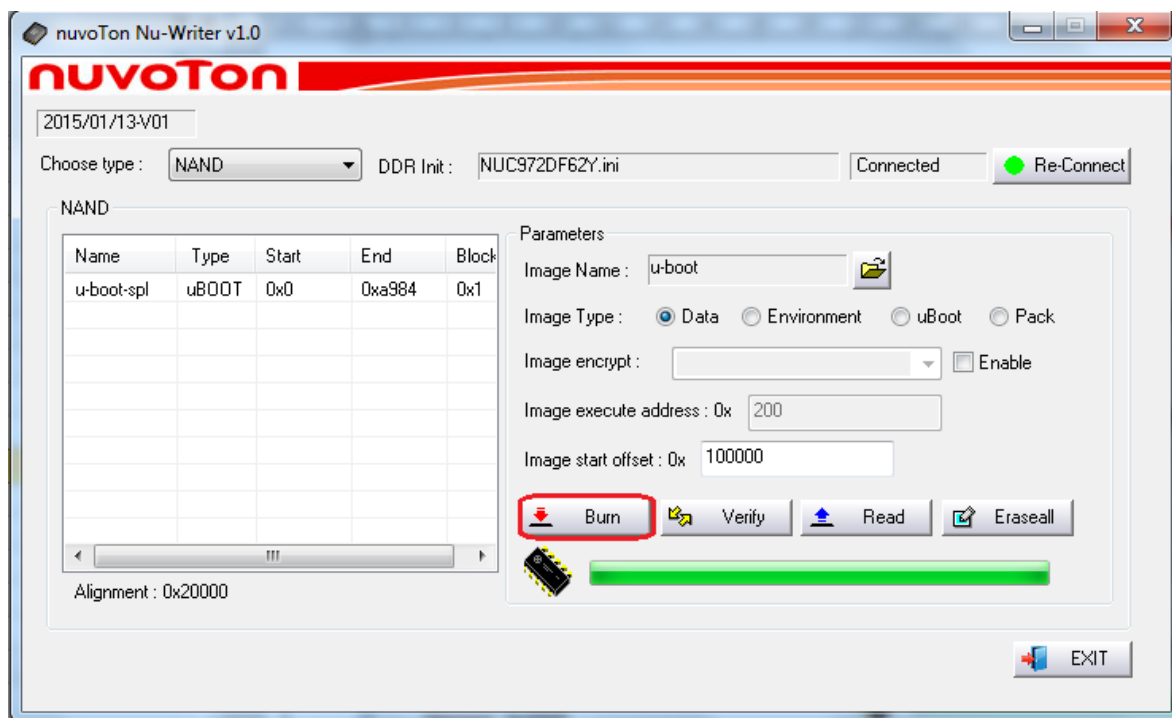
Image Name: select u-boot.bin ,

Image Type: select Data ,

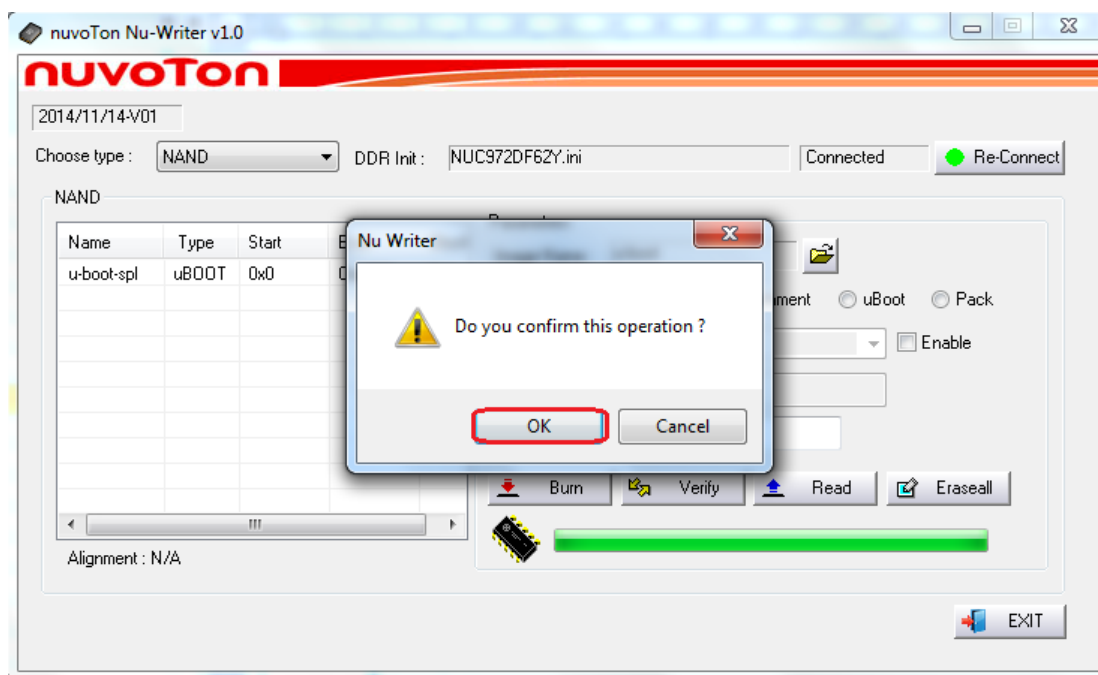
Image execute address:0x Fill in 100000



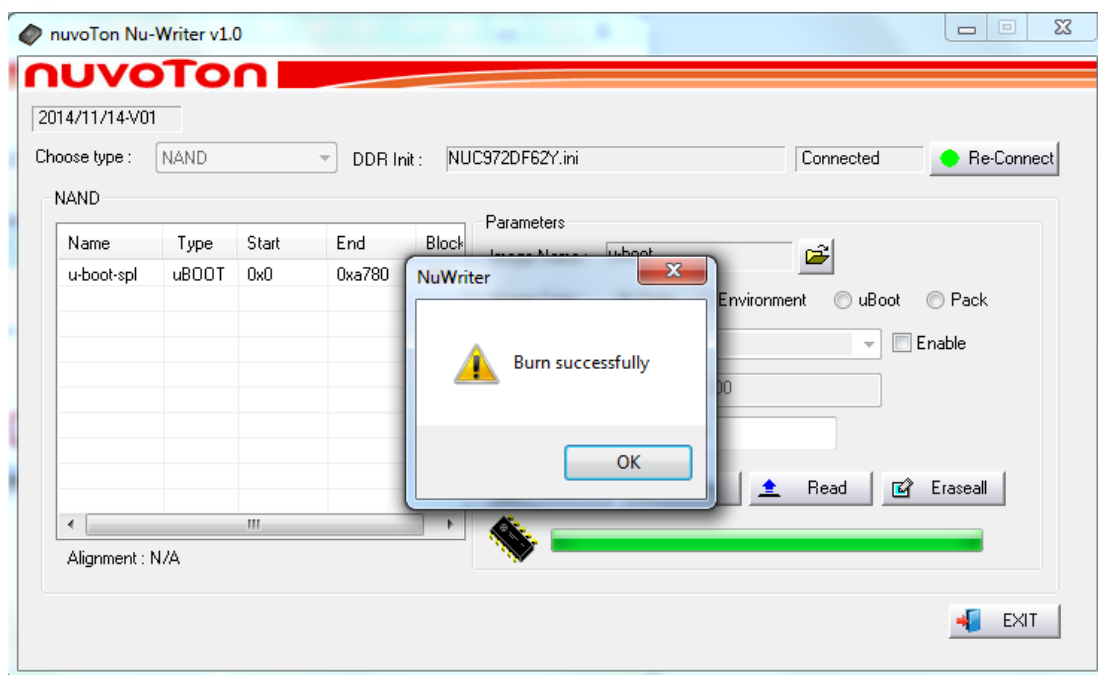
Press “Burn” button.



A dialog box ask if confirm the operation , select “OK”.

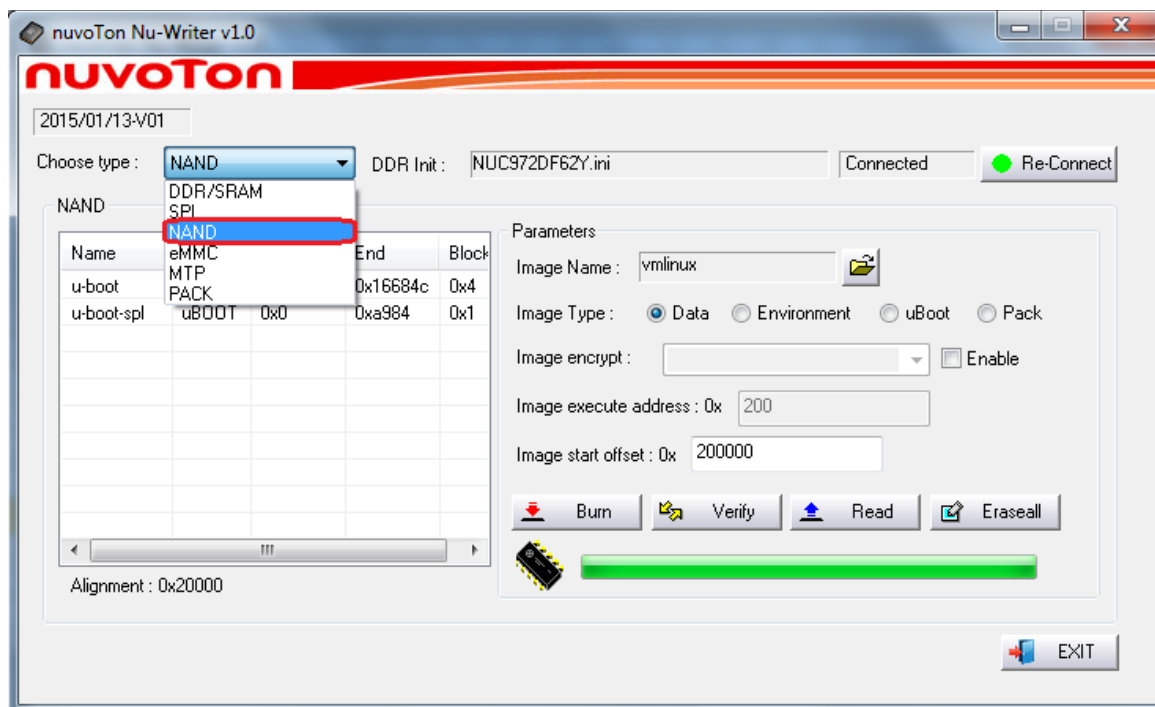


A dialog box shows burn successfully message.



4.4.4 Burn Linux kernel

Choose type : select "NAND".

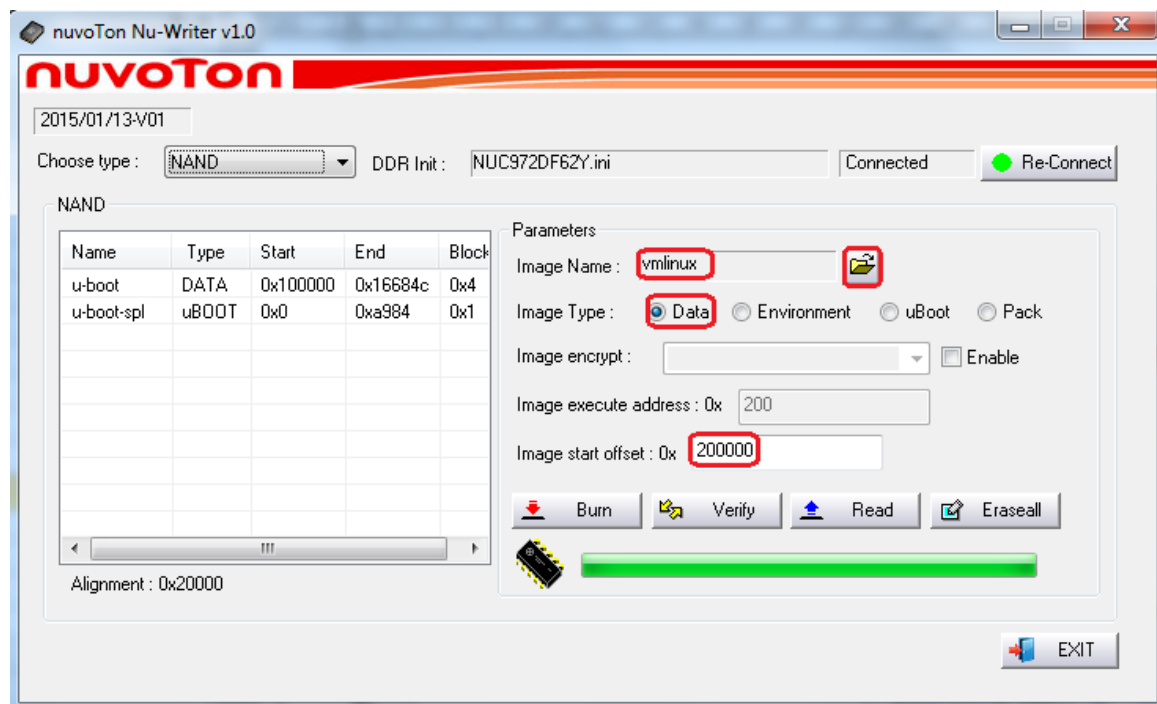


Set burning parameters.

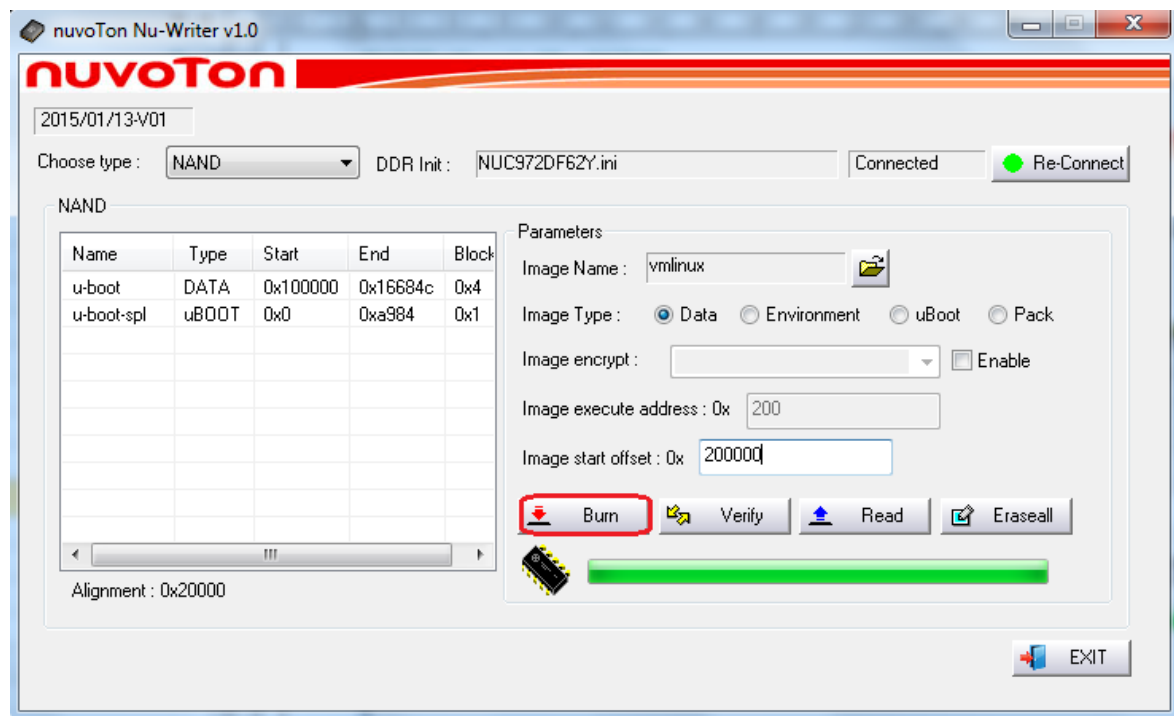
Image Name: select vmlinux.ub (Please reference 4.7 to know how to generate vmlinux.ub).

Image Type: select Data ,

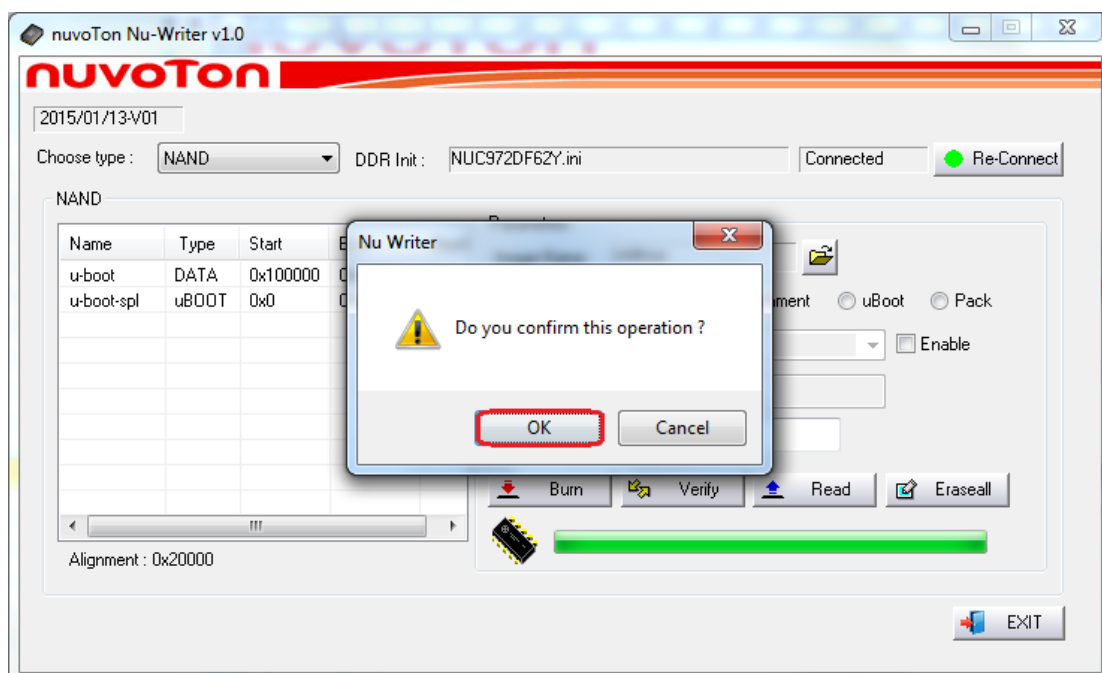
Image execute address:0x Fill in 200000



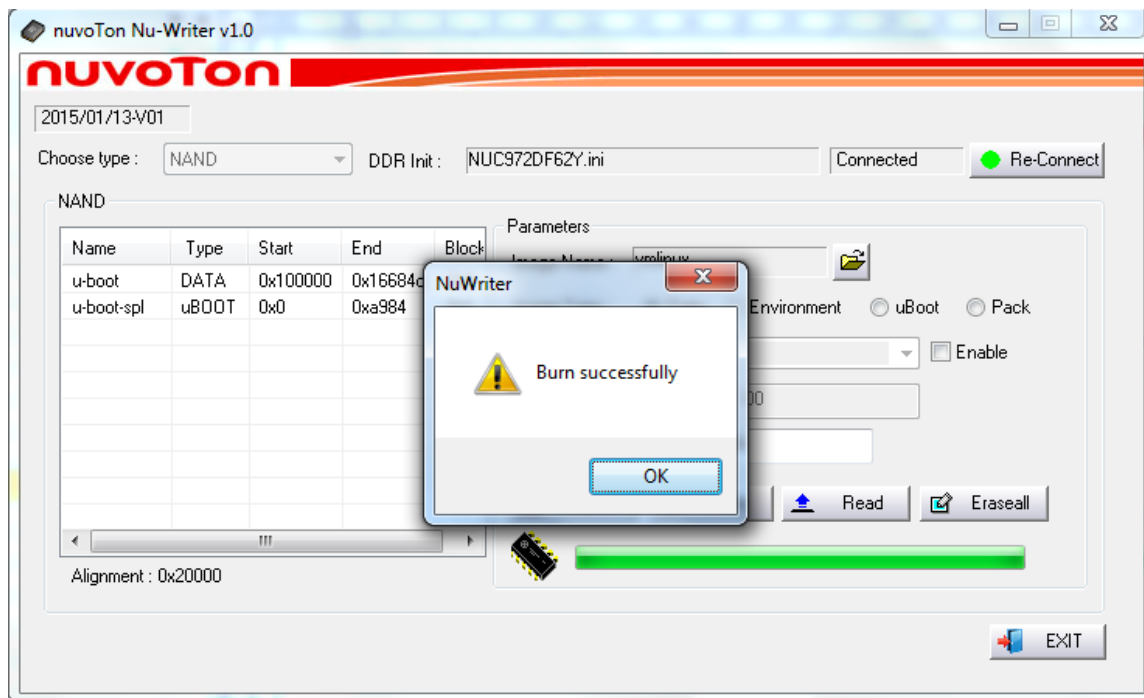
Press "Burn" button.



A dialog box ask if confirm this operation , choose "OK"



A dialog box shows burn successfully message.



4.4.5 nboot command to boot Linux kernel in NAND flash

Following example demonstrate use nboot command to read Linux kernel image stored in NAND flash offset 0x200000 to DDR address 0x7fc0. Then use bootm command to boot Linux kernel.

```

U-Boot> nboot 0x7fc0 0 0x200000

Loading from nand0, offset 0x200000
Image Name:
Image Type:  ARM Linux Kernel Image (uncompressed)
Data Size:  1639744 Bytes = 1.6 MiB
Load Address: 00007FC0
Entry Point: 00008000
U-Boot> bootm 0x7fc0
## Booting kernel from Legacy Image at 00007fc0 ...
Image Name:
Image Type:  ARM Linux Kernel Image (uncompressed)
Data Size:  1639744 Bytes = 1.6 MiB
Load Address: 00007FC0
Entry Point: 00008000
Verifying Checksum ... OK
XIP Kernel Image ... OK
OK

Starting kernel ...

```

4.5 U-Boot Command

U-boot provides a powerful command line interface which may be accessed through a terminal emulator connected to the target board's serial port. For example type "help" at the command prompt will print a list of all the available commands:

```

U-Boot> help
0    - do nothing, unsuccessfully
1    - do nothing, successfully
?    - alias for 'help'
base  - print or set address offset
bdinfo - print Board Info structure
boot  - boot default, i.e., run 'bootcmd'
bootd - boot default, i.e., run 'bootcmd'
...

```

For most commands, you do not need to type in the full command name; instead it is sufficient to type a few characters. For instance, help can be abbreviated as h. Almost all U-Boot commands expect numbers to be entered in hexadecimal input format. (Exception: for historical reasons, the sleep command takes its argument in decimal input format.)

4.5.1 Bootm command

Before describing network 、 NAND 、 SPI 、 USB 、 MMC command , I introduce bootm command .

Since Linux kernel image is stored in network 、 NAND 、 SPI 、 USB 、 MMC , we can download Linux kernel to DDR by those storage relative command , then boot Linux kernel by bootm command.

Hence , bootm command is used to boot Linux kernel image or other application program image that is generated by mkimage tool.

The command “go” we introduce in 4.5.2 , is used to boot Linux kernel binary file or other application program binary file that is “NOT” generated by mkimage tool.

bootm command format is as below:

```
U-Boot> help bootm
bootm - boot application image from memory

Usage:
bootm [addr [arg ...]]
    - boot application image stored in memory
      passing arguments 'arg ...'; when booting a Linux kernel,
      'arg' can be the address of an initrd image
```

Suppose we have downloaded Linux kernel to DDR address 0x7fc0 , then we can boot Linux kernel by bootm command.

```
U-Boot> bootm 0x7fc0
## Booting kernel from Legacy Image at 00007fc0 ...
Image Name:
Image Type: ARM Linux Kernel Image (uncompressed)
Data Size: 1639744 Bytes = 1.6 MiB
Load Address: 00007FC0
Entry Point: 00008000
Verifying Checksum ... OK
XIP Kernel Image ... OK
OK

Starting kernel ...
```

4.5.2 Go command

- go: start application

```
U-Boot> help go
go - start application at address 'addr'
```

```
Usage:
go addr [arg ...]
    - start application at address 'addr'
      passing 'arg' as arguments
```

Below is an example that run a “hello world” application program loaded at 0x100000.

```
U-Boot> go 0x100000
## Starting application at 0x00100000 ...

Hello world!
```

4.5.3 Network relative command

- ping

Transmit ICMP ECHO_REQUEST packet to network host

```
U-Boot> help ping
ping - send ICMP ECHO_REQUEST to network host

Usage:
ping pingAddress
U-Boot>
```

Before using this command, you have to set variables ipaddr that is the IP address of your platform.

Below is an example that set IP address of NUC970 to 192.168.0.101 and ping a remote PC whose IP address is 192.168.0.100

```
U-Boot> set ipaddr 192.168.0.101
U-Boot> ping 192.168.0.100
Using emac device
host 192.168.0.100 is alive
U-Boot>
```

- tftp

Download image via network using TFTP protocol.

```
U-Boot> help tftp
```

tftpboot - boot image via network using TFTP protocol

Usage:

tftpboot [loadAddress] [[hostIPAddr:]bootfilename]

U-Boot>

Before using this command, you have to set variables ipaddr and serverip.

Below is an example to download a Linux kernel image by TFTP protocol. First, set IP address of NUC970 and TFTP server to 192.168.0.101 and 192.168.0.100 respectively. Second, download Linux kernel image to address 0x200000 by TFTP protocol. Third, boot Linux kernel by command bootm.

```
U-Boot> set ipaddr 192.168.0.101
U-Boot> set serverip 192.168.0.100
U-Boot> tftp 0x7fc0 vmlinux.ub
Using emac device
TFTP from server 192.168.0.100; our IP address is 192.168.0.101
Filename 'vmlinux.ub'.
Load address: 0x7FC0
Loading: *#####
          #####
          887.7 KiB/s
done
Bytes transferred = 1639808 (190580 hex)
U-Boot> bootm 0x7FC0
## Booting kernel from Legacy Image at 007FC0 ...
   Image Name:
   Image Type:  ARM Linux Kernel Image (uncompressed)
   Data Size:  1639744 Bytes = 1.6 MiB
   Load Address: 00007FC0
   Entry Point: 00008000
   Verifying Checksum ... OK
   Loading Kernel Image ... OK
OK

Starting kernel ...
```

- dhcp

Download image via network using DHCP/TFTP protocol

U-Boot> help dhcp

dhcp - boot image via network using DHCP/TFTP protocol

```
Usage:
dhcp [loadAddress] [[hostIPAddr:]bootfilename]
U-Boot>
```

Below is an example to download Linux kernel image to address 0x7fc0 by DHCP/TFTP protocol. You don't have to set ipaddr for your platform, since DHCP server will assign an IP for you.

```
U-Boot> dhcp 0x7fc0 vmlinux.ub
BOOTP broadcast 1
*** Unhandled DHCP Option in OFFER/ACK: 7
*** Unhandled DHCP Option in OFFER/ACK: 7
DHCP client bound to address 192.168.0.102
Using emac device
TFTP from server 192.168.0.100; our IP address is 192.168.0.102; sending through gateway
192.168.0.100
Filename 'vmlinux.ub'.
Load address: 0x7fc0
Loading: *#####
#####
1 MiB/s
done
Bytes transferred = 1639808 (190580 hex)
U-Boot> bootm 0x7fc0
## Booting kernel from Legacy Image at 00007fc0 ...
Image Name:
Image Type: ARM Linux Kernel Image (uncompressed)
Data Size: 1639744 Bytes = 1.6 MiB
Load Address: 00007FC0
Entry Point: 00008000
Verifying Checksum ... OK
XIP Kernel Image ... OK
OK

Starting kernel ...
```

- bootp

Download image via network using BOOTP/TFTP protocol

```
U-Boot> help bootp
bootp - boot image via network using BOOTP/TFTP protocol

Usage:
```

```
bootp [loadAddress] [[hostIPaddr:]bootfilename]
U-Boot>
```

Below is an example to download Linux kernel image to address 0x7fc0 by BOOTP/TFTP protocol. You don't have to set ipaddr for your platform, since DHCP server will assign an IP for you.

```
U-Boot> bootp 0x7fc0 vmlinux.ub
BOOTP broadcast 1
*** Unhandled DHCP Option in OFFER/ACK: 7
*** Unhandled DHCP Option in OFFER/ACK: 7
DHCP client bound to address 192.168.0.102
Using emac device
TFTP from server 192.168.0.100; our IP address is 192.168.0.102; sending through gateway
192.168.0.100
Filename 'vmlinux.ub'.
Load address: 0x7fc0
Loading: *#####
          #####
          1 MiB/s
done
Bytes transferred = 1639808 (190580 hex)
U-Boot> bootm 0x7fc0
## Booting kernel from Legacy Image at 00007fc0 ...
   Image Name:
   Image Type:  ARM Linux Kernel Image (uncompressed)
   Data Size:  1639744 Bytes = 1.6 MiB
   Load Address: 00007FC0
   Entry Point: 00008000
   Verifying Checksum ... OK
   XIP Kernel Image ... OK
OK

Starting kernel ...
```

4.5.4 Nand flash commands

- nand: NAND Sub-system

U-Boot supports NAND flash relative commands, including nand info/device/erase/read/write.

Command format is as below:

```
U-Boot> help nand
nand - NAND sub-system
```


Usage:

```
nand info - show available NAND devices
nand device [dev] - show or set current device
nand read - addr off|partition size
nand write - addr off|partition size
    read/write 'size' bytes starting at offset 'off'
    to/from memory address 'addr', skipping bad blocks.
nand read.raw - addr off|partition [count]
nand write.raw - addr off|partition [count]
    Use read.raw/write.raw to avoid ECC and access the flash as-is.
nand erase[.spread] [clean] off size - erase 'size' bytes from offset 'off'
    With '.spread', erase enough for given file size, otherwise,
    'size' includes skipped bad blocks.
nand erase.part [clean] partition - erase entire mtd partition'
nand erase.chip [clean] - erase entire chip'
nand bad - show bad blocks
nand dump[.oob] off - dump page
nand scrub [-y] off size | scrub.part partition | scrub.chip
    really clean NAND erasing bad blocks (UNSAFE)
nand markbad off [...] - mark bad block(s) at offset (UNSAFE)
nand biterr off - make a bit error at offset (UNSAFE)
U-Boot>
```

Below example show NAND Page size/OOB size/Erase size information by nand info/device command.

```
U-Boot> nand info

Device 0: nand0, sector size 128 KiB
Page size   2048 b
OOB size    64 b
Erase size  131072 b
U-Boot> nand device

Device 0: nand0, sector size 128 KiB
Page size   2048 b
OOB size    64 b
Erase size  131072 b
U-Boot>
```

nand erase.chip erase whole chip.

```
U-Boot> nand erase.chip
```

```
NAND erase.chip: device 0 whole chip
99% complete.Erasing at 0x7fe0000 -- 100% complete.
OK
U-Boot>
```

Below is an example to write a Linux kernel image to NAND flash. The Linux kernel image is allocated at DDR 0x500000 and its size is 0x190580 bytes. We will write it to NAND flash offset 0x200000. Then, read the Linux kernel image back to DDR 0x7fc0. At last, use command, bootm, to boot Linux kernel image.

```
U-Boot> nand write 0x500000 0x200000 0x190580
```

```
NAND write: device 0 offset 0x200000, size 0x190580
1639808 bytes written: OK
```

```
U-Boot> nand read 0x7FC0 0x200000 0x190580
```

```
NAND read: device 0 offset 0x200000, size 0x190580
1639808 bytes read: OK
```

```
U-Boot> bootm 0x7FC0
```

```
## Booting kernel from Legacy Image at 007FC0 ...
```

```
Image Name:
```

```
Image Type: ARM Linux Kernel Image (uncompressed)
```

```
Data Size: 1639744 Bytes = 1.6 MiB
```

```
Load Address: 00007FC0
```

```
Entry Point: 00008000
```

```
Verifying Checksum ... OK
```

```
Loading Kernel Image ... OK
```

```
OK
```

```
Starting kernel ...
```

- nboot: boot from NAND device

Command format is as below:

```
U-Boot> help nboot
```

```
nboot - boot from NAND device
```

```
Usage:
```

```
nboot [partition] | [[[loadAddr] dev] offset]
U-Boot>
```

Below example use nboot to read Linux kernel image from NAND flash offset 0x200000 to DDR address 0x7fc0. Then boot Linux kernel by command bootm.

```
U-Boot> nboot 0x7fc0 0 0x200000

Loading from nand0, offset 0x200000
Image Name:
Image Type:  ARM Linux Kernel Image (uncompressed)
Data Size:  1639744 Bytes = 1.6 MiB
Load Address: 00007FC0
Entry Point: 00008000
U-Boot> bootm 0x7fc0
## Booting kernel from Legacy Image at 00007fc0 ...
Image Name:
Image Type:  ARM Linux Kernel Image (uncompressed)
Data Size:  1639744 Bytes = 1.6 MiB
Load Address: 00007FC0
Entry Point: 00008000
Verifying Checksum ... OK
XIP Kernel Image ... OK
OK

Starting kernel ...
```

4.5.5 SPI flash commands

U-Boot supports SPI flash relative commands including sf probe/read/write/erase/update. The command format is as below.

```
U-Boot> help sf
sf - SPI flash sub-system

Usage:
sf probe [[bus:]cs] [hz] [mode]      - init flash device on given SPI bus and chip select
sf read addr offset len - read `len' bytes starting at `offset' to memory at `addr'
sf write addr offset len - write `len' bytes from memory at `addr' to flash at `offset'
sf erase offset [+]len  - erase `len' bytes from `offset' `+len' round up `len' to block size
sf update addr offset len  - erase and write `len' bytes from memory at `addr' to flash at `offset'
U-Boot>
```

Note that you have to run command, sf probe, first before using sf read/write/erase/update.

You can designate SPI speed in argument of sf probe. Below is an example to set SPI clock to 18 MHz.

```
U-Boot> sf probe 0 18000000
```

Below is an example to read a Linux kernel image from SPI flash. First, use “sf probe” command to set SPI clock to 18 MHz. Then, “sf read” command read Linux kernel image stored at SPI flash offset 0x200000 to DDR 0x7fc0. Finally, use command, bootm, to boot Linux kernel image.

```
U-Boot> sf probe 0 18000000
SF: Detected EN25QH16-104HIP with page size 64 KiB, total 16 MiB
U-Boot> sf read 0x7FC0 0x200000 0x190580
U-Boot> bootm 0x7FC0
## Booting kernel from Legacy Image at 007FC0 ...
Image Name:
Image Type: ARM Linux Kernel Image (uncompressed)
Data Size: 1639744 Bytes = 1.6 MiB
Load Address: 00007FC0
Entry Point: 00008000
Verifying Checksum ... OK
Loading Kernel Image ... OK
OK

Starting kernel ...
```

4.5.6 Memory commands

- md: Memory display.

```
U-Boot> help md
md - memory display

Usage:
md [.b, .w, .l] address [# of objects]
U-Boot>
```

Below is an example to display the memory content from address 0x10000 to 0x100ff.

```
U-Boot> md 0x1000
00001000: bffb5c5f 5ffb56ff fcff5f41 ff67760b \....V._A_...vg.
00001010: fcd227e3 dffefeeb 70cf7cb3 dbefc7cb .'.....|.p....
00001020: fbda3e3b eb3e9ebb aa3abc95 e5fbbb2f ;>....>....:/...
```

```
00001030: ffbbb319 effe9d7d bfbeeb09 ff7b4f31  ....}.....1O{.
00001040: f7bf3973 eaff296c e6fce35e 6ffcd7f  s9..l)..^.....o
00001050: cfd28a65 8cd69f2b efecce87 677f3b8f  e...+.....;g
00001060: def67b1d deff7ece 3ffd4003 ffbf32c2  .{...~...@.?.2..
00001070: feef5b67 fdfa2e6 b7ffe1d3 efffb707  g[.....
00001080: ed2fee4b 6fd852b9 cbf765dd 796dc3de  K./..R.o.e....my
00001090: ff9fcff9 ef7bae38 efb0aff3 f8fdf324  ....8.{.....$..
000010a0: fda577b7 cfbbebcc d5936aa0 088f362f  .w.....j../6..
000010b0: ff6bae5a beff9df1 eaddd74 3de9fd3d  Z.k.....t...=.=
000010c0: dbff79bf 6f32ccf1 89bfa6b1 fbafeebf  .y....2o.....
000010d0: 77f5b6cd bd7fe7fc 6e2366f2 dff7a5fc  ...w.....f#n....
000010e0: f9ff160b edba6d61 fbf88f79 ffef7b76  ....am..y...v{..
000010f0: 3efabd8c fbfaebe2 6f7d807a ffae9ace  ...>....Z.}o....
U-Boot>
```

- mw: Memory write

```
U-Boot> help mw
mw - memory write (fill)

Usage:
mw [.b, .w, .l] address value [count]
U-Boot>
```

Below is an example to write 4 words 0s to address 0x10000.

```
U-Boot> mw 0x10000 0 4
U-Boot>
```

Display the memory content of address 0x10000. The first 4 words of address 0x10000 are 0s.

```
U-Boot> md 0x10000
00010000: 00000000 00000000 00000000 00000000  ....
00010010: e58c3004 e59c3008 e0843003 e58c3008  .0...0...0...0..
00010020: e1a01105 e1a03305 e0613003 e0833005  ....3...0a..0..
00010030: e1a02103 e0632002 e1a02102 e0862002  .!... c..!... ..
00010040: e58282d0 e58242d4 e59f3220 e0831001  ....B.. 2.....
00010050: e5913110 e58232d8 e58262c8 e3a0300c  .1...2...b...0..
00010060: e58232b4 e59f321c e5823014 e254a000  .2...2...0....T.
00010070: 0a000006 e1a02305 e0422105 e0822005  n....#...!B.. ..
00010080: e1a03102 e0623003 e1a03103 e0863003  .1...0b..1...0..
00010090: e59342d8 e51b0038 eb015a3f e1a03000  .B..8...?Z...0..
```

```
000100a0: e59f01e4 e1a01004 e1a0200a eb007cc5 ..... ..|..
000100b0: ea00005e e2813040 e1a03183 e083300e ^...@0...1...0..
000100c0: e0863003 e2832004 e5822000 e5832008 .0... ..
000100d0: e08c3001 e283308e e1a03103 e0863003 .0...0...1...0..
000100e0: e2833004 e5837000 e2811001 e2800001 .0...p.....
000100f0: e3500005 1afffee e1a03305 e0433105 ..P.....3...1C.
U-Boot>
```

- cmp: Memory compare

```
U-Boot> help cmp
cmp - memory compare

Usage:
cmp [.b, .w, .l] addr1 addr2 count
U-Boot>
```

Below is an example to compare 64 words of address 0x8000 with address 0x9000.

```
U-Boot> cmp 0x8000 0x9000 64
word at 0x00008000 (0xe321f0d3) != word at 0x00009000 (0xe59f00d4)
Total of 0 word(s) were the same
U-Boot>
```

- mtest: simple RAM read/write test

```
U-Boot> help mtest
mtest - simple RAM read/write test

Usage:
mtest [start [end [pattern [iterations]]]]
U-Boot>
```

Below is an example to test RAM read/write from address 0xa00000 to address 0xb00000 0x20 (32) iterations.

```
U-Boot> mtest 0xa00000 0xb00000 5a5a5a5a 20
Testing 00a00000 ... 00b00000:
Iteration: 32Pattern A5A5A5A5 Writing... Reading...Tested 32 iteration(s) with 0 errors.
U-Boot>
```

4.5.7 USB commands

- usb: USB sub-system

usb: USB sub-system

U-Boot> help usb

usb - USB sub-system

Usage:

usb start - start (scan) USB controller

usb reset - reset (rescan) USB controller

usb stop [f] - stop USB [f]=force stop

usb tree - show USB device tree

usb info [dev] - show available USB devices

usb storage - show details of USB storage devices

usb dev [dev] - show or set current USB storage device

usb part [dev] - print partition table of one or all USB storage devices

usb read addr blk# cnt - read `cnt' blocks starting at block `blk#'
to memory address `addr'

usb write addr blk# cnt - write `cnt' blocks starting at block `blk#'
from memory address `addr'

U-Boot>

- usb reset

U-Boot> usb reset

(Re)start USB...

USB0: USB EHCI 0.95

scanning bus 0 for devices... 2 USB Device(s) found

scanning usb for storage devices... 1 Storage Device(s) found

U-Boot>

- usb start

U-Boot> usb start

(Re)start USB...

USB0: USB EHCI 0.95

scanning bus 0 for devices... 2 USB Device(s) found

scanning usb for storage devices... 1 Storage Device(s) found

U-Boot>

- usb tree

```
U-Boot> usb tree
USB device tree:
 1 Hub (480 Mb/s, 0mA)
 | u-boot EHCI Host Controller
 |
|+-2 Mass Storage (480 Mb/s, 200mA)
     Kingston DT 101 II 0019E000B4955B8C0E0B0158

U-Boot>
```

- usb info

```
U-Boot> usb info
1: Hub, USB Revision 2.0
- u-boot EHCI Host Controller
- Class: Hub
- PacketSize: 64 Configurations: 1
- Vendor: 0x0000 Product 0x0000 Version 1.0
  Configuration: 1
- Interfaces: 1 Self Powered 0mA
  Interface: 0
- Alternate Setting 0, Endpoints: 1
- Class Hub
- Endpoint 1 In Interrupt MaxPacket 8 Interval 255ms

2: Mass Storage, USB Revision 2.0
- Kingston DT 101 II 0019E000B4955B8C0E0B0158
- Class: (from Interface) Mass Storage
- PacketSize: 64 Configurations: 1
- Vendor: 0x0951 Product 0x1613 Version 1.0
  Configuration: 1
- Interfaces: 1 Bus Powered 200mA
  Interface: 0
- Alternate Setting 0, Endpoints: 2
- Class Mass Storage, Transp. SCSI, Bulk only
- Endpoint 1 In Bulk MaxPacket 512
- Endpoint 2 Out Bulk MaxPacket 512

U-Boot>
```


- usb storage

```
U-Boot> usb storage
Device 0: Vendor: Kingston Rev: PMAP Prod: DT 101 II
Type: Removable Hard Disk
Capacity: 3875.0 MB = 3.7 GB (7936000 x 512)
U-Boot>
```

- usb dev

```
U-Boot> usb dev

USB device 0: Vendor: Kingston Rev: PMAP Prod: DT 101 II
Type: Removable Hard Disk
Capacity: 3875.0 MB = 3.7 GB (7936000 x 512)
U-Boot>
```

- usb part

```
U-Boot> usb part

Partition Map for USB device 0 -- Partition Type: DOS

Part      Start Sector  Num Sectors  UUID          Type
1         8064        7927936      1dfc1dfb-01   0b Boot
U-Boot>
```

- usb read: read `cnt` blocks starting at block `blk#` to memory address `addr`.
- usb write: write `cnt` blocks starting at block `blk#` from memory address `addr`.

Below is an example that write device 0 block #2, 1 block from 0x10000, and read back device 0 block #2, 1 block to 0x20000. Then compare the memory content of 0x10000 and 0x20000 with 1 block (512 bytes).

```
U-Boot> usb write 0x10000 2 1

USB write: device 0 block # 2, count 1 ... 1 blocks write: OK
U-Boot> usb read 0x20000 2 1

USB read: device 0 block # 2, count 1 ... 1 blocks read: OK
U-Boot>
U-Boot> cmp 0x10000 0x20000 80
Total of 128 word(s) were the same
U-Boot>
```

- usbboot: boot from USB device

```
U-Boot> help usb boot
usbboot - boot from USB device

Usage:
usbboot loadAddr dev:part
U-Boot>
```

Before using usbboot, you have to write Linux kernel image into USB device.

It can be achieved by command, usb write. However, we have to know the start block(sector) number where Linux kernel image put to. Below we use command, usb part, to show the partition map of USB device 0.

```
U-Boot> usb part

Partition Map for USB device 0 -- Partition Type: DOS

Part      Start Sector  Num Sectors  UUID          Type
  1         8064         7927936    1dfc1dfb-01   0b Boot
U-Boot>
```

The start sector (block) number is 369 (0x171). Therefore, we use command, usb write, to write Linux kernel image to device 0 block # 369(0x171). The block count can be computed as below.

The Linux kernel image is downloaded at 0x200000. It can be downloaded by ICE or TFTP or other tools. And the Linux kernel size is 1639808 bytes. $1639808/512 = 3202.75$. So, it needs 3203 (0xc83) blocks to store the Linux kernel.

```
U-Boot> usb write 0x200000 1f80 c83

USB write: device 0 block # 8064, count 3203 ... 3203 blocks write: OK
U-Boot>
```

Now, the Linux kernel is stored in device 0 block # 369(0x171). So, We can load Linux kernel from USB device 0 partition 1 by command, usbboot.

```
U-Boot> usbboot 0x7fc0 0:1

Loading from usb device 0, partition 1: Name: usbda1 Type: U-Boot
Image Name:
Image Type:  ARM Linux Kernel Image (uncompressed)
Data Size:   1639744 Bytes = 1.6 MiB
Load Address: 00007FC0
```

```

Entry Point: 00008000
U-Boot> bootm 0x7fc0
## Booting kernel from Legacy Image at 00007fc0 ...
Image Name:
Image Type: ARM Linux Kernel Image (uncompressed)
Data Size: 1639744 Bytes = 1.6 MiB
Load Address: 00007FC0
Entry Point: 00008000
Verifying Checksum ... OK
XIP Kernel Image ... OK
OK

Starting kernel ...

```

Besides, U-Boot supports command `fatls` and `fatload` that can access USB device files from file system. Below is an example that lists USB device file by command `fatls` and loads USB device file by command `fatload`.

```

U-Boot> fatls usb 0:1
1639808 vmlinux.ub

1 file(s), 0 dir(s)

U-Boot>
U-Boot> fatload usb 0:1 0x7fc0 vmlinux.ub
reading vmlinux.ub
1639808 bytes read in 90 ms (17.4 MiB/s)
U-Boot>
U-Boot> bootm 0x7fc0
## Booting kernel from Legacy Image at 00007fc0 ...
Image Name:
Image Type: ARM Linux Kernel Image (uncompressed)
Data Size: 1639744 Bytes = 1.6 MiB
Load Address: 00007FC0
Entry Point: 00008000
Verifying Checksum ... OK
XIP Kernel Image ... OK
OK

Starting kernel ...

```

4.5.8 Environment variable commands

- setenv: set environment variables

```
U-Boot> help setenv
setenv - set environment variables

Usage:
setenv [-f] name value ...
  - [forcibly] set environment variable 'name' to 'value ...'
setenv [-f] name
  - [forcibly] delete environment variable 'name'
U-Boot>
```

Below is an example to set environment variable, ipaddr, to 192.168.0.101

And use command, echo, to show the value of ipaddr.

```
U-Boot> setenv ipaddr 192.168.0.101
U-Boot> echo $ipaddr
192.168.0.101
U-Boot>
```

- saveenv: save environment variables to persistent storage.

```
U-Boot> help saveenv
saveenv - save environment variables to persistent storage

Usage:
saveenv
U-Boot>
```

- env: environment handling commands

```
U-Boot> help env
env - environment handling commands

Usage:
env default [-f] -a - [forcibly] reset default environment
env default [-f] var [...] - [forcibly] reset variable(s) to their default values
env delete [-f] var [...] - [forcibly] delete variable(s)
env edit name - edit environment variable
env export [-t | -b | -c] [-s size] addr [var ...] - export environment
env import [-d] [-t | -b | -c] addr [size] - import environment
```

```
env print [-a | name ...] - print environment
env run var [...] - run commands in an environment variable
env save - save environment
env set [-f] name [arg ...]

U-Boot>
```

4.5.9 Decrypt command

In addition to the original commands U-Boot support. NUC970 U-Boot added a command for decryption. It supports AES decryption only. The command format is as below.

```
U-Boot> help decrypt
decrypt - Decrypt image(kernel)

Usage:
decrypt decrypt aes SrcAddr DstAddr Length - Decrypt the image from SrcAddr to DstAddr with
lenth [Length].
Example : decrypt aes 0x8000 0x10000 0x200- decrypt the image from 0x8000 to 0x10000 and
lenth is 0x200

decrypt program aes EnSecure - program AES key to MTP and [Enable/Disable] secure boot.
Example : decrypt program aes 1 - program AES key to MTP and Enable secure boot.
Example : decrypt program aes 0 - program AES key to MTP but Disable secure boot.

Note that before enabling secure boot, you have to burn U-Boot with the same AES key!
Otherwise, your system will be locked!!!
```

For instance, decrypt a Linux kernel image from 0x200000 to 0x400000 with length 0x190580. The command is as below.

```
U-Boot> decrypt aes 0x200000 0x400000 0x190580
```

```
U-Boot> decrypt aes 0x800000 0x7fc0 0x190580
```

The command “decrypt program” can burn AES key to MTP , and set if enable secure boot mode or not.

For instance , if burn AES key to MTP , but "NOT" enable secure boot , command is:

```
U-Boot> decrypt program aes 0
```

if burn AES key to MTP , and also enable secure boot , command is:

```
U-Boot> decrypt program aes 1
```

Note that before enable secure boot , you have to confirm using Nu-Writer to burn U-Boot encrypted with the same AES key , otherwise , your system will be locked and boot failed when you reset or power-on again , therefore , it must be very carefully to use this command.

4.5.10 MMC command

- mmc : MMC sub-system

U-Boot support MMC relative command , include read/write/erase/list/dev.

The command format is as below.

```
U-Boot> help mmc
mmc - MMC sub system

Usage:
mmc read addr blk# cnt
mmc write addr blk# cnt
mmc erase blk# cnt
mmc rescan
mmc part - lists available partition on current mmc device
mmc dev [dev] [part] - show or set current mmc device [partition]
mmc list - lists available devices
U-Boot>
```

mmc list : list all mmc device

```
U-Boot> mmc list
mmc: 0
mmc: 1
mmc: 2
U-Boot>
```

NUC970 supports mmc device including SD port 0, SD port 1 and eMMC.

User can modify following three definitions in nuc970_evb.h according to your platform.

```
#define CONFIG_NUC970_SD_PORT0
#define CONFIG_NUC970_SD_PORT1
#define CONFIG_NUC970_EMMC
```

The default setting enables SD port 0 and SD port 1 , eMMC and NAND can not be used at the same time , NAND is disabled by default setting.

If SD port 0, SD port 1 and eMMC are all enabled , mmc device numbers are as below:

device number 0 is SD port 0

device number 1 is SD port 1

device number 2 is eMMC

If your platform supports SD port 0 and eMMC (doesn't support SD port 1) , you have to disable the definition CONFIG_NUC970_SD_PORT1 in nuc970_evb.h

The command, mmc list, can see the following result:

```
U-Boot> mmc list
mmc: 0
mmc: 1
U-Boot>
```

device number 0 is SD port 0

device number 1 is eMMC

If your platform supports eMMC (doesn't support SD port 0 and SD port 1) , you have to disable the definition CONFIG_NUC970_SD_PORT0 and CONFIG_NUC970_SD_PORT1 in nuc970_evb.h

The command, mmc list, can see the following result:

```
U-Boot> mmc list
mmc: 0
U-Boot>
```

device number 0 is eMMC

Following example is user enable SD port 0, SD port 1 and eMMC.

User set current device to SD port 1 by “mmc dev” command , then use mmcinfo command to display SD relative information.

```
U-Boot> mmc dev 1
mmc1 is current device
U-Boot> mmcinfo
Device: mmc
Manufacturer ID: 3
OEM: 5344
Name: SD02G
Tran Speed: 25000000
Rd Block Len: 512
SD version 2.0
High Capacity: No
Capacity: 1.8 GiB
Bus Width: 4-bit
U-Boot>
```

Following example is user sets current device to device 0 (SD port 0) by “mmc dev” command.

Use “mmc erase” command to erase SD card block 0x30 and 0x31, and copy data from DDR address 0x8000 to SD card block 0x30 and 0x31, then read SD card block 0x30 and 0x31 to DDR 0x500000. Finally compare the data in DDR address 0x8000 with address 0x500000 to validate SD access.

```
U-Boot> mmc dev 0
```

```

mmc0 is current device
U-Boot> mmc erase 0x30 2

MMC erase: dev # 0, block # 48, count 2 ... 2 blocks erase: OK
U-Boot> mmc write 0x8000 0x30 2

MMC write: dev # 0, block # 48, count 2 ... 2 blocks write: OK
U-Boot> mmc read 0x500000 0x30 2

MMC read: dev # 0, block # 48, count 2 ... 2 blocks read: OK
U-Boot> cmp.b 0x8000 0x500000 0x400
Total of 1024 byte(s) were the same
U-Boot>

```

Following example is user sets current device to device 2 (eMMC) by “mmc dev” command.

Use “mmc erase” command to erase SD card block 1024 to 2047, and copy data from DDR address 0x8000 to SD card block 1024~2047, then read SD card block 1024~ 2047 to DDR 0x500000. Finally compare the data in DDR address 0x8000 with address 0x500000 to validate SD access.

```

U-Boot> mmc dev 2
mmc2(part 0) is current device
U-Boot> mmc erase 0x400 0x400

MMC erase: dev # 2, block # 1024, count 1024 ... 1024 blocks erase: OK
U-Boot> mmc write 0x8000 0x400 0x400

MMC write: dev # 2, block # 1024, count 1024 ... 1024 blocks write: OK
U-Boot> mmc read 0x500000 0x400 0x400

MMC read: dev # 2, block # 1024, count 1024 ... 1024 blocks read: OK
U-Boot> cmp.b 0x8000 0x500000 0x4000
Total of 16384 byte(s) were the same
U-Boot>

```

We can access SD/eMMC card by “mmc” command , Besides , we can access the files in SD/eMMC card by “fatls” and “fatload” command.

Following example use “fatls” command to list the file in SD port 0 , then “fatload” command to read Linux kernel image (vmlinux.ub) to DDR address 0x7fc0 , finally boot Linux kernel by “bootm” command.

```

U-Boot> fatls mmc 0
1639808 vmlinux.ub
0 4gsd.txt

```



```
2 file(s), 0 dir(s)

U-Boot> fatload mmc 0 0x7fc0 vmlinux.ub
reading vmlinux.ub
1639808 bytes read in 301 ms (5.2 MiB/s)
U-Boot> bootm 0x7fc0
## Booting kernel from Legacy Image at 00007fc0 ...
   Image Name:
   Image Type:  ARM Linux Kernel Image (uncompressed)
   Data Size:  1639744 Bytes = 1.6 MiB
   Load Address: 00007FC0
   Entry Point: 00008000
   Verifying Checksum ... OK
   XIP Kernel Image ... OK
OK

Starting kernel ...
```

4.5.11 UBI commands

- mtdparts : define flash/nand partitions

U-Boot supports MTD partition relative command , including add/del/list.

```
U-Boot> help mtd
mtdparts - define flash/nand partitions

Usage:
mtdparts
  - list partition table
mtdparts delall
  - delete all partitions
mtdparts del part-id
  - delete partition (e.g. part-id = nand0,1)
mtdparts add <mtd-dev> <size>[@<offset>] [<name>] [ro]
  - add partition
mtdparts default
  - reset partition table to defaults
-----
this command uses three environment variables:
'partition' - keeps current partition identifier
partition := <part-id>
<part-id> := <dev-id>,part_num
```

```
'mtdids' - linux kernel mtd device id <-> u-boot device id mapping
mtdids=<idmap>[,<idmap>,...]
<idmap>  := <dev-id>=<mtd-id>
<dev-id>  := 'nand'|'nor'|'onenand'<dev-num>
<dev-num> := mtd device number, 0...
<mtd-id>  := unique device tag used by linux kernel to find mtd device (mtd->name)
'mtdparts' - partition list
mtdparts=mtdparts=<mtd-def>[;<mtd-def>...]
<mtd-def> := <mtd-id>:<part-def>[,<part-def>...]
<mtd-id>  := unique device tag used by linux kernel to find mtd device (mtd->name)
<part-def> := <size>[@<offset>][<name>][<ro-flag>]
<size>    := standard linux memsize OR '-' to denote all remaining space
<offset>  := partition start offset within the device
<name>    := '(' NAME ')
<ro-flag> := when set to 'ro' makes partition read-only (not used, passed to kernel)
U-Boot>
```

mtdparts list all the mtd partitions

```
U-Boot> mtdparts
device nand0 <nand0>, # parts = 3
#: name      size      offset      mask_flags
0: u-boot    0x00100000  0x00000000  0
1: kernel    0x01400000  0x00100000  0
2: user      0x06b00000  0x01500000  0
active partition: nand0,0 - (u-boot) 0x00100000 @ 0x00000000
defaults:
mtdids : nand0=nand0
mtdparts: mtdparts=nand0:0x100000@0x0(u-boot),0x1400000@0x100000(kernel),-(user)
U-Boot>
```

- ubi : ubi commands

U-Boot supports UBI relative command , including info/create/read/write.

```
U-Boot> help ubi
ubi - ubi commands
Usage:
ubi part [part] [offset]
- Show or set current partition (with optional VID header offset)
ubi info [[ayout]] - Display volume and ubi layout information
ubi create[vol] volume [size] [type] - create volume name with size
ubi write[vol] address volume size - Write volume from address with size
```

ubi read[vol] address volume [size] - Read volume to address with size

ubi remove[vol] volume - Remove volume

[Legends]

volume: character name

size: specified in bytes

type: s[tatic] or d[ynamic] (default=dynamic)

U-Boot>

ubi part : Display or set current partition

U-Boot> ubi part user

Creating 1 MTD partitions on "nand0":

0x0000001500000-0x000008000000 : "mtd=2"

UBI: attaching mtd1 to ubi0

UBI: physical eraseblock size: 131072 bytes (128 KiB)

UBI: logical eraseblock size: 126976 bytes

UBI: smallest flash I/O unit: 2048

UBI: VID header offset: 2048 (aligned 2048)

UBI: data offset: 4096

UBI: attached mtd1 to ubi0

UBI: MTD device name: "mtd=2"

UBI: MTD device size: 107 MiB

UBI: number of good PEBs: 855

UBI: number of bad PEBs: 1

UBI: max. allowed volumes: 128

UBI: wear-leveling threshold: 4096

UBI: number of internal volumes: 1

UBI: number of user volumes: 1

UBI: available PEBs: 17

UBI: total number of reserved PEBs: 838

UBI: number of PEBs reserved for bad PEB handling: 8

UBI: max/mean erase counter: 6/4

U-Boot>

ubi info : display capacity and ubi information

U-Boot> ubi info l

UBI: volume information dump:

UBI: vol_id 0

UBI: reserved_pebs 826

UBI: alignment 1

UBI: data_pad 0

```

UBI: vol_type      3
UBI: name_len     9
UBI: usable_leb_size 126976
UBI: used_ebs     826
UBI: used_bytes   104882176
UBI: last_eb_bytes 126976
UBI: corrupted    0
UBI: upd_marker   0
UBI: name         nandflash

```

UBI: volume information dump:

```

UBI: vol_id       2147479551
UBI: reserved_pebs 2
UBI: alignment    1
UBI: data_pad     0
UBI: vol_type     3
UBI: name_len     13
UBI: usable_leb_size 126976
UBI: used_ebs     2
UBI: used_bytes   253952
UBI: last_eb_bytes 2
UBI: corrupted    0
UBI: upd_marker   0
UBI: name         layout volume

```

U-Boot>

ubifsmount : mount ubifs volume

U-Boot> help ubifsmount

ubifsmount - mount UBIFS volume

Usage:

ubifsmount <volume-name>

- mount 'volume-name' volume

U-Boot> ubifsmount ubi0:nandflash

UBIFS: mounted UBI device 0, volume 0, name "nandflash"

UBIFS: mounted read-only

UBIFS: file system size: 103485440 bytes (101060 KiB, 98 MiB, 815 LEBs)

UBIFS: journal size: 5206016 bytes (5084 KiB, 4 MiB, 41 LEBs)

UBIFS: media format: w4/r0 (latest is w4/r0)

UBIFS: default compressor: LZO

UBIFS: reserved for root: 5114338 bytes (4994 KiB)

```
U-Boot>
```

ubifsls : list files in a directory

```
U-Boot> help ubifsls
ubifsls - list files in a directory
Usage:
ubifsls [directory]
    - list files in a 'directory' (default '/')
U-Boot> ubifsls
<DIR>      160  Thu Jan 01 00:08:09 1970  tt
U-Boot>
```

ubifsumount : unmount UBIFS volume

```
U-Boot> help ubifsumount
ubifsumount - unmount UBIFS volume
Usage:
ubifsumount  - unmount current volume
U-Boot> ubifsumount
Unmounting UBIFS volume nandflash!
U-Boot>
```

4.6 Environment variables

4.6.1 Environment variables configuration

Environment variables can store in NAND flash 、SPI flash or eMMC, user can modify below three definitions in nuc970_evb.h:

- CONFIG_ENV_IS_IN_NAND: environment variables are stored in NAND flash
- CONFIG_ENV_IS_IN_SPI_FLASH: environment variables are stored in SPI flash
- CONFIG_ENV_IS_IN_MMC: environment variables are stored in eMMC

Note that only one of them can be defined.

User can configure the flash offset address environment variables stored and the space reserved for environment variables in nuc970_evb.h:

- CONFIG_ENV_OFFSET: the flash offset address environment variables stored
- CONFIG_ENV_SIZE: the space reserved for environment variables

4.6.2 Default environment variables

U-Boot has some default environment variables. If the variables are not stored in flash, U-Boot will assign default value to the variables.

Below are the default environment variables.

- baudrate

Console baudrate , the value is from CONFIG_BAUDRATE in nuc970_evb.h

- bootdelay

It's the boot delay time when U-Boot run the command script in bootcmd. Its unit is second. Before it is countdown to 0, hit any key to stop running script in bootcmd.

- ethact

It sets which Ethernet interface state is active , since nuc970 ethernet driver set device name to emac , ethact can be set to emac only.

- ethaddr

Ethernet mac address. ethaddr value is from CONFIG_ETHADDR in nuc970_evb.h

- stderr

Set stderr , default value is serial

- stdin

Set stdin , default value is serial

- stdout

Set stdout , default value is serial

4.6.3 command script

Below are script relative commands

- bootcmd

Whenever U-Boot boots up , U-Boot executes the script in bootcmd.

Following example set bootcmd as : read Linux kernel from SPI flash offset 0x200000 to DDR address 0x7fc0 , and boot Linux kernel.

Remember to save the environment variables to flash.

```
U-Boot> set bootcmd sf probe 0 18000000\; sf read 0x7fc0 0x200000 0x190580\; bootm 0x7fc0
U-Boot> saveenv
Saving Environment to SPI Flash...
SF: Detected EN25QH16-104HIP with page size 64 KiB, total 16 MiB
Erasing SPI flash...Writing to SPI flash...done
U-Boot>
```

4.6.4 New added environment variable

NUC970 U-Boot adds some new environment variables.

- spimode: defines SPI transmit mode to one-bit or Quad mode.

```
U-Boot> setenv spimode mode
```

The parameter “ mode” can be 1 or 4.

1: one-bit mode

4: Quad mode

For instance, set SPI mode to Quad mode

```
U-Boot> setenv spimode 4
```

If your SPI flash does not support Quad mode, you can set spimode to one-bit mode

```
U-Boot> setenv spimode 1
```

Remember to set environment variables to flash.

```
U-Boot> saveenv
```

- watchdog: Enable or disable watchdog timer function.

```
U-Boot> setenv watchdog mode
```

The parameter "mode" can be on or off.

on: watchdog timer function enabled

off: watchdog timer function disabled

For instance, disable watchdog function

```
U-Boot> setenv watchdog off
```

Enable watchdog function

```
U-Boot> setenv watchdog on
```

Remember save the environment variables to flash.

```
U-Boot> saveenv
```

4.7 mkimage tool

U-Boot supports a number of different image formats that can be downloaded, saved to flash and executed. The types of such image files supported by U-Boot, include:

- Linux Kernel
- Script files
- Standalone binaries
- RAM disk images

These images are often referred to as a ".ub" files, as that is the file extension name that is often used to name them.

4.7.1 Use mkimage to generate Linux kernel image

The mkimage tool is located in tools/mkimage. Below is an example to encapsulate an ARM Linux kernel binary file (970image). Linux kernel download address is 0x7fc0 and execution address is 0x8000.

```
u-boot/tools# ./mkimage -A arm -O linux -T kernel -a 0x7fc0 -e 0x8000 -d 970image 970image.ub
```

```
Image Name:
Created:    Fri Aug 8 14:38:39 2014
Image Type: ARM Linux Kernel Image (uncompressed)
Data Size:  1639744 Bytes = 1601.31 kB = 1.56 MB
Load Address: 00007FC0
Entry Point: 00008000
```

- A: set CPU architecture to arm
- O: set operating system to linux
- T: set image type to kernel
- a: set load address to 0x7fc0
- e: set entry point to 0x8000
- d: set image data from 970image

4.7.2 Checksum calculation (SHA-1 or crc32)

mkimage tool calculates the checksum of Linux kernel image and puts it in the header.

The command “bootm” also calculates the checksum of Linux kernel image loaded to DDR and compares the checksum with the value in header of Linux kernel image. If they are different, it shows the error message “Verifying Checksum ... Bad Data CRC” and stop booting Linux kernel. If they are the same, it shows the message “Verifying Checksum ... OK” and continues to boot Linux kernel. The default option of mkimage tool is using software crc32 to calculate the checksum of Linux kernel image. It’s time-consuming. If you want to speed up booting Linux kernel, you can use hardware SHA-1 to calculate the checksum of Linux kernel.

NUC970 adds a new parameter “-S” to calculate Linux kernel checksum.

The original checksum calculation method of mkimage tool is crc32, NUC970 provides another option, SHA-1, below is an example uses SHA-1 to calculate Linux kernel checksum. Add option “-S sha1”. Remember to enable CONFIG_NUC970_HW_CHECKSUM in nuc970_evb.h

```
u-boot/tools# ./mkimage -A arm -O linux -T kernel -S sha1 -a 0x7fc0 -e 0x8000 -d 970image
970image.ub
Image Name:
Created:    Fri Aug 8 14:39:47 2014
Image Type: ARM Linux Kernel Image (uncompressed)
Data Size:  1639744 Bytes = 1601.31 kB = 1.56 MB
Load Address: 00007FC0
Entry Point: 00008000
```

- A: set CPU architecture to arm
- O: set operating system to linux

- T: set image type to kernel
- a: set load address to 0x7fc0
- e: set entry point to 0x8000
- S: Set checksum calculation to sha1
- d: set image data from 970image

If you select crc32 to calculate Linux kernel checksum , below is an example : Added an option “-S crc32”. Remember to disable CONFIG_NUC970_HW_CHECKSUM in nuc970_evb.h

```
u-boot/tools# ./mkimage -A arm -O linux -T kernel -S crc32 -a 0x7fc0 -e 0x8000 -d 970image
970image.ub
Image Name:
Created:   Fri Aug 8 14:39:47 2014
Image Type: ARM Linux Kernel Image (uncompressed)
Data Size: 1639744 Bytes = 1601.31 kB = 1.56 MB
Load Address: 00007FC0
Entry Point: 00008000
```

- A: set CPU architecture to arm
- O: set operating system to linux
- T: set image type to kernel
- a: set load address to 0x7fc0
- e: set entry point to 0x8000
- S: Set checksum calculation to crc32
- d: set image data from 970image

4.7.3 AES encrypt

Besides , NUC970 mkimage tool adds AES encrypt function added two new parameters,

-E AES, Encrypt image ; and -K , designate the key file that AES encryption uses. Following example encapsulate and encrypt an ARM Linux kernel image (970image) , AES key is in key.dat. Linux kernel load address is 0x7fc0 , execution address is 0x8000.

```
u-boot/tools# ./mkimage -A arm -O linux -T kernel -a 0x7fc0 -e 0x8000 -E AES -K key.dat -d
970image 970image.ub
Image Name:
Created:   Fri Aug 8 14:39:47 2014
```

```
Image Type: ARM Linux Kernel Image (uncompressed)
Data Size: 1639744 Bytes = 1601.31 kB = 1.56 MB
Load Address: 00007FC0
Entry Point: 00008000
```

- A: set CPU architecture to arm
- O: set operating system to linux
- T: set image type to kernel
- a: set load address to 0x7fc0
- e: set entry point to 0x8000
- d: set image data from 970image
- E: Set encrypt type to AES
- K: Designate the key file that AES encryption uses

There is a file named key.dat in directory tools/ , the file content is AES key. User can edit key.dat to modify key

The content of key.dat is as below:

```
0x34125243
0xc41a9087
0xa14cae80
0xc4c38790
0x672ca187
0xd4f785a1
0x562fa1c5
0x78561198
```

Each line has 4 bytes , totally 8 lines.

Each line starts with 0x , please do not modify it. If you want to modify key , please directly modify the value after 0x.

Please modify this file in Linux environment , if you edit it in Windows and copy to Linux , you can use dos2unix tool to transform key.dat , otherwise the AES key for encryption is wrong.

```
u-boot/tools$ dos2unix key.dat
dos2unix: converting file key.dat to Unix format ...
```

4.8 Security issue

4.8.1 Encrypt

To protect the security of image file and avoid the hacker, we encrypt image by mkimage tool.

Below is an example for AES encryption.

```
u-boot# ./mkimage -A arm -O linux -T kernel -a 0x8000 -e 0x8000 -E AES -K key.dat -d
vmlinux.bin vmlinux.ub
```

4.8.2 Decrypt

We can also decrypt the image in U-Boot command line , below is an example to decrypt an encrypted Linux kernel located in address 0x200000 to 0x400000 , Linux kernel size is 0x190580.

```
U-Boot> decrypt aes 0x200000 0x400000 0x190580
```

4.8.3 Risk

After the image is decrypted, hacker can use the command “md” to display the memory and know the image content.

There is a solution for this security issue :

- Disable md command ◦

Modify include/config_cmd_default.h

Remark the line #define CONFIG_CMD_MEMORY

4.9 Watchdog timer

4.9.1 Watchdog timer configuration

Modify below two definitions in nuc970_evb.h to enable NUC970 watchdog timer function.

```
#define CONFIG_NUC970_WATCHDOG
#define CONFIG_HW_WATCHDOG
```

Remark the two definitions will disable NUC970 watchdog timer function.

4.9.2 Watchdog timer environment variables

When NUC970 watchdog timer configuration is enabled , user can set environment variable “watchdog” to “off” or “0” , the watchdog timer function will be disabled without modifying configuration file or recompilation.

```
U-Boot> set watchdog off
U-Boot> saveenv
```

Set environment variable “watchdog” to “on” will enable watchdog timer function again.

```
U-Boot> set watchdog on
U-Boot> saveenv
```

After modifying the environment variable “watchdog” , remember to use command “saveenv” to save variable “watchdog” to flash.

If the configuration of watchdog timer in nuc970_evb.h is disabled , it is meaningless to modify environment variable “watchdog”.

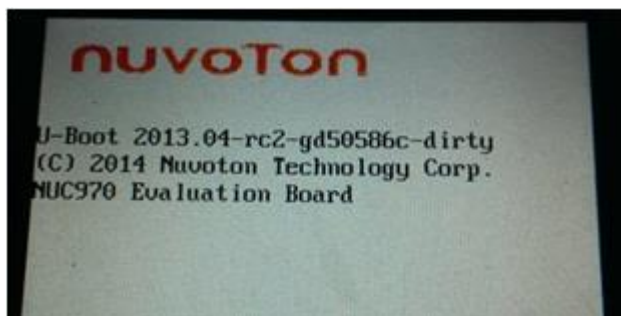
4.9.3 Watchdog timer period

When Watchdog timer function is enabled and system is idle more than 14 seconds , Watchdog timer will reset system ; Whenever use enter a command (input Enter key) , the idle time will be reset to 0.

4.10 U-Boot LCD

4.10.1 NUC970 LCD display content

During U-Boot boot up , Nuvoton LOGO and U_Boot relative information is displayed in LCD panel.



4.10.2 How to replace LOGO

To change Nuvoton LOGO to another one , if the company name is abc , LOGO file name is abc.bmp , the steps are as below:

- Put abc.bmp in tools/logos
- Modify tools/Makefile

Search the below three lines ,

```
ifeq ($(VENDOR),nuvoton)
```

```
LOGO_BMP= logos/nuvoton.bmp
```

```
endif
```

Behind the above three lines, add the following three lines

```
ifeq ($(VENDOR),abc)
```

```
LOGO_BMP= logos/abc.bmp
```

```
endif
```

4.11 GPIO

U-Boot GPIO can be used for LED .

NUC970 U-Boot provides the function to set GPIO , user can access GPIO by NUC970 GPIO driver interface.

4.11.1 NUC970 GPIO

NUC970 GPIO port includes port A ~ port J , each port has 16 pins.

Note that GPIO port C pin 15 and GPIO port J pin 5~15 are reserved , please do not use them.

NUC970 U-Boot assign each pin a GPIO number , for instance , GPIO port A pin 0 number is GPIO_PA0 , GPIO port B pin 2 number is GPIO_PB2

User has to pass GPIO number when calling NUC970 GPIO driver function.

4.11.2 GPIO driver interface

NUC970 提供以下的 GPIO APIs

```
int gpio_request(unsigned gpio, const char *label);
int gpio_direction_input(unsigned gpio);
int gpio_direction_output(unsigned gpio, int value);
int gpio_get_value(unsigned gpio);
int gpio_set_value(unsigned gpio, int value);
```

The first parameter of each API is GPIO number.

- gpio_request

Confirm GPIO is in used or not , the second parameter is not used , you can fill in 0.

If the designated GPIO pin is switched to other function (not GPIO) , there will be error message.

For instance , when we use GPIO port D0 , and calling gpio_request():

```
gpio_request(GPIO_PD0,NULL);
```

If port D0 is switched to other function , you will get below error message.

```
[gpio_request] Please Check GPIO pin [96], multi-function pins = 0x6
```

- gpio_direction_input

Set GPIO pin to input mode.

- gpio_direction_output

Set GPIO pin to output mode and output value °

- gpio_get_value

Read GPIO pin value

- gpio_set_value

Set GPIO pin output value °

4.11.3 Example

Below example set GPIO port G0 ~ port G5 output value to 0x101010 °

```
gpio_request(GPIO_PG0,NULL);
gpio_direction_output(GPIO_PG0, 0);
gpio_request(GPIO_PG1,NULL);
gpio_direction_output(GPIO_PG1, 1);
```

```
gpio_request(GPIO_PG2,NULL);
gpio_direction_output(GPIO_PG2, 0);
gpio_request(GPIO_PG3,NULL);
gpio_direction_output(GPIO_PG3, 1);
gpio_request(GPIO_PG4,NULL);
gpio_direction_output(GPIO_PG4, 0);
gpio_request(GPIO_PG5,NULL);
gpio_direction_output(GPIO_PG5, 1);
```

4.12 Network test environment

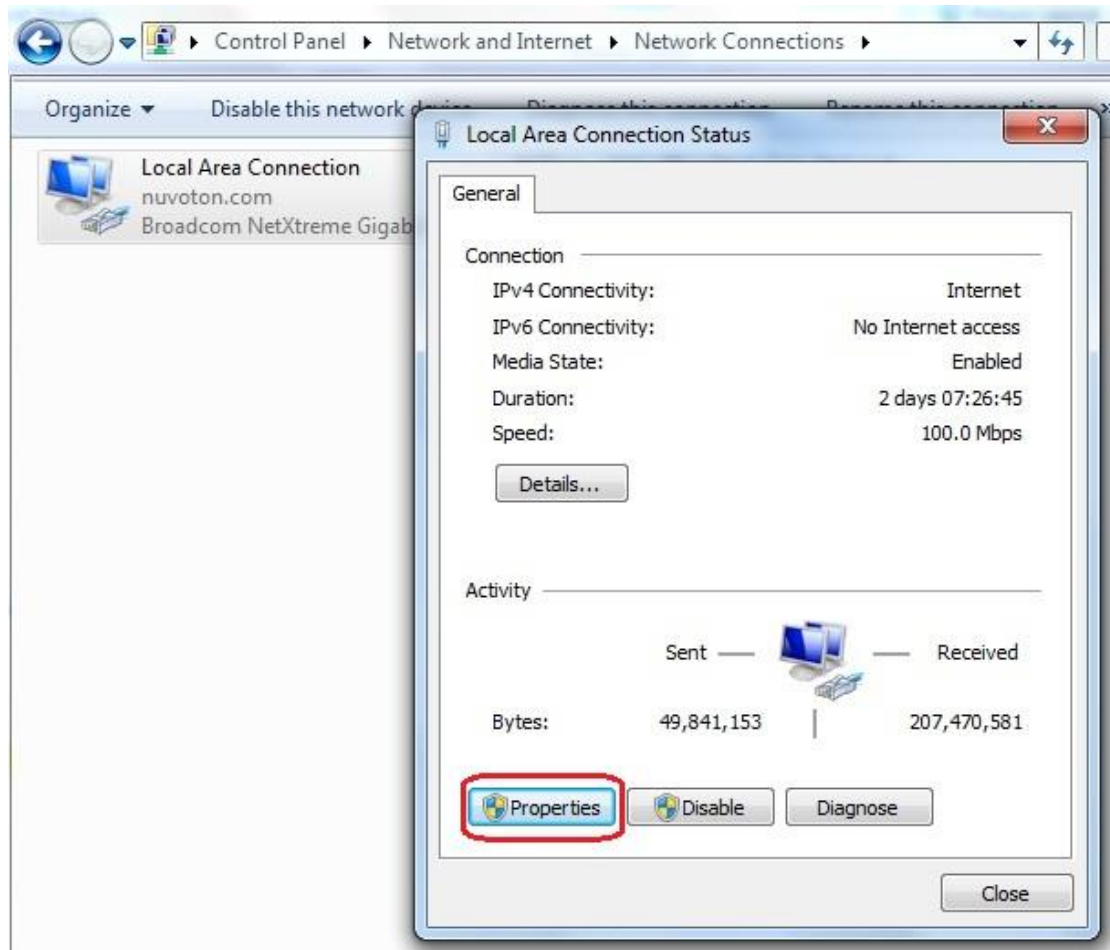
We need a PC that has static IP address and installed with TFTP/DHCP server application.

4.12.1 Set static IP address

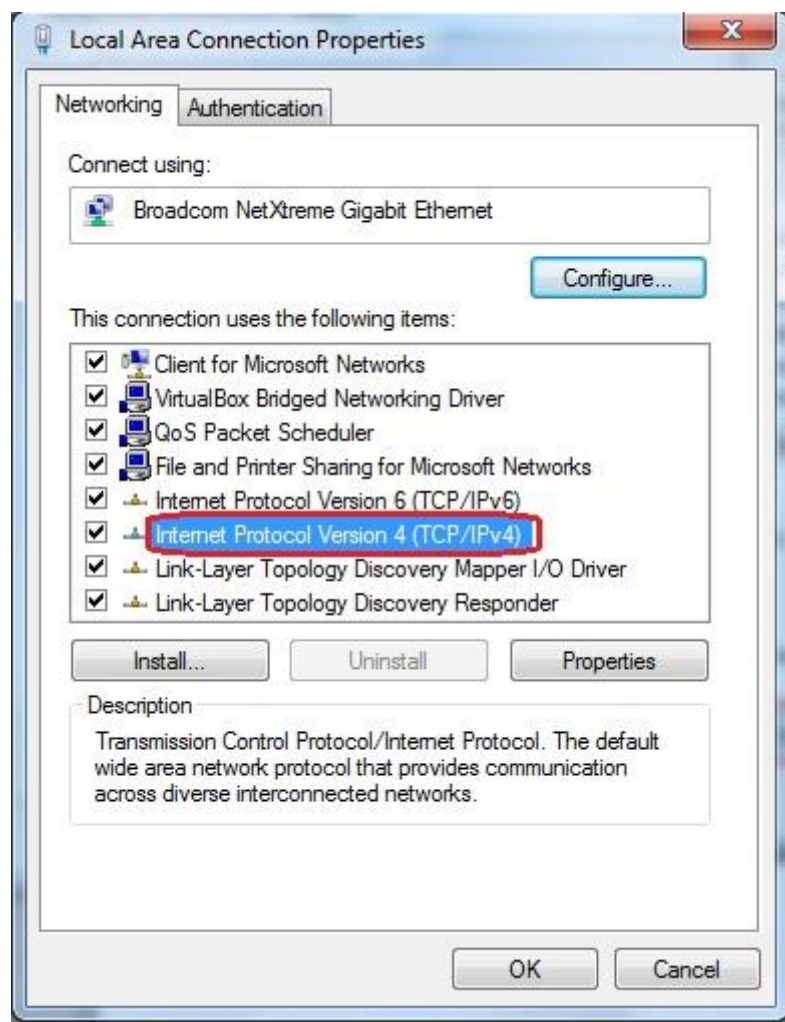
1. Choose Local Area Connection (under Control Panel -> Network and Internet -> Network Connections)



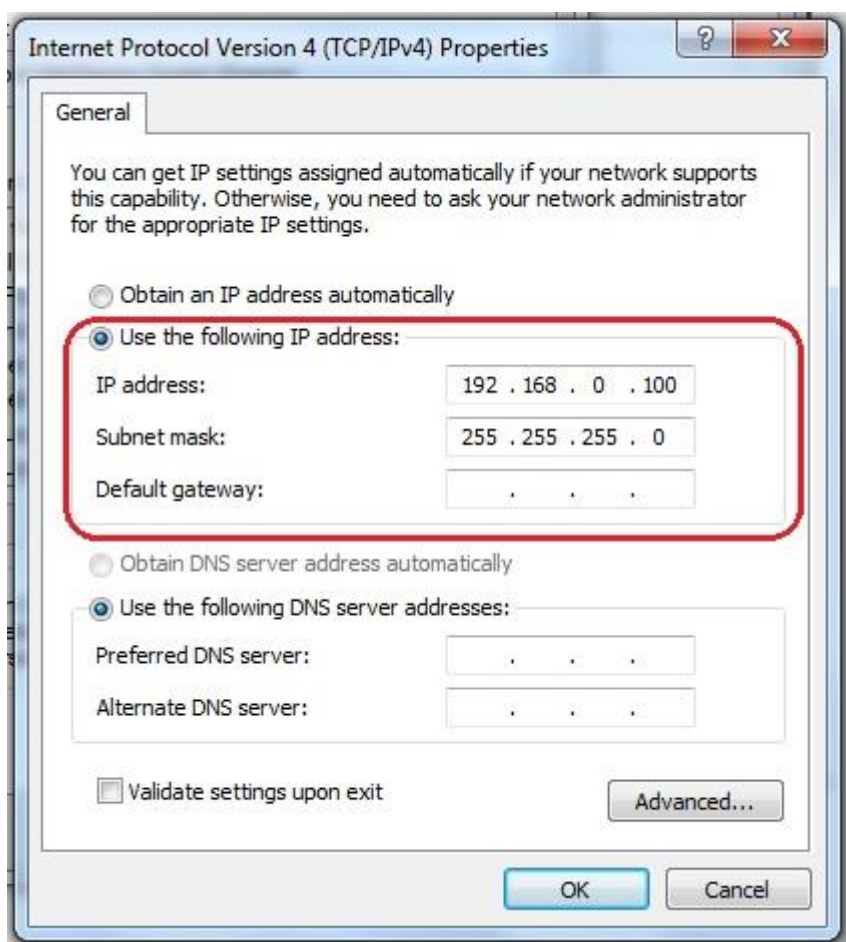
2. Choose Properties



3. Choose Internet Protocol Version 4 (TCP/IPv4)



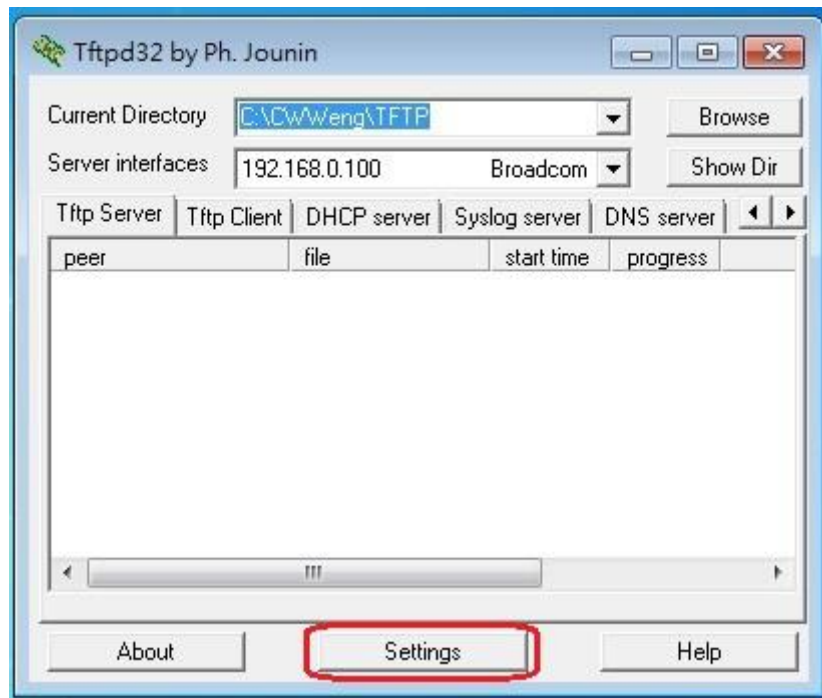
4. Set static IP address



4.12.2 TFTP and DHCP server

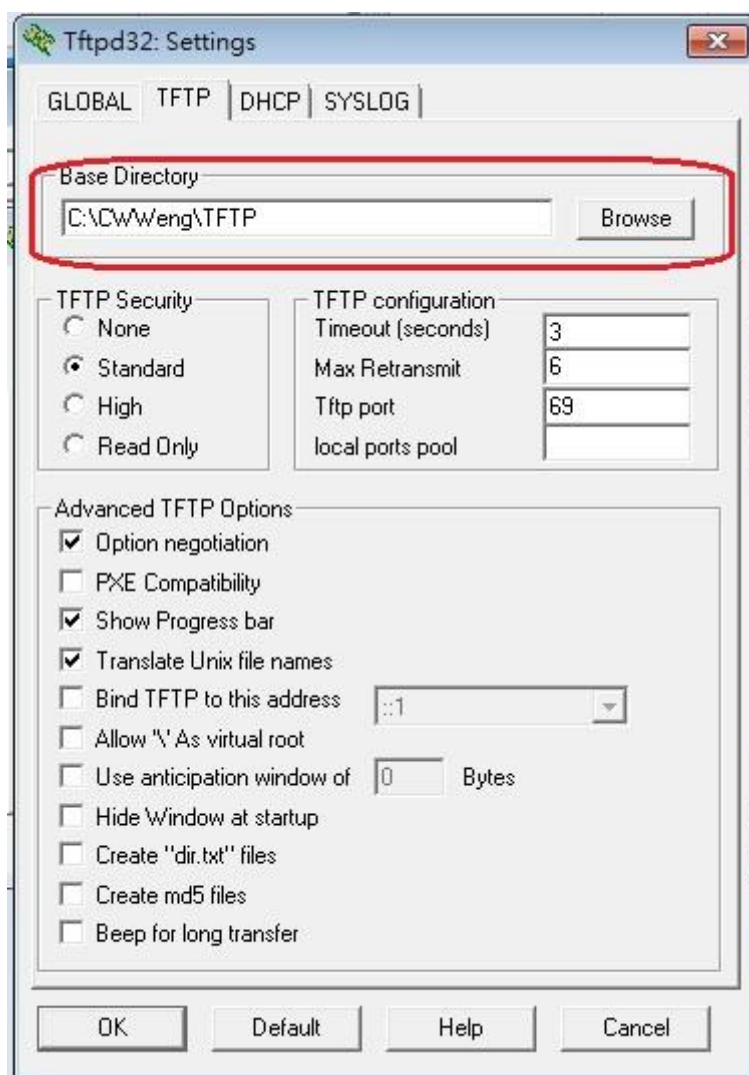
There is a free, open source application, TFTP32, which includes TFTP and DHCP server. It can be downloaded by following URL

<http://www.jounin.net/tftp-server-for-windows.html>

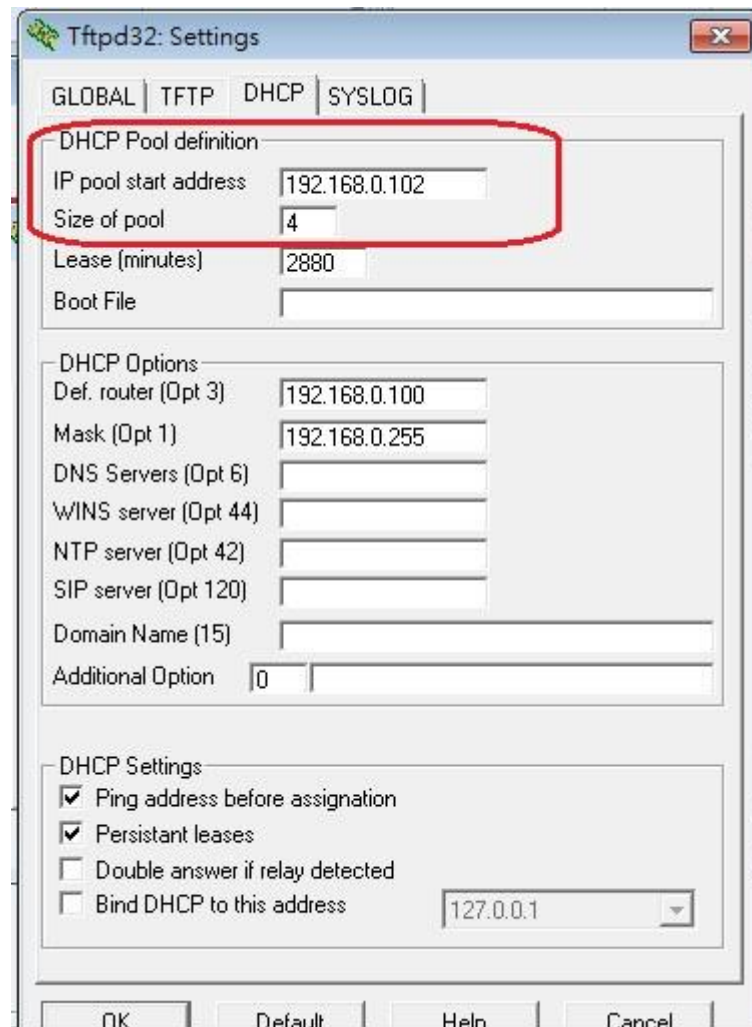


Choose Settings to setup TFTP server and DHCP server.

Set base directory for TFTP server.



Set DHCP pool definitions. Below is an example to set IP pool start from 192.168.0.102



4.13 Speed up SPI flash boot

In SPI flash boot, U-Boot uses the command “sf read” to read Linux kernel from SPI flash to DDR , and uses the command “bootm” to boot Linux kernel. There are two methods to shorten the SPI flash boot up time including speed up SPI and calculate checksum of Linux kernel image by SHA-1.

4.13.1 Speed up SPI

There are two methods to speed up SPI.

- Set SPI in Quad mode
- Set SPI clock to higher rate

Below is an example that sets the environment variable “spimode” to 4, remember to use “saveenv” command to save the environment variable to flash. Then, use “sf probe” command to set SPI clock to 18 MHz that is the highest SPI clock.

```
U-Boot> set spimode 4
U-Boot> saveenv
Saving Environment to SPI Flash...
```

```
Erasing SPI flash...Writing to SPI flash...done
U-Boot> sf probe 0 18000000
SF: Detected w25q128 with page size 4 KiB, total 16 MiB
U-Boot>
```

4.13.2 Calculate checksum of Linux kernel image by SHA-1

The command “bootm” calculates the checksum of Linux kernel by hardware can also shorten boot time. Please reference 0 that introduce how-to use SHA-1 to calculate checksum to speed up booting Linux kernel.

4.14 Notice

U-Boot SPI sub-system uses NUC970 SPI0. Its multi-function pins are B6, B7, B8, B9, B10, B11. Hence, you have to check whether these pins are connected.

5 Revision History

Version	Date	Description
0.9	July. 10, 2014	Initial release

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.*