NUC970 Linux BSP User Manual

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

www.nuvoton.com

1 NUC970 Linux BSP Introduction

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.

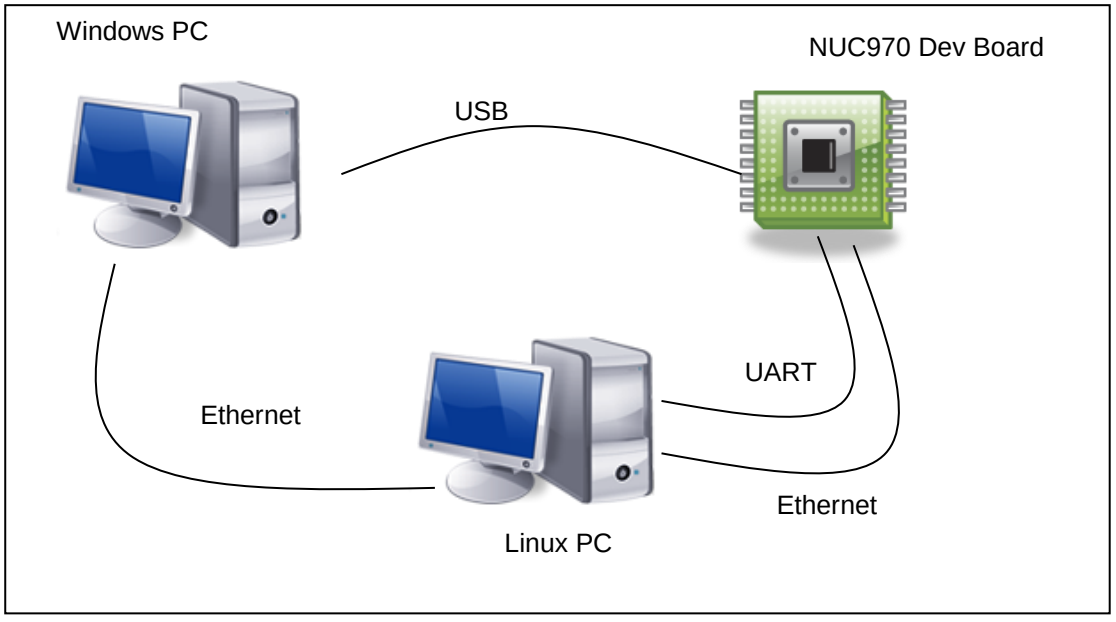
The NUC970 series processor integrates two 10/100 Mb Ethernet MAC controllers, USB 2.0 HS HOST/Device controller with HS transceiver embedded, TFT type LCD controller, CMOS sensor I/F controller, 2D graphics engine, DES/3DES/AES crypto engine, I2S I/F controller, SD/MMC/NAND FLASH controller, GDMA and 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)

NUC970 Linux BSP includes following contents:

- Linux 3.10 kernel sorce code and NUC970 device drivers.
- GCC 4.3.4 cross compiler with EABI support.
- uClibc-0.9.29
- Binutils-2.19.1
- Demo program for device drivers, busybox, mtd-util, and other open source applications.
- U-Boot source code including NUC970 device drivers.
- Flash programming tool Nu-Writer, and its Windows driver.
- User manuals.

1.1 Develop Environment

This BSP only provides cross development tool chain in Linux environment. So Linux platform is a must to build Linux kernel, U-Boot, and applications using the cross compiling tool chain in BSP. This platform could be a dedicate Linux server or running on virtual machine. PC can communicate with NUC970 Dev Board via different communication interfaces, such as UART, USB or Ethernet. As well as debug port, JTAG. Above interfaces could be used to load binary file to EV board for execution. JTAG interface could be used for chip level debug. USB interface is the interface used by NuWriter to program NAND, SPI, and eMMC. Following figure is an example of development environment.



1.2 Dev Board Setting

NUC970 family supports different boot 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 multiplefunctions, 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 Installation

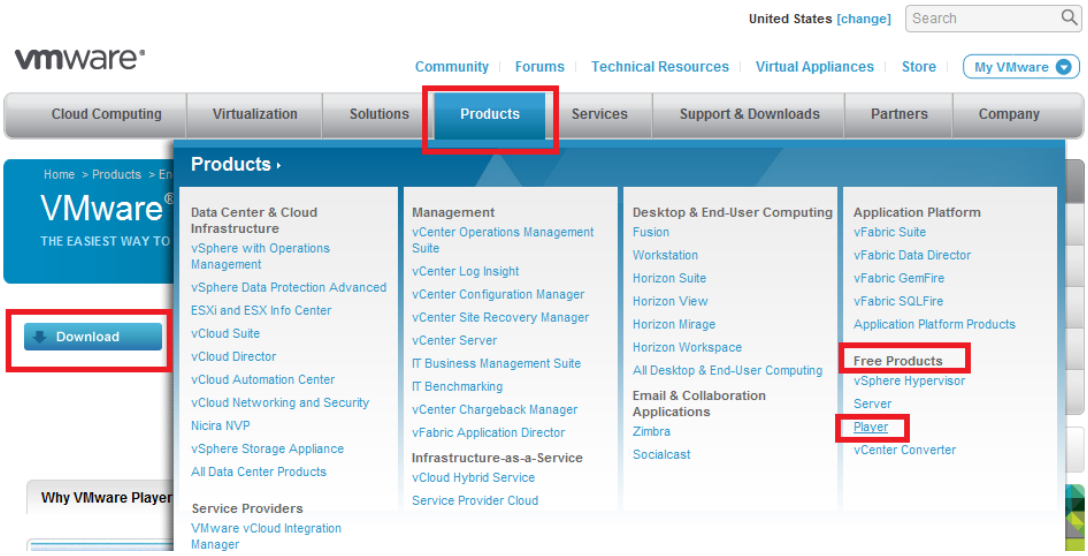
2.1 System Requirement

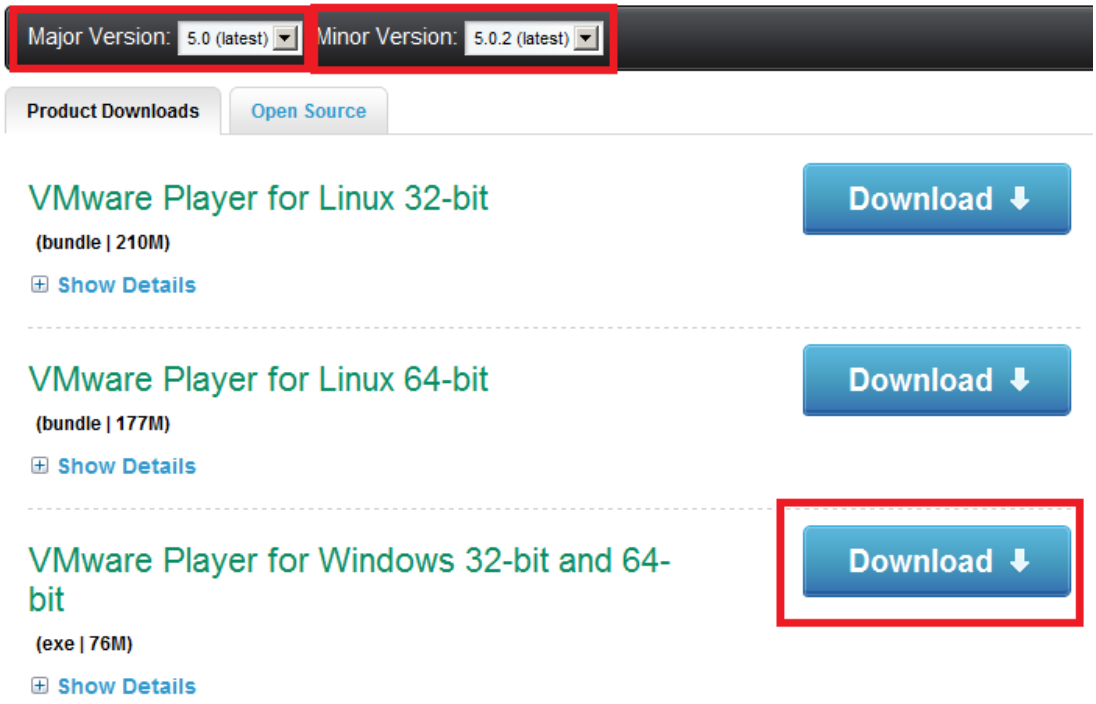
NUC970 Linux BSP provides cross compilation tools based on Linux operating system. We have tested this BSP in different x86 Linux distributions, including Ubuntu, CentOS, and Debian…etc. Because there are so many distributions out there with different system configuration, sometimes it is necessary to change system setting or manually install some missing component in order to cross compile.

Linux development environment could either be native, or install in a virtual machine execute on top of other operating system. This chapter introduces how to install CentOS Linux to VMware virtual machine, and the steps to install NUC970Linux BSP.

2.2 Download and installation VMware virtual machine

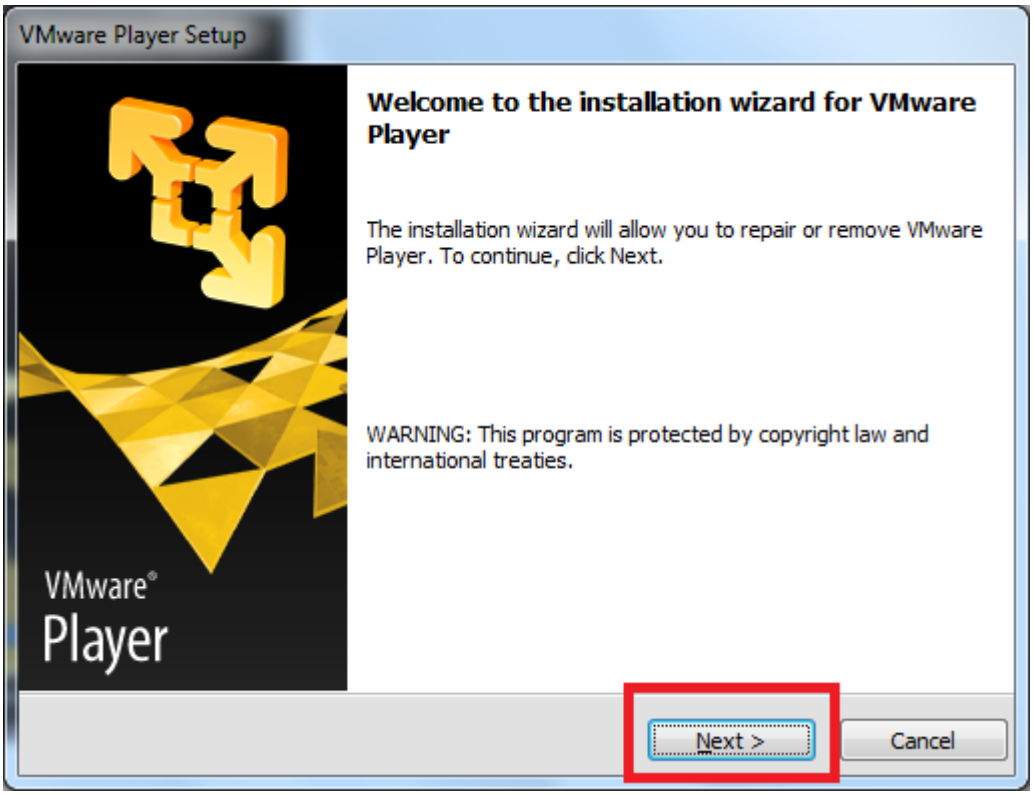
VMware provides free virtual machine VMware player 5.0.2 for users to download from VMware official website http://www.vmware.com/. Select “Products” → “Free Products” → “Player”, click “Download” button, select “5.0 (latest)” as “Major Version” and “5.0.2 (latest)” as “Minor Version” and then download “VMware Player for Windows 32-bit and 64-bit”. Please refer to the figure below.



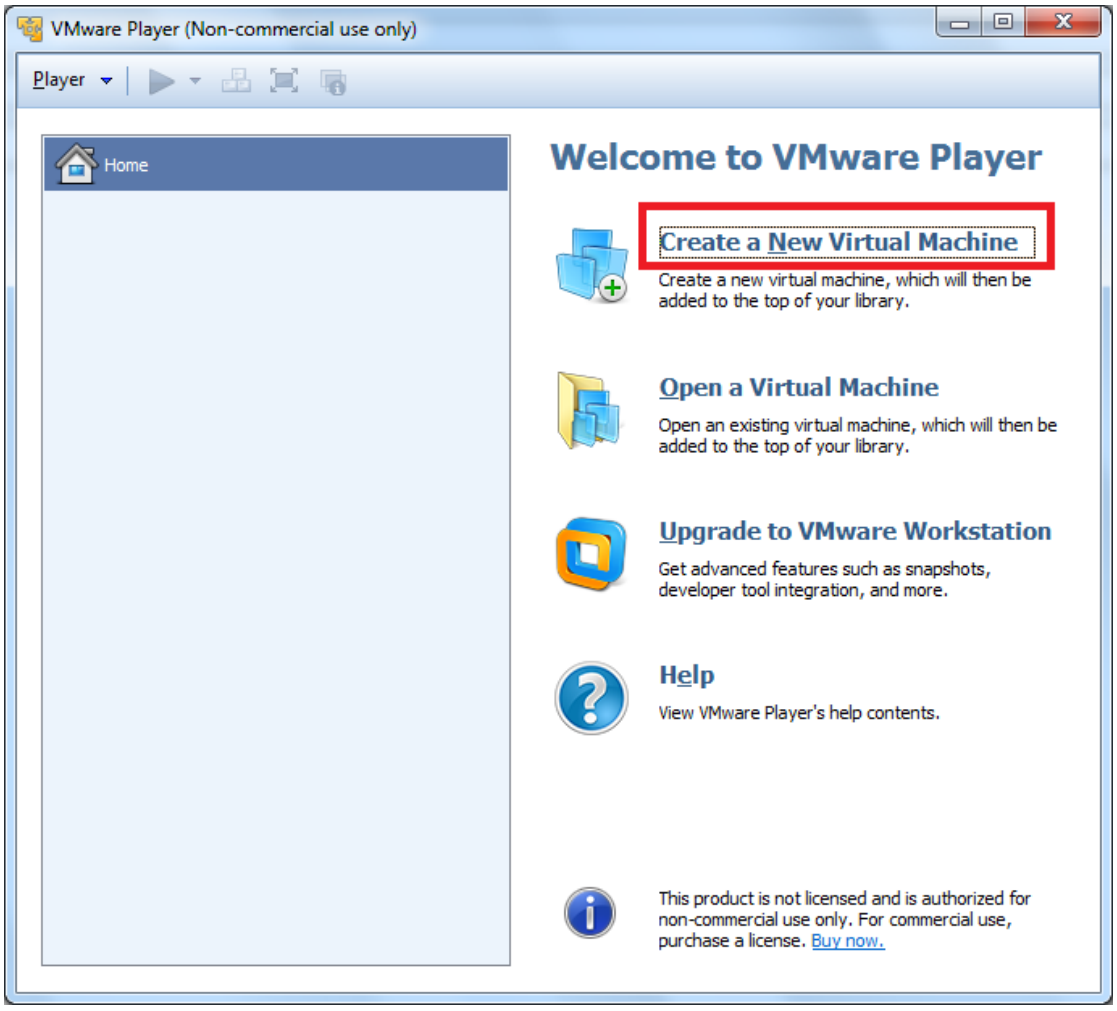


After download complete, double click the downloaded file.

And then click “Next” to continue installation steps.

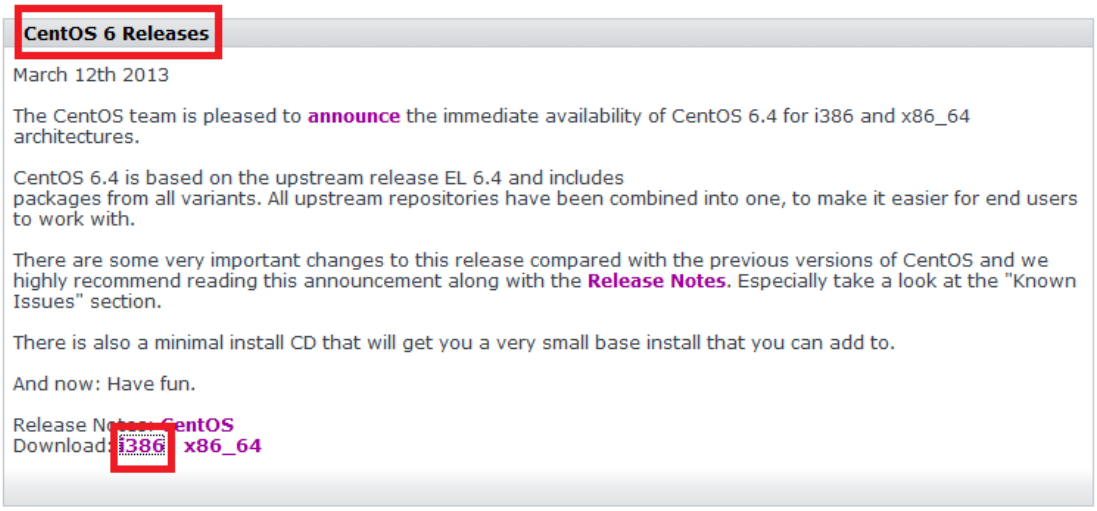


At last, double click the installed file to create a virtual machine.



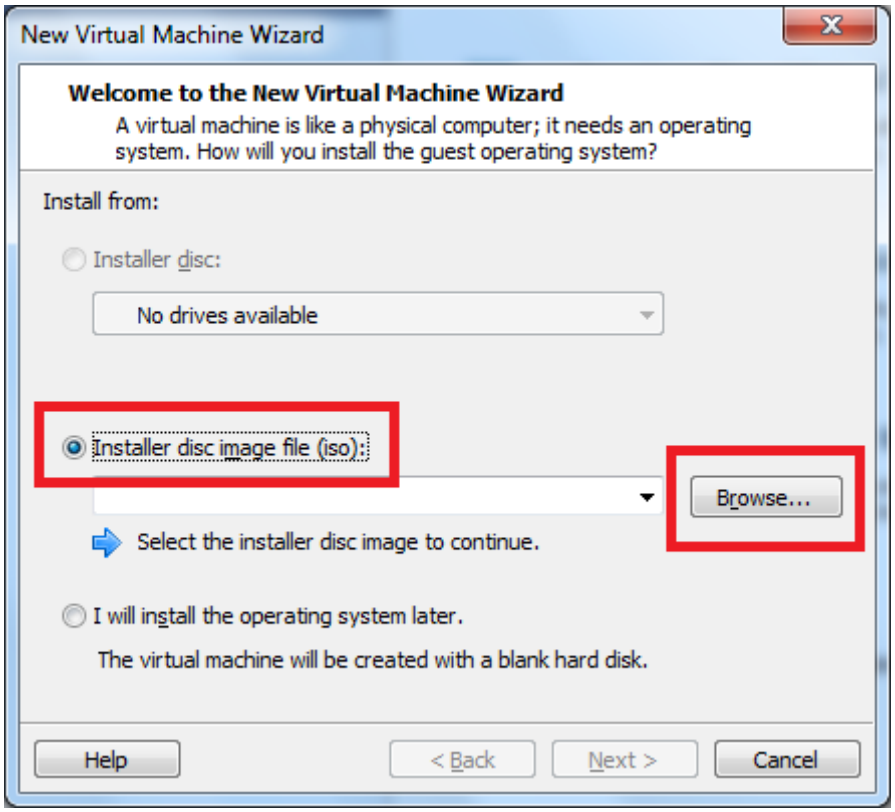
2.3 Download and installation CentOS Linux

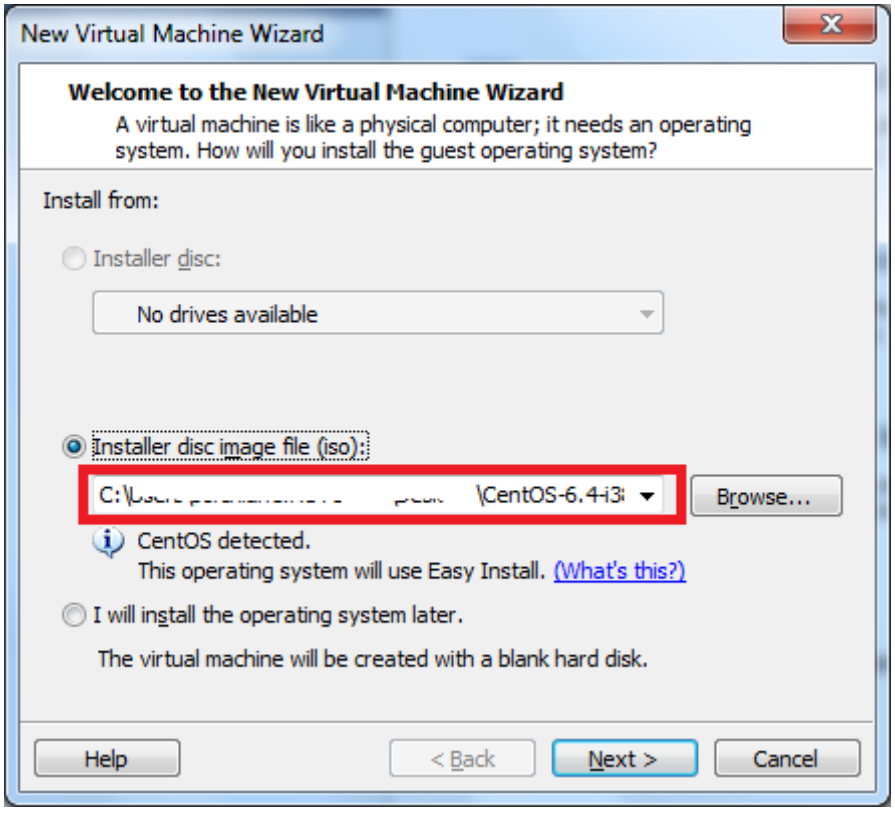
Here introduces the procedure to install CentOS 6.4 under VMWare. It is pretty much the same with installing as native operating system. First connect to http://www.centos.org/, and enter the download page by selecting “CentOS 6 Releases” → “i386”. Select “CentOS-6.4-i386-bin-DVD 1.iso” to download CentOS 6.4.



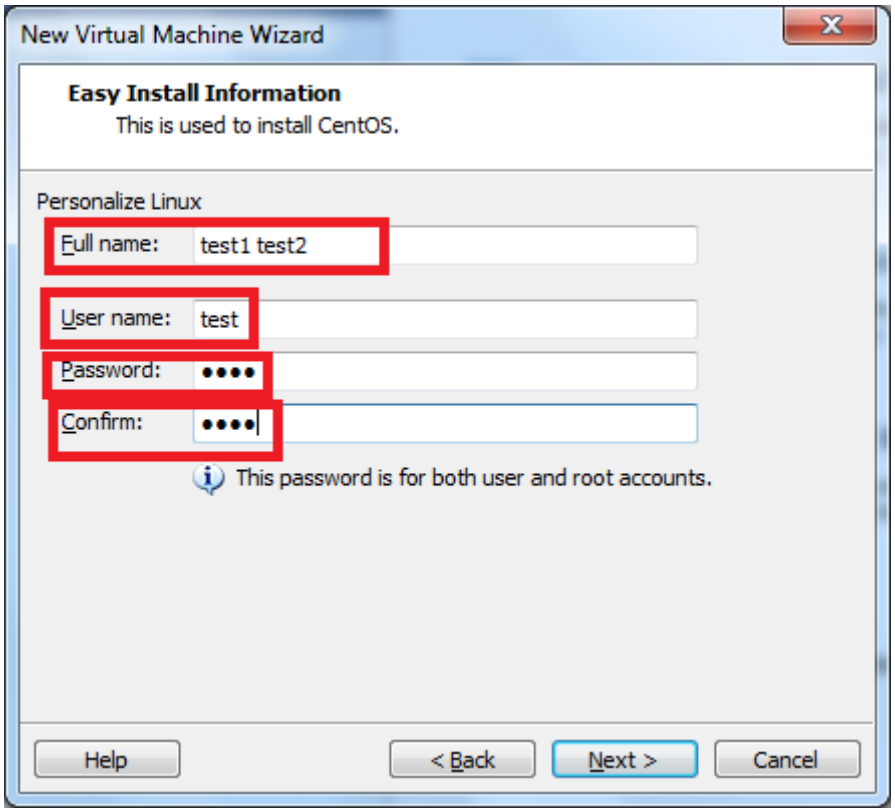
If VMware virtual machine is already installed, click “Create a New Virtual Machine” to install Linux virtual machine, otherwise follow the steps in previous section to complete VMware installation.

First, click “Installer disc image file (iso):” and “Browse…” and select the downloaded CentOS 6.4 iso as image source file.

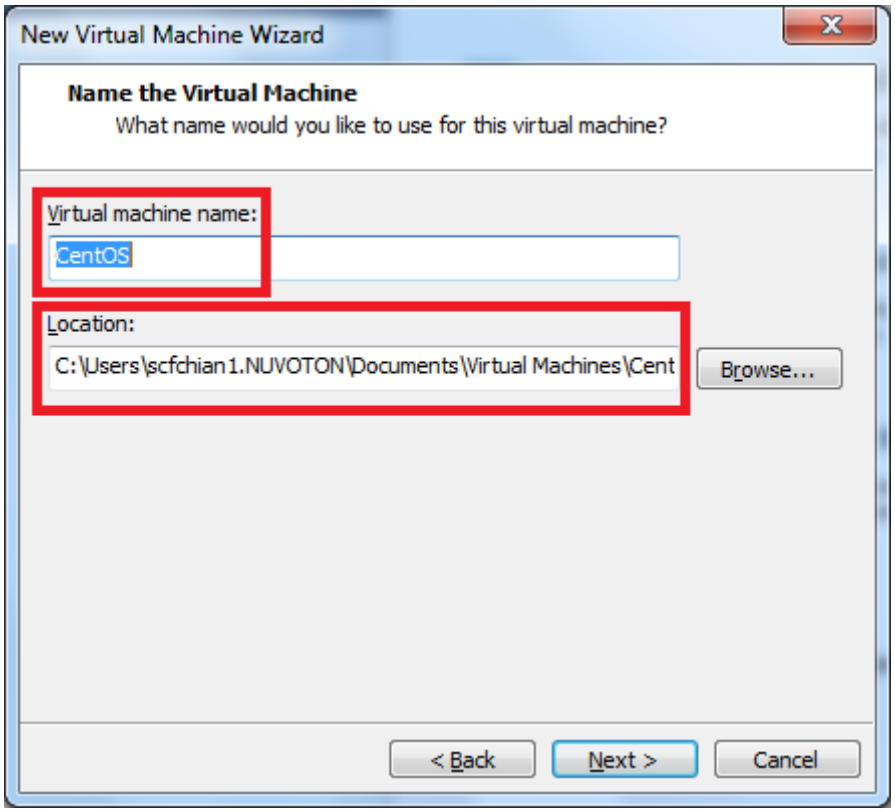




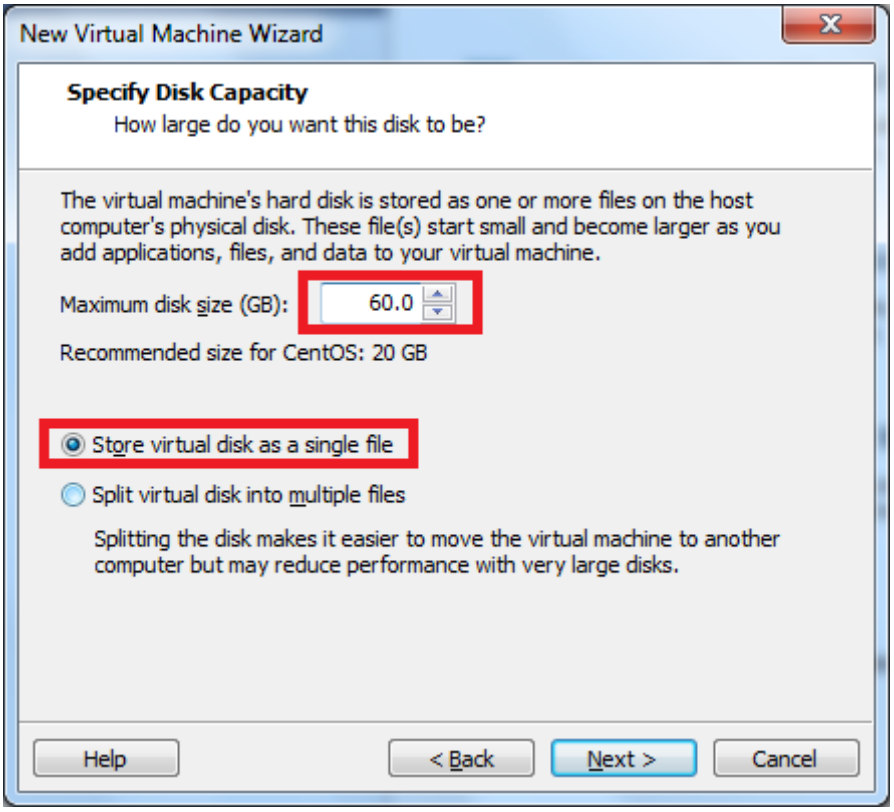
Input “Full name:”, “User name:”, “Password:”, and “Confirm:” click Confirm button. Please remember the username and password, they’ll be used to login CentOS later.



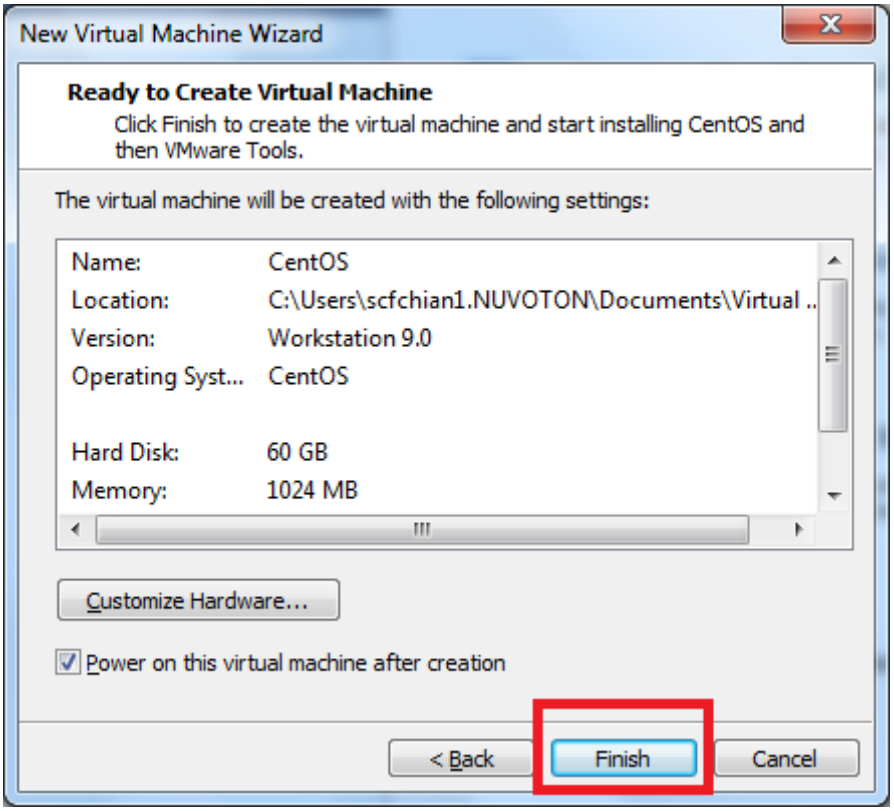
Next, input “Virtual machine name:” and “Location:” if necessary, otherwise keep the default value.



In the next step, input “Maximum disk size (GB):” value, it is recommend to reserve at least 60GB disk space, and select “Store virtual disk as a single file”.

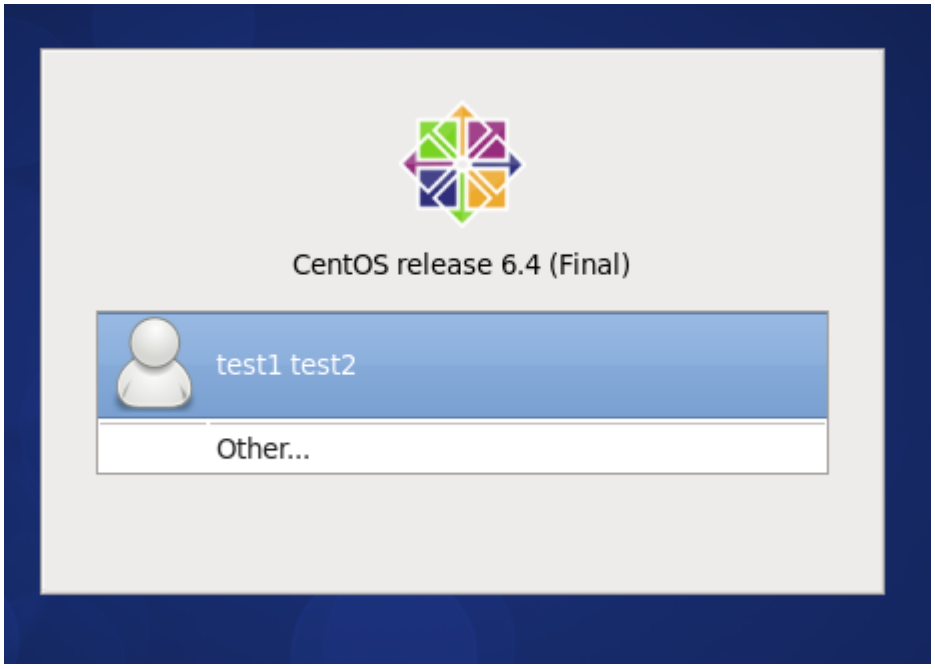


The last step is to click “Finish” to complete the CentOS configuration.



VMware will automatically complete reset of the CentOS installation procedures.

A CentOS login window will shows up after installation complete. You can login using the username and password set in previous step.



2.4 Install missing packages

Each Linux distribution selects different packages to install. And most distributions do not install all packages mandatory for NUC970 Linux BSP. And also lack of some optional packages which might be useful during development. Below listed some packages that may be missing after installed a Linux distribution but are mandatory or recommend, and could install later manually.

Package name	Function	Mandatory/ Optional
Patch	Application for apply patch file	Mandatory
libc6-dev	Contains required libraries to link with cross compiling tool chain. (i386 version)	Mandatory
libncurses5-dev	Contains required header files to build menuconfig interface	Mandatory
Minicom or cutecom	Serial terminal which could display the bootloader message or Linux console output.	Optional

Each Linux distribution has its own package management system. Ubuntu users could use apt-get command or Synaptic Package Manager to install packages. And Fedora users could use rpm command or Package Manager to install packages. Please refer to the manual of the Linux distribution you use to install missing components.

2.5 BSP installation procedures

Linux BSP contains three directories. Content of each directory listed in following table:

Directory name	Content
BSP	A tar ball contains U-Boot, Linux kernel, sample application source code. As well as cross compiler and root file system.

Documents	BSP related documents.
Tools	Writer tool and its driver executing in Windows OS.

Please copy the tar ball under BSP directory to Linux machine and use following command to extract the file.

```
$ tar zxvf nuc970bsp.tar.gz
$ cd nuc970bsp
```

After enter nuc970bsp directory, execute the installation script install.sh. This script requires the administrator privilege to execute. You can use “su” command to switch to root and execute the installation script.

```
$ su
Password: (Enter password of root)
# ./install.sh
```

Or execute this script as root by using sudo command. (This method works for those distributions do not open the root account privilege, such as Ubuntu)

```
# sudo ./install.sh
```

Below is the console output during installation:

```
Extract arm_linux_4.3.tar.gz to /usr/local/
Wait for a while
Successfully installed tool chain
Install rootfs.tar.gz, applications.tar.gz and linux-3.10.x.tar.gz
Please enter absolute path for installing(eg:/home/<user name>) :
/home/someone/nuc970_bsp
Please wait for a while, it will take some time
NUC970 BSP installion complete
```

If your Linux server has already installed the arm_linux_4.3 tools, the installation script will ask whether or not to overwrite existing tool chain. Otherwise the script will install the tool chain into /usr/local/arm_linux_4.3 without asking. For the first case, if you want to update the tool chain, you can select Y(or yes、y、YES), then hit Enter key.

After install the tool chain, the installation script will ask for the absolute path for install kernel and applications. The table below listed the item will be installed in the specified location

Directory name	Content
applications	Demo applications and other open source applications, such as busybox, wireless tool…
image/kernel	Pre build kernel using default configuration
image/U-Boot	Pre build U-Boot images support either NAND or SPI flash and env.txt file which stores U-Boot’s environment variable. The default execution address of U-Boot images is 0xE00000. Please refer to section 3.11 for the procedure to program U-Boot.
linux-3.10.x	Linux kernel source code
rootfs	Root file system
U-Boot	U-Boot source code

The installation script will try to configured the installed directory with correct owner and group, and add the path of compiler into $PATH. However, this doesn’t work correctly in every Linux distribution. User might need to set the owner/group of installed directory with correct user’s

name, and add /usr/local/arm_linux_4.3/usr/bin to $PATH manually.
Note: Please logout and re-login after installation complete to make the changes take effect.

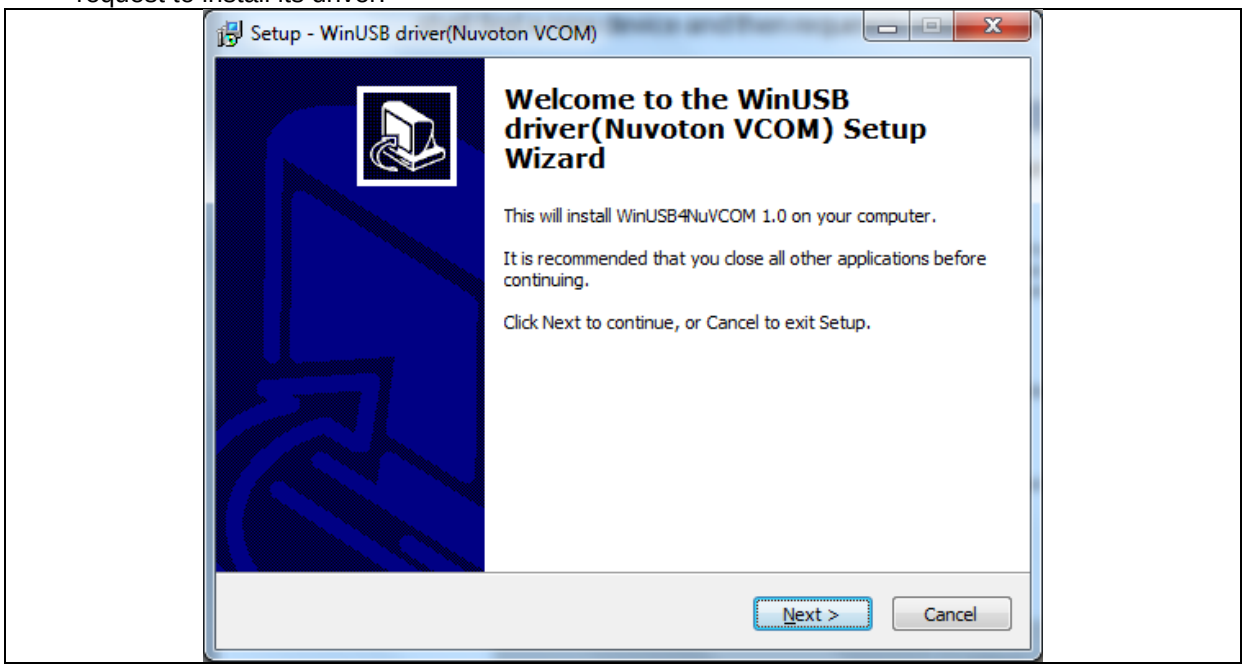
3 Nu-Writer Usage Guide

3.1 Overview

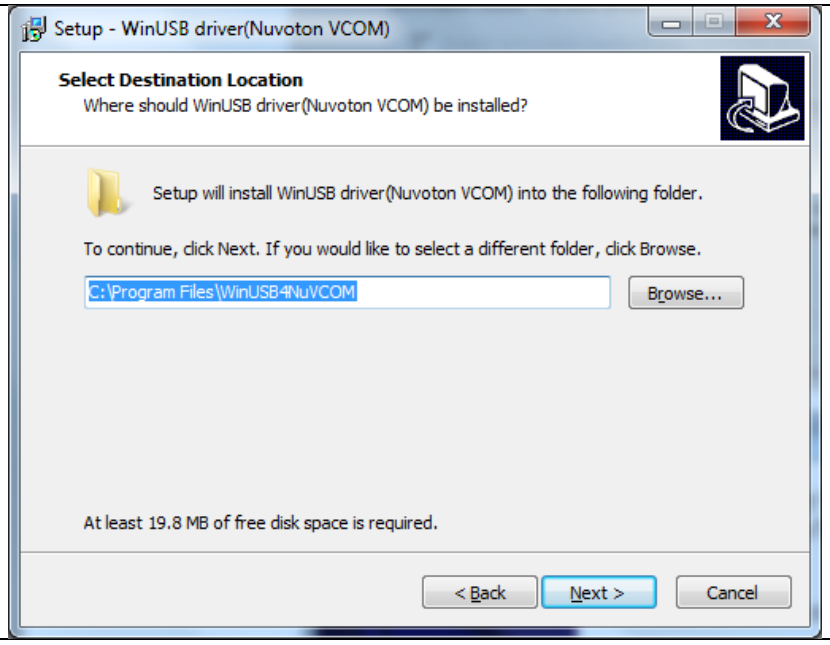
This tool can help users to program their images into the on-board ROM device when the system enters USB ISP mode. On-Board ROM device includes eMMC device, SPI Flash device and NAND Flash.

3.2 Installing the Nu-Writer Driver

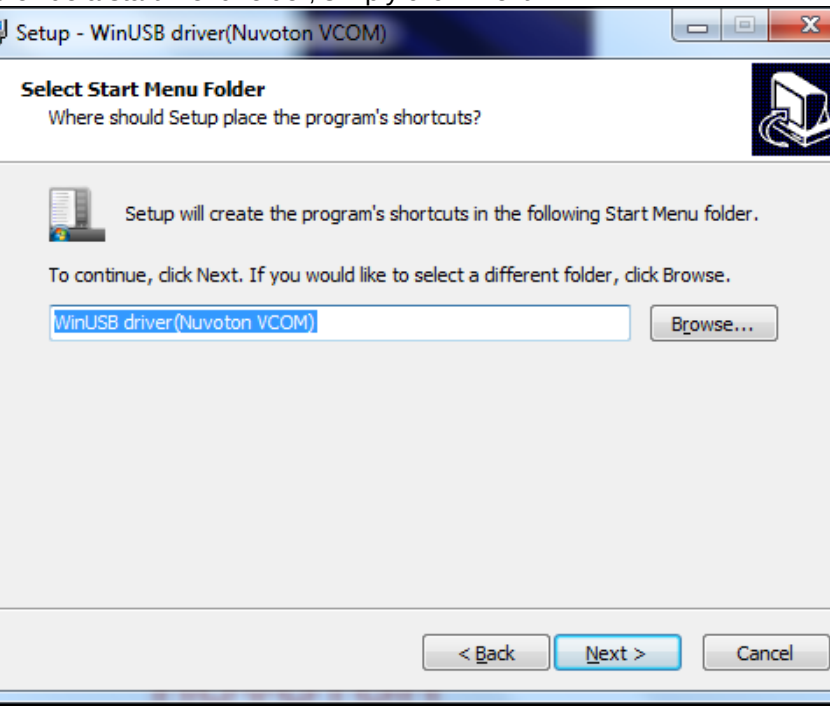
This programming tool requires a Nu-Writer driver to be installed on PC first. Please follow the steps below to install the driver.
Execute the “WinUSB4NuVCOM.exe” before the USB cable is plugged in. The “WinUSB4NuVCOM.exe” can be found in the “Tool” directory. Power on the NUC970 Series MCU EVB and plug the USB cable into PC, the Windows shall find a new device and then request to install its driver.



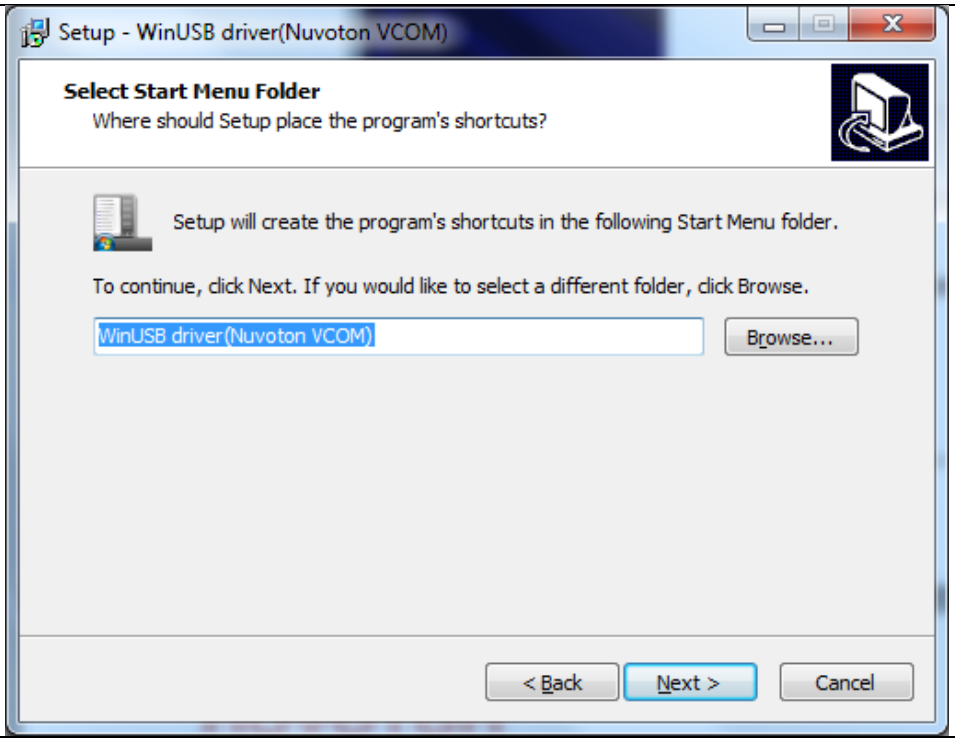
Click “Next”. The software installation will ask you how to install the driver as following figure..



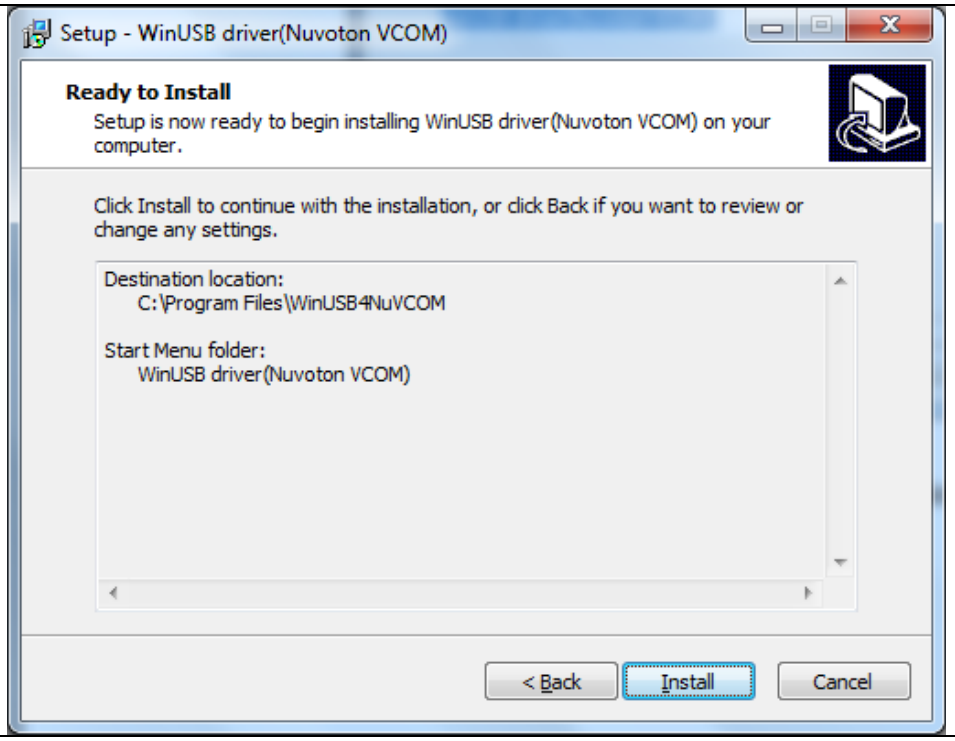
Select “setup path to specific location (Advanced), and then click “Next”. The installation software will ask to provide a start menu folder, simply click “Next”.



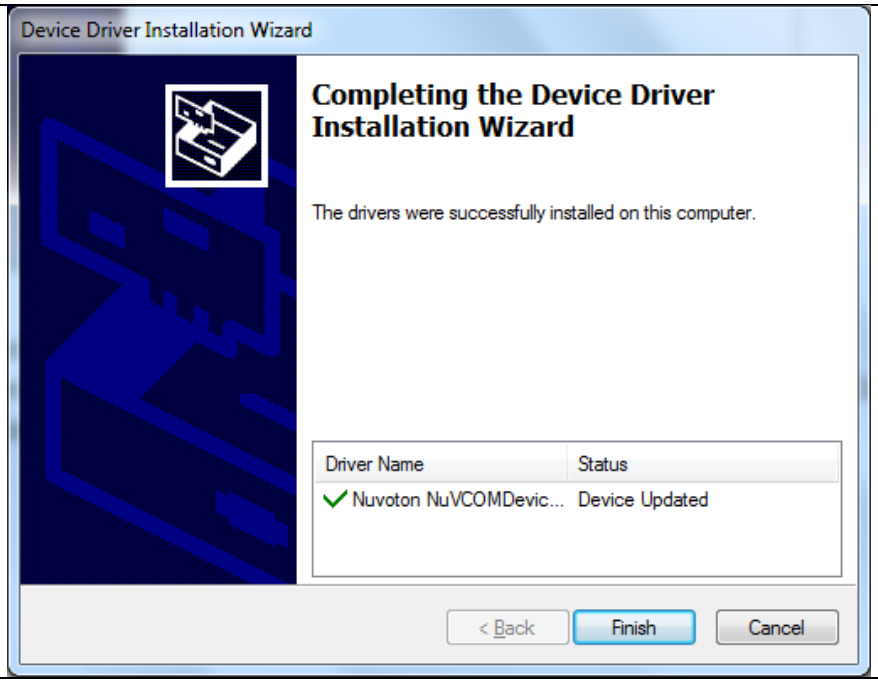
Click “Next”, as follows.



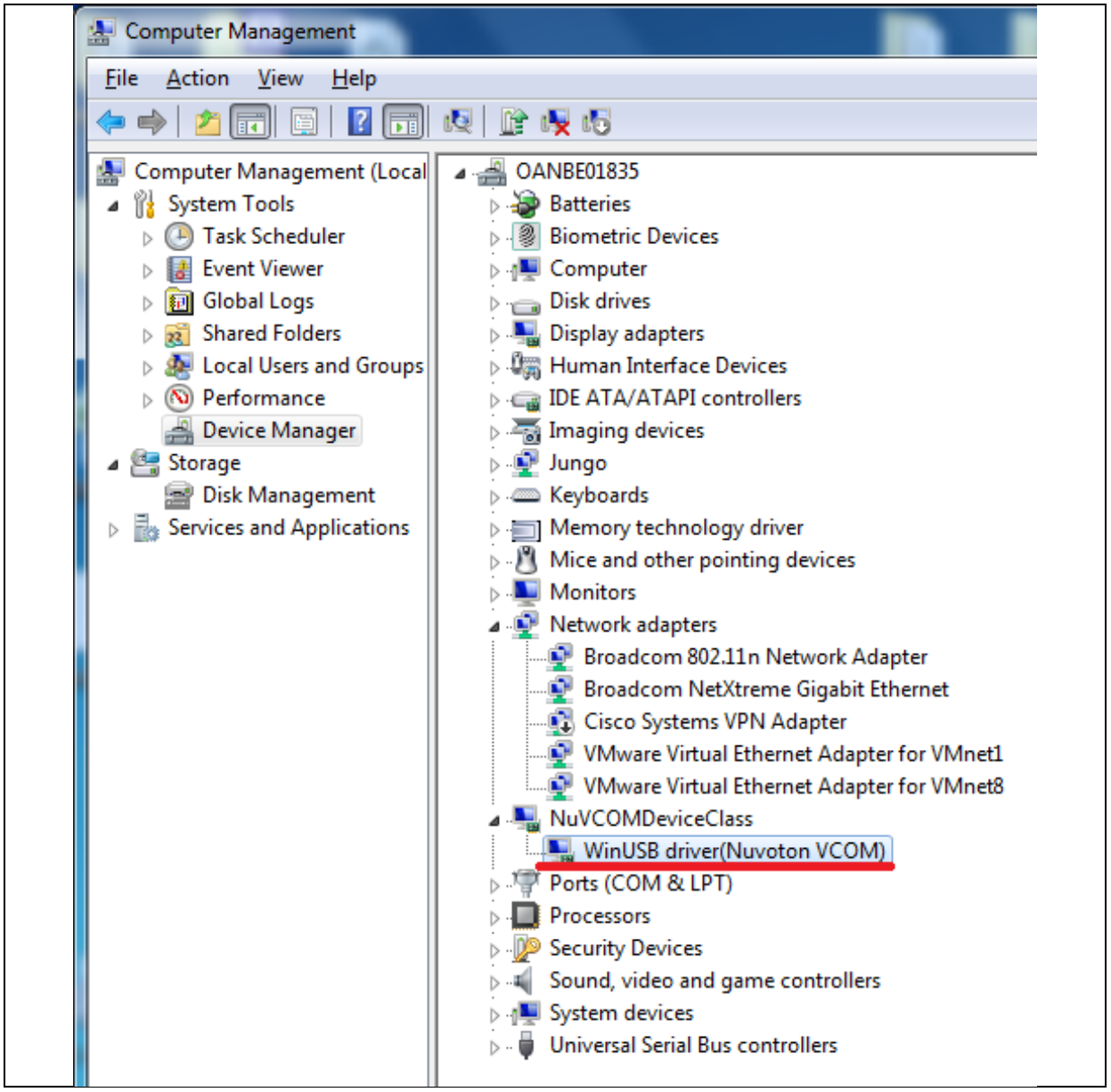
Click “Install”.



Click “Finish” to finish install driver.



If the installation is successful, and Dev board is connected to PC, a virtual COM port named “WinUSB driver (Nuvoton VCOM)” can be found by using “Device Manager” to check the ports devices.



3.3 USB ISP

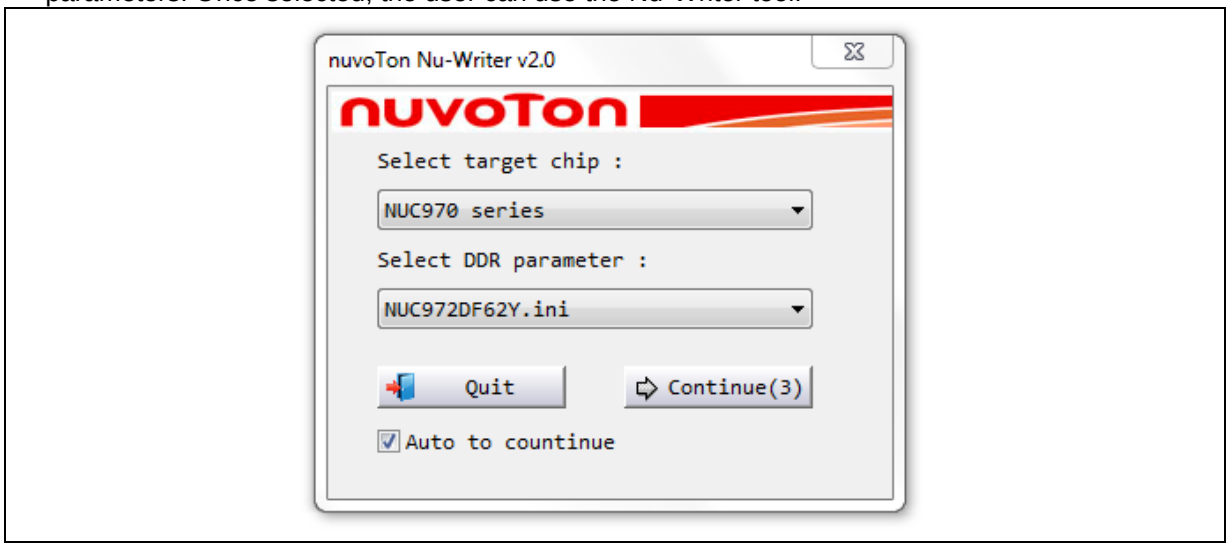
The NUC970 Series Dev board provides jumpers to select boot-up conditions. To select USB ISP mode, PA1 must be set to low and PA1 must be set to low. Other boot select can refer to the following table:

Power-on setting	PA1	PA0
USB ISP	Low	Low
Boot from eMMC	Low	High
Boot from NAND	High	Low
Boot from SPI	High	High

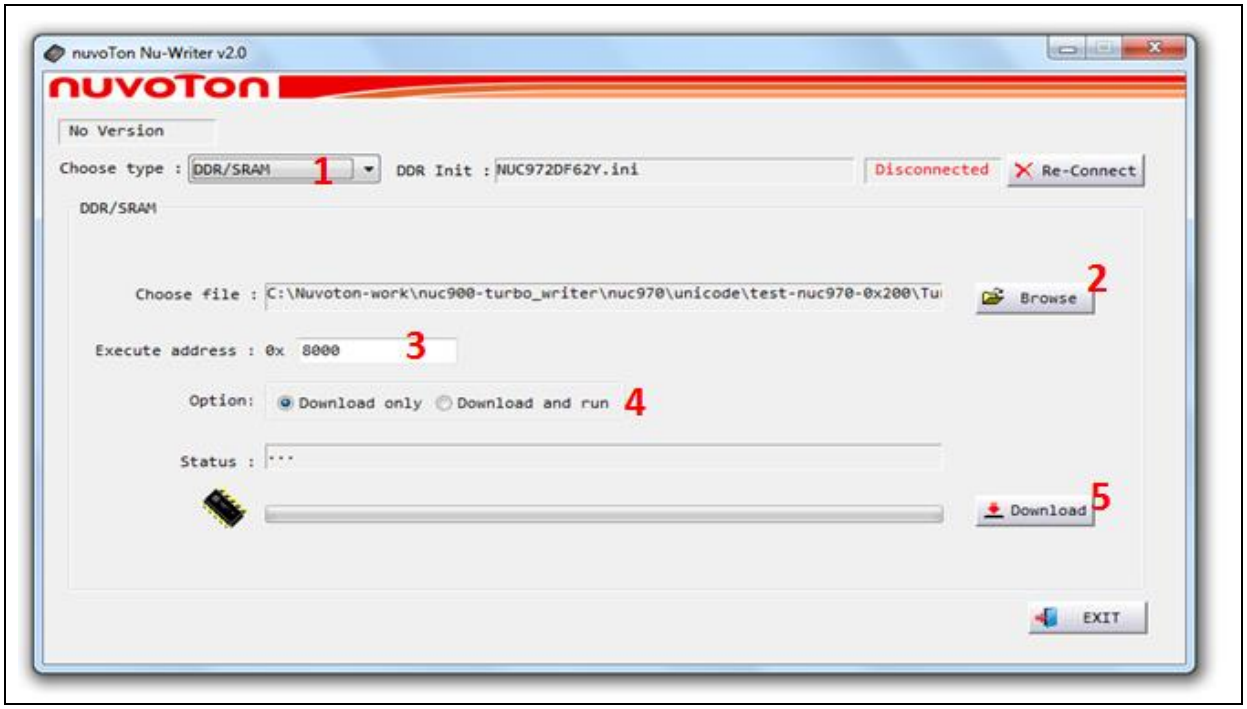
Power-on NUC970 Series Dev board, and then open the programming tool, “Nu-Writer” on the PC. Note that the tool cannot work if the “WinUSB4NuVCOM” driver is not found.

3.4 Select Chips

Extract Nu-Writer-xxxxxxxx.7z (in BSP/Tools folder). After double click “nuwriter.exe”, as follows. Nu-writer support NUC970 series (NUC972, NUC976…) chips, user can select chip and DDR parameters. Once selected, the user can use the Nu-Writer tool.



3.5 DDR/SRAM Mode



The DDR/SRAM mode is used to download an image to DDR or SRAM for debugging purpose. Follow the steps is listed below:

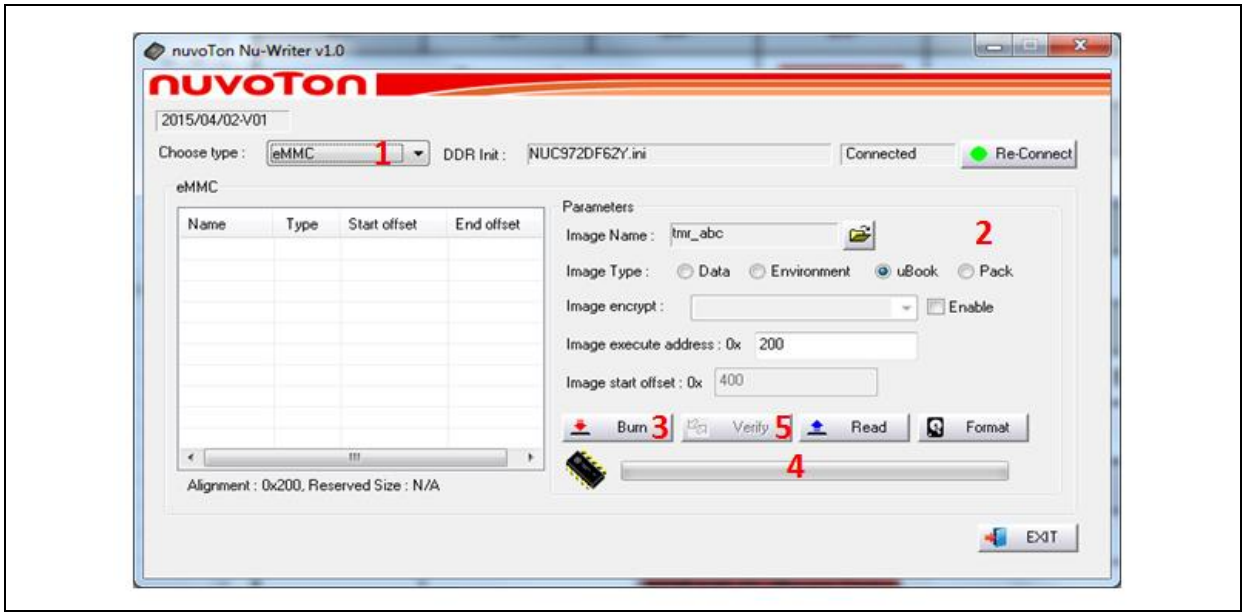
1. Select “DDR/SRAM”.
2. Select the image.
3. Enter the image execution address. Note: Execution address between 0x00000000 ~ 0x01F00000 (31MB).
4. Select ”Download only” or ”Download and run”

5. Click “Download”

3.6 eMMC Mode

This mode can write a new image to eMMC. The Image type can be U-Boot, Data, Environment or Pack.

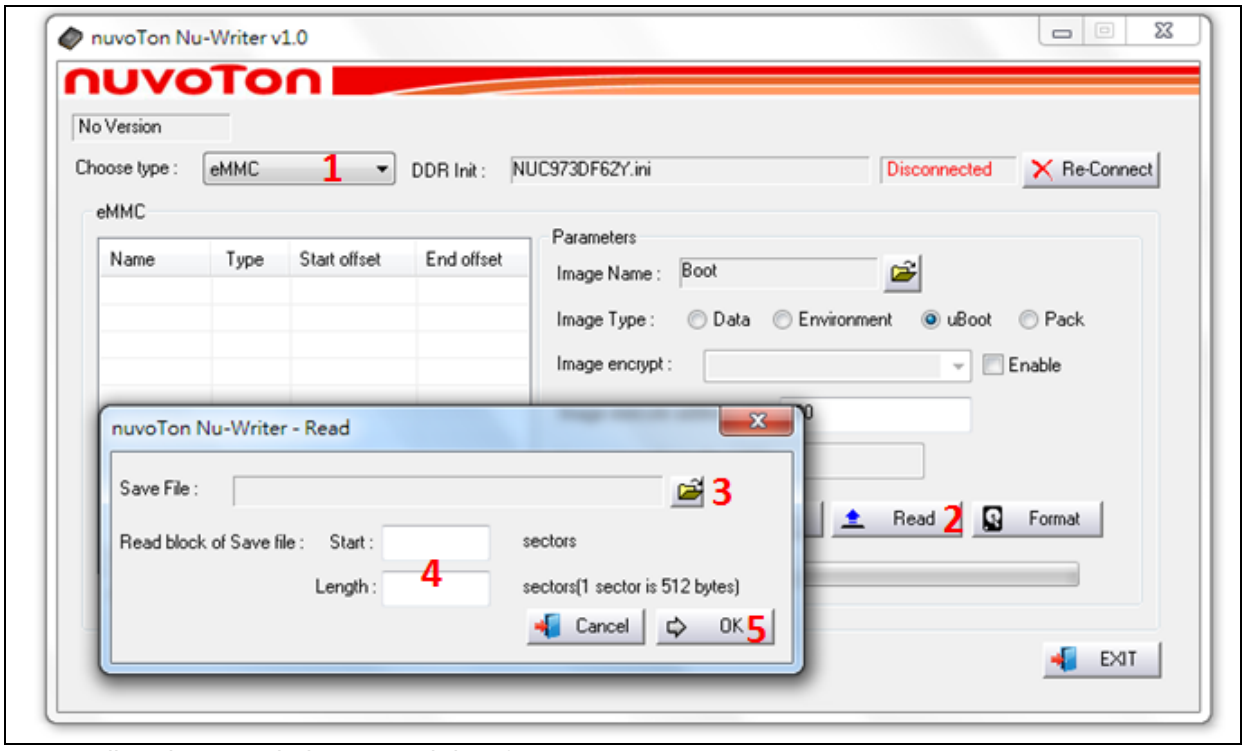
3.6.1 Add a New Image



Follow the steps below to add a new image to eMMC:

1. Select the “eMMC” type, which will not list the pre-programmed images in the eMMC.
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 “Burn”.
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.

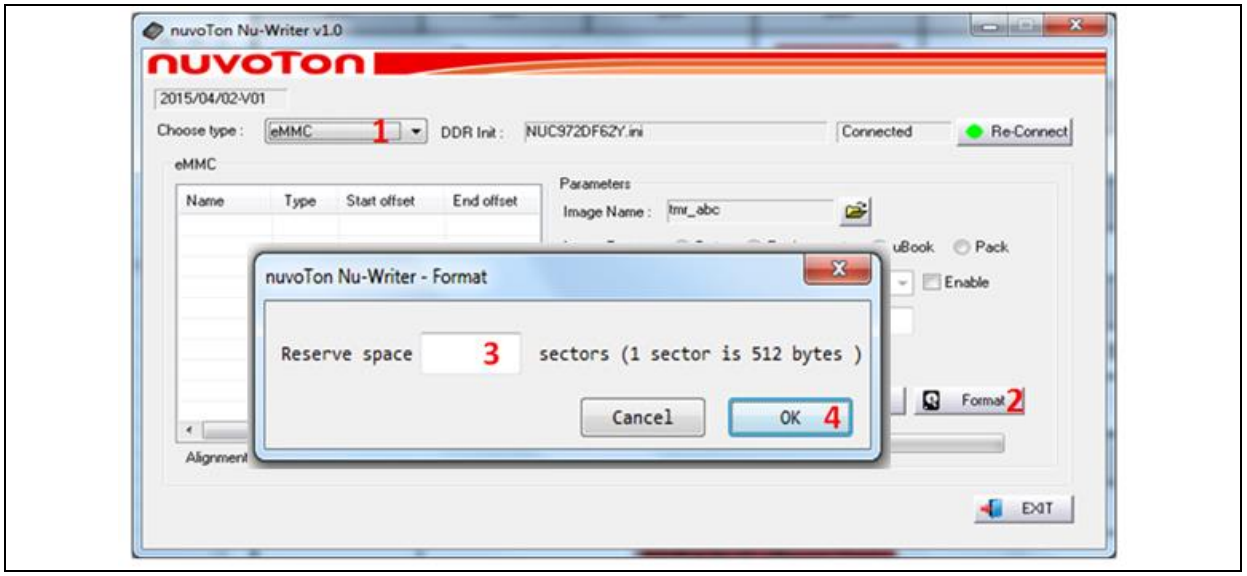
3.6.2 Read Raw Data



Follow the steps below to read data from eMMC:

1. Select the “eMMC”.
2. Click “Read”.
3. Set the file name of read back image.
4. Enter the **sectors** to read back. Each sector is 512 bytes.
   - Start: First sector to read back (In decimal format)
   - Length: Total secotrs to read back.(In decimal format)
5. Click “OK”.

3.6.3 Format (FAT32)



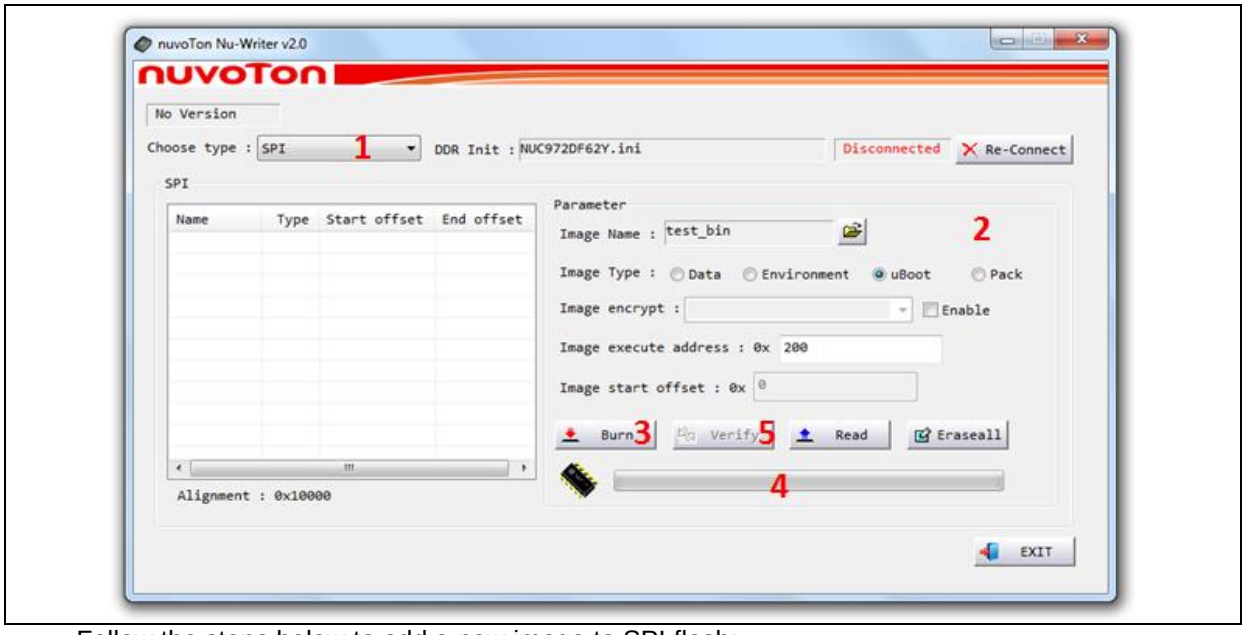
Follow the steps below to format eMMC:

1. Select “**eMMC**”.
2. Click “Format”.
3. Enter Reserve space (1 sector is 512bytes). **Note:** The reserved space is intent to store loader, and kernel. There must be sufficient space reserved for them, and rest of the space will be formatted to a single FAT32 parition.
4. Click “OK”.

3.7 SPI Mode

This mode can program a new image to SPI flash and specify the type of the image. These types can be recognized by boot loader or Linux. The Image type is set u-Boot, Data, Environment or Pack.

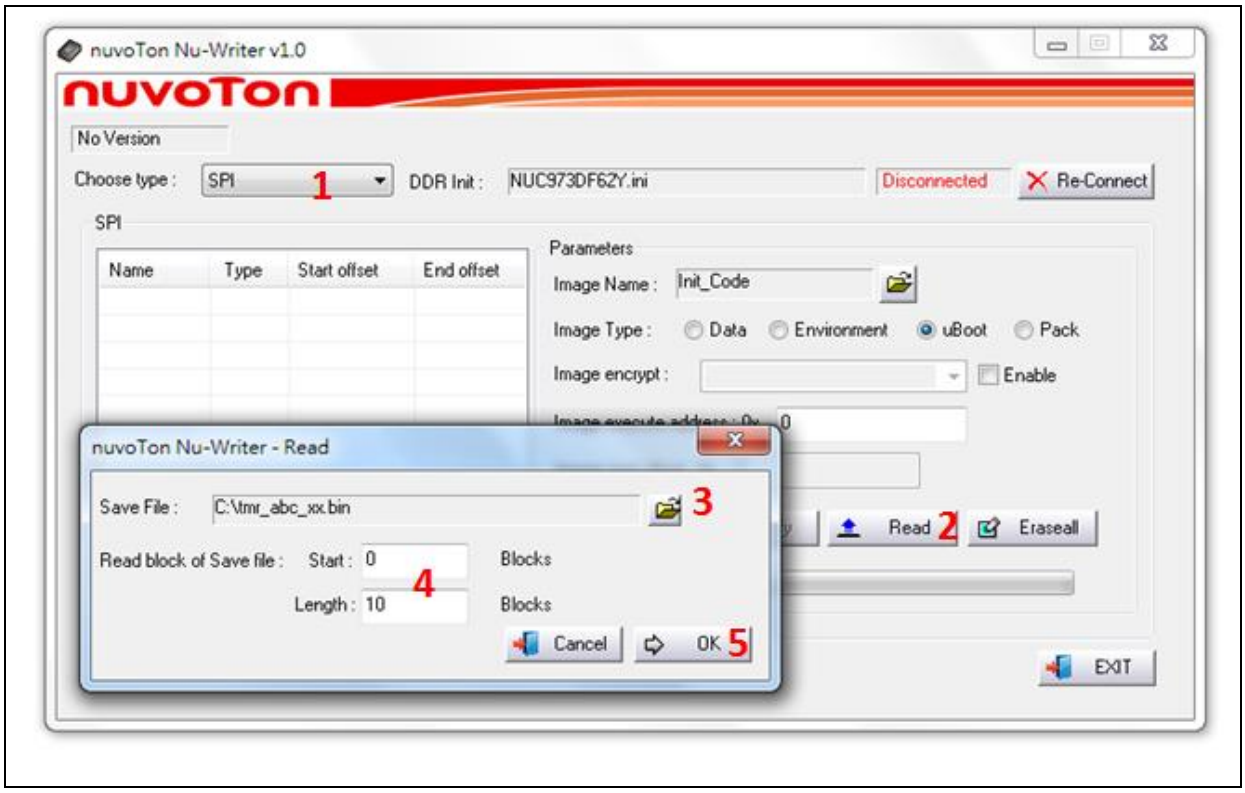
3.7.1 Add a New Image



Follow the steps below to add a new image to SPI flash:

1. Select the “SPI” type, which will not list the pre-programmed images in the SPI Flash.
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 “Burn”.
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.

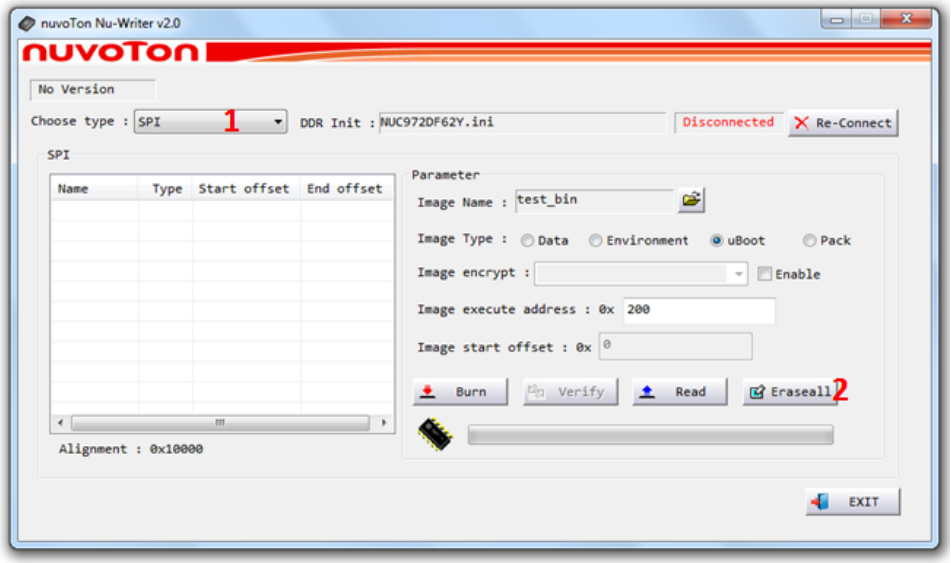
3.7.2 Read Raw Data



Follow the steps below to read data from SPI flash:

1. Select the “SPI”.
2. Click “Read”.
3. Set the file name of read back image.
4. Enter the blocks to read back. Block size depends on the SPI flash in use.
   - Start: Start of blocks
   - Length: Length of blocks
5. Click “OK”.

3.7.3 Erase SPI Flash



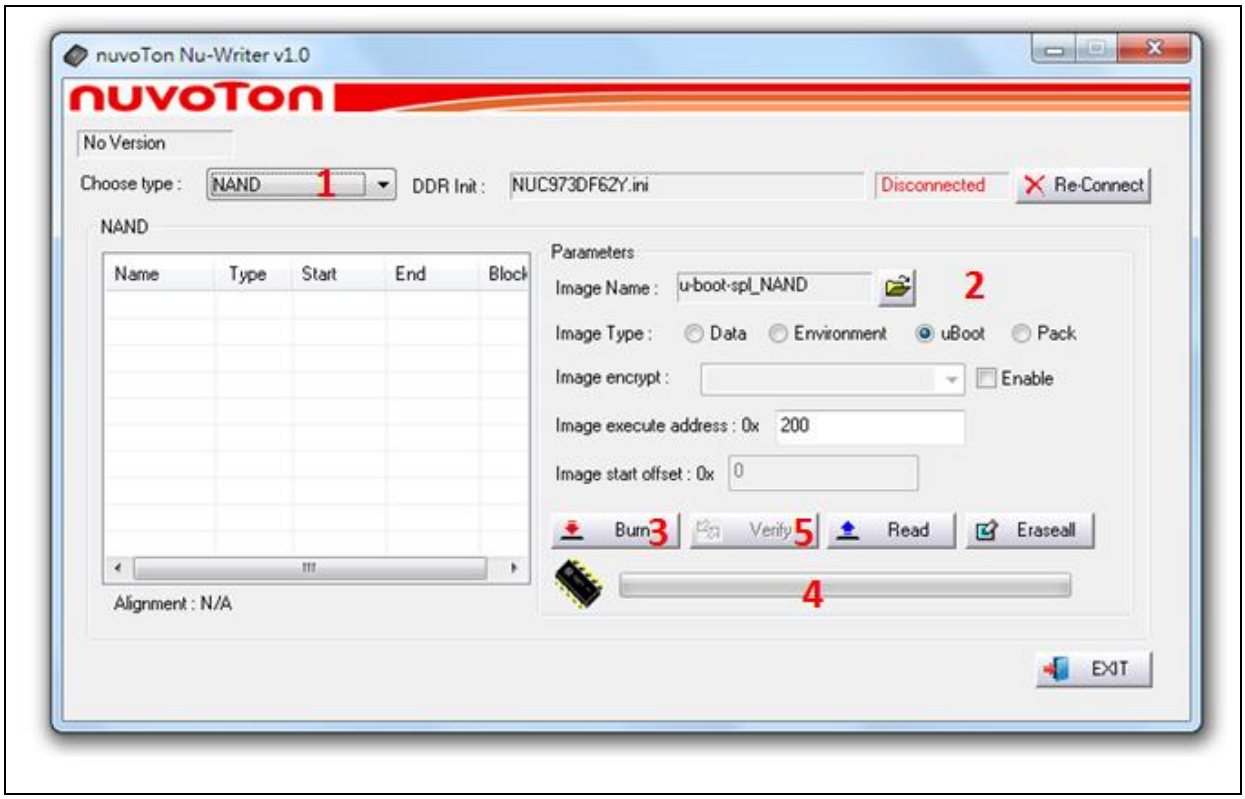
According to the figure above, follow the steps below to erase SPI flash:

1. Select the “SPI” type.
2. Click “Erase all”, Erase all data on the SPI flash.

3.8 NAND Mode

This mode can program a new image to NAND flash and specify the type of the image. These types can be recognized by u-boot or Linux. The Image type is set u-Boot, Data, Environment or Pack. Both YAFFS2 (in-band tags mode) and UBIFS file system image should be programmed as “Date”.

3.8.1 Add a New Image



Follow the steps below to add a new image to NAND flash:

1. Select the “NAND” type, which will not list the pre-programmed images in the NAND Flash.
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. It cannot use by Environment type.
   - Image execute address: Enter image execute address. This setting is only valid for U-Boot.
   - Image start offset: Enter image start offset.
3. Click “Burn”.
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.

3.8.2 Read Raw Data

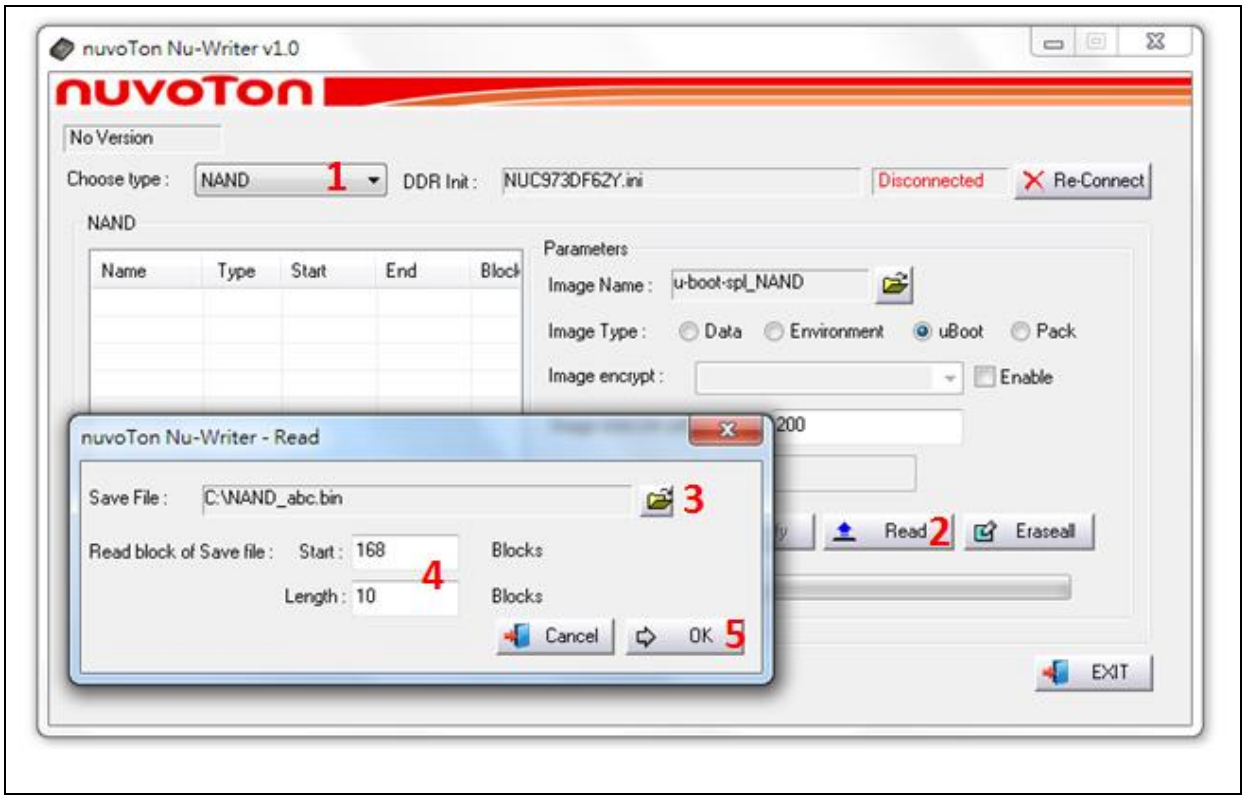
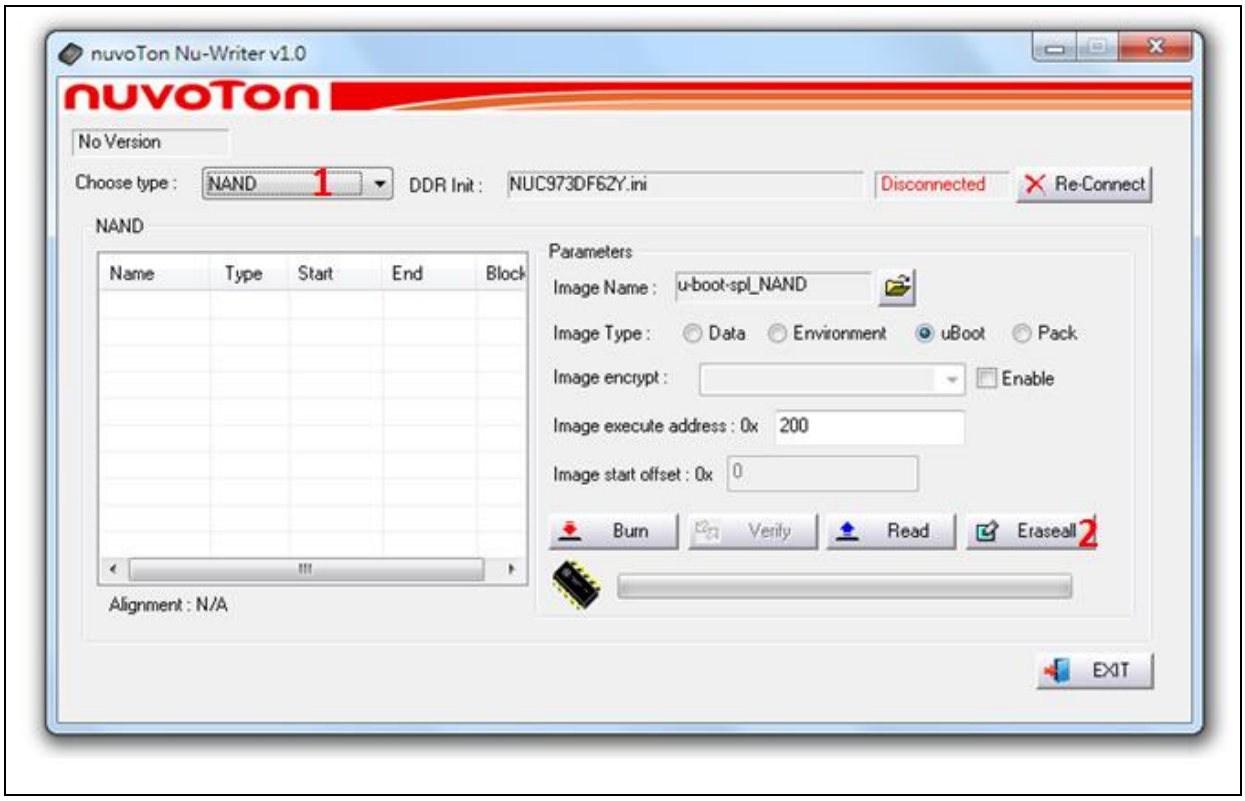


Figure 3-1 NAND – Read

According to the figure above, follow the steps below to read data from NAND flash:

1. Select the “NAND”.
2. Click “Read”.
3. Browse save file.
4. Enter the read back of blocks; Aligned on block size boundary, Block size is based on NAND specifications.
   - Start: Start address of blocks
   - Length: Length of blocks
5. Click “OK”.

3.8.3 Erase NAND Flash



According to the figure above, follow the steps below to erase NAND flash:

1. Select the “NAND” type.
2. Click “Erase all”; Erase all data on the NAND flash.

3.8.4 Make file system image

Make a new yaffs2 image with in-band tags, as follow (yaffs2 tags stored in data blocks)

```
# mkyaffs2 -–inband-tags -p 2048 rootfs rootfs_yaffs2.img
```

--inband-tags：Tags stored in data blocks.

-p：Set NAND flash page size.

rootfs folder can be compressed into rootfs_yaffs2.img, user can use Nu-Writer tool to burn rootfs_yafffs2.img into NAND flash.

Enter the following command will mount yaffs2 file system on the Linux.

```
mount -t yaffs2 –o “inband-tags” /dev/mtdblock2 /flash
```

Yaffs2 command can be found in yaffs2utils.tar.gz

Make a new ubifs image, as follows:

```
# mkfs.ubifs -F -x lzo -m 2048 -e 126976 -c 732 -o rootfs_ubifs.img -d ./rootfs
# ubinize -o ubi.img -m 2048 -p 131072 –O 2048 -s 2048 rootfs_ubinize.cfg
```

mkfs.ubifs description parameter, as follows:

-r, -d, --root=DIR	build file system from directory DIR
-m, --min-io-size=SIZE	minimum I/O unit size
-e, --leb-size=SIZE	logical erase block size
-c, --max-leb-cnt=COUNT	maximum logical erase block count
-o, --output=FILE	output to FILE
-j, --jrn-size=SIZE	journal size

-R, --reserved=SIZE how much space should be reserved for the super-user
-x, --compr=TYPE compression type - "lzo", "favor_lzo", "zlib" or "none" (default: "lzo")
-X, --favor-percent may only be used with favor LZO compression and defines how many percent better zlib should compress to make mkfs.ubifs use zlib instead of LZO (default 20%)
-f, --fanout=NUM fanout NUM (default: 8)
-F, --space-fixup file-system free space has to be fixed up on first mount (requires kernel version 3.0 or greater)
-k, --keyhash=TYPE key hash type - "r5" or "test" (default: "r5")
-p, --orph-lebs=COUNT count of erase blocks for orphans (default: 1)
-D, --devtable=FILE use device table FILE
-U, --squash-uids squash owners making all files owned by root
-l, --log-lebs=COUNT count of erase blocks for the log (used only for debugging)
-v, --verbose verbose operation
-V, --version display version information
-g, --debug=LEVEL display debug information (0 - none, 1 - statistics, 2 - files, 3 - more details)
ubinize description parameter, as follows:
-o, --output= output file name
-p, --peb-size= size of the physical eraseblock of the flash this UBI image is created for in bytes, kilobytes (KiB), or megabytes (MiB) (mandatory parameter)
-m, --min-io-size= minimum input/output unit size of the flash in bytes
-s, --sub-page-size= minimum input/output unit used for UBI headers, e.g. sub-page size in case of NAND flash (equivalent to the minimum input/output unit size by default)
-O, --vid-hdr-offset= offset if the VID header from start of the physical eraseblock (default is the next minimum I/O unit or sub-page after the EC header)
-e, --erase-counter= the erase counter value to put to EC headers (default is 0)
-x, --ubi-ver= UBI version number to put to EC headers (default is 1)
-Q, --image-seq= 32-bit UBI image sequence number to use (by default a random number is picked)
-v, --verbose be verbose
rootfs_ubinize.cfg content as follows：

```
[rootfs-volume]
mode=ubi
image=rootfs_ubifs.img
vol_id=0
vol_size=92946432
vol_type=dynamic
vol_name=system
vol_flags=autoresize
```

rootfs folder can be compressed into ubi.img, user can use Nu-Writer tool to burn ubi.img into NAND flash.
Enter the following command will mount UBIFS file system on the Linux.
Refer to “/sys/class/misc/ubi_ctrl/dev” content, assuming that the content is "10:56",user can set as follows.

```
mknod /dev/ubi_ctrl c 10 56
ubiattach /dev/ubi_ctrl -p /dev/mtd2
mount -t ubifs ubi0:system /flash
```

UBIFS command can be found in mtd-utils.tar.gz
Linux kernel must also be configuring, as follows:

YAFFS2：

```
File systems  --->
    [*] Miscellaneous filesystems  --->
        <*>  yaffs2 file system support
        <*>  Autoselect yaffs2 format
        <*>  Enable yaffs2 xattr support
```

UBIFS：

```
Device Drivers  --->
    -*- Memory Technology Device (MTD) support  --->
        <*>  Enable UBI - Unsorted block images  --->
File systems  --->
    [*] Miscellaneous filesystems  --->
        <*>  UBIFS file system support
        [*]  Advanced compression options
        [*]   LZO compression support
        [*]   ZLIB compression support
```

Pakages	Description

lzo-2.09.tar.gz	The compression / decompression tool.
	Cross compiler command is as follows:
	$ cd lizo-2.09
	$ ./configure --host=arm-linux --prefix=$PWD/../install
	$ make
	$ make install

libuuid-1.0.3.tar.gz	A universally unique identifier tool.
	Cross compiler command is as follows:
	$ cd libuuid-1.0.3
	$ ./configure --host=arm-linux --prefix=$PWD/../install
	$ make
	$ make install

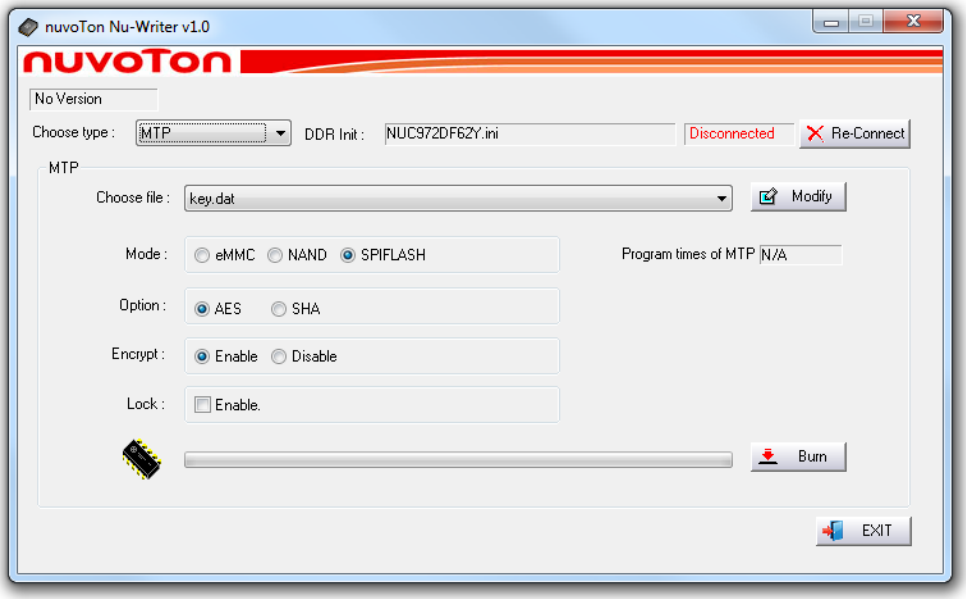
mtd-utils.tar.gz	mtd-utils source code.
	Cross compiler command is as follows:
	Thie packages need to use lzo-2.09.tar.gz and libuuid-1.0.3.tar.gz.
	$ cd mtd-utils
	$ export CROSS=arm-linux-
	$ export WITHOUT_XATTR=1
	$ export DESTDR=$PWD/../install
	$ export LZOCPPFLAGS=-I/home/install/include
	$ export LZOLDFLAGS=-L/home/install/lib
	$ make
	$ make install

yaffs2utils.tar.gz	yaffs2 command tool $ make

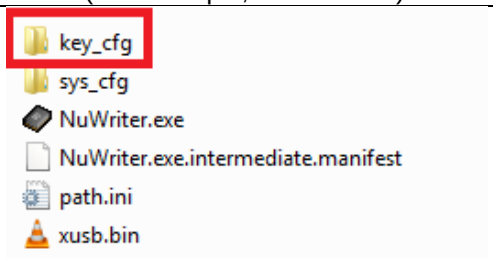
3.9 MTP Mode

On the MTP form, user can select MTP keys file to burn into NUC970 Series MCU. This MTP keys can protect user's binary code in device (eMMC, NAND Flash, SPI Flash).

3.9.1 Add a New Key



1. Under the Nu-Writer folder (For example, C:\NuWriter). Enter the key_cfg folder.



2. Create a text file and enter password. The password in the following format, the first line must be 256, and eight consecutive big-endian modes key. (For example, C:\NuWriter key_cfg\key.dat).

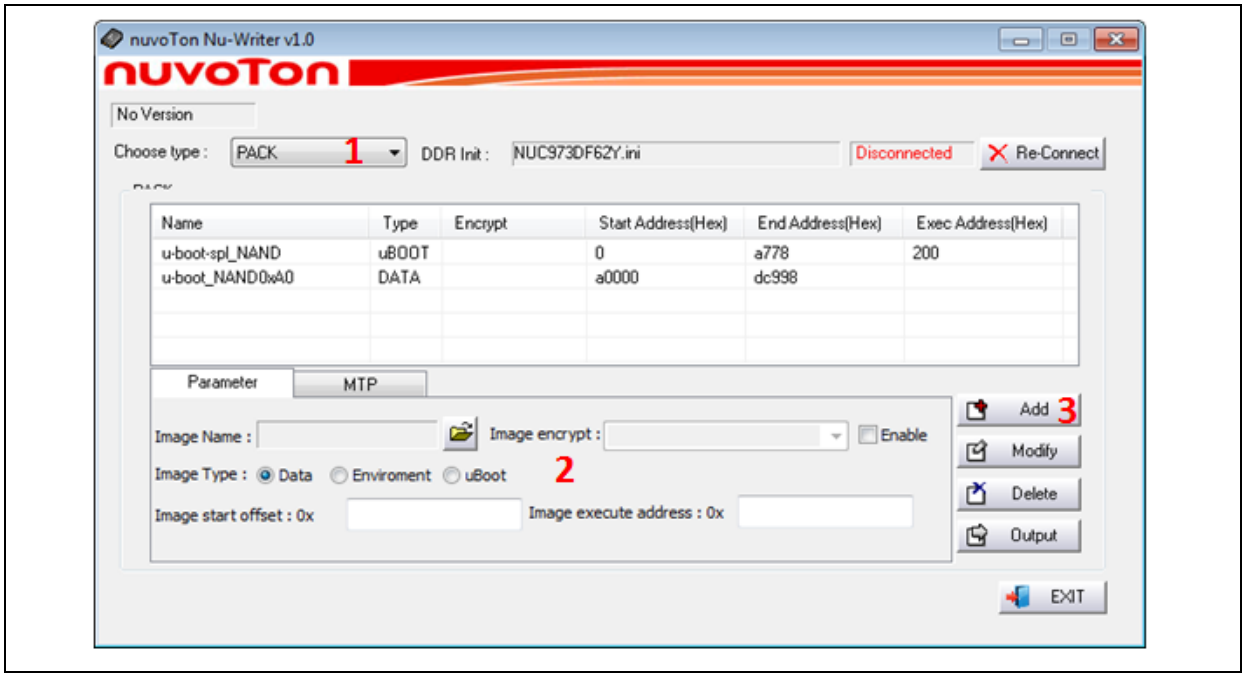
```
256
0x12345678
0x23456789
0x3456789a
0x456789ab
0x56789abc
0x6789abcd
0x789abcde
0x89abcdef
```

3. Re-open “Nu-Writer” tool, and select “MTP”.
4. Select “key.dat”.
5. Select burning option:
   - Boot mode selection: eMMC, NAND, or SPI.
   - Protection mode selection: SHA or AES.
6. Click “Burn”.

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).

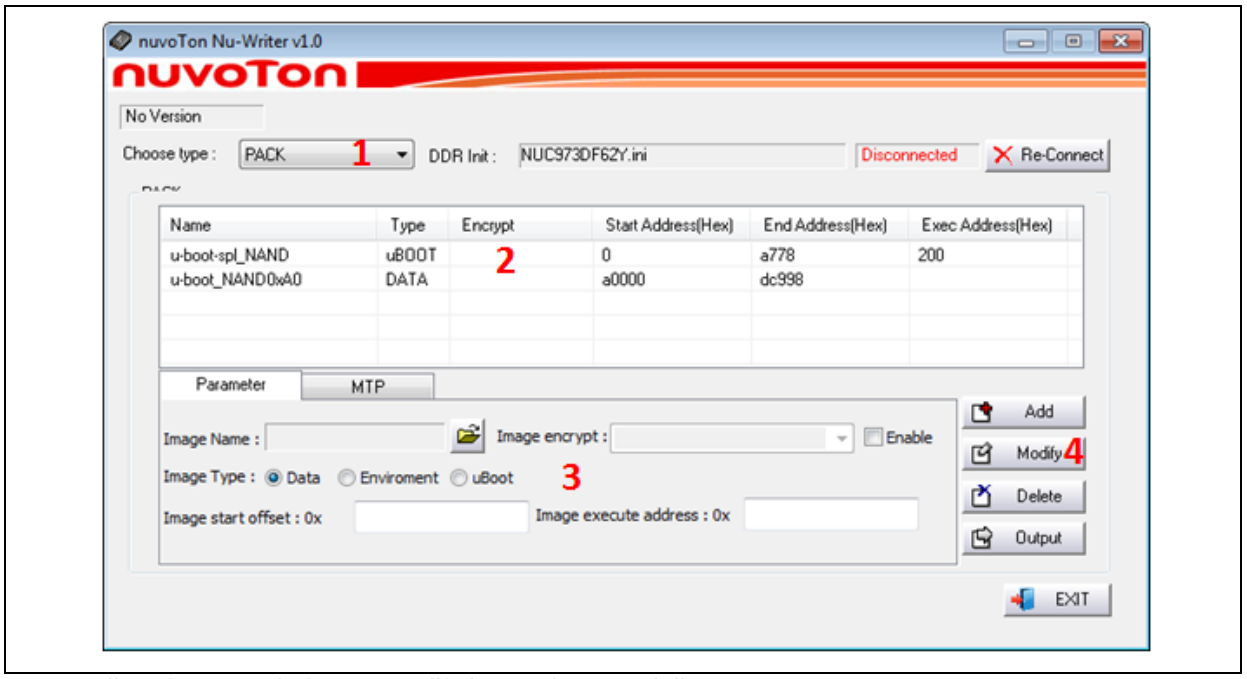
3.10.1 Add a New Image



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.

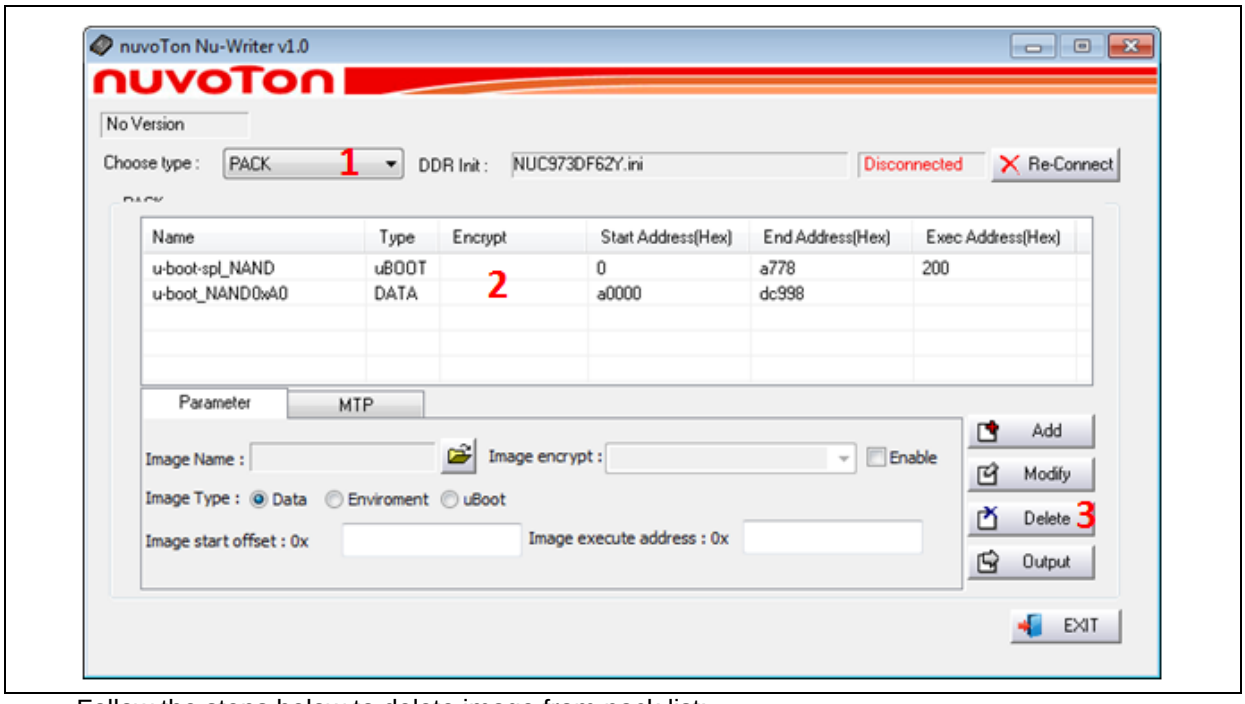
3.10.2 **Modify an Image**



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”

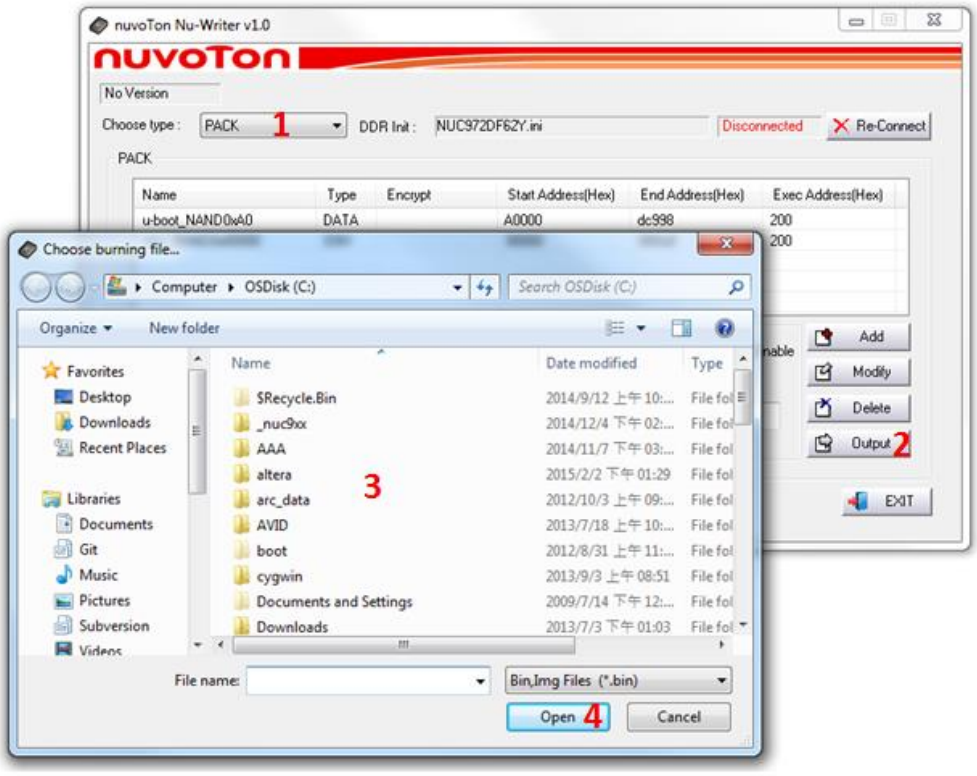
3.10.3 **Delete an Image**



Follow the steps below to delete image from pack list:

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

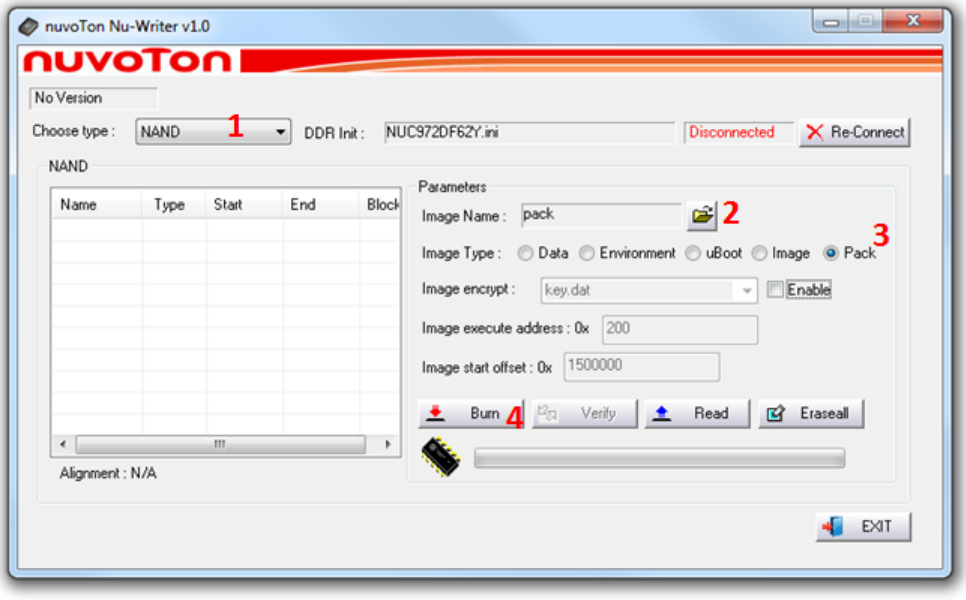
3.10.4 **Create** a **Pack** Image



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”.

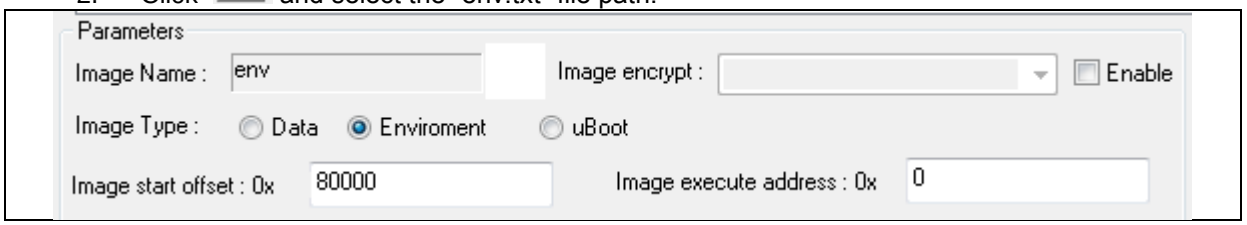
3.10.6 Create and Program a Pack Image

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).

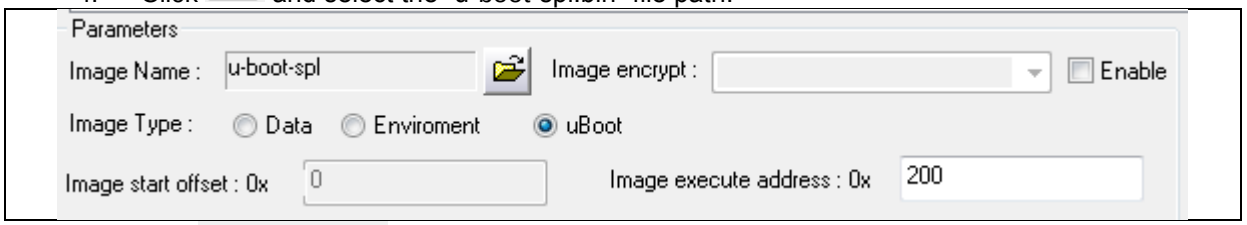
Suppose user wants to build a pack image includes env.txt, u-boot.bin and u-boot-spl.bin. And program to NAND flash. Follow the steps below to build pack image:

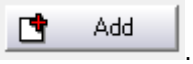
1. Select “PACK”.
2. Click  and select the “env.txt” file path:



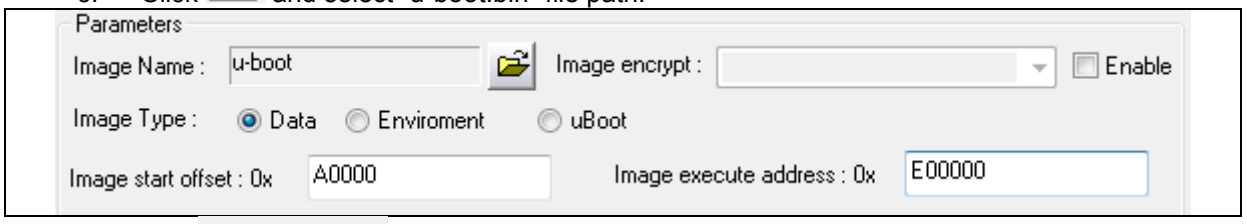
3. Click 

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

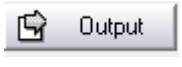


5. Click 

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



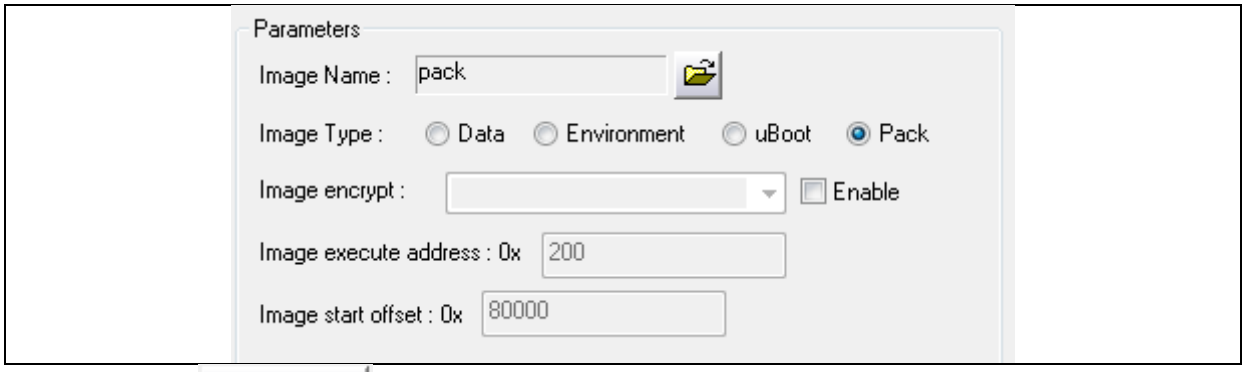
7. Click 

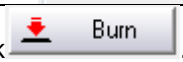
.
8. Click 

, 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 

.

3.11 Program U-Boot

Follow the steps below to program u-boot into NAND flash.

3.11.1 Preparation of related files:

1. u-boot.bin (Default: offset address is set to 0x100000,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).

The detail can reference 4.1 chapters.

3.11.2 U-Boot environment variable files (env.txt)

env.txt stored U-boot environment variable and value, as follows:

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

Each line as an environment variable, format as follows:
Variable = value
Space cannot exist between variable and value. Line breaks is used (0x0d, 0x0a).
Environment variable detail can reference to chapter 4.6.2.

3.11.3 Burn U-Boot into NAND Flash

1. Select “NAND”
2. Select u-boot-spl.bin , set image type to u-Boot mode, set image execute address to 0x200, Click “burn” to burn u-boot-spl.bin.
3. Select u-boot.bin, set image type to Data, set image start offset to 0x100000, click “burn” to burn u-boot.bin.
4. Select env.txt, set image type to Environment, set image start offset as 0x80000, click “burn” to burn env.txt.

3.11.4 Burn U-Boot into SPI Flash

1. Select “SPI”

2. Select u-boot.bin, set image type to u-Boot, set image execute address to 0xE00000, click “burn” to burn u-boot.bin.
3. Select env.txt, set image type to Environment mode, set image start offset to 0x80000, click “burn” to burn env.txt.

3.11.5 Burn U-Boot into eMMC

1. Select “eMMC”.
2. Select u-boot.bin, set image type is u-Boot, set image execute address to 0xE00000, click “burn” to burn u-boot.bin.
3. （image execute address is changed,detail can reference 4.3.3 chapters）
4. Select env.txt, set image type to Environment, set image start offset to 0x80000, click “burn” to burn env.txt .

3.12 Nu-Writer Trouble Shooting

Nu-Writer development is based on the Microsoft visual C++ 2008. If Nu-Writer cannot work, then your PC needs to install “Microsoft Visual C++ 2008 Redistributables” which can be downloaded from following URL: http://www.microsoft.com/en-us/download/details.aspx?id=29

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 4.7.2 .

- 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_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_EMAC0
//#define CONFIG_NUC970_EMAC1
#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_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_CMD_NAND_YAFFS2 1
#define CONFIG_YAFFS2          1
#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 needs 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 is stored.
- 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
```

- 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

```
/* 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 is under arch/arm/cpu/arm926ejs/nuc970.
- board: This directory contains board relative source code.
- The board relative source code of NUC970 is 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 is 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 the directory object code will be generated to, and can be omitted)

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 the directory object code will be generated to, and can be omitted)

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,
In this example, “RAM_TEXT = 0xE00000” means 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

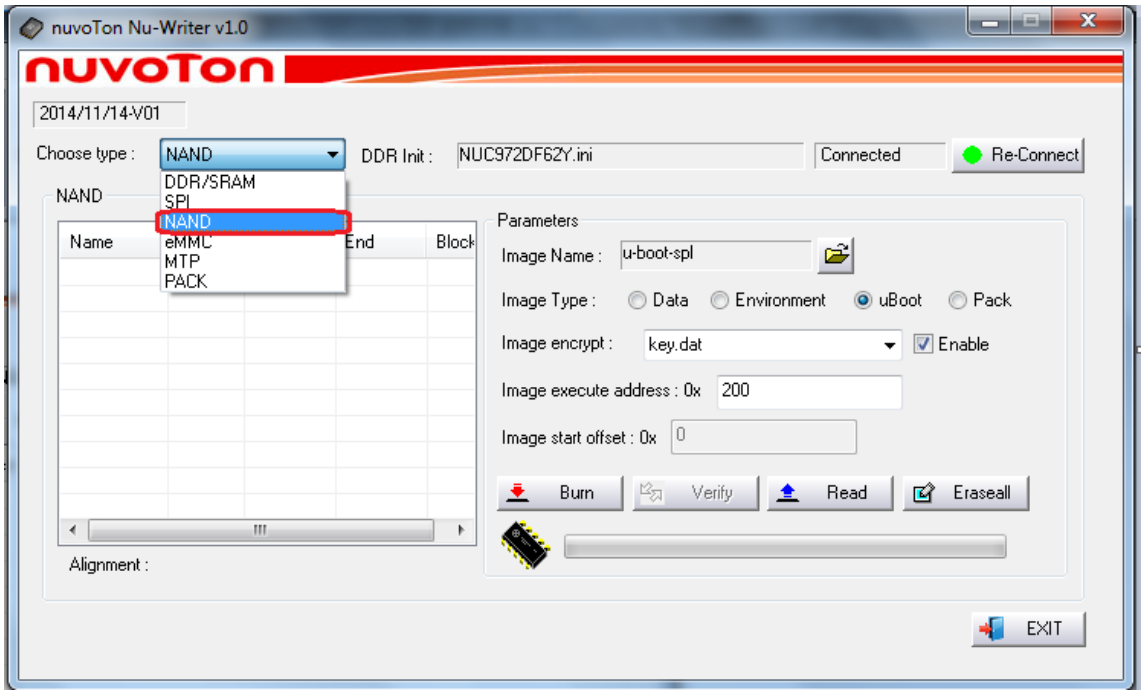
```
# make O=../build/nuc970_uboot/ distclean
# make O=../build/nuc970_uboot/ nuc970_config
# make O=../build/nuc970_uboot/ all
```

(make option “O” designates the directory object code will be generated to, and can be omitted)
After compilation success, find out the binary file of Main U-Boot and SPL U-Boot.

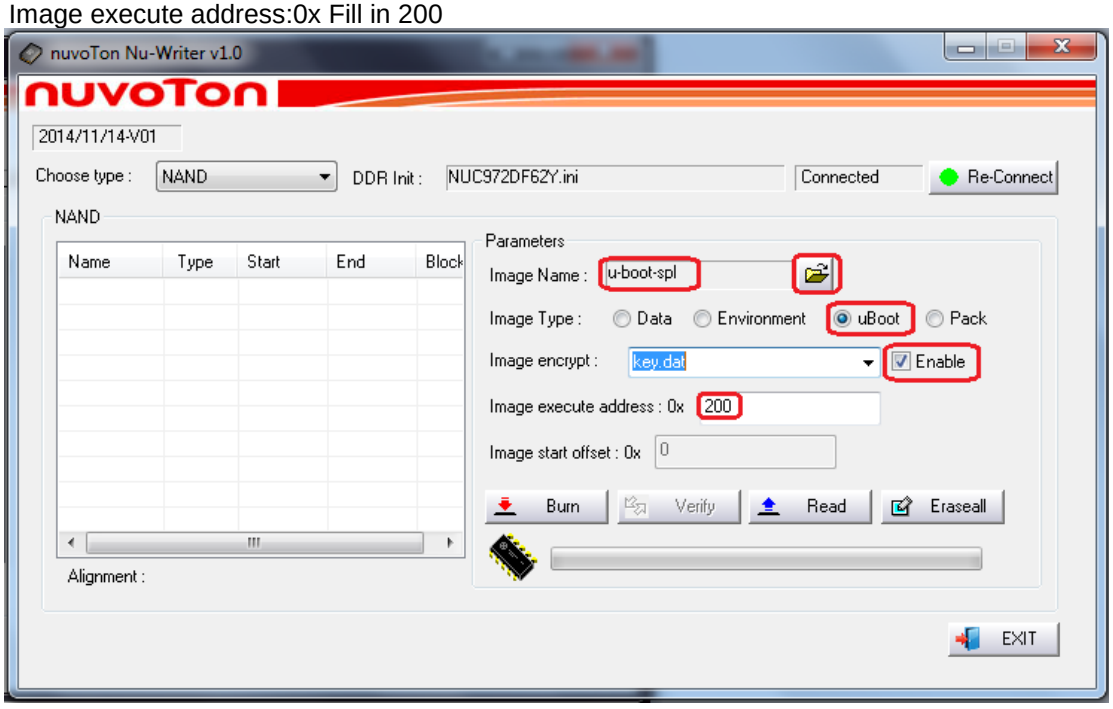
- Main U-Boot binary file is generated in root directory,file name is u-boot.bin
- SPL U-Boot binary file is generated in sub-directory nand_spl,file name is u-boot-spl.bin

4.4.2 Burn SPL U-Boot

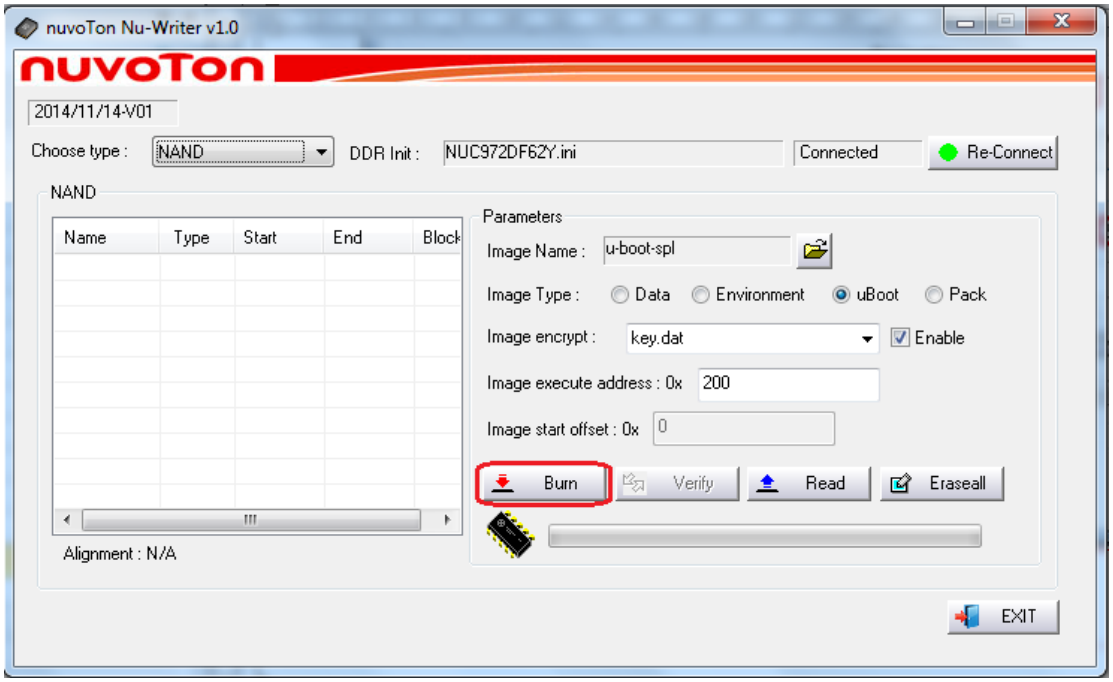
Choose type : select “NAND”.



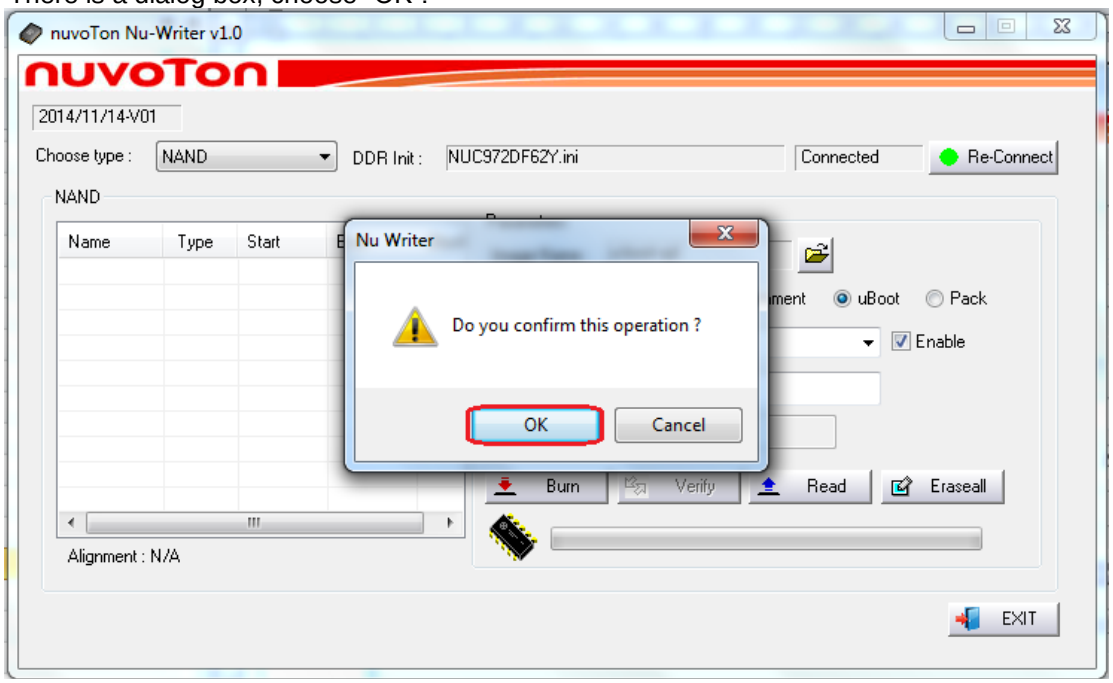
Then set Parameters as below picture,
Image Name: select u-boot-spl.bin,
Image Type: select uBoot,
Image encrypt: select Enable
Image execute address:0x Fill in 200



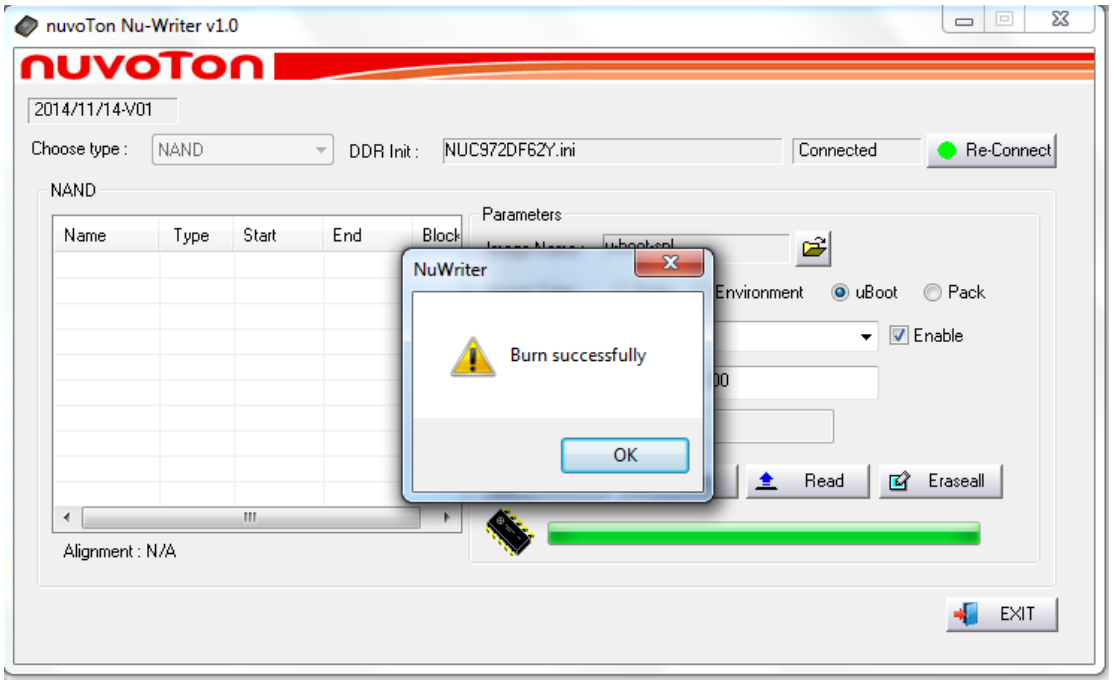
Then press “Burn” button.



There is a dialog box, choose “OK”.

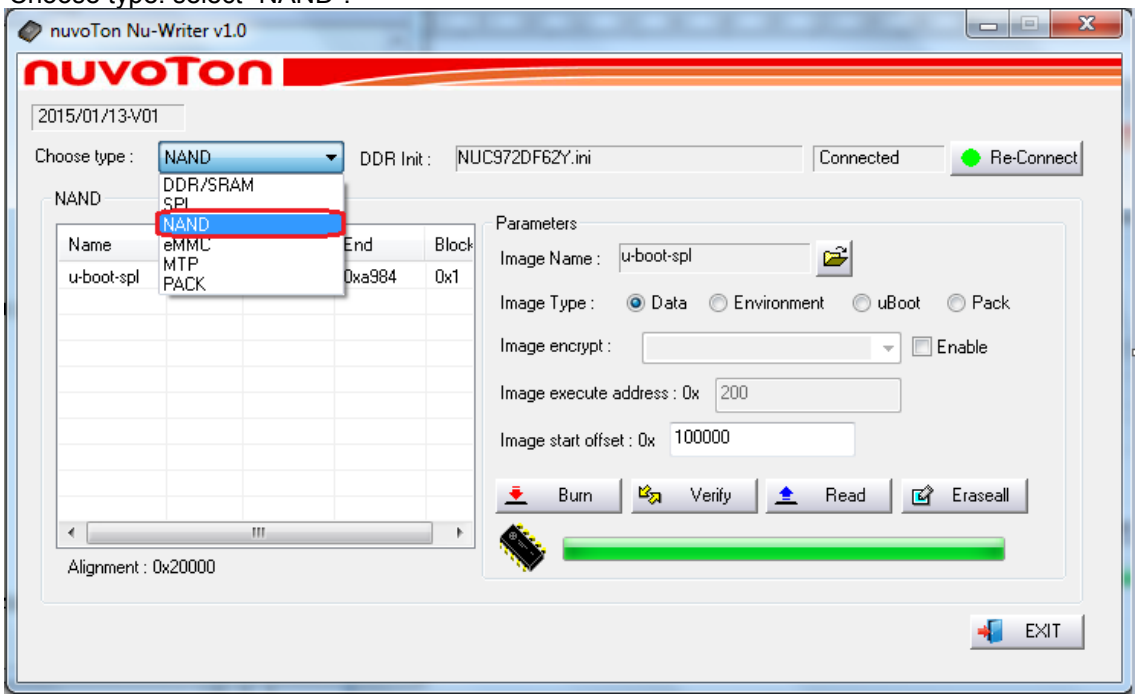


A dialog box shows burn successfully message.

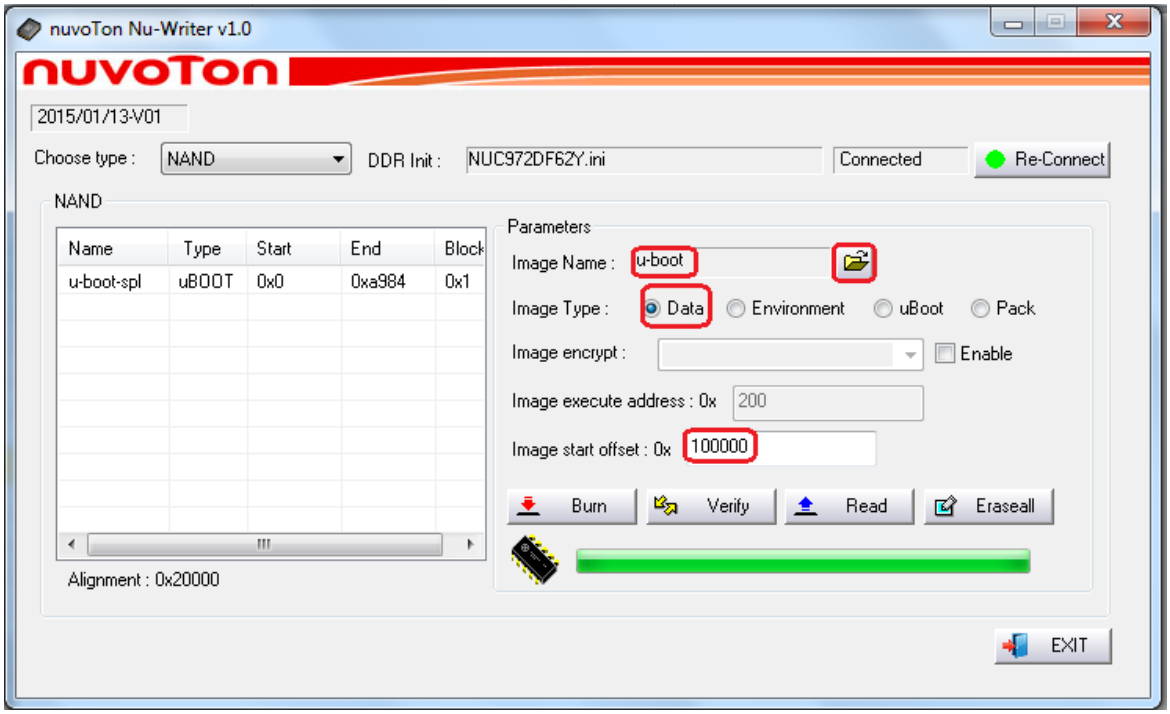


4.4.3 Burn Main U-Boot

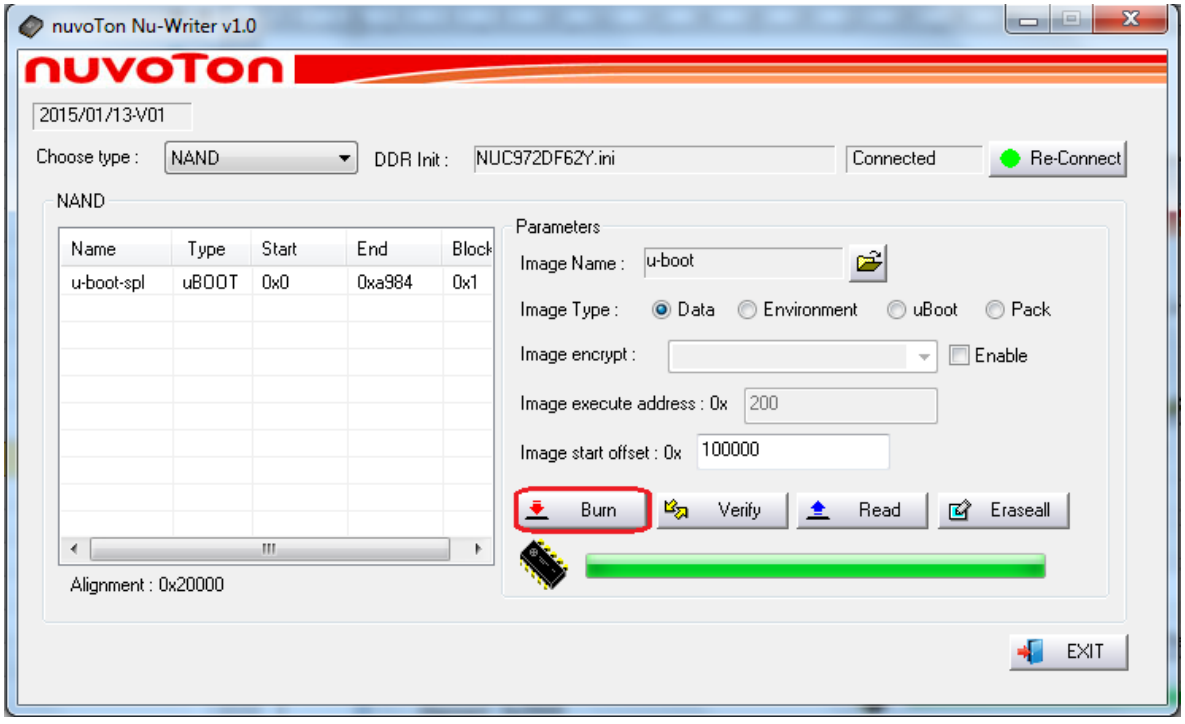
Choose type: select “NAND”.



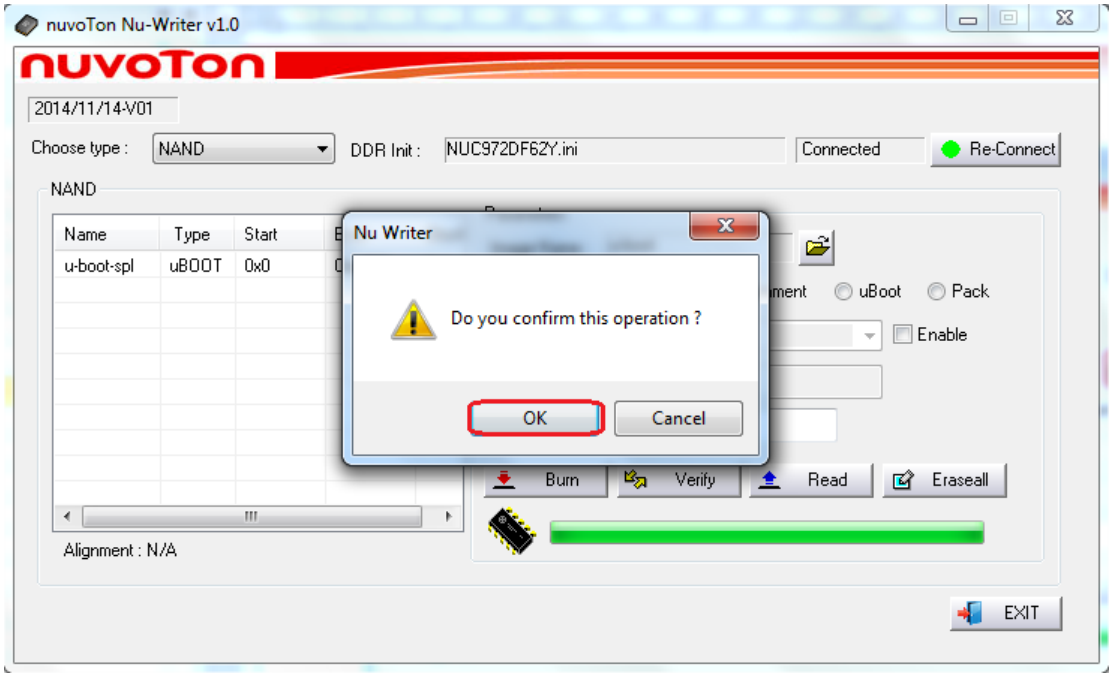
Set burning parameters.
Image Name: select u-boot.bin,
Image Type: select Data,
Image execute address:0x Fill in 100000



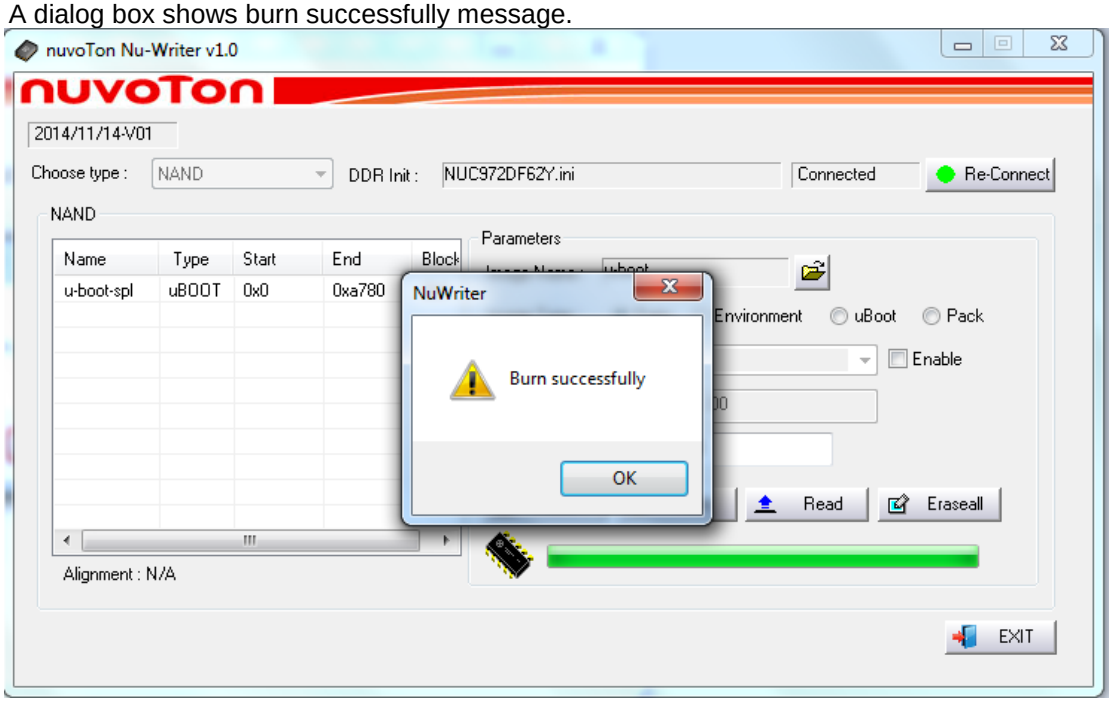
Press “Burn” button.



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

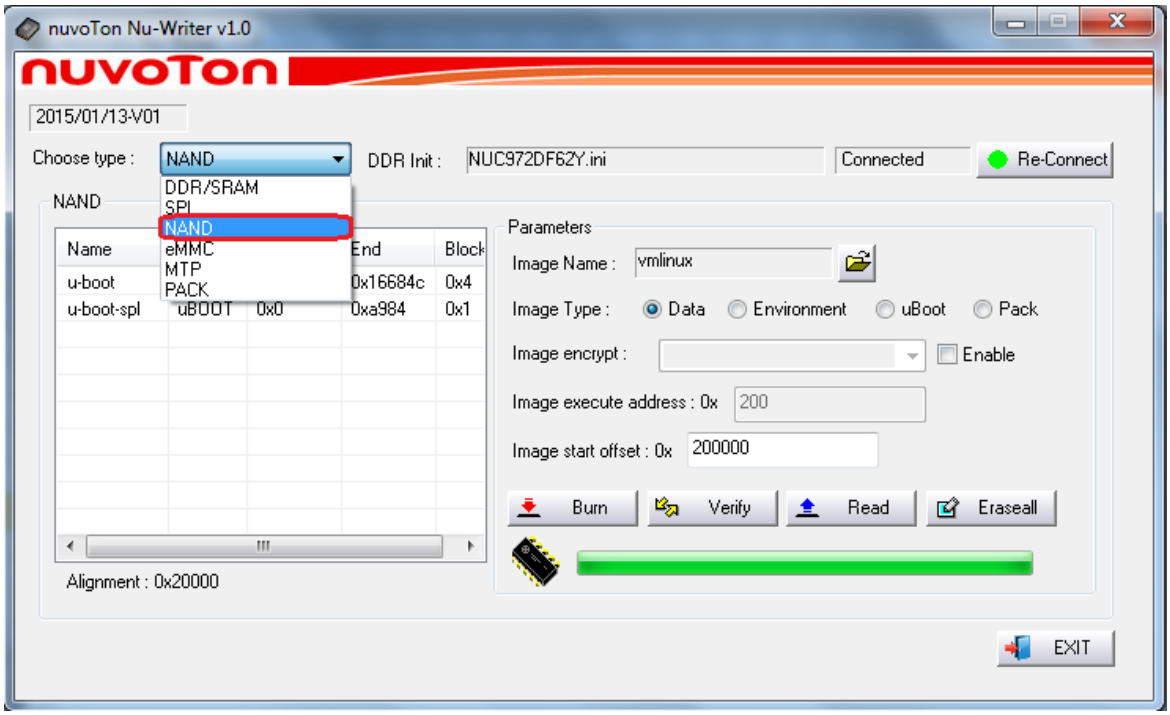


A dialog box shows burn successfully message.

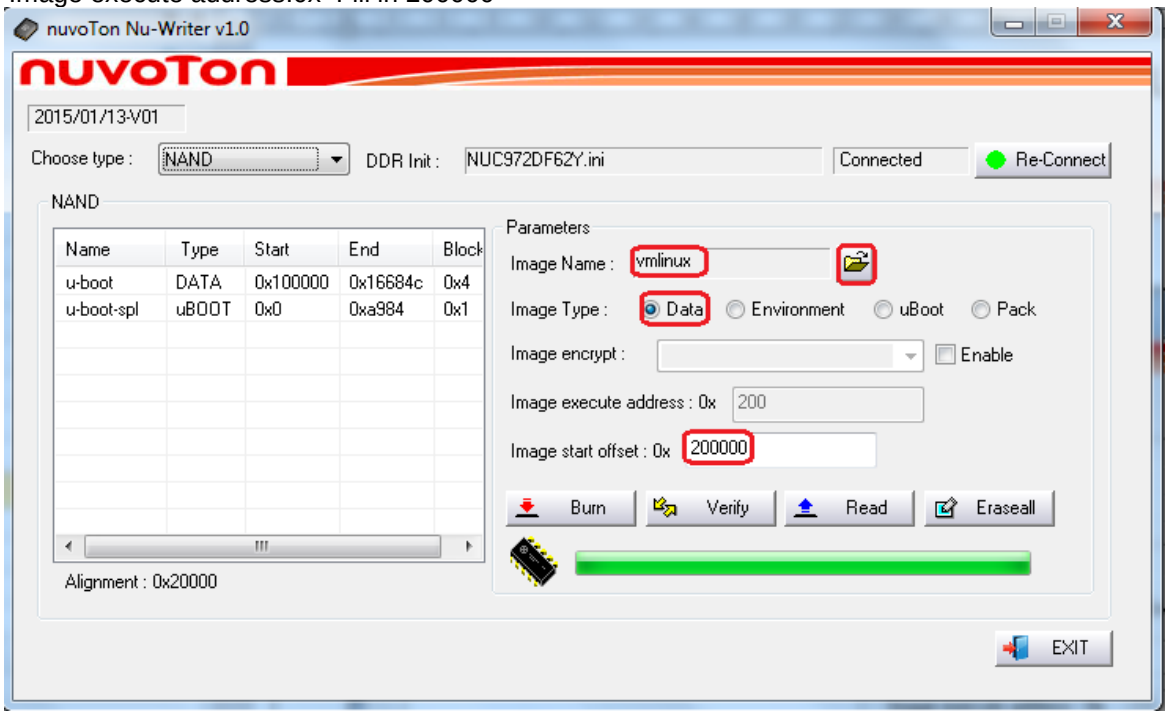


4.4.4 Burn Linux kernel

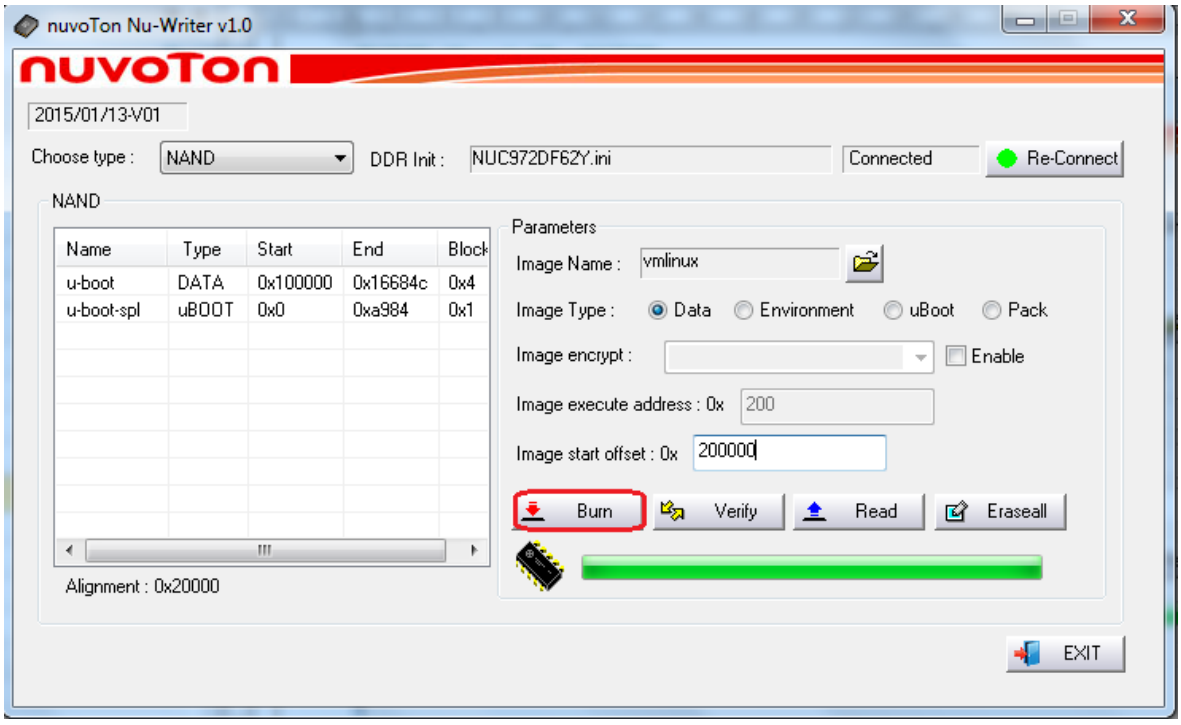
Choose type : select “NAND”.



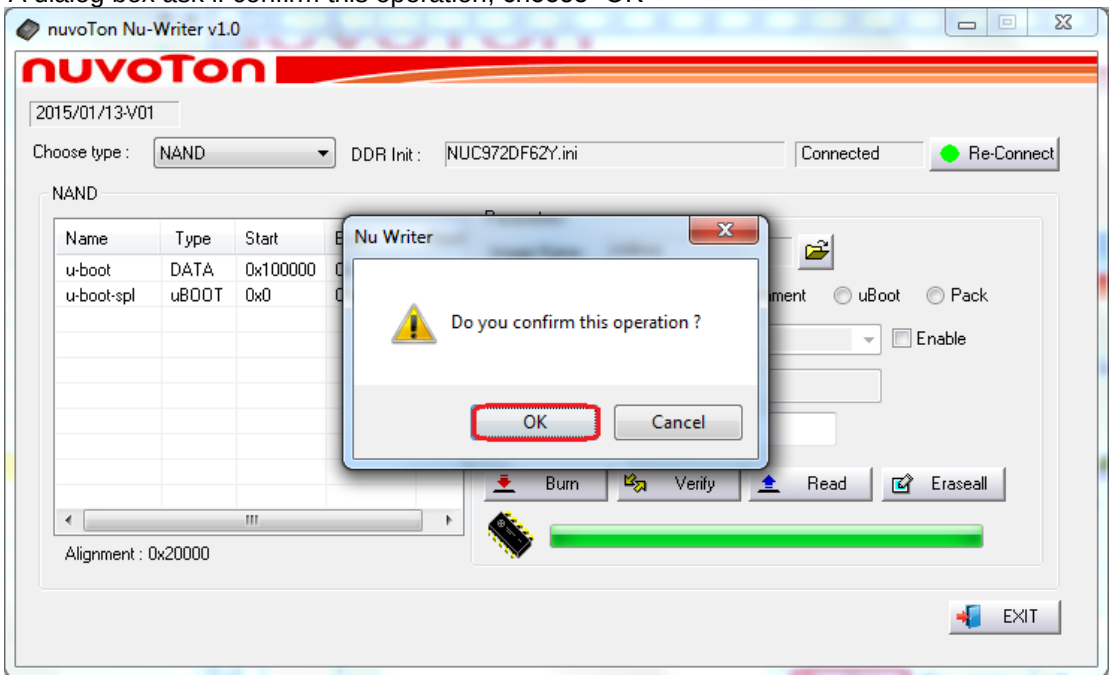
Set burning parameters.
Image Name: select vmlinux.ub (Please reference 4.7 to know how to generate vmlinux.ub).
Image Type: select Data,
Image execute address:0x Fill in 200000



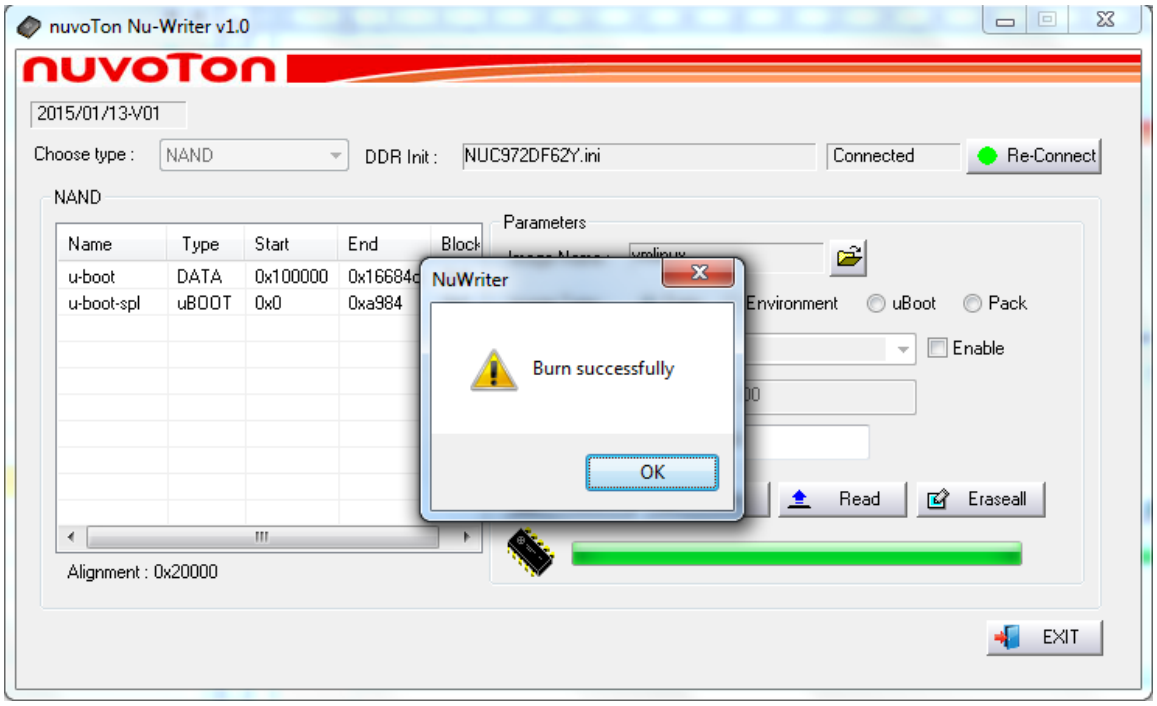
Press “Burn” button.



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

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 or other application program.
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 example is to start an application program that has been downloaded to DDR 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: bffbcf5c 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 6fffcd7f    s9..l)..^......o
00001050: cfd28a65 8cd69f2b efeece87 677f3b8f    e...+........;.g
00001060: def67b1d deff7ece 3ffd4003 ffbf32c2    .{...~...@.?.2..
00001070: feef5b67 ffdfa2e6 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 eadded74 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: 0a00006e 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 1affffee 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 commands

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 commands

- 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 mmclist 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 devide 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 MTD 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 default set MTD default partition value, the default value is defined in nuc970_evb.h. The setting is set MTD partition ID to nand0, the three default partitions are u-boot, kernel and user.
First partition: u-boot, start from 0x0, size is 0x200000.
Second partition: kernel, start from 0x200000, size is 0x1400000.
Third partition: user, start from 0x1600000, size is the rest space.

```
#define MTDIDS_DEFAULT “nand0=nand0”
#define MTDPARTS_DEFAULT “mtdparts=nand0:0x200000@0x0(u-
boot),0x1400000@0x200000(kernel),-(user)”
```

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>
```

4.5.12 **UBI commands**

- 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 [l[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":
0x000001500000-0x000008000000 : "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>
```

4.5.13 YAFFS2 commands

- yaffs : yaffs commands

U-Boot supports YAFFS commands, including mount/list/mkdir/rmdir/rd/w.
Format is as below:

```
U-Boot> help
ydevconfig- configure yaffs mount point
ydevls  - list yaffs mount points
yls     - yaffs ls
ymkdir  - YAFFS mkdir
ymount  - mount yaffs
ymv     - YAFFS mv
yrd     - read file from yaffs
yrdm    - read file to memory from yaffs
yrm     - YAFFS rm
yrmdir  - YAFFS rmdir
ytrace  - show/set yaffs trace mask
yumount - unmount yaffs
```

```
ywr     - write file to yaffs
ywrm    - write file from memory to yaffs
```

ydevconfig configure YAFFS mount point

```
U-Boot> ydevconfig
Bad arguments: ydevconfig mount_pt mtd_dev start_block end_block
U-Boot> ydevconfig nand 0 0xb0 0x3ff
Configures yaffs mount nand: dev 0 start block 176, end block 1023 using
inband tags
```

ydevls list YAFFS mount points

```
U-Boot> ydevls
nand            0 0x000b0 0x003ff using inband tags, not mounted
```

ymount mount YAFFS

```
U-Boot> ymount
Bad arguments: ymount mount_pt
U-Boot> ymount nand
Mounting yaffs2 mount point nandnand
U-Boot> ydevls
nand            0 0x000b0 0x003ff using inband tags, free 0x6573800
```

yls list the content of YAFFS file system, a mount point is a directory, previous example nand is a directory

```
U-Boot> yls
Bad arguments: yls [-l] dir
U-Boot> yls -l nand
lost+found                          2032     2 directory
```

ymkdir make a directory

```
U-Boot> ymkdir nand/test
U-Boot> yls -l nand
test                                2032   257 directory
lost+found                          2032     2 directory
```

yrmdir delete a directory

```
U-Boot> yrmdir nand/test
U-Boot> yls -l nand
lost+found                          2032     2 directory
```

ywr / ywrm write a file / save a block of memory to a file

```
U-Boot> ywr nand/wr.bin 0x55 100
Writing value (55) 100 times to nand/wr.bin... done
U-Boot> ywrm nand/wrm.bin 0xe00000 0x1000
U-Boot> yls -l nand
wrm.bin                             4096   259 regular file
```

```
wr.bin                                256   258 regular file
lost+found                           2032     2 directory
```

yrd / yrdm read a file / read a file to memory

```
U-Boot> yrd nand/wr.bin
Reading file nand/wr.bin
Done
U-Boot> yrdm nand/wrm.bin 0x200000
Copy nand/wrm.bin to 0x00200000...      [DONE]
```

yrm delete a file

```
U-Boot> yls -l nand
wrm.bin                              4096   259 regular file
wr.bin                                256   258 regular file
lost+found                           2032     2 directory
U-Boot> yrm nand/wr.bin
U-Boot> yls -l nand
wrm.bin                              4096   259 regular file
lost+found                           2032     2 directory
```

yumount unmounts YAFFS

```
U-Boot> yumount nand
Unmounting yaffs2 mount point nand
U-Boot> ydevls
nand           0 0x000b0 0x003ff using inband tags, not mounted
```

4.6 Environment variables

4.6.1 Environment variables configuration

Environment variables can store in NAND flash or SPI flash, user can modify below two 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

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>
```

- bootargs

This argument will be passed to Linux kernel. Below example is to pass the bootargs about NAND MTD partition to Linux kernel. Finally, remember to save environment variables to NAND flash.

```
U-Boot> set bootargs "root=/dev/ram0 console=ttyS0,115200n8
rdinit=/sbin/init mem=64M mtdparts=nand0:0x200000@0x0(u-
boot),0x1400000@0x200000(kernel),-(user)"
U-Boot> saveenv
Saving Environment to NAND...
Erasing Nand...
Erasing at 0xe0000 -- 100% complete.
Writing to Nand... 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 an ".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 (vmlinux.bin). Linux kernel download address is 0x7fc0 and execution

address is 0x8000.

```
u-boot# arm-linux-objcopy -O binary vmlinux vmlinux.bin
u-boot/tools# ./mkimage -A arm -O linux  -T kernel -a 0x7fc0 -e 0x8000 -d
vmlinux.bin vmlinux.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 vmlinux.bin

4.7.2 Checksum calculation (SHA-1 or crc32)

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 vmlinux.bin vmlinux.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 vmlinux.bin

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 vmlinux.bin vmlinux.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 vmlinux.bin

4.7.3 **AES encrypt**

Besides, NUC970 mkimage tool adds AES encrypt function added two new parameters, -E AES, Encrypt image. –K, designates the key file that AES encryption uses. Following example encapsulate and encrypt an ARM Linux kernel image (vmlinux.bin), 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 vmlinux.bin vmlinux.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 vmlinux.bin
-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, 8 lines total.
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 recomplation.

```
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 provides following 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.

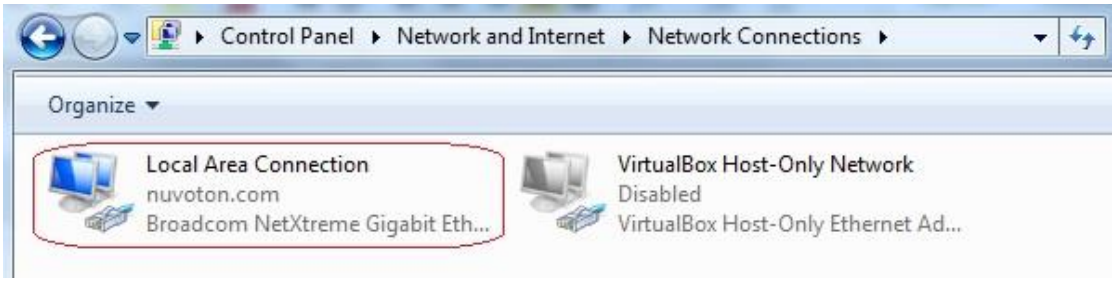
```
gpio_request(GPIO_PG0,NULL);
gpio_direction_output(GPIO_PG0, 0);
gpio_request(GPIO_PG1,NULL);
gpio_direction_output(GPIO_PG1, 1);
gpio_request(GPIO_PG2,NULL);
gpio_direction_output(GPIO_PG2, 0);
gpio_request(GPIO_PG3,NULL);
gpio_direction_output(GPIO_PG3, 1);
gpio_request(GPIO_PG4,NULL);
gpio_direction_output(GPIO_PG4, 0);
gpio_request(GPIO_PG5,NULL);
gpio_direction_output(GPIO_PG5, 1);
```

4.12 Network test environment

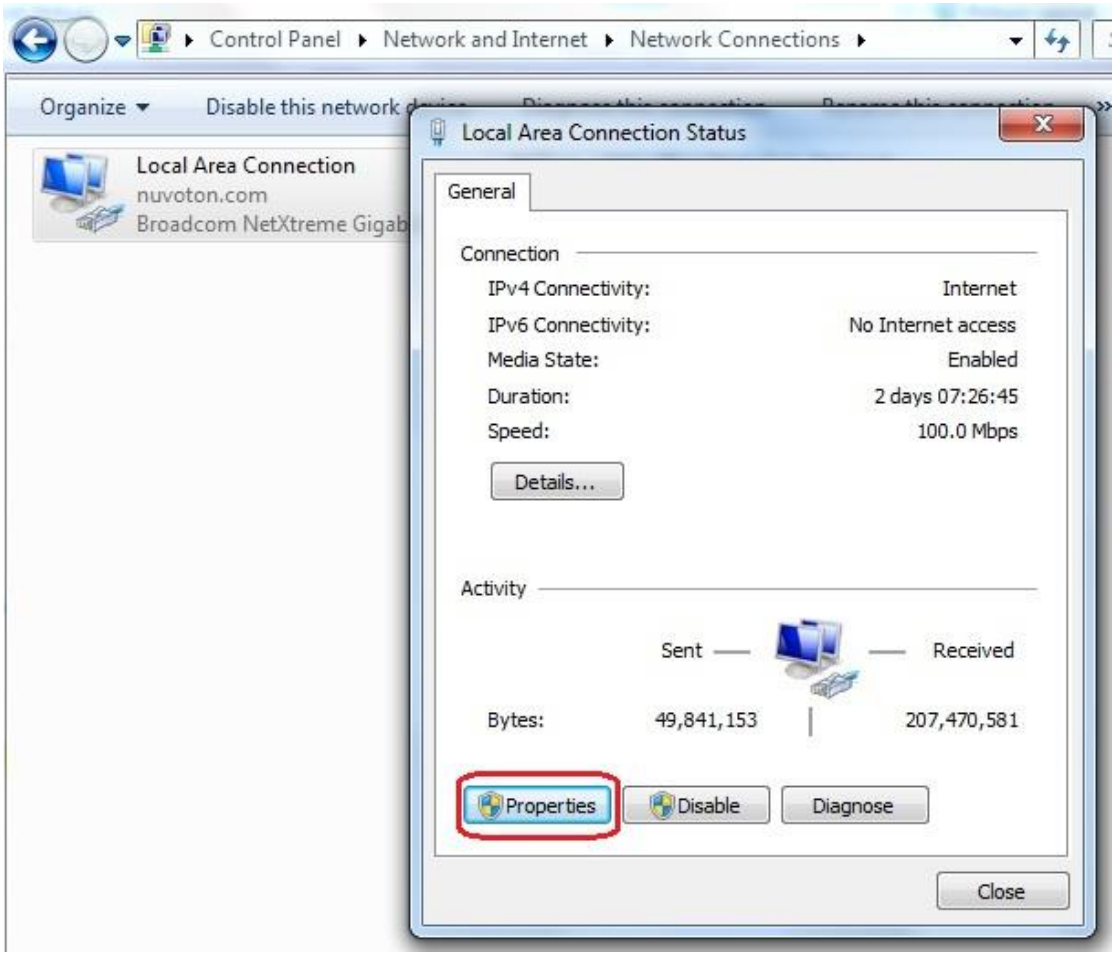
We need a PC that has static IP address and installed with TFTP/DHCP server application.

4.12.1 Set static IP address

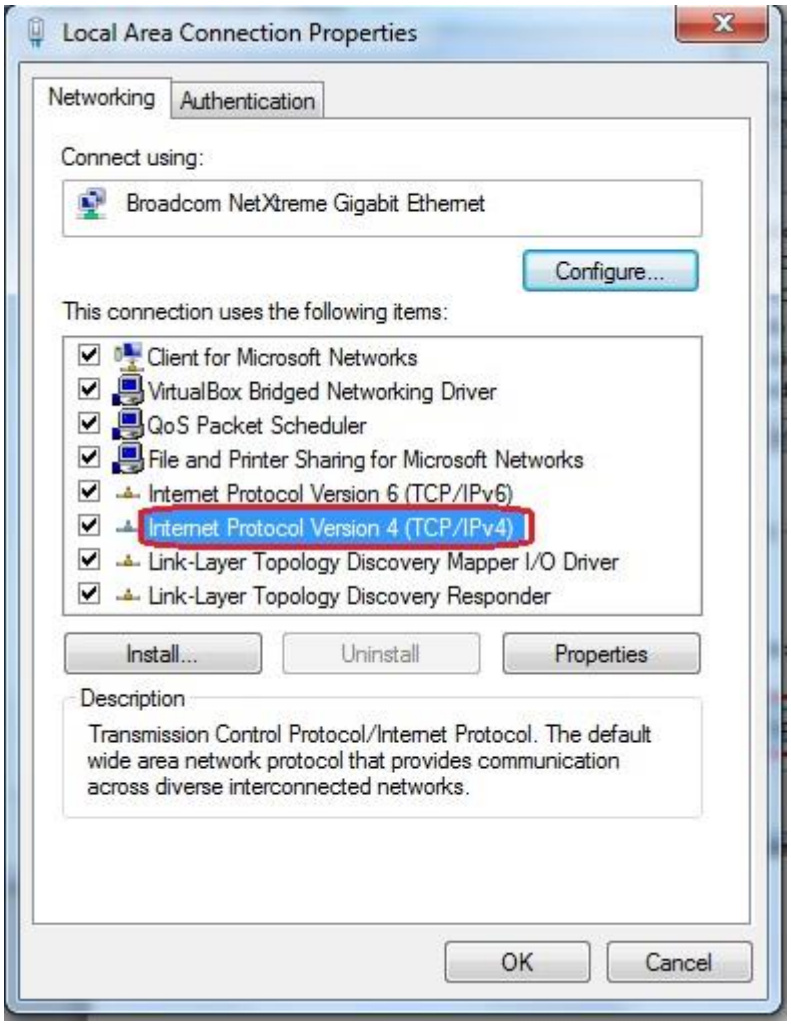
Choose Local Area Connection (under Control Panel -> Network and Internet -> Network Connections



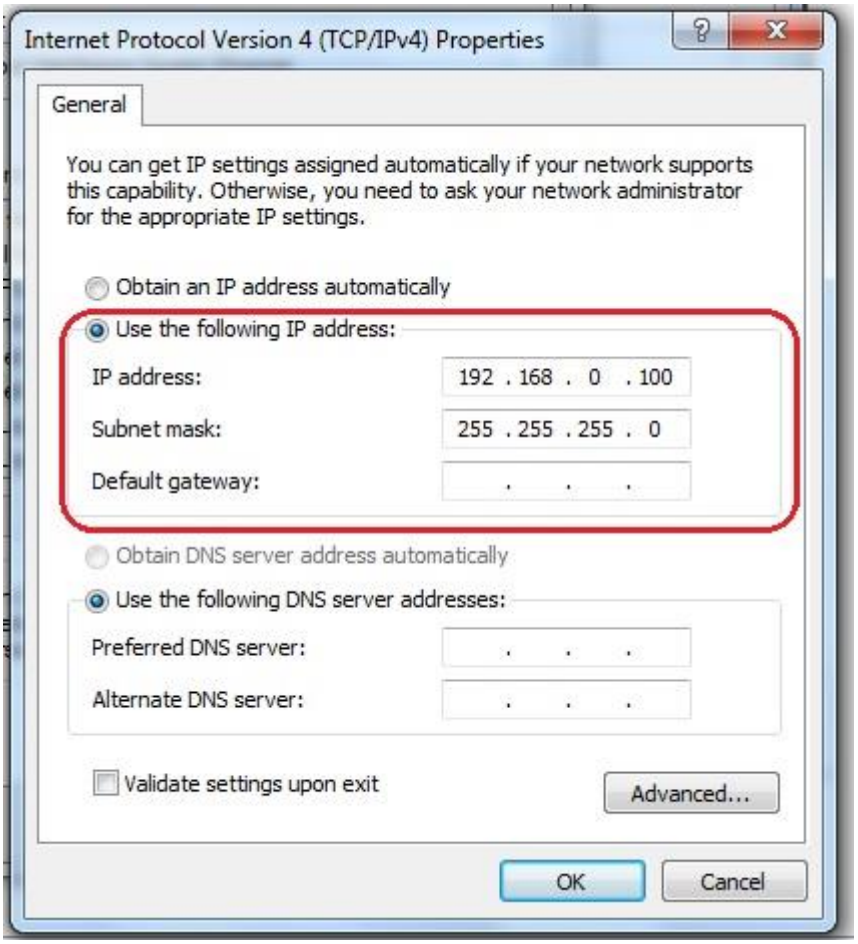
- Choose Properties



- Choose Internet Protocol Version 4 (TCP/IPv4)

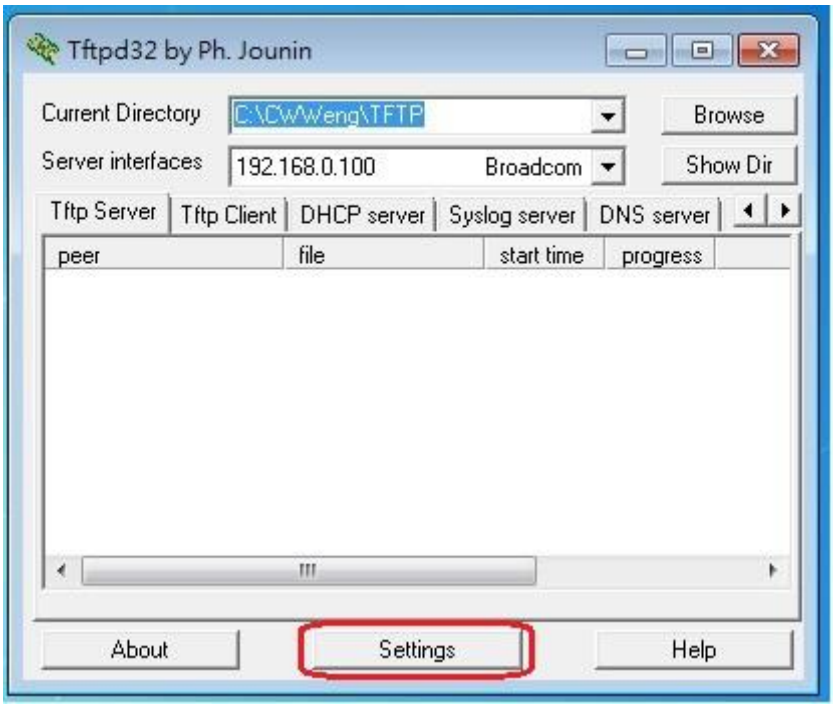


- Set static IP address

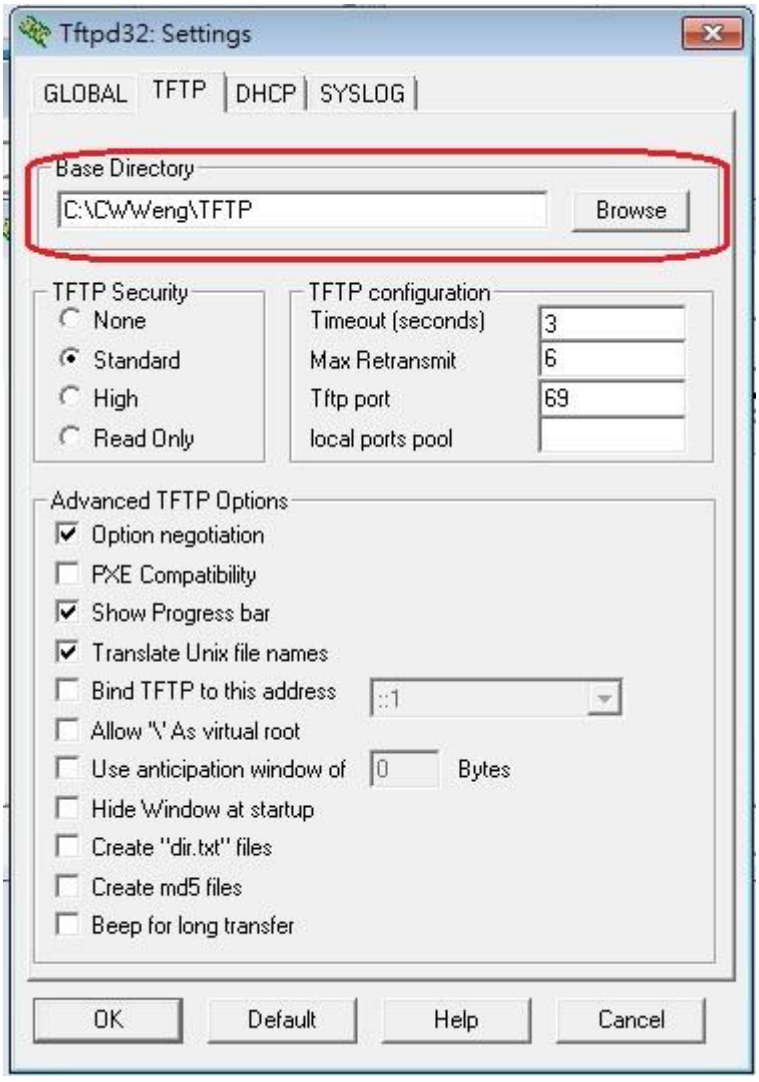


4.12.1 **TFTP and DHCP server**

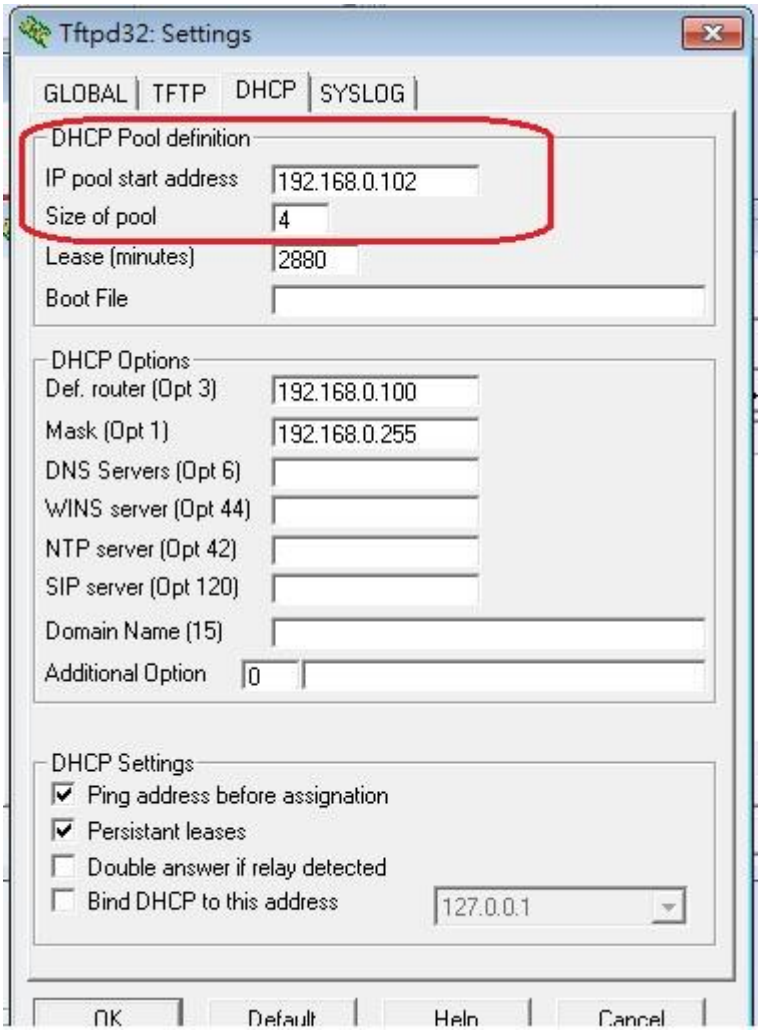
There is a free, open source application, TFTPD32, which includes TFTP and DHCP server. It can be downloaded by following URL
http://www.jounin.net/tftp-server-for-windows.html



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 Notice

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

5 Linux Kernel

5.1 The Configuration Interface for the Kernel

Linux supports different kinds of configuration. Users can disable some unnecessary functions to save resource of kernel system.
To enter the page of Linux configuration, please type “make menuconfig” command in shell.
It’s multi-layer menu in configuration system. In the current page, user can press “up”, “down”, ”left”, ”right” four keys to control the layer of configuration system. Select kernel function by pressing “up” or “down” key and select menu function in the bottom of page by pressing “left” or “right” key. To enter the next layer of configuration page, user can press “enter” key.
There are five functions at the bottom of menu page. User can disable or enable kernel function by pressing space key when cursor stays at “Select”. The symbol in front of the selection function “[ ]” stands for this function is disabled, “[*]” stands for this function is enabled and “[M]” stands for this function is built as module and can be loaded dynamically.
Menu page can be returned to upper layer by pressing space key when cursor stays at “Exit” at the bottom of menu page. If it’s at the top layer of configuration system, system will inform user if wants to save the configuration and exit.
The help screen will show when cursor is at “Help” by pressing space key. To save current configuration or load old configuration, use can press space key when cursor is at “Save” or “Load” at the bottom of menu page.
The kernel configuration file will be named “.config” and be saved in the linux-3.10.x directory.

5.2 Default Configuration

There is a default configuration for the NUC970 series chips provided by Nuvoton. Before modifying any configuration of kernel, we recommend to load the default configuration of kernel first. User can type “make <mcu name>_defconfig” command to do that. The option “<Mcu name> can be nuc972, nuc973, nuc976, nuc977. For example, type “make nuc972_defconfig” to load default configuration of nuc972. Sometimes if system can’t boot up, user can load the default configuration which is descripted above to recovery kernel to safe status.

5.3 Linux Kernel Configuration

This section introduces the configuration to enable kernel function according to different NUC970 driver or functions.

5.3.1 Basic Configuration of System

- Mount the module

Some drivers only support dynamic load, for example “USB WiFi driver” or “USB device driver” … and so on. Please enable the following function to support that. When system is booted up at shell, user can use “insmod <module name>” to load module.

```
[*] Enable loadable module support  --->
```

- Remove module

If some module drivers need to be removed by system, please enable the following function to support module removing. To remove module, user can use “rmmod <module name>” command to do that.

```
[*] Enable loadable module support  --->
     [*]   Module unloading
```

- Boot Options – root file system is based on RAM

Boot option can configure system, including the type of root file system, the size of memory, baud-rate of uart console… and so on. The following example is a simple configuration and there are many commands can be supported by kernel. User can refer to the document which is at Documentation/kernel-parameters.txt.

```
Boot options  --->
     (root=/dev/ram0 console=ttyS0, 115200n8 rdinit=/sbin/init mem=64M)
Default kernel command string
          Kernel command line type (Use bootloader arguments if available)
--->
```

- Boot Options – root file system is based on YAFFS2 (NAND Flash)

If root file system is at NAND flash and use YAFFS2 file system, user needs to enable YAFFS2 file system (please refer to 5.3.4) and disable RAM file system function.

```
General setup  --->
     [ ] Initial RAM filesystem and RAM disk (initramfs/initrd) support
```

The following is an example to boot up YAFFS2 root file system. A YAFFS2 root file system image (please refer to 3.8.4) needs to be done first and write it to the mtdblock2 in Linux system.

```
Boot options  --->
     (noinitrd root=/dev/mtdblock2 rootfstype=yaffs2 rootflags=inband-tags
console=ttyS0, 115200n8 rdinit=/sbin/init mem=64M) Default kernel command
string
          Kernel command line type (Use bootloader arguments if available)
--->
```

- Boot Options – root file system is based on JFFS2 (SPI Flash)

If root file system is at SPI flash and use JFFS2 file system, user needs to enable JFFS2 file system (please refer to 5.3.4) and disable RAM file system function.

```
General setup  --->
     [ ] Initial RAM filesystem and RAM disk (initramfs/initrd) support
```

The following is an example to boot up JFFS2 root file system. A JFFS2 root file system image needs to be done first by mkfs.jffs2 utility and write it to the mtd1 in Linux system

```
Boot options  --->
     (root=/dev/mtdblock1 rw rootfstype=jffs2 console=ttyS0,115200n8
rdinit=/sbin/init mem=64M) Default kernel command string
          Kernel command line type (Use bootloader arguments if available)
--->
```

- Boot Options – root file system is based on UBIFS (NAND Flash)

If root file system is at NAND flash and use UBIFS file system, user needs to enable UBIFS file system (please refer to 5.3.4) and disable RAM file system function.

```
General setup  --->
     [ ] Initial RAM filesystem and RAM disk (initramfs/initrd) support
```

The following is an example to boot up UBIFS root file system. A UBIFS root file system image (please refer to 3.8.4) needs to be done first and write it to the mtd2 in Linux system

```
Boot options  --->
     (noinitrd ubi.mtd=2 root=ubi0:system rw rootfstype=ubifs
console=ttyS0, 115200n8 rdinit=/sbin/init mem=64M) Default kernel command
string
          Kernel command line type (Use bootloader arguments if available)
--->
```

- Boot Options – root file system is based NFS (Network File System)

At the development stage of Linux application, user oftern wants to modify testing application. This method can reduce some development time by mounting NFS rootfs.

```
Boot options  --->
     (noinitrd root=/dev/nfs nfsroot=x.x.x.x:/path_to_nfs_rootfs
ip=y.y.y.y:z.z.z.z:g.g.g.g:m.m.m.m console=ttyS0,115200n8 rdinit=/sbin/init
mem=64M) Default kernel command string
```

x.x.x.x and z.z.z.z is the server ip, y.y.y.y is the client ip, g.g.g.g is the gateway ip and m.m.m.m is the net mask.

And user needs to enable network function (please refer to 5.3.2) and the following item additionally.

```
[*] Networking support  --->
     Networking options  --->
           [*]   IP: kernel level autoconfiguration
```

Of course, NFS function must be enabled.

```
File systems  --->
      [*] Network File Systems  --->
      <*>   NFS client support
      [*]   Root file system on NFS
```

5.3.2 **Network**

- TCP/IP

To enable basic network functions, please enable the following configurations.

```
 [*] Networking support  --->
     Networking options  --->
          <*> Packet socket
          <*> Unix domain sockets
          [*] TCP/IP networking
          [*]   IP: multicasting
```

- WiFi Wireless

If wireless device is used, user needs to enable the following functions additionally.

```
[*] Networking support  --->
     [*]   Wireless  --->
          <*>   cfg80211 - wireless configuration API
          [*]     cfg80211 wireless extensions compatibility
Device Drivers  --->
```

```
     [*] Network device support  --->
          [*]   Network core driver support
          [*]   Wireless LAN  --->
               <*>   IEEE 802.11 for Host AP (Prism2/2.5/3 and
WEP/TKIP/CCMP)
```

5.3.3 **Drivers**

- Audio Interface

The following is I²S interface configuration:

```
Device Drivers  --->
     <*> Sound card support  --->
          <*>   Advanced Linux Sound Architecture  --->
               <*>   ALSA for SoC audio support  --->
                    <*>   SoC Audio for NUC970 series
                    <*>     NUC970 I2S support for demo board
                              I2S Mode Selection (Master Mode)  --->
```

I2S supports master and slave mode, user can decide which mode is used by selecting in the configuration menu.
If I2S function is enabled, NAU8822 codec driver is also enabled automatically. In order to use I2S with audio codec function well, user needs to enable I2C function at the same time.
If audio application is wrote by old OSS architecture, user can enable the following two functions to do that. User can refer to the example in the BSP which source code is at BSP/applications/demos/alsa_audio.

```
Device Drivers  --->
     <*> Sound card support  --->
          <*>   Advanced Linux Sound Architecture  --->
               <*>   OSS Mixer API
               <*>   OSS PCM (digital audio) API
```

- Cryptographic Accelerator

In order to support Cryptographic Accelerator function, user needs to enable PF_KEY sockets function support in Networking support menu page.

```
[*]  Networking support  --->
     Networking options --->
          <*> PF_KEY sockets
```

Then enable Cryptographic API related functions.

```
Cryptographic API  --->
     <*>  Userspace cryptographic algorithm configuration
     [*]  Disable run-time self ests
     <*>  Software async crypto daemon
     <*>  User-space interface for hash algorithm
     <*>  User-space interface for symmetric key cipher algorithms
```

```
    [*]  Hardware crypto devices --->
         <*> Support for NUC970 Cryptographic Accelerator
```

NUC970 Cryptographic Accelerator function supports AES, DES and 3-DES crypto algorithm. It also supports SHA and HMAC hash algorithm. User can refer to the example named crypto (source is at BSP/applications/demos/crypto directory)

- DMA

DMA function is supported by NUC970 series chip. In order to support it in kernel, user needs to enable "NUC970 DMA support" in "DMA Engine support" menu page.
User can learn DMA functions in kernel by referring to source which is at linux-3.10.x/drivers/dma/dmatest.c. A test client will be also enabled by enabling "DMA Test Client" function, it's a shortcut to understand the procedure of DMA in kernel.

```
Device Drivers  --->
    [*]DMA Engine support  --->
         <*>   NUC970 DMA support
         <*>   DMA Test client
```

- User space memory management

If user wants to get a physical and virtual address of memory by enabling this user space memory management function.

```
Device Drivers  --->
 Character devices  --->
      [*] Support for /dev/nuc970-mem
```

- Ethernet

NUC970 series support two Ethernet ports. They can be enabled simulataniously. To support network port, PHY driver also needs to be enabled additionally.
The PHY chip on the development board is provided by ICPlus, the configuration will need to be modified if different PHY is used.

```
Device Drivers  --->
    [*] Network device support  --->
         <*>Dummy net driver support
         [*]   Ethernet driver support  --->
              <*>     Nuvoton NUC970 Ethernet MAC 0
              <*>     Nuvoton NUC970 Ethernet MAC 1
         -*-   PHY Device support and infrastructure  --->
              <*>   Drivers for ICPlus PHYs
```

- Etimer

When Linux kernel runs, it uses basic timer function of NUC970 to be timer. NUC970 also supports four enhanced timers which can output 50% duty cycle or capture function. Four channel of Etimer can be controlled individually.
The following is an example which Etimer channel 0 and channel 1 are as output by general purpose pin PC.6 and PC.8. And channel 2 and channel 3 are as capture pins by PC.11 and PC.13.
If the channel is unused, select the function to "No output" or "No input".

```
Device Drivers  --->
     Misc devices  --->
          <*> NUC970 Enhance Timer (ETIMER) support
               NUC970 ETIMER channel 0 toggle output pin (Output to PC6)  -
-->
               NUC970 ETIMER channel 0 capture input pin (No input)  --->
               NUC970 ETIMER channel 1 toggle output pin (Output to PC8)  -
-->
               NUC970 ETIMER channel 1 capture input pin (No input)  --->
               NUC970 ETIMER channel 2 toggle output pin (No output)  --->
               NUC970 ETIMER channel 2 capture input pin (Input from PC11)
--->
               NUC970 ETIMER channel 3 toggle output pin (No output)  --->
               NUC970 ETIMER channel 3 capture input pin (Input from PC13)
--->
```

Application can control etimer function by ioctl() function. The driver supports toggle out, tiger counting mode and free counting mode functions now.
The value captured by PWM at capture mode (trigger counting mode and free counting mode) can be read back by using read() function. The unit of value is us, it stands for time interval between two triggers. No matter using toggle out or capture mode, the unit is us.
User can refer to example code in the BSP (source code path is at BSP/applications/demos/etimer) to develop the related application.

- Smartcard

NUC970 series has two smartcard interfaces that comply with ISO-7816 and EMV 2000 specification. If the system needs to access smartcard, please refer to the kernel configuration below to enable smartcard driver. This driver supports both T = 0 and T = 1 protocols. The card detection level and power-on level, which is depend on the on board circuit and card slot design can be configured individually. Besides, ether Port I or Port G can be selected for smartcard interface 0. When enable Perform EMV check checkbox, the driver will perform a more strict protocol check comply with EMV 2000, so some smartcards will be reported as faliuare cards.

```
Device Drivers  --->
     Misc devices  --->
          <*> NUC970 Smartcard Interface support
          [ ]   Perform EMV check
          [*]   NUC970 SC0 support
                  NUC970 SC0 pin selection (Use port I)  --->
                  NUC970 SC0 CD pin config (CD high as card insert)  --->
          [ ]     Inverse SC0 power pin level
          [*]   NUC970 SC1 support
                  NUC970 SC1 CD pin config (CD high as card insert)  --->
          [ ]     Inverse SC1 power pin level
```

User applications can control smartcard using ioctl() function call. Below listed the commands support by smartcard driver and their purpose. User can refer to example code in the BSP (source code path is at BSP/applications/demos/sc) for the usage of these commands.
SC_IOC_GETSTATUS: Check slot staus, for example card inserted or removed..
SC_IOC_ACTIVATE: Activate smartcard, report ATR length if success.

SC_IOC_READATR: Read the ATR (Answer to reset) of activated smartcard.
SC_IOC_DEACTIVATE: Deactivate smartcard.
SC_IOC_TRANSACT: Send ADPU command and read response through sc_transact structure.

- GPIO

In order to support GPIO function controlled by kernel, please enable “NUC970 GPIO support” and “/sys/class/gpio/…”function.

```
Device Drivers  --->
     [*] GPIO Support  --->
          [*]   /sys/class/gpio/... (sysfs interface)
               <*>   NUC970 GPIO support
```

The number of each GPIO pin will be descripted at the following.
Driver will keep 32 numbers for each group of GPIO from port A to port J. So the number for the GPIOA will be 0x000~0x01F, GPIOB will be 0x020~0x03F, GPIOC will be 0x040~0x05F, GPIOD will be 0x060~0x07F, GPIOE will be 0x080~0x09F, GPIOF will be 0x0A0~0x0BF, GPIOG will be 0x0C0~0x0DF, GPIOH will be 0x0E0~0x0FF, GPIOI will be 0x100~0x11F and GPIOJ will be 0x120~0x13F.
Application can control each GPIO port by using sysfs. The following is the description of GPIO action based on sysfs interface.

- /sys/class/gpio/export : which GPIO pin will be exported
- /sys/class/gpio/unexport: which GPIO pin will be un-exported
- /sys/class/gpio/gpio0/direction : set GPIOA0 direction to in or output
- /sys/class/gpio/gpio0/value : set or read the value to/from GPIOA0

The following is an example to let GPIOA0 output high:

```
$ echo 0 > /sys/class/gpio/export
$ echo out >/sys/class/gpio/gpio0/direction
$ echo 1 >/sys/class/gpio/gpio0/value
```

User also can refer to the example which source code is at BSP/applications/demos/gpio.
The driver can also control GPIO pin by the following steps.

- Add #inlcude <linux/gpio.h> in the target driver.
- Decide which GPIO pin will be use according to the definition in the arch\arm\mach-nuc970\inlcude\mach\gpio.h.

Take NUC970_PC7 GPIO pin as example.

Set to input mode	gpio_direction_input(NUC970_PC7);
Set to output mode and value	gpio_direction_output(NUC970_PC7,1);
Set to output high	gpio_set_value(NUC970_PC7, 1);
Set to output low	gpio_set_value(NUC970_PC7, 0);
Read the value	gpio_get_value(NUC970_PC7);
Check if GPIO is in use	gpio_request(NUC970_PC7, "NUC970_PC7");
Get the GPIO interrupt number:	gpio_to_irq(NUC970_PC7);

Example:

```
static irqreturn_t PC7IntHandler(int irq, void *dev_id)
{
printk(KERN_INFO "PC7IntHandler:irq=%d \n",irq);
    return IRQ_HANDLED;
}
```

```
int xxx_init(void)
{
int ret,irqno;
ret = gpio_request(NUC970_PC7, "NUC970_PC7");
if (ret) printk("NUC970_PC7 failed ret=%d\n",ret);
irqno=gpio_to_irq(NUC970_PC7);
request_irq(irqno,  PC7IntHandler,
                 IRQF_TRIGGER_RISING | IRQF_TRIGGER_FALLING,
                "NUC970_PC7",
                 NULL);
}
```

- Use GPIO to simulate I²C interface

User can use GPIO to simulate I2C function. Please enable the following function to do that.

```
Device Drivers  --->
<*> I2C support  --->
I2C Hardware Bus support  --->
<* > GPIO-based bitbanging I2C
```

User can select I2C pin by modifying i2c_gpio_adapter_data structure in arch/arm/mach-nuc970/dev.c. For example, .sda_pin = NUC970_PG1, .scl_pin = NUC970_PG0 will use PG0 as SCL pin and PG1 will be SDA pin.

- I²C

The configuration of I²C is list as following:

```
Device Drivers  --->
     <*> I2C support  --->
          I2C Hardware Bus support  --->
               <*> NUC970 I2C Driver for Port 0
               <*> NUC970 I2C Driver for Port 1
               NUC970 I2C1 pin selection (Port G)  --->
```

There are many groups can be select for I2C port 1, like GPIO port-B, port-G, port-H or port-I.
If I2C function support is selected in kernel configuration, kernel will use build in I2C interface of NUC970 to communicate with other device.
The BSP builds in five I2C port 0 client devices. There are OV7725, OV5640, NT99050, NT99141 and NAU8822 by default. User can modify those devices to I2C port1 by modifying nuc970_i2c_client1 structure in arch/arm/mach-nuc970/dev.c.

- LCD

To enable LCD function support, please enable the following function in kernel configurations.

```
Device Drivers  --->
     Graphics support  --->
          <*> Support for frame buffer devices  --->
                [*]   NUC970 LCD framebuffer support
```

```
                    NUC970 LCD panel selection (800x480 5-Inch Color TFT
LCD)  --->
                    LCD source format (RGB888 support)  --->
          Console display driver support  --->
               <*> Framebuffer Console support
```

There is a LCD display screen in on development board which resolution is 800x480 and use 24-bit data bus connected with NUC970 LCD interface. So the color of this display screen is RGB888(24-bit). Color depth can be adjusted according to user space application.
To display Linux content on the LCD screen, please enable the following functions.

```
[*] Bootup logo  --->
[*]   Standard 16-color Linux logo
[*]   Standard 224-color Linux logo
```

There is an example demonstrated the operations of frame buffer which source code is at BSP/applications/demos/lcm directory.

- 2D Graphic Engine

NUC970 supports 2D graphic drawing function like line, rectangle, rotation, scale up/down and BitBlt.

```
Device Drivers  --->
    Generic Driver Options  --->
         Misc devices  --->
              <*> NUC970 2D support
```

- MTD NAND flash

To enable NAND flash function, user needs to enable the following function in kernel configuration. There are two groups of pin can be selected when using NAND flash interface. There are GPIO port C and port I, it depends on the connection of hardware on board.

```
Device Drivers  --->
    Generic Driver Options  --->
         <*> Nuvoton NUC970 FMI function selection
              Select FMI device to support (Support MTD NAND Flash)  --->
    -*- Memory Technology Device (MTD) support  --->
         <*>   Command line partition table parsing
         <*>   Caching block device access to MTD devices
         -*- NAND Device Support  --->
              -*-   Nuvoton NUC970 MTD NAND  --->
                   NUC970 NAND Flash pin selection (Port C)  --->
```

It's necessary to enable "Command line partition table parsing" function when the basic configuration of flash driver is passed from U-Boot.
The default configuration of flash driver will partition MTD into three blocks, there are /dev/mtdblock0, /dev/mtdblock1, /dev/mtdblock2 when system boots up.
The first block is the space for the U-Boot, second one is for the kernel and the last one is for mounting YAFFS2 or UBIFS.
If user wants to modify the block size or increase or decrease number of partition, please modify uboot/include/nuc970_evb.h or drivers/mtd/nand/nuc970_nand.c.

- PWM

To enable PWM function, user needs to enable the following functions. The PWM pin maybe needs to change according to hardware connection.
"No output" is stand for those unused PWM channels.

```
Device Drivers  --->
     [*] Pulse-Width Modulation (PWM) Support  --->
          <*>   NUC970 PWM support
                NUC970 PWM channel 0 output pin (Output from PB2)  --->
                NUC970 PWM channel 1 output pin (Output from PB3)  --->
                NUC970 PWM channel 2 output pin (No output)  --->
                NUC970 PWM channel 3 output pin (No output)  --->
```

This section will descript PWM control method by using sysfs. After system boots up, there are four PWM (pwmchip0~3) in /sys/class/pwm directory. Each group stands for one PWM channel. Before using it, enter target PWM directory and execute "echo 0 > export" command to enable this PWM channel. If enable success, there is a pwm0 directory will be created and user can control this PWM channel.
There are some files in the new created directory, their meaning is list as the following table.

File Name	Purpose
period	Control cycle, which the unit is ns. The shortest time supported by the driver is us. Example (control cycle is 20us) $ echo 20000 > period
duty_cycle	Set duty cycle of PWM, which the unit is ns. The shortest time supported by the driver is us. Example (duty cycle is 15us) $ echo 15000 > duty_cycle
polarity	Set polarity, it can be normal or inverse output. Example: Normal output: $ echo normal > polarity Inverse output:$ echo inversed > polarity
enable	Enable or disable function. Example: Enable function: $echo 1 > enable Disable function: $echo 0 > enable

The following is a PWM0 example which control cycle is 300us and duty cycle is 33%.

```
$ cd sys/class/pwm
$ ls
pwmchip0  pwmchip1  pwmchip2  pwmchip3
$ cd pwmchip0
$ ls
device     export     npwm       power      subsystem  uevent     unexport
$ echo 0 > export
$ ls
device     npwm       pwm0       uevent
export     power      subsystem  unexport
$ cd pwm0/
```

```
$ ls
duty_cycle  enable      period      polarity    power       uevent
$ echo 1 > enable
$ echo 300000 > period
$ echo 100000 > duty_cycle
```

- Ralink RT3070 802.11 WiFi

To support RT3070 USB WiFi module, user needs to enable wireless network function, USB host, loadable module support and the following function.

```
Generic Driver Options -->
[*] Contiguous Memory Allocator
```

And add the additional command in boot command.

```
coherent_pool=2M
```

The RT3070 driver source code is in BSP/applications/DPO_RT3070_LinuxSTA_V2.3.0.2_20100412 directory. The output after compiling is kernel module and can be loaded dynamically.
Before compiling the source code, user needs to modify Makefile like the following in the source file directory.

```
ifeq ($(PLATFORM),SMDK)
LINUX_SRC = /home/bhushan/itcenter/may28/linux-2.6-samsung
CROSS_COMPILE = /usr/local/arm/4.2.2-eabi/usr/bin/arm-linux-
endif

ifeq ($(PLATFORM),NUC900)
LINUX_SRC = /home/andy/hdb/linux_kernel/linux-2.6.35.y
CROSS_COMPILE = arm-linux-
endif

ifeq ($(PLATFORM),NUC970)
LINUX_SRC = /PATH_TO_LINUX_KERNEL/linux-3.10.x
CROSS_COMPILE = arm-linux-
endif
```

After compiling, module driver (rt39070sta.ko) will be output. To use this driver, RT2870STA.dat file will also need to copy to root file system etc/Wireless/RT2870STA directory.

```
$ ls
chips                       Makefile          rt3070sta.ko
common                      os                sta
include                     README_STA_usb    sta_ate_iwpriv_usage.txt
iwpriv_usage.txt            RT2870STACard.dat tools
LICENSE_ralink-firmware.txt RT2870STA.dat
```

The usage of this driver is described as following,

1. Load driver module by using insmod command.

```
$ insmod rt3070.ko
```

2. Enable wireless interface

```
$ ifconfig ra0 up
```

3. Connect to wireless AP by using wireless utility included in BSP.
4. Use WEP method to connect

```
$ iwconfig ra0 essid "name of AP"
$ iwconfig ra0 key open
$ iwconfig ra0 key “secret key”
```

5. Use WPA-PSK method to connect

```
$ iwpriv ra0 set NetworkType=Infra
$ iwpriv ra0 set AuthMode=WPAPSK
$ iwpriv ra0 set EncrypType=TKIP
$ iwpriv ra0 set WPAPSK=”secret key”
$ iwpriv ra0 set SSID=“name of AP”
```

6. Use WPA2-PSK method to connect

```
$ iwpriv ra0 set NetworkType=Infra
$ iwpriv ra0 set AuthMode=WPA2PSK
$ iwpriv ra0 set EncrypType=AES
$ iwpriv ra0 set WPAPSK=”secret key”
$ iwpriv ra0 set SSID=“name of AP”
```

After successful connection with wireless AP, user can set static IP or use following command to get IP from DHCP server.

```
$ udhcpc –i ra0
```

- Realtek RTL8188 802.11 WiFi

To support RTL8188 USB WiFi module, user needs to enable wireless network function, USB host, loadable module support and the following function.

```
 [*] Networking support --->
-*- Wireless --->
     <*>  Nuvoton external WiFi driver support
```

The usage of this driver is described as following,

1. Load driver module by using insmod command.

```
$ insmod rtl8188eu.ko
```

2. Enable wireless interface

```
$ ifconfig lo up
$ ifconfig wlan0 up
```

3. Use wpa_supplicant utility to connect with wireless AP

```
$ ./wpa_supplicant –Dwext –i wlan0 –c <config file> -B
```

Wpa_supplicant needs configuration file, the following are examples of configuration file for it.

```
network={
     ssid="TESTTEST"
     proto=WPA
     key_mgmt=WPA-PSK
     pairwise=CCMP
     psk="ZZZZZZZZ"
}
```

NOTE: Nuvoton cannot provide RTL8188 driver source code.

- RS232, RS485, IrDA

NUC970 series support 11 serial ports which can be configured individually. Please follow the instruction below to enable serial port function.
User can enable or disable each port on configuration page. Most of the ports have various GPIO pins can be selected except UART0, UART3 and UART5.
The UART0 is kept for console and user doesn't need to configure it.

```
Device Drivers  --->
     Character devices  --->
          Serial drivers  --->
               [*]  NUC970 serial support
               [*]    NUC970 UART1 support
                        NUC970 UART1 pin selection (Tx:PE2, Rx:PE3)  --->
               [*]    NUC970 UART2 support
                        NUC970 UART2 pin selection (Tx:PF11, Rx:PF12)  --->
               [*]    NUC970 UART3 support
               [*]    NUC970 UART4 support
                        NUC970 UART4 pin selection (Tx:PC10, Rx:PC11)  --->
               [*]    NUC970 UART5 support
               [*]    NUC970 UART6 support
                        NUC970 UART6 pin selection (Tx:PB2, Rx:PB3)  --->
               [*]    NUC970 UART7 support
                        NUC970 UART7 pin selection (Tx:PG4, Rx:PG5)  --->
               [*]    NUC970 UART8 support
                        NUC970 UART8 pin selection (Tx:PE10, Rx:PE11)  --->
               [*]    NUC970 UART9 suppor
                        NUC970 UART9 pin selection (Tx:PH2, Rx:PH3)  --->
               [*]    NUC970 UART10 support
                        NUC970 UART10 pin selection (Tx:PB10, Rx:PB11)  ---
>
               [*]     Console on NUC970 serial port
```

If serial port is configured as IrDA function, user needs to enable serial port function and IrDA

function additionally like the following items.

```
[*] Networking support  --->
          <*>   IrDA (infrared) subsystem support  --->
               Infrared-port device drivers  --->
                    <*> NUC970 SIR on UART
```

- SD Card

To enable SD interface, user needs to enable the following functions. NUC970 series support two SD card interface and BSP only supports one of them now (can't operate both of them). If SD1 is used, user needs to select which GPIO pins to be used. The GPIO pins can be Port E, Port H or Port I.

```
Device Drivers  --->
     <*>MMC/SD/SDIO card support  --->
          <*>   MMC block device driver
          <*>     Use bounce buffer for simple hosts
          <*>   Nuvoton NUC970 SD Card support
               NUC970 SD port selection (SD1)  --->
               NUC970 SD1 pin selection (Port E)  --->
```

After system booting, if any card is detected the device mmcblk0 will be create in /dev directory. If more than one partition are created in the card, the device will be created sequentially, like mmcblk0, mmcblk1 and so on

- SPI

NUC970 series support two SPI interfaces. They can be enabled individually or not. The following is description for configuring two SPI interfaces.

```
Device Drivers  --->
     [*] SPI support  --->
          <*>   Nuvoton NUC970 Series SPI Port 0
          NUC970 SPI0 pin selection (Normal mode)  --->
          <*>   Nuvoton NUC970 Series SPI Port 1
          NUC970 SPI1 pin selection (Port B – Normal mode)  --->
```

There are normal (4-pin) or quad (6-pin) mode can be selected for the SPI0. Normal mode quad mode in SPI1 can be selected at GPIO port B or port I.
If SPI flash device is also used, user needs to enable MTD function like the following items.

```
Device Drivers  --->
     <*> Memory Technology Device (MTD) support  --->
          <*>   Caching block device access to MTD devices
          Self-contained MTD device drivers  --->
               <*> Support most SPI Flash chips (AT26DF, M25P, W25X, ...)
```

User also needs to enable JFFS2 file system functions in order to use SPI flash device correctly. The configuration of JFFS2 is descripted in the file system section. User can refer to it for more detail.

If user wants to use new SPI flash device which is not included in BSP, it's necessary to modify id table in driver and it can be identified be kernel correctly.
Please modify the m25p_ids structure in drivers/mtd/devices/m25p80.c file.

```
static const struct spi_device_id m25p_ids[] = {
/* Atmel -- some are (confusingly) marketed as "DataFlash" */
{ "at25fs010",  INFO(0x1f6601, 0, 32 * 1024,   4, SECT_4K) },
{ "at25fs040",  INFO(0x1f6604, 0, 64 * 1024,   8, SECT_4K) },
...
{ "en25qh16", INFO(0x1c7015, 0, 64 * 1024, 32, 0) },
...
{ "cat25128", CAT25_INFO(2048, 8, 64, 2) },
     { },
};
```

And nuc970_spi_flash_data structure also needs to be modified for the same purpose.
The string (name) at type argument must be the same with the first argument of m25p_ids structure otherwise system can't recognize it correctly.

```
static struct flash_platform_data nuc970_spi_flash_data = {
     .name = "m25p80",
     .parts =  nuc970_spi_flash_partitions,
     .nr_parts = ARRAY_SIZE(nuc970_spi_flash_partitions),
     .type = "en25qh16",
};
```

If user wants to modify partition number of SPI flash, nuc970_spi_flash_partitions structure in arch/arm/mach-nuc970/dev.c file also needs to be modified.

```
static struct mtd_partition nuc970_spi_flash_partitions[] = {
     {
          .name = "SPI flash",
          .size = 0x0200000,
          offset = 0,
     },
};
```

User can use mkfs.jffs2 utility to generate jffs2 image.

```
mkfs.jffs2 -e 0x10000 -r jffs2directory  -o image.jffs2
```

Note that the "-e" argument (erase block size) must be the same with flash device, otherwise kernel will mount fail.
After generating image, use NuWriter utility to burn the image to address 0x200000 of flash.

- USB host

To enable USB Host function, please check "USB support" in "Device Drivers" menu. NUC970 USB Host equips with EHCI (USB 2.0) and OHCI (USB1.1) Host controllers. All of the following

items must be checked to enable both Host controllers.

```
Device Drivers  --->
     [*] USB support  --->
          <*>   Support for Host-side USB
          <*>     EHCI HCD (USB 2.0) support
          <*>     NUC970 EHCI (USB 2.0) support
          NUC970 USB Host port power pin select (No USBH_PPWRx and
USBH_OVRCUR) --->
          <*>     OHCI HCD support
          [*]     NUC970 OHCI (USB 1.1) support
```

According to target NUC970 chip's pin configuration, select the corresponding multi-function pin setting.

```
[ ] PE.14 and PE.15 for USBH_PPWR0/1, PH.1 for USBH_OVRCUR
[ ] PF.10, PH.1 for USBH_OVRCUR
[ ] No USBH_PPWRx, PH.1 for USBH_OVRCUR
[X] No USBH_PPWRx and USBH_OVRCUR
```

If target board's USB port power is controlled by a Power Switch Controller, NUC970 must have USBH_PPWRx and USBH_OVRCUR pins to communicate with it. Depend on the target NUC970 chip's pin configuration, USB Host port0 and port1 power can be controlled by PE.14 and PE.15 respectively; or both be controlled by PF.10. PH.1 is dedicated assigned to USBH_OVRCUR for over-current detection.
If VBUS of USB Host ports are connected to +5V directly, USBH_PPWRx pins are not required any more. In this condition, USBH_PPWRx can be configured as GPIO pins and user must select "No USBH_PPWRx" items from this menu.
User can also select the last item "No USBH_PPWRx and USBH_OVRCUR" to release USBH_PPWRx and USBH_OVRCUR pins. In this condition, user can obtain 2 or 3 free GPIO pins, but take the risk of being unaware of over-current dangerous.

- USB mass storage class device support

Besides selecting NUC970 USB Host controller driver, user may have to select supporting device classes. For example, if user want to support mass storage device, it's necessary to enable "SCSI device support" first. After enabled SCSI device support, "USB Mass Storage Support" option will be present in "USB support" menu. Select it to enable Mass Storage Device supporting.

```
Device Drivers  --->
     SCSI device support  --->
          <*>   SCSI device support
               <*> legacy / proc/scsi/ support
               <*> SCSI disk support
               <*> SCSI media changer support
               [*] Asynchronous SCSI scanning
               [*] SCSI low-level drivers
     [*] USB support  --->
          <*>   USB Mass Storage Support
```

- USB host and HID device

To support HID class devices, such as USB mouse and USB keyboard, except to enable USB Host function, user must also enable “HID bus support” and “Input device support”. As the following:

```
Device Drivers --->
     HID bus support  --->
          <*>   User-space I/O driver support for HID subsystem
          <*>   Generic HID driver
          USB HID support  --->
               <*> USB HID transport layer
     Input device support  --->
          <*>   Mouse interface
          [*]     Provide legacy /dev/psaux device
          <*>   Event interface
          [*]   Keyboards  --->
               <*>   AT keyboard
          [*]   Mice  --->
               <*>   PS/2 mouse
```

- USB Device

```
Device Drivers --->
     [*] USB support --->
          <*> USB Gadget Support --->
               USB Peripheral Controller --->
                    <*> NUC970 USB Device Controller
               <M> USB Gadget Driver
               <M> Mass Storage Gadget
```

After compiling kernel, three driver module files will be outputted. (fs/configfs/configfs.ko, drivers/usb/gadget/libcomposite.ko and drivers/usb/gadget/g_mass_storage.ko)
User needs to copy those file to rootfs or somewhere they can be accessed by system. (Like USB mass storage device)
The following is an example by using USB mass storage gadget function.

```
$ insmod configfs.ko
$ insmod libcomposite.ko
$ insmod g_mass_storage.ko file=/dev/mmcblk0p1 stall=0 removable=1
```

- Video Capture

To support video capture function, user needs to enable “Cameras/video grabbers support” item first and enable “NUC970 Video-in support” in “Encoders, decoders, sensors and other helper chips” item. Finally, user can select model of video capture device. BSP supports OV7725, OV5640, NT99050 and NT99141 drivers now.

```
Device Drivers  --->
     <*> I2C support  --->
```

```
          I2C Hardware Bus support  --->
               <*> GPIO-based bitbanging I2C
     [*] Multimedia support  --->
          [*]   Camera/video grabbers support
          [*]   Media Controller API
          [*]   V4L2 sub-device userspace API
          [*]   V4L platform devices  --->
               Encoders, decoders, sensors and other helper chips  --->
                    <*> Nuvoton NUC970 Video-In Support
                    (3)   Max frame buffer
                    (24000000) Video frequency
                    Nuvoton NUC970 Image Sensor Selection (NT99141)  --->
```

If I2C interface is used by video capture device to configure arguments, I2C function also needs to be enabled first. User can refer to I2C section to do that.
The V4L2 API is supported by video capture driver in BSP and user can refer to the example in BSP/applications/demos/cap directory.

- Watchdog Timer

To support watchdog timer function, please enable the following items.
Timeout period in default is 2.03 seconds and this time can be modified via ioctl() command function (WDIOC_SETTIMEOUT) by application program.
There are three different time cycles can be supported by watchdog driver. If command argument is smaller than 2 and the timeout period will be 0.53 second. If command argument is between 2 to 8 and timeout period will be 2.03 seconds. And if argument is larger than 8 and timeout period will be 8 seconds. There is an example in BSP/applications/demos/wdt directory for user to reference.

```
Device Drivers  --->
     [*] Watchdog Timer Support  --->
          <*>   Nuvoton NUC970 Watchdog Timer
```

- Window Watchdog Timer

Please enable the following items to support window watchdog timer function.

```
Device Drivers  --->
     [*] Watchdog Timer Support  --->
          <*>   Nuvoton NUC970 Window Watchdog Timer
```

There are two different between window watchdog timer and watchdog timer. First, the configuration of window watchdog timer cannot be modified after enabling its function. Second, window watchdog timer only can be reset in specific time slot, but watchdog timer can be reset at any time if timeout doesn't occur. In application, user needs to use WDIOC_GETTIMELEFT ioctl() argument to get the available time to reset. If return value is 0 and application can use WDIOC_KEEPALIVE argument to let system reset otherwise system will be reset right now.
An example code is in BSP/applications/demos/wwdt for reference.

- Keypad

```
Device Drivers  --->
```

```
     Input device support --->
          [*] Keyboards  --->
               <*>  NUC970 Matrix Keypad support
                    NUC970 matrix keypad pin selection (Keypad pins are 4x8
matrix PA pin)  --->
                         ( ) Keypad pins are 4x2 matrix PA pin
                          ( ) Keypad pins are 4x4 matrix PA pin
                          (X) Keypad pins are 4x8 matrix PA pin
                          ( ) Keypad pins are 4x2 matrix PH pin
                          ( ) Keypad pins are 4x4 matrix PH pin
                          ( ) Keypad pins are 4x8 matrix PH pin
```

The item “Keypad pins are 4x2 matrix PH pin” must be enabled when user uses the keypad on NUC970 develop board.
An example code is in BSP/applications/demos/keypad for reference.

- RTC

```
Device Drivers --->
     [*] Real Time Clock --->
          <*> NUC970 RTC driver
```

- CAN

NUC970 series support 2 CAN ports which can be configured individually. Please follow the instruction below to enable CAN port function.
User can enable or disable each port on configuration page. CAN0 port has various GPIO pins can be selected.

```
-*- Networking support  --->
     <*>   CAN bus subsystem support  --->
          --- CAN bus subsystem support
          <*>  CAN Gateway/Router (with netlink configuration)
               CAN Device Drivers  --->
                    <*> Platform CAN drivers with Netlink support
                    [*] CAN bit-timing calculation
                    <*> NUC970 CAN0/CAN1 devices  --->
                         --- NUC970 CAN0/CAN1 devices
                         [*]  NUC970 CAN0 support
                              NUC970 CAN0 pin selection (Tx:PB11, Rx:PB10)
--->
                                   (X) Tx:PB11, Rx:PB10
                                   ( ) Tx:PH3, Rx:PH2
                                   ( ) Tx:PI4, Rx:PI32
                         [*]  NUC970 CAN1 support
                              NUC970 CAN1 pin selection (Tx:PH15, Rx:PH14)
--->
                                   (X) Tx:PH15, Rx:PH14
```

An example code is in BSP/applications/demos/CAN for reference

- ADC Battery

Please enable ”NUC970 ADC battery driver” function in “Power supply class support” item to support ADC battery interface.

```
Device Drivers  --->
     [*] Power supply class support  --->
          <*> NUC970 ADC battery driver --->
```

User can check battery current status in ”sys/class/power_supply” directory after system boots up.
User can use “cat” command to read the current status like,
$ cat voltage_now, → read the current battery voltage
$ cat present. → read the battery capacity in percentage

```
# cd /sys/class/power_supply
# ls
NUC970 Battery(ADC)
# cd NUC970\ Battery\ (ADC\)
# ls
present             technology          uevent              voltage_now
subsystem           type                voltage_max_design
```

- ADC keypad/touch screen

Please enable the following items to support ADC keypad or touch screen function.

```
Device Drivers  --->
     [*] Input device support  --->
          <*> Event interface
          <*> Input NUC970 ADC  --->
               <*>Keypad support
               <*>Touchscreen support
```

When use keypad function, user can adjust return value of each button or ADC threshold value by modifying nuc970_keycode or nuc970_key_th structure in drivers/input/nuvoton/nuc970adc.c driver.
The following is a brief description and example to configure those two structures.
If user wants to have 8 keys via ADC interface and name of them will be “KEY_A”, “KEY_B”~ “KEY_H” in nuc970_keycode structure. Then try to get reasonable value of ADC range for each key in nuc970_key_th structure.

```
static int nuc970_keycode[] = {
    KEY_A,
    KEY_B,
    KEY_C,
    KEY_D,
    KEY_E,
    KEY_F,
    KEY_G,
```

```
    KEY_H,
};

static struct key_threshold nuc970_key_th[] = {
    {0x500,0x5ff},
    {0x600,0x6ff},
    {0x700,0xaff},
    {0xa00,0xb4f},
    {0xb50,0xbff},
    {0xc00,0xcff},
    {0xd00,0xd49},
    {0xd50,0xe00},
};
```

- Touch screen calibration by using tslib utility

When use touchscreen function, user can adjust return Z_TH threshold value by "#define Z_TH" in drivers/input/nuvoton/nuc970adc.c driver. Z_TH can be to avoid pendown detection wrong. TSLIB-1.1 source code is included in this BSP and can be found in applications/tslib-1.1 directory.

The usage of touch screen library tslib list as below.

1. Compile tslib-1.1
   - ./configure --prefix=$(pwd)/install --enable-static --enable-shared --host=arm-linux
   - make
   - make install
2. Copy all of files in "install" directory to rootfs directory.
3. Modify rootfs/etc/profile and add the following commands.

```
export TSLIB_CONFFILE=/etc/ts.conf
export TSLIB_PLUGINDIR=/lib/ts
export TSLIB_TSDEVICE=/dev/input/event0
export TSLIB_CALIBFILE="/etc/pointercal"
export TSLIB_CONSOLEDEVICE="none"
```

4. Modify rootfs/etc/ts.conf file.

```
# Uncomment if you wish to use the linux input layer event interface
module_raw input

Uncomment if you're using a Sharp Zaurus SL-5500/SL-5000d
# module_raw collie

# Uncomment if you're using a Sharp Zaurus SL-C700/C750/C760/C860
# module_raw corgi

# Uncomment if you're using a device with a UCB1200/1300/1400 TS interface
# module_raw ucb1x00
```

```
Uncomment if you're using an HP iPaq h3600 or similar
# module_raw h3600

# Uncomment if you're using a Hitachi Webpad
# module_raw mk712

# Uncomment if you're using a Hitachi Webpad
# module_raw mk712

# Uncomment if you're using an IBM Arctic II
# module_raw arctic2

module pthres pmin=1
module variance delta=30
module dejitter delta=100
module linear
```

5. Please follow the instruction displayed on screen by using ts_calibrate calibration program. After that, can use ts_test to do the test. If result is poor for the test, user can run the calibration program again.

```
# ts_calibrate
xres = 800, yres = 480
Took 26 samples...
Top left : X = 3505 Y =  353
Took 24 samples...
Top right : X = 3421 Y = 3740
Took 37 samples...
Bot right : X =  546 Y = 3736
Took 27 samples...
Bot left : X =  585 Y =  342
Took 30 samples...
Center : X = 1993 Y = 2041
-20.572632 -0.000537 0.206449
508.422333 -0.131128 -0.002377
Calibration constants: -1348248 -35 13529 33319966 -8593 -155 65536
```

6. The following content will be shown on screen after executing ts_test program.

```
# ts_test
```

,.,,.

- SCUART

There are two smart card interfaces built in NUC970 series. They have additional UART function to simulate as basic UART port when there is not enough UART to be used in system.
In this mode, the SC_CLK pin will be used to as transmit function and SC_DATA will be receive function. Those two pins can be enabled individually and there are some pin selections at SCUART0 interface.

```
Device Drivers --->
     Character devices --->
          Serial drivers --->
                [*] NUC970 Smartcard UART mode support
                [*] NUC970 SCUART0 support
                    NUC970 SCUART0 pin selection (Tx:PG11, Rx:PG12) --->
                [*] NUC970 SCUART1 support
```

The device node for the SCUART is /dev/ttySCU0 or /dev/ttySCU1. The basic operation of SCUART is the same with normal UART but have a lot of limitations, for example, there are only four levels of FIFO and can't support flow control function, can't support RS485 and IrDA transmission mode. It is better to use normal UART only if they are all occupied by system.

- Loopback device

A loop device is a pseudo-device that makes a file accessible as a block device. Before use, a loop device must be connected to an existing file in the file system.
Please enable the following items to support loopback device.

```
Device Drivers --->
     Block devices --->
          <*>   Loopback device support
```

The usage of loopback device lists as the following steps.
1. Create image file for mounting on loopback device.

```
$ dd if=/dev/zero bs=1M count=1 of=fat.img
```

2. Format image (take FATFS for example)

```
$ busybox mkfs.vfat fat.img
```

3. Mount image

```
$ mount -o loop fat.img /mnt/loop
```

5.3.4 File System

- FAT

FAT is common file system and can be seen usually on SD card or USB mass storage device. User can enable the following items to support it.

```
File systems  --->
     DOS/FAT/NT Filesystems  --->
          <*> MSDOS fs support
          <*> VFAT (Windows-95) fs support
          (437) Default codepage for FAT
          (iso8859-1) Default iocharset for FAT
```

Command for mounting the first partition on SD card is list as following.

```
$ mount –t vfat /dev/mmcblk0p1 /mnt
```

- JFFS2

JFFS2 is one of the file system used on NAND flash. Please enable the following items to support it.

```
File systems  --->
     [*] Miscellaneous filesystems  --->
          <*>   Journalling Flash File System v2 (JFFS2) support
          [*]   JFFS2 write-buffering support
```

- ROMFS

ROMFS is one of the file system used on root file system. Please enable the following items to support it.

```
File systems  --->
     [*] Miscellaneous filesystems  --->
          <*>   ROM file system support
               RomFS backing stores (Block device-backed ROM file system
support)  --->
```

- YAFFS2

YAFFS2 is one of the file system used on NAND flash. Before enabling the following items, user needs to enable MTD item “Caching block device access to MTD devices Device drivers” first.

```
File systems  --->
     [*] Miscellaneous filesystems  --->
          <*>   yaffs2 file system support
          <*>   Autoselect yaffs2 format
          <*>   Enable yaffs2 xattr support
```

Command for mounting YAFFS2 is list as following.

```
$ mount –t yaffs2 –o”inband-tags” /dev/mtdblock2 /flash
```

- exFAT

exFAT is a new generation file system created by Microsoft. It is more flexible about size of single file and total capacity of device.

```
File systems  --->
     DOS/FAT/NT Filesystems  --->
          <*> exFAT fs support
```

Command for mounting the first partition on SD card is list as following.

```
$ mount –t exfat /dev/mmcblk0p1 /mnt
```

- FUSE and NTFS

FUSE (Filesystem in Userspace) is a kind of file system that is implemented for user space. User can implement much kind of file systems by FUSE. The famous file system that is implement by FUSE are NTFS-3G or SSHFS and so on. The following is an example that implements Microsoft NTFS (NTFS-3G) by FUSE.
Please enable the following item to support FUSE function.

```
File systems  --->
     <*> FUSE (Filesystem in Userspace) support
```

NTFS-3G is an open source project developed and implemented by Tuxera. It is a driver which can read and write NTFS on Linux and source code can be downloaded from http://www.tuxera.com/community/ntfs-3g-download page. Please refer to user's manual of ntfs-3g to compile it. And mount it by the following command.

```
$ ./ntfs-3g /dev/mmcblk0p1 /mnt/mmc
```

- UBIFS

Please enable the following items to support it.

```
Device Drivers  --->
-*- Memory Technology Device (MTD) support  --->
<*>   Enable UBI - Unsorted block images  --->
File systems  --->
     [*] Miscellaneous filesystems  --->
          <*>   UBIFS file system support
          [*]     Advanced compression options
          [*]       LZO compression support
```

5.3.5 **FIQ**

In order to make sure the real time of interrupt, user can use FIQ instead of IRQ. This section includes an example which describes how to use timer2 FIQ.
Please enable the following item to support FIQ in system.

```
Kernel Configuration
     System Type  --->
          [*] Nuvoton NUC970 FIQ support
```

Example

```
/*IRQ handler for the timer*/
void nuc970_timer2_interrupt(void) {
    // ... add some code here
    __raw_writel(0x04, REG_TMR_TISR); /* clear timer2 flag */
}

static uint8_t fiqStack[1024];
extern unsigned char fiq_handler, fiq_handler_end;
static struct fiq_handler timer2_fiq = {
     .name     = "timer2_fiq_handler"
};

void use_fiq(void) {
int ret;
struct pt_regs regs;

init_FIQ(0);
ret = claim_fiq(&timer2_fiq);
if (ret)
return;
set_fiq_handler(&fiq_handler, &fiq_handler_end - &fiq_handler);
// set some registers use in FIQ handler
regs.ARM_r8 = (long)nuc970_timer2_interrupt;
regs.ARM_r10 = (long)REG_AIC_IPER;
regs.ARM_sp = (long)fiqStack + sizeof(fiqStack) - 4;
set_fiq_regs(&regs);
/* Enable the FIQ */
__raw_writel(__raw_readl(REG_AIC_SCR7) & ~0x00070000, REG_AIC_SCR7);
enable_fiq(IRQ_TMR2);
}
```

Note that, the regs.ARM_r8 must be the address of fiq handler function and regs.ARM_r10 must be the address of REG_AIC_IPER register.
User needs to configure timer2 if necessary additionally.

5.4 Linux Kernel Compilation

After finish the configuration of kernel, type “make” command to compile kernel in linux-3.10.x directory. If no error happens, the kernel image file and kernel zip file will be output to upper image directory.

```
$ make
......
  Kernel: arch/arm/boot/Image is ready
cp arch/arm/boot/Image   ../image/970image
```

```
zip ../image/970image.zip ../image/970image
updating: ../image/970image (deflated 31%)
  GZIP    arch/arm/boot/compressed/piggy.gzip
  CC      arch/arm/boot/compressed/misc.o
  AS      arch/arm/boot/compressed/piggy.gzip.o
  LD      arch/arm/boot/compressed/vmlinux
  OBJCOPY arch/arm/boot/zImage
  Kernel: arch/arm/boot/zImage is ready
$ ls ../image/
970image  970image.zip
```

6 Linux user applications

6.1 Sample applications

There are some sample applications in the applications/ directory. Content of each directory listed in the following table

Directory	Description
alsa-utils-1.0.23	ALSA command line utilities.* Cross compilation command as below: $ ./configure –host=arm-linux –disable-nls --disable-xmlto CPPFLAGS=-I/usr/local/arm_linux_4.3/usr/include/ncurses LDFLAGS=-L/usr/local/arm_linux_4.3/usr/lib Sample mixer setting for playback: ● $./amixer set PCM 85% ● $./amixer set Headphone 90% Sample mixer setting for recording: When source is Mic: ● $./amixer set “Mic Bias” on ● $./amixer set “Input PGA” 100% ● $./axmier set ADC 90% When source is Line In: ● $./axmier set "Right Input Mixer R2" on ● $./axmier set "Left Input Mixer L2" on ● $./axmier set "L2/R2 Boost" 100% ● $./axmier set ADC 90% Playback command: ● $./aplay <file name> To playback the sample sound file in BSP, please use following command: $ cd usr $./aplay –c 2 –f S16_LE alsa/8k2ch.pcm Recording command: ● $./arecord –d 10 –f S16_LE –c2 –r8000 –t wav –D plughw:0,0 <file name>

	Command to record and play simultaneously: ● $./arecord -f S16_LE -r 8000 -c 2 -D plughw:0,0 | ./aplay
benchmark/netperf-2.6.0	Network performance benchmarking tool. Cross compilation command below: $./configure –host=arm-linux
busybox-1.22.1/	Busybox source code. Cross compilation command below: 1. $ make menuconfig 2. Select applets to be build 3. $ make
demos/2d	2D graphic engine sample application (Need to enable user's space memory management in kernel, please refer to **Error! Reference source not ound.** section)
demos/alsa_audio	Audio sample application. *
demos/cap	Video capture sample application. *
demos/can	CAN bus sample application
demos/crypto	Encryption/decryption sample application. *
demos/etimer	Enhanced timer sample application. *
demos/gpio	GPIO sample application. *
demos/irda	IrDA sample application. *
demos/lcm/	LCD sample application. *
demos/thread	Thread sample applications. *
demos/rtc	RTC sample application. *
demos/uart	UART sample application. *
demos/wdt	Watchdog timer sample application. *
demos/wwdt	Window watchdog timer sample application. *
DPO_RT3070_LinuxSTA_V2.3.0.2_20100412	RT3070 USB WiFi dongle open source driver.
wireless_tools.29	WiFi configuration tools. Including iwconfig, iwlist, iwpriv…
tslib-1.1	Touch screenlibrary, including calibration and test utilities. *
yaffs2utils.tar.gz	yaffs2 command tool. Simply type make to compile $ make

lzo-2.09.tar.gz	Compress/decompress utility.
	Cross compilation command below:
	$ cd lizo-2.09
	$ ./configure --host=arm-linux --prefix=$PWD/../install
	$ make
	
	$ make install

libuuid-1.0.3.tar.gz	Utility to create UUID. Cross compilation command below: $ cd libuuid-1.0.3 $ ./configure --host=arm-linux --prefix=$PWD/../install

	$ make
	
	$ make install

mtd-utils.tar.gz	mtd-utils source code. Required to use lzo-2.09.tar.gz and libuuid-1.0.3.tar.gz
	Cross compilation command below:
	$ cd mtd-utils
	$ export CROSS=arm-linux-
	$ export WITHOUT_XATTR=1
	$ export DESTDR=$PWD/../install
	$ export LZOCPPFLAGS=-I/home/install/include
	$ export LZOLDFLAGS=-L/home/install/lib
	$ make
	
	$ make install

qt-everywhere-opensource-src-4.8.5	QT gui source code 1. Please make following modifications to support touch screen using tslib: qt-everywhere-opensource-src-4.8.5/mkspecs/qws/linux-nuc970-g++/qmake.conf Set QMAKE_INCDIR and QMAKE_LIBDIR as below: QMAKE_INCDIR = path to /tslib-1.1/install/include QMAKE_LIBDIR = path to /tslib-1.1/install/lib 2.Set environment variable: $ export MY_CC_QT4_PREFIX=/usr/local/Trolltech/QtEmbedded-4.8.5 3.Configure： ./configure \ -prefix ${MY_CC_QT4_PREFIX} \ -release \ -opensource \ -static \ -qconfig dist \ -no-exceptions \ -no-accessibility \ -no-stl \ -no-qt3support \ -no-xmlpatterns \ -no-multimedia \ -no-audio-backend \ -no-phonon \ -no-phonon-backend \ -no-svg \ -no-webkit \ -no-javascript-jit \ -no-script \ -no-scripttools \ -no-declarative \ -no-declarative-debug \ -qt-zlib \ -qt-freetype \ -no-gif \ -qt-libpng \

	-no-libmng \
	-no-libtiff \
	-qt-libjpeg \
	-no-openssl \
	-nomake tools \
	-nomake demos \
	-nomake examples \
	-nomake docs \
	-nomake translations \
	-no-nis \
	-no-cups \
	-no-iconv \
	-no-pch \
	-no-dbus \
	-embedded arm \
	-platform qws/linux-x86-g++ \
	-xplatform qws/linux-nuc970-g++ \
	-no-gtkstyle \
	-no-nas-sound \
	-no-opengl \
	-no-openvg \
	-no-sm \
	-no-xshape \
	-no-xvideo \
	-no-xsync \
	-no-xinerama \
	-no-xcursor \
	-no-xfixes \
	-no-xrandr \
	-no-xrender \
	-no-mitshm \
	-no-fontconfig \
	-no-xinput \
	-no-xkb \
	-no-glib \
	-qt-gfx-linuxfb \
	-qt-mouse-tslib \
	-qt-kbd-linuxinput
	
	4. Cross compilation command:
	$ make
	
	5. Compile example
	$ cd /path/to/qt-everywhere-opensource-src-4.8.5/examples/dialogs/trivialwizard
	$ ../../../bin/qmake
	$ make
	
	6. Run QT
	a.Copy trivialwizard where system can access
	b.Set tslib environment variable
	$ export QWS_MOUSE_PROTO=Tslib:/dev/input/event0
	c.Execute
	$ trivialzardwizard –qws

Minigui-3.0.12	Cross compilation command below:
	
	1. Build libminigui-gpl-3.0.12
	
	$ ./configure --prefix=$PWD/../build CC=arm-linux-gcc --host=arm-linux --build=i386-linux --with-osname=linux --with-targetname=fbcon --disable-pcxvfb --enable-videonuc970 --enable-videofbcon --enable-autoial --disable-vbfsupport --disable-screensaver
	$ make
	$ make install
	
	2. Build minigui-res-be-3.0.12
	
	$ ./configure --prefix=$PWD/../build
	$ make
	$ make install
	
	3. Build mg-samples-3.0.12
	
	$ export PKG_CONFIG_PATH="$PWD/../build/lib/pkgconfig"
	$ ./configure --prefix=$PWD/../build CC=arm-linux-gcc --host=arm-linux --build=i386-linux CFLAGS=-I$PWD/../build/include
	$ make
	$ make install
	
	4. Copy all the files and directories in build directory to rootfs
	5. Modify /etc/MiniGUI.cfg and put together with executable file of minigui in the same directory.
	
	[system]
	# GAL engine and default options
	gal_engine=nuc970
	defaultmode=800x480-32bpp
	# IAL engine
	ial_engine=fbcon
	mdev=/dev/input/mice
	mtype=IMPS2
	[nuc970]
	defaultmode=800x480-32bpp
	
	[fbcon]
	defaultmode=800x480-32bpp

	[cursorinfo]
	# Edit following line to specify cursor files path
	cursorpath=/share/minigui/res/cursor/
	
	[resinfo]
	respath=/share/minigui/res/
	
	6. Must enable ps2 mouse function in kernel. (Please refer to **Error! Reference source not found.** ection)

*. The execution result will be incorrect if the driver is not enabled in kernel configuration and/or jumper/ switch setting on EV board setting is inconsistent with kernel configuration.

6.2 Cross compilation

Sometimes a project requires porting an application to ARM platform. Many open source projects already supports cross compiling. Simply follow these projects' document to configure for cross compiling to build executable files or libraries for ARM platform.
If the application's Makefile doesn't support cross compilation options, the modification of Makefile is necessary. The Makefile used for cross compiling could be alike with the original one, only part of it needs to be modified

- The prefix of tool chain must be set. For example, the original Makefile use gcc for compiling, the new Makefile use arm-linux-gcc for cross compiling. Other tools for example, as and ld need to change to arm-linux-as and arm-linux-ld respectively.
- The path of library and include files need to be set. The cross compiler doesn't use the glibc or other library using in x86 system. Rather it links with uClibc which consumes less system resource.

Here is a simple Makefile for your reference.

```
.SUFFIXES : .x .o .c .s

ROOT = /usr/local/arm_linux_4.3/usr
LIB =$(ROOT)/lib
LIB1 = $(ROOT)/lib/gcc/arm-linux-uclibcgnueabi/4.3.4
INC :=$(ROOT)/include
INC1:=$(ROOT)/ lib/gcc/arm-linux-uclibcgnueabi/4.3.4/include

CC=arm-linux-gcc -O2 -I$(INC) -I$(INCSYS)
WEC_LDFLAGS=-L$(LIB) -L$(LIB1)
STRIP=arm-linux-strip

TARGET = hello

SRCS := hello.c

LIBS=  -lc -lgcc -lc
```

```
all:
	$(CC) $(WEC_LDFLAGS) $(SRCS) -o $(TARGET) $(LIBS)
	$(STRIP) $(TARGET)

clean:
	rm -f *.o
	rm -f $(TARGET)
	rm -f *.gdb
```

7 Revision Hisotry

Verison	Date	Description
0.08	Aug. 15, 2014	Initial release
0.09	Oct. 8, 2014	1. Editorial change 2. Added driver configuration for USB Device, keypad, RTC, ADC/tslib
0.10	Oct. 24, 2014	1. Added U-Boot SPL configuration description 2. Updated Nu-Writer usage guide
0.11	Oct. 28, 2014	1. Added a section describes hot to program U-Boot using Nu-Writer. 2. Added Linux kernel SCUART setting descriotion
0.12	Nov. 26, 2014	1. Added U-Boot MMC configuration description 2. Added U-Boot LCD configuration description 3. Added Loop back device configuration description 4. Added exFAT/NTFS configuration description 5. Added U-Boot eMMC configuration description 6. Added QT application 7. Modified tslib configuration description
0.13	Mar. 6, 2015	1. Added U-Boot bootm command description 2. Added U-Boot UBI command description 3. Modified U-Boot mkimage tool description 4. Added U-Boot GPIO configuration description 5. Added U-Boot Watchdog timer configuration description
0.14	Apr. 8, 2015	1. Modified alsa-utils-1.0.23 usage description 2. Modified LCD device driver description 3. Modified U-Boot SPI configuration description 4. Modified U-Boot NAND configuration description 5. Modified U-Boot CONFIG_SYS_NAND_U_BOOT_OFFS configuration description 6. Modified U-Boot NAND AES secure boot example 7. Modified U-Boot SPI command description 8. Modified U-Boot NAND command description 9. Modified U-Boot env command script sample 10. Added U-Boot CONFIG_BOOTP_SERVERIP configuration description 11. Added U-Boot CONFIG_ENV_RANGE configuration description
0.15	May. 6, 2015	1. Modified kernel default setting description 2. Modified NuWriter NAND/eMMC/SPI Read mode configuration description 3. Added lzo-2.09/libuuid-1.03/mtd-utils/yaffs2utils description in 4. Added UBIFS filesystem configuration description 5. Added section 3.8.4 for create different FS Image 6. Added Linux kernel FIQ usage description

0.16	May. 29, 2015	1. Added YAFFS2 inband-tags Image usage description 2. Added YAFFS2 command usage description 3. Added description for using YAFFS2 as root file system 4. Added description for using UBIFS as root file system 5. Added RTL8188 USB dongle usage description
0.17	Nov. 06, 2015	1. Added NFS boot description. 2. Added SPI flash boot description. 3. Added 2D driver configuration. 4. Added user's space memory management description. 5. Added color format selection description. 6. Added description for using aplay. 7. Added description for U-Boot go command. 8. Modify menu selection of USB host. 9. Added description for RTL8188. 10. Added CAN driver configuration. 11. Correct typo.

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.