NUC970 Linux BSP 使用手冊

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

Apr. 22, 2016 Rev. 1.00

內容

1 NUC970 Linux BSP 簡介

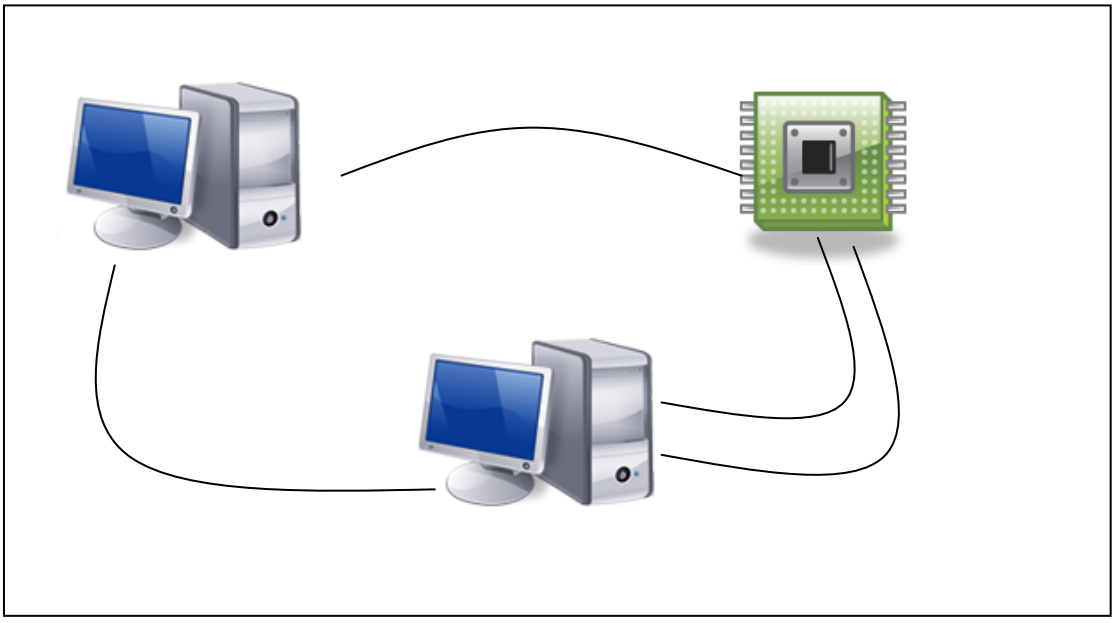
這包 BSP 支持了NUC970 系列芯片. 新唐科技的 NUC970 系列芯片是以 ARM926EJS 為核心的系统级單芯片. 包含了 16kB I-Cache 以及 16kB D-Cache 以及MMU 記憶體管理模塊. 最高支援到 300MHz 的頻率, 並且提供了豐富的外設接口周邊. 有USB 快速Host/Device, SDHC, 支援TFT LCD介面, 網路接口 和I2S audio介面, 有11 組UART…等. 並可以由 NAND flash, SPI Flash 開機.

這包 Linux BSP 包含了以下內容:

- Linux 3.10 內核源碼, 以及 NUC970 使用的驅動程式
- GCC 4.8.4 交叉編譯器, 支持 EABI.
- uClibc-0.9.33 庫文件
- Binutils-2.24 交叉開發工具
- 演示個接口功能的範例程式源碼, 以及一些開源軟件
- U-Boot 源碼, 以及 NUC970 使用的驅動程式
- Windows 端燒錄程序 Nu-Writer, 以及所需的驅動
- 說明文檔

1.1 開發環境連線

在Linux 環境下, 基本的系統訊息以及 shell 環境的溝通都是透過串口來達成. 不論是U-Boot 或是Linux 均使用 UART0 來做為訊息溝通的接口. 在 U-Boot 環境下, 也支援了網口 TFTP 的傳輸. 另外新唐也提供了基於 Windows 平台的 USB 介面燒寫工具. 以下是開發環境連線的示意圖. 若是使用虛擬機, 則只需要一台 PC 即可



1.2 開發板設置

NUC970 系列芯片支持不同的開機模式, 可從 SPI, NAND, eMMC 開機, 或是進入 USB ISP 模式. 這些設置是透過 PA[1:0] 的 jumper 控制. 另外, 因為複用腳位的關係, 開發版上會有些 jumper 須依不同系統需求來設置. 請參考開發版的文件來做系統相應的設置.

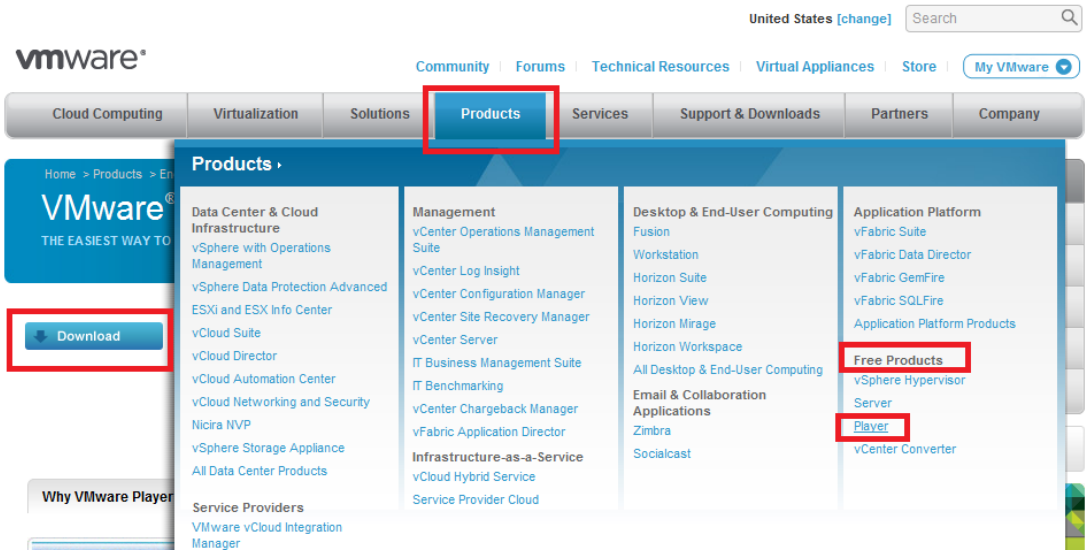
2 BSP 安裝

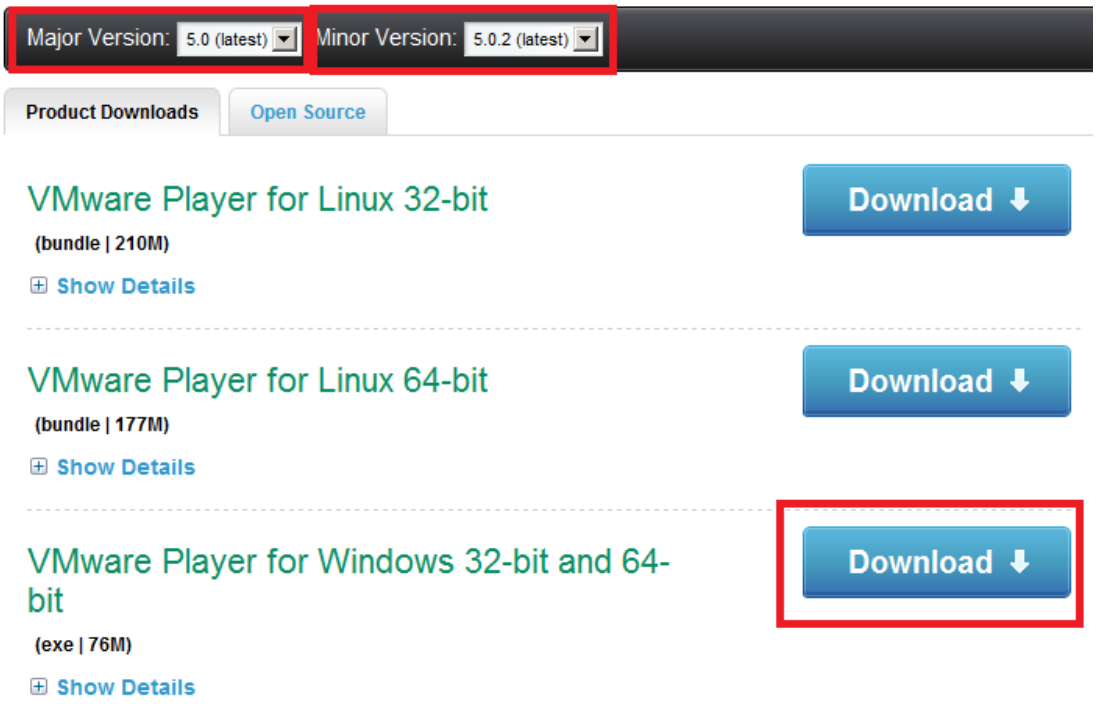
2.1 系統需求

NUC970 Linux BSP 提供了基於 Linux 作業系統的交叉編譯環境. 新唐有在不同的 x86 Linux 環境測試了本 BSP, 包含了 Ubuntu, CentOS, Debian…等. 因 Linux 發行版眾多, 系統設置會有些許差異, 有時使用這需更改系統設置, 使開發環境順利執行.
Linux 開發環境可選擇架設原生環境, 或是選擇架設於 Windows 作業系統中的虛擬機上. 本章節介紹了VMware 虛擬機安裝, CentOS Linux 開發環境安裝, 以及 BSP 的安裝步驟

2.2 VMware 虛擬機下載及安裝

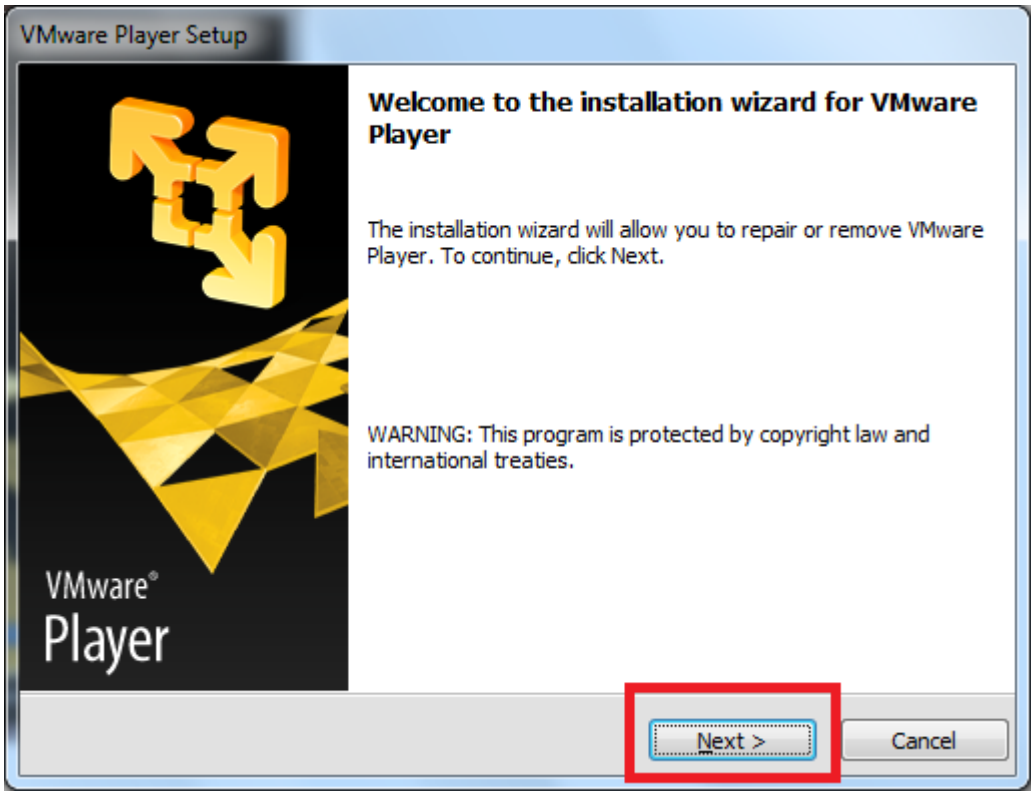
VMware 提供了免費的虛擬機 VMware player 5.0.2 供使用者下載. 從 VMware 官網 http://www.vmware.com/. 的頁面進入 “Products” → “Free Products” → “Player”, 然後按下 “Download” 鍵, 選擇 “5.0 (latest)” 作為 “Major Version” 以及 “5.0.2 (latest)” 作為 “Minor Version” 之後下載 “VMware Player for Windows 32-bit and 64-bit”. 請參考以下的截圖.



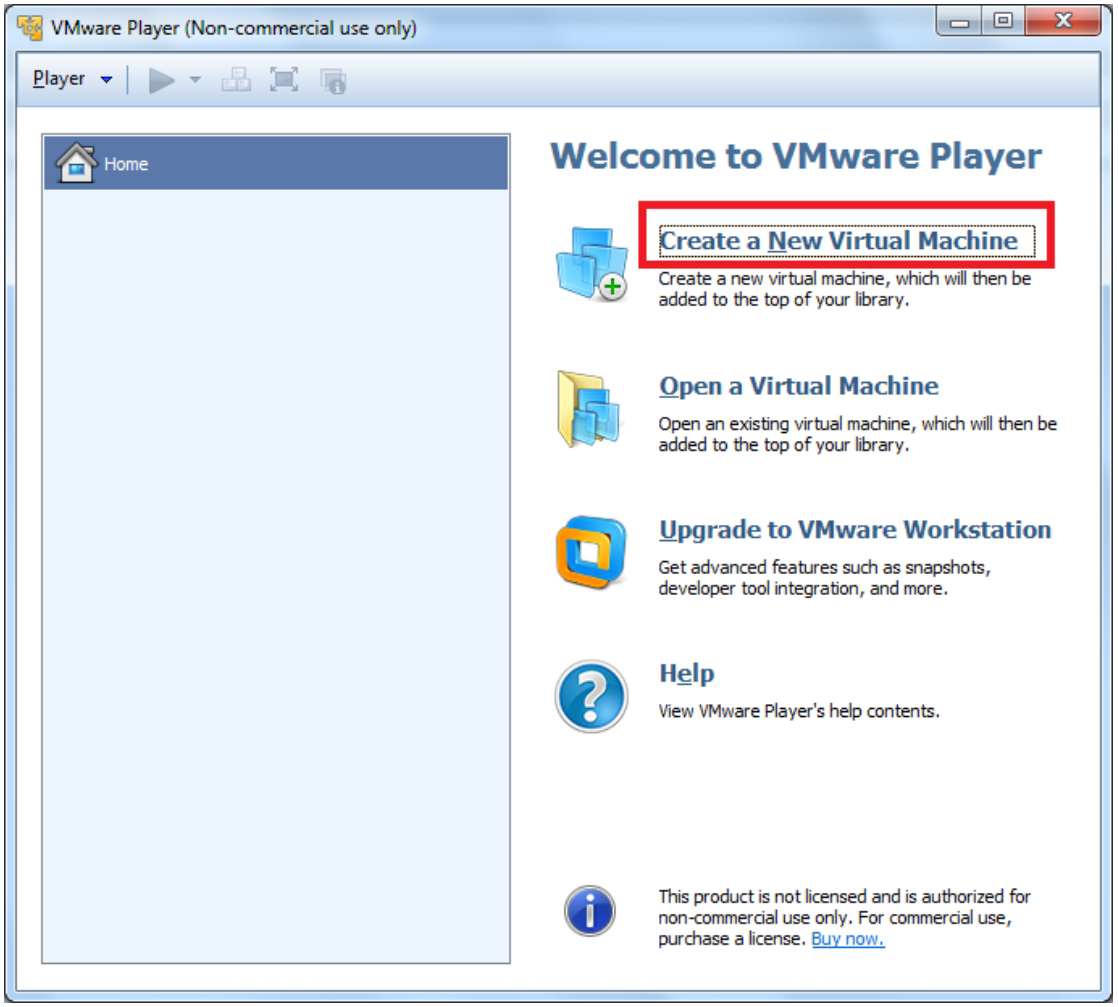


下載完成後, 請先雙擊下載的 VMware檔案.

然後按下 “Next” 繼續完成安裝步驟.

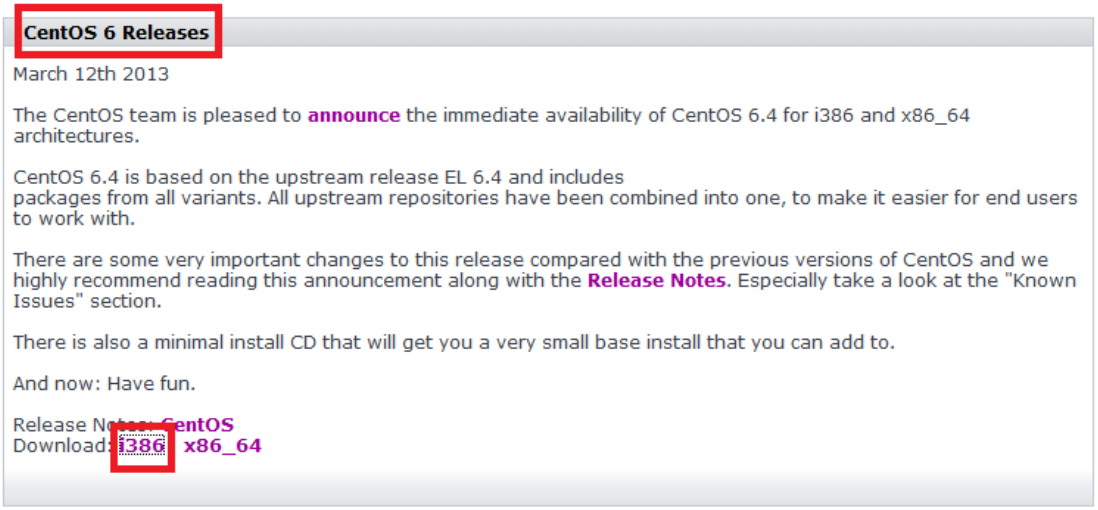


最後, 雙擊安裝完成的檔案來建立虛擬機.



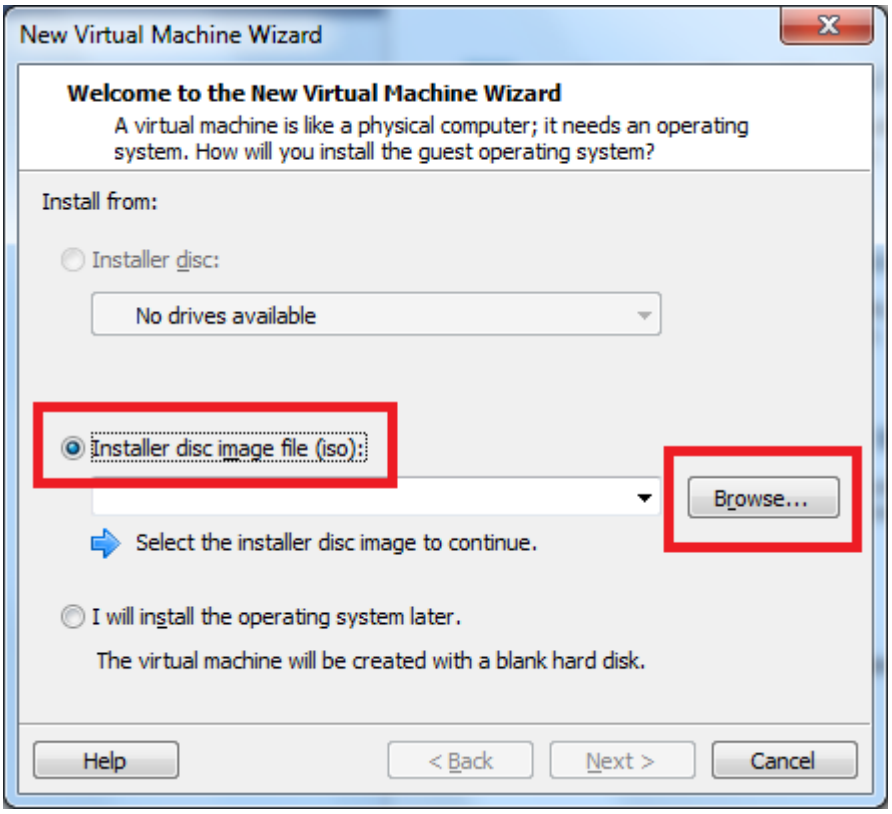
2.3 CentOS Linux 下載及安裝

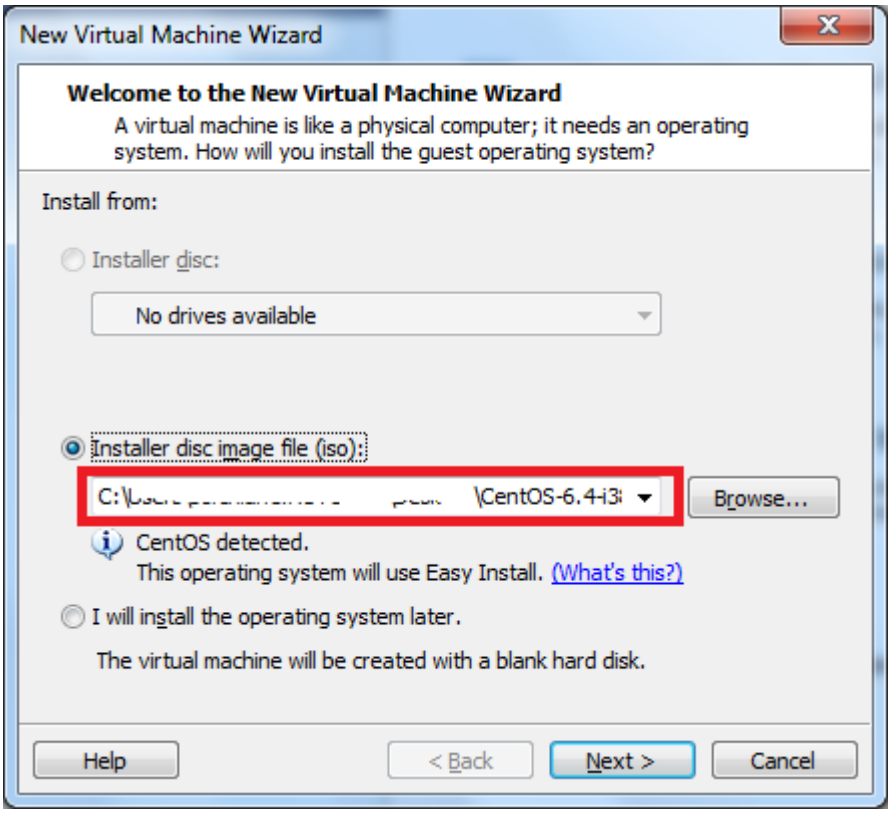
在此介紹了在 VMware 下的 CentOS 6.4 安裝步驟. 若是要安裝原生 PC 機, 步驟也類似. CentOS 6.4 請先連線至以下網址下載: http://www.centos.org/. 下載的網頁由 “CentOS 6 Releases” → “i386” 進入. 選擇 “CentOS-6.4-i386-bin-DVD 1.iso” 下載.



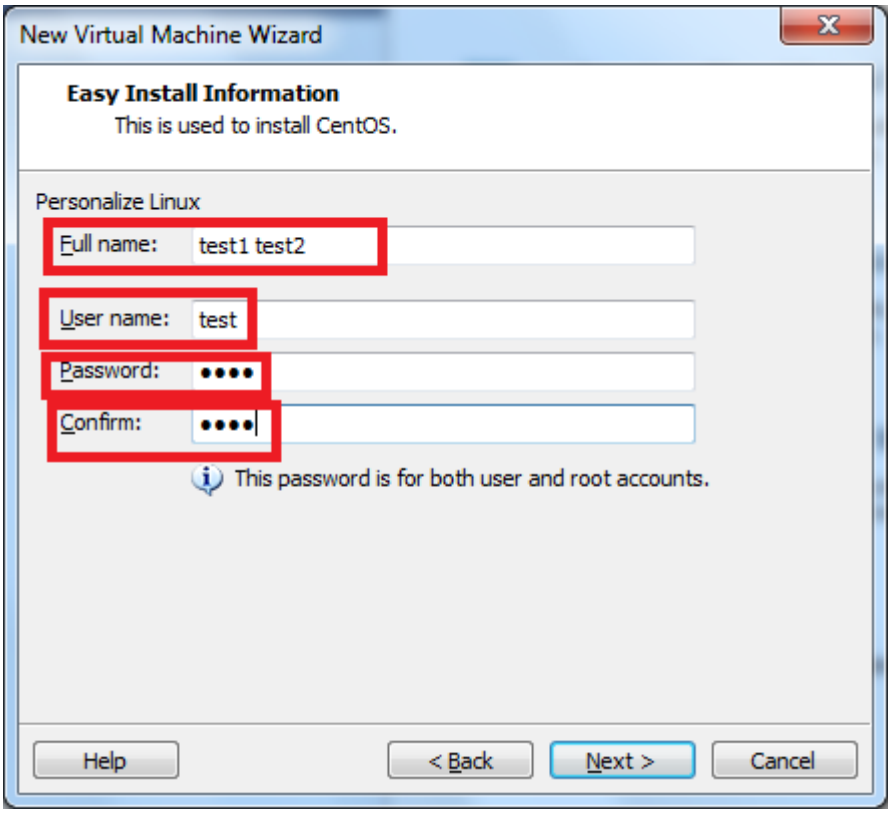
若是 VMware 已安裝完成, 點選 “Create a New Virtual Machine” 繼續安裝Linux 虛擬機, 或返回上一章節繼續完成 VMware 安裝.

首先, 點選 “Installer disc image file (iso):” and “Browse…” 並指定下載的 CentOS 6.4 iso 無安裝來源檔案.

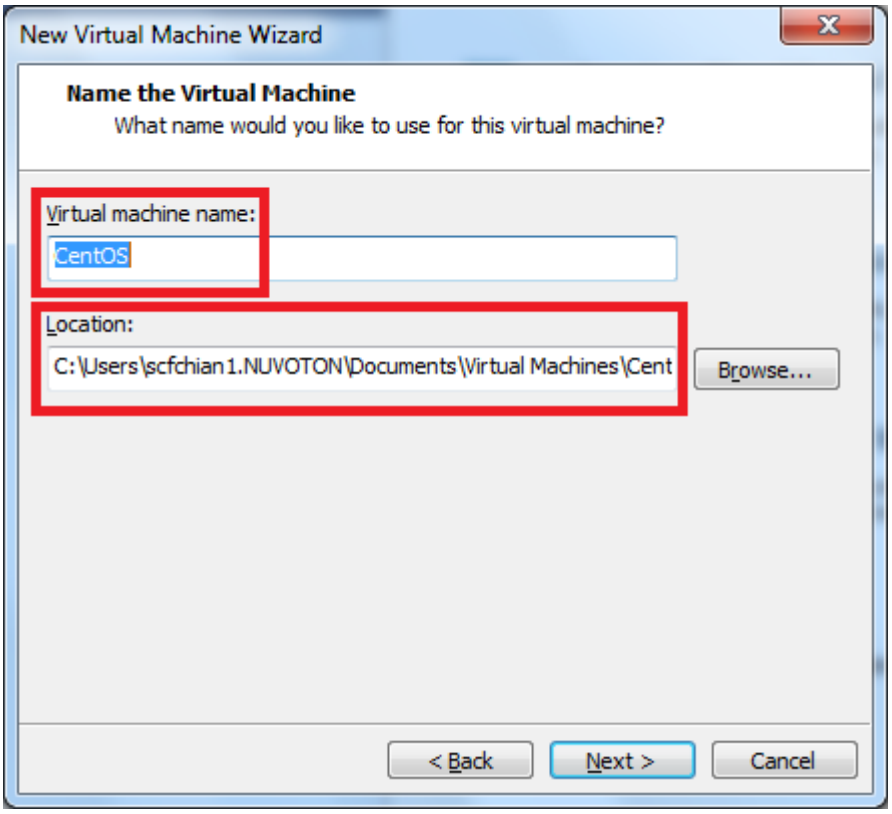

New Virtual Machine Wizard
Welcome to the New Virtual Machine Wizard
A virtual machine is like a physical computer; it needs an operating system. How will you install the guest operating system?
Install from:
Installer disc:
No drives available
Installer disc image file (iso):
Browse...
Select the installer disc image to continue.
I will install the operating system later.
The virtual machine will be created with a blank hard disk.
Help
< Back
Next >
Cancel



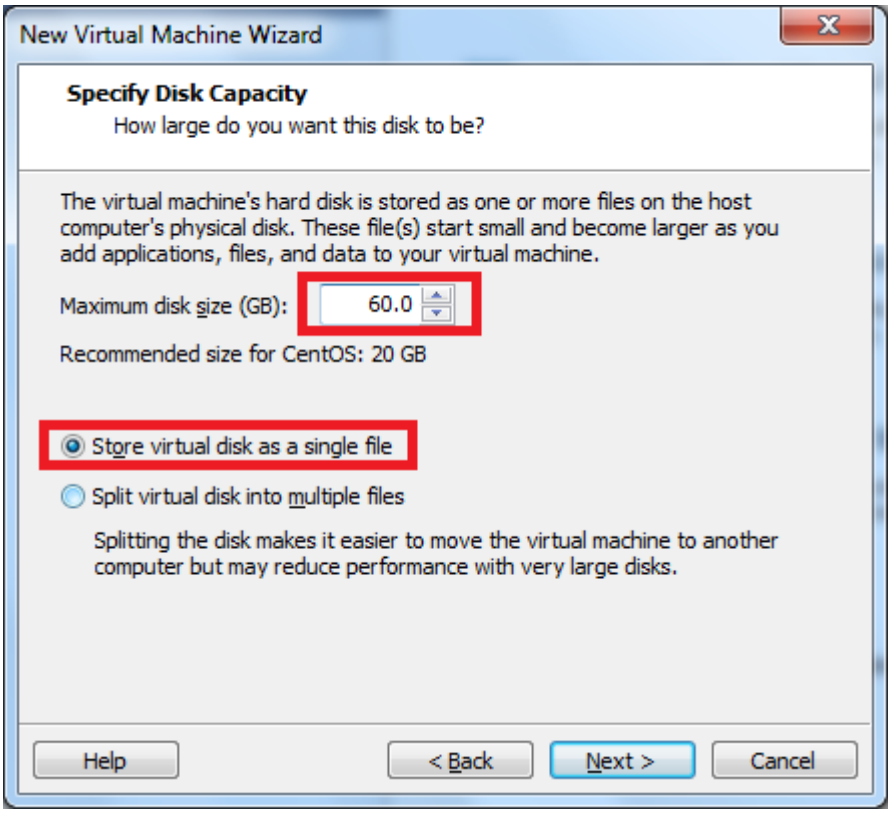
接著輸入 “Full name:”, “User name:”, “Password:”, and “Confirm:” 按下 Confirm 鍵. 請記得在此設置的用戶名稱及密碼, 它們將用來登入安裝的 CentOS 作業系統.



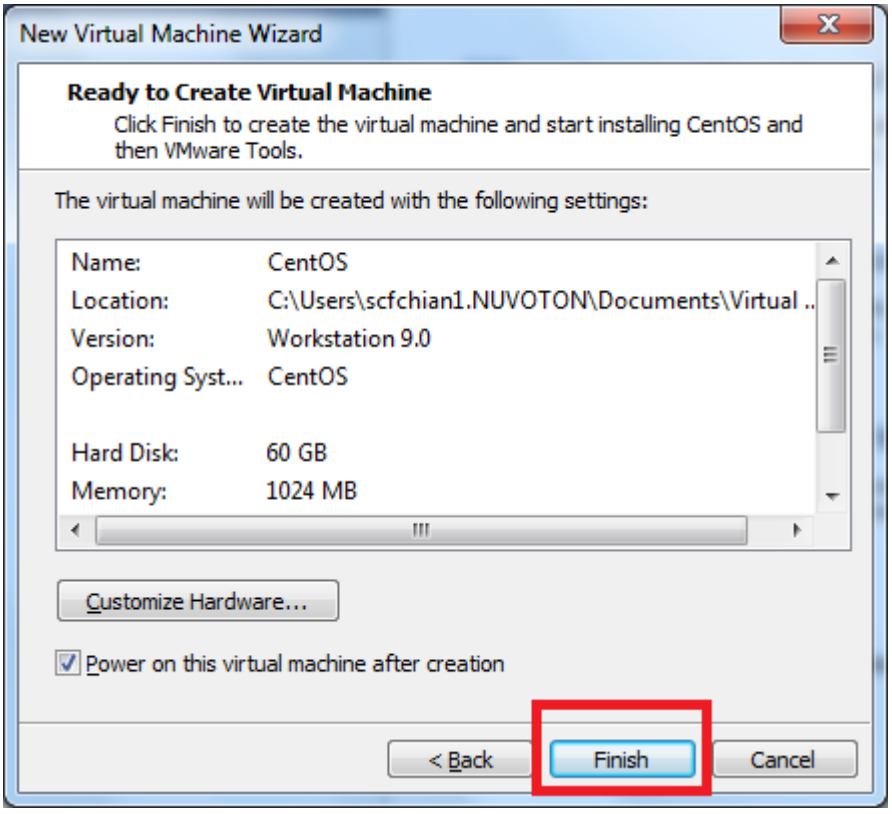
接下來, 若有必要則輸入 “Virtual machine name:” 以及 “Location:”, 否則保留默認值即可.



下一步則是輸入 “Maximum disk size (GB):” 值, 建議至少 60GB, 然後選擇 “Store virtual disk as a single file”.

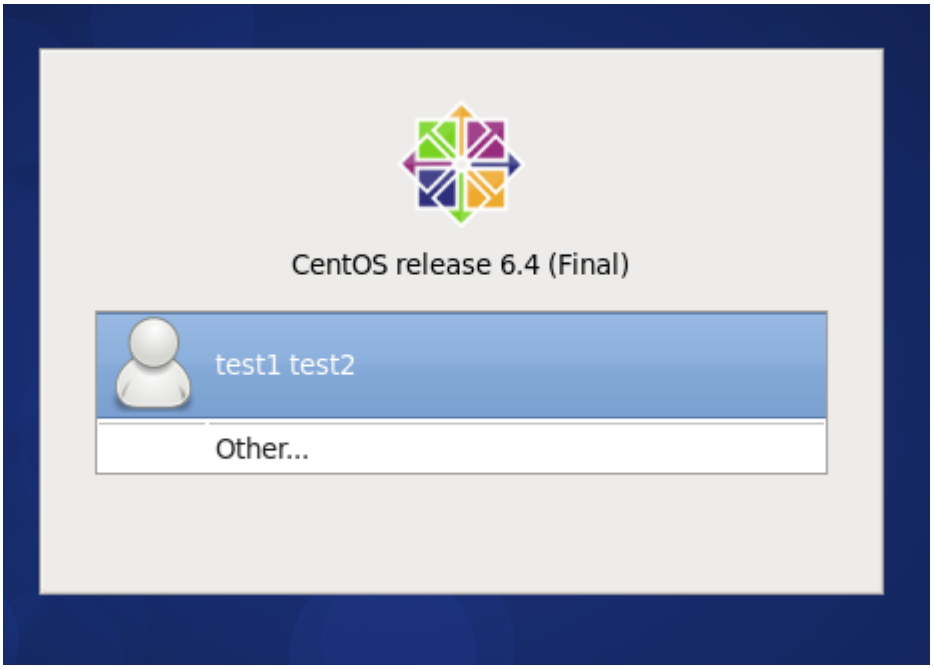

New Virtual Machine Wizard
Specify Disk Capacity
How large do you want this disk to be?
The virtual machine's hard disk is stored as one or more files on the host computer's physical disk. These file(s) start small and become larger as you add applications, files, and data to your virtual machine.
Maximum disk size (GB):
60.0
Recommended size for CentOS: 20 GB
Store virtual disk as a single file
Split virtual disk into multiple files
Splitting the disk makes it easier to move the virtual machine to another computer but may reduce performance with very large disks.
Help
< Back
Next >
Cancel

最後一步則是按下 “Finish” 來完成 CentOS 的設定.



VMware 會繼續完成 CentOS 的安裝.

安裝完成後, 會顯示 CentOS 的登入畫面, 此時可用之前設置的用戶名稱及密碼來登入



2.4 補齊元件

多數的 Linux 發行版不會將所有的原件都安裝. 但有些元件在使用 BSP 的開發過程中是必須的. 另外有些元件可讓開發的過程更加順利. 以下列了一些Linux 安裝過程會省略, 但必須及建議安裝的原件.

元件名稱	功用	必須/建議
patch	用來打補丁的工具	必須
libc6-dev	交叉編譯去所需動態鏈結的 32 位函式庫 (i386 版本)	必須
libncurses5-dev	設置內核介面所需使用的動態鏈結庫	必須
Minicom 或cutecom	串口控制台, 可用來顯示以及控制 U-Boot 及內核.	建議

各個Linux 發行版的元件安裝介面不盡相同. Ubuntu 的使用者可以使用 apt-get 命令或是 Synaptic Package Manager 來安裝元件. 而 Fedora的使用者可以使用 rpm 命令或是 Package Manager來安裝元件. 請參考所使用的 Linux 發行版文件來安裝缺少的元件.

2.5 BSP 安裝步驟

Linux BSP 包含了三個目錄. 各目錄的內容列在下表:

目錄名稱	內容
BSP	一個壓縮包包含了 U-Boot, Linux 內核, 範例程序的源碼. 交叉編譯工具, 以及根文件系統.
Documents	BSP 相關文件
Tools	Windows 上的燒錄工具以及驅動程式

請將 BSP 目錄中的壓縮包複製到 Linux 開發機器上. 並使用以下的命令解壓縮 :

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

在此目錄中有安裝腳本 install.sh. 此腳本需要管理者權限才可以執行. 可以選擇使用 “su” 命令切換到管理者來執行:

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

或是使用 sudo 來執行安裝腳本 (若是安裝的 Linux 沒有開放 root 權限, 例如 Ubuntu, 則可以使用本方式來安裝 BSP)

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

以下為整個安裝過程：

```
Extract arm_linux_4.8.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
```

若是使用的 Linux 開發環境之前已經安裝過新唐提供的交叉編譯工具, 安裝腳本會詢問是

否須複寫編譯工具, 否則腳本並不會詢問使用者, 而是直接在 /usr/local/arm_linux_4.8目錄安裝編譯工具. 在第一種已安裝過編譯工具的情況下, 若是要複寫, 可按Y (或是 yes、y、YES), 然後按 Enter 鍵.
安裝完成交叉編譯工具後, 安裝腳本會詢問安裝 Linux 內核, U-Boot, 以及範例程序的***絕對路徑***. 下表列出了會裝在指定目錄中的項目.

目錄名稱	內容
applications	範例程序以及開源軟件, 例如 busybox, wireless tool...
image/kernel	使用默認設置預先編譯好的內核
image/U-Boot	使用默認設置預先編譯好的支持 NAND 或 SPI flash 的 U-Boot 執行檔及環境變數env.txt檔。其中 U-Boot 的默認執行位址均為 0xE00000。燒錄方式請參考3.11章節。
linux-3.10.x	內核源碼
rootfs	根文件系統(arm_linux_4.3 環境使用)
rootfs_48	根文件系統(arm_linux_4.8 環境使用)
Rootfs_tomato	根文件系統 (Tomato開發板使用)
U-Boot	U-Boot 源碼

安裝腳本會嘗試將安裝的目錄設置正確權限, 並將交叉編譯器的路經加至系統搜尋路徑($PATH)中. 但在有些 Linux 版本中, 可能發生無法正確設置的問題. 此時需麻煩使用者手動設置正確的權限並且將/usr/local/arm_linux_4.8/bin加到 $PATH 中.
請注意, 在安裝完成後, 使用者須先登出再登入, $PATH 的設定才會生效.

3 Nu-Writer 使用說明

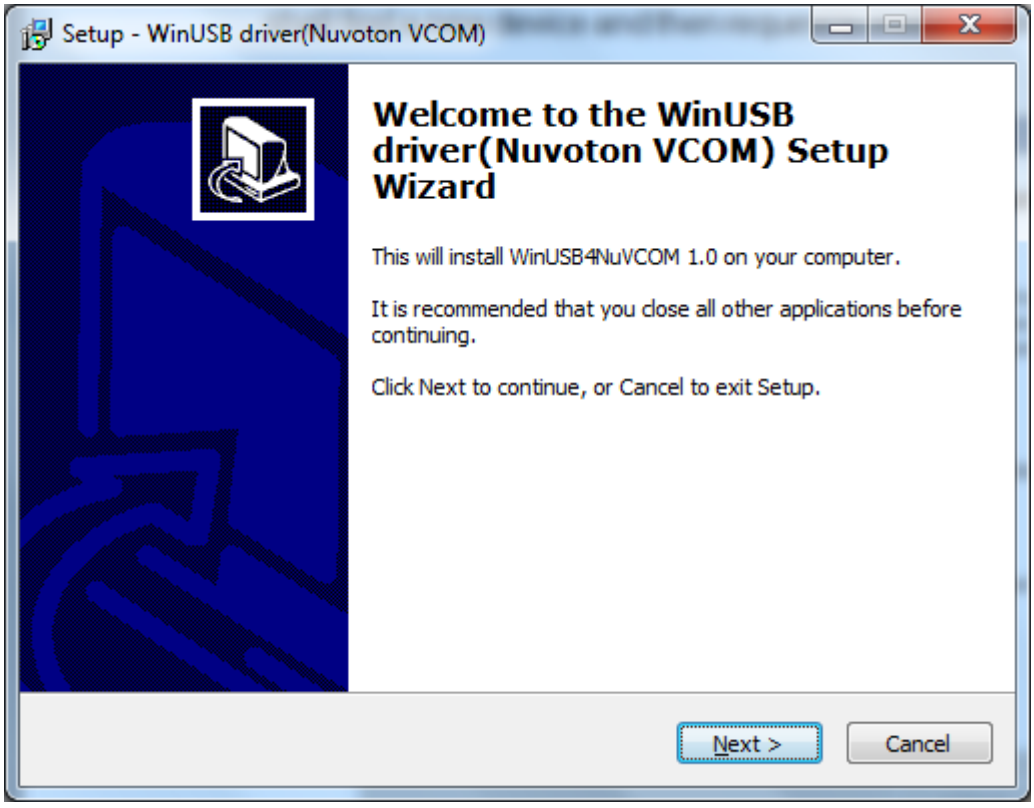
3.1 簡介

Nu-Writer 工具能幫助使用者透過 USB ISP模式, 將Image檔案放入儲存體中, 例如：SPI Flash 設備或 NAND Flash設備.

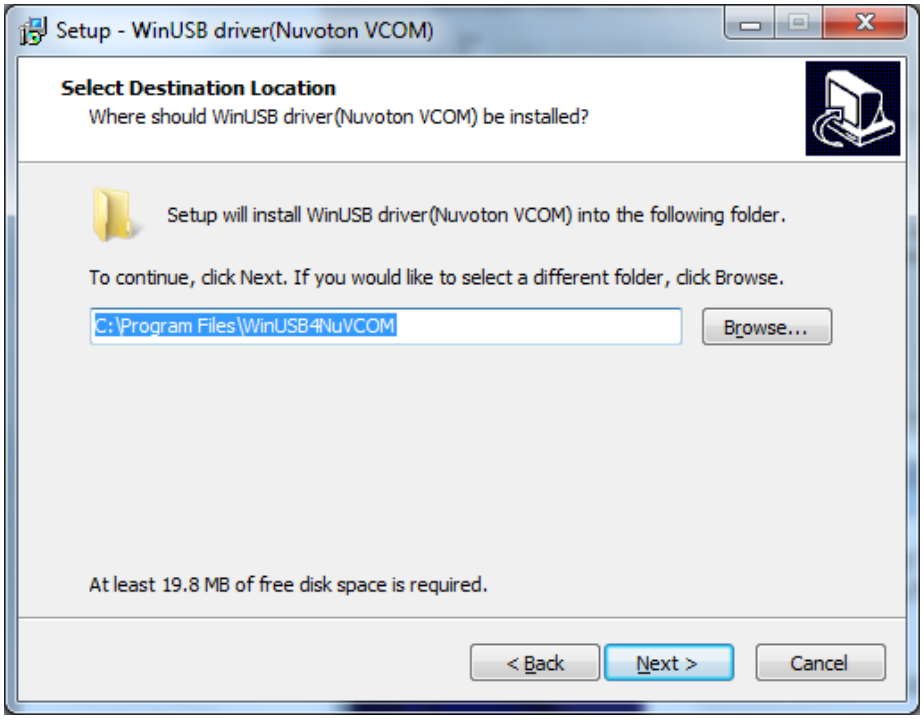
3.2 驅動程式安裝

Nu-Writer 必須在電腦中安裝VCOM驅動程式才能使用Nu-Writer工具. 請依據下列步驟來安裝VCOM驅動程式：

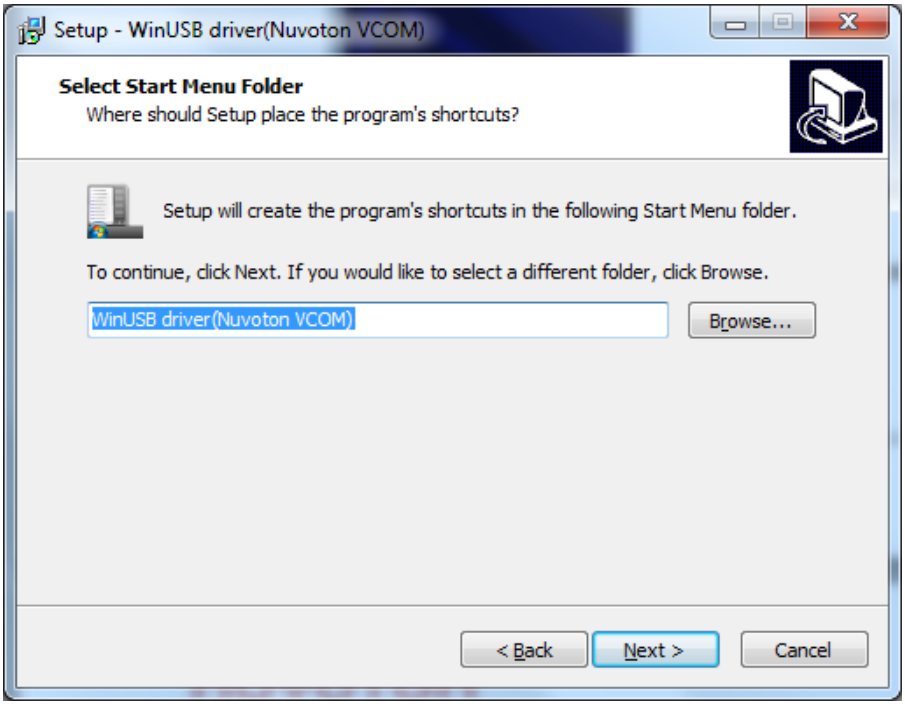
1. 將電腦與NUC970系列晶片開發板透過USB cable連接起來後.在電腦中執行 WinUSB4NuVCOM.exe 開始安裝驅動程式. (在Linux BSP的Tools目錄中)
2. 開啟NUC970系列晶片開發板的電源之後, Windows 會發現新的設備, 然後會要求你安裝驅動程式.



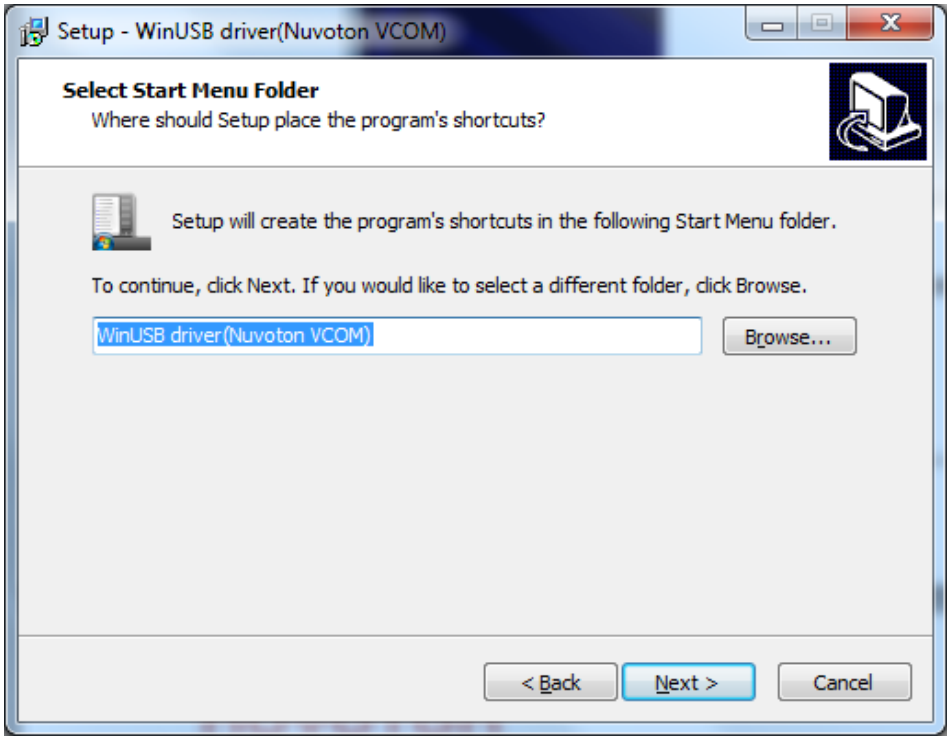
3. 按下 “Next”. 這個畫面告訴你即將要安裝WinUSB4NuVCOM 1.0 驅動程式. 如下圖：



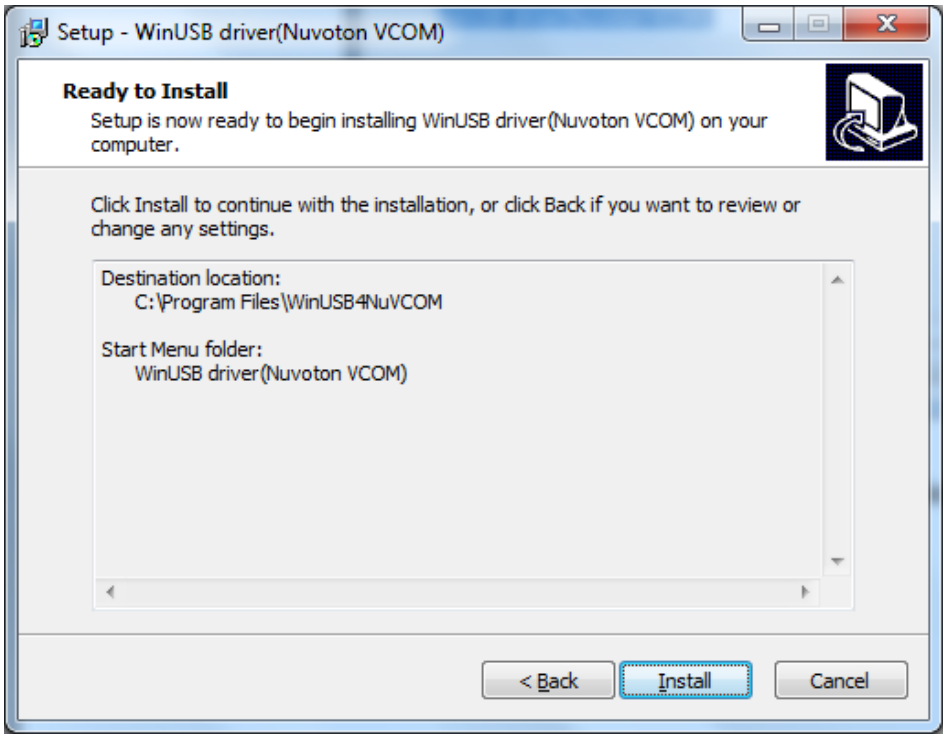
4. 選擇使用者想要安裝的路徑或使用預設的路徑, 確定以後按下 “Next”. 如下圖：



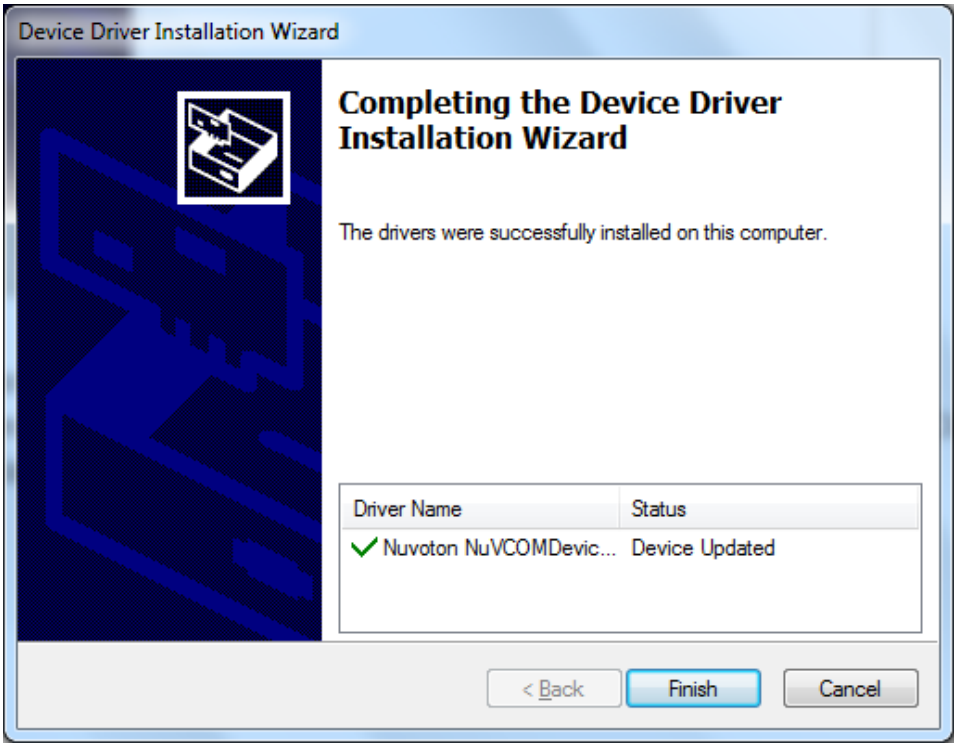
5. 按下 “Next”. 如下圖：



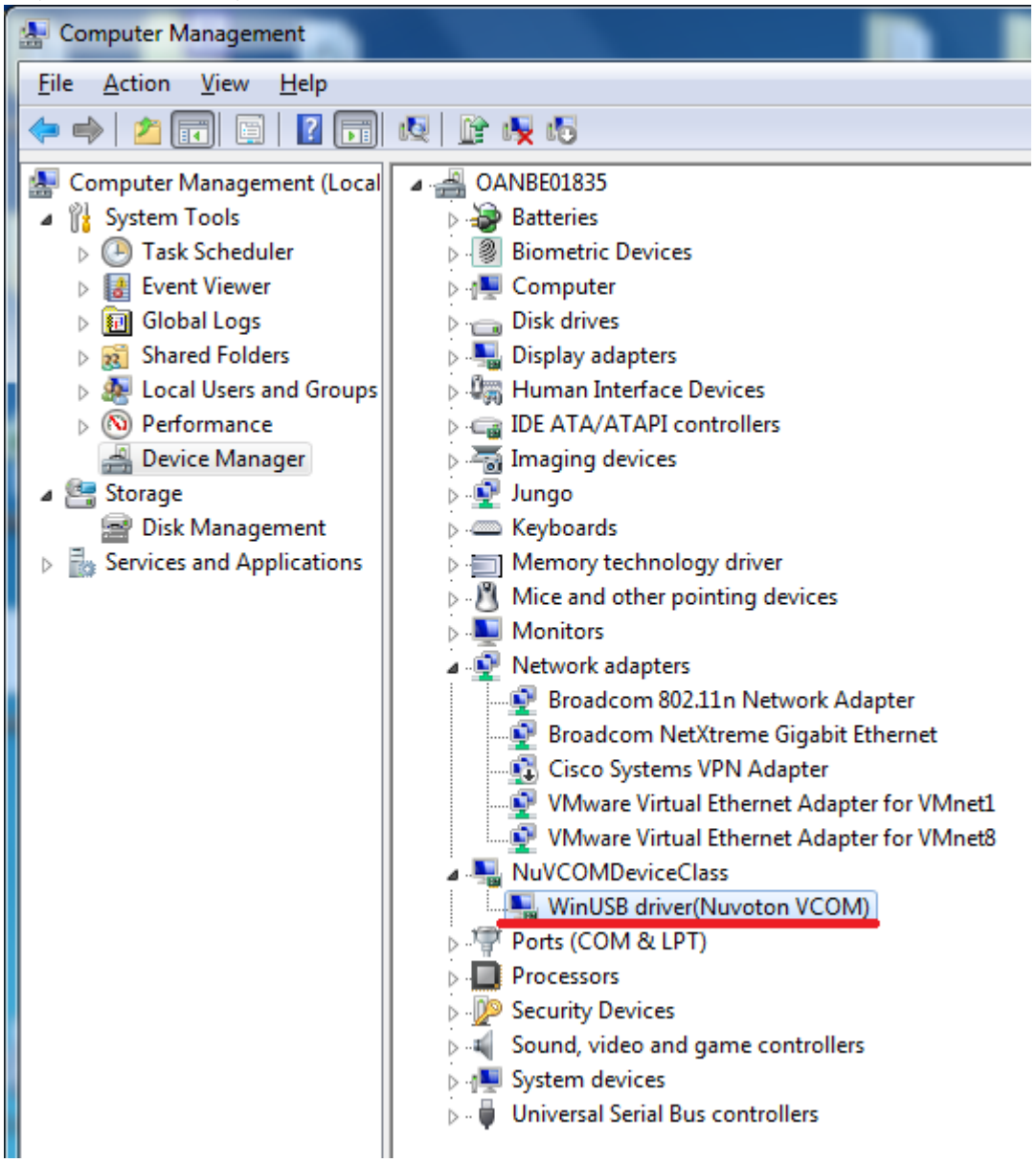
6. 按下 “Install”. 如下圖：



7. 按下 “Finish”, 完成VCOM驅動程式的安裝. 如下圖：



8. 如果VCOM驅動程式是安裝成功, 可以在”Device Manager”中看到“WinUSB driver (Nuvoton VCOM)”. 如下圖：



3.3 USB ISP 模式設置

NUC970系列晶片提供jumpers 去選擇開機的方法. 選擇USB ISP 模式, 則 PA0和PA1 必須設定為低電平 . 其他開機設定可以參考下表：

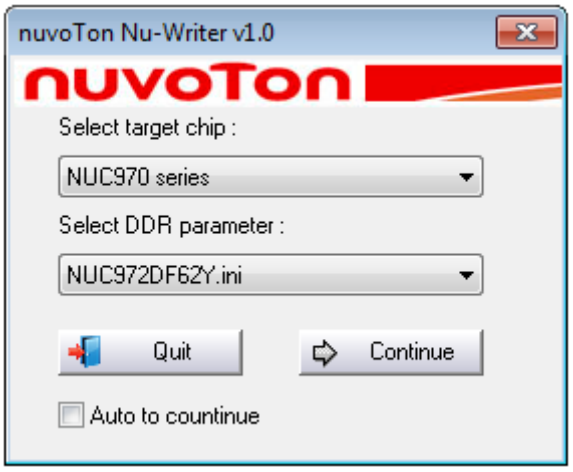
開機設定	PA1	PA0

USB ISP 開機	Low	Low
eMMC 開機	Low	High
NAND 開機	High	Low
SPI 開機	High	High

開啟NUC970系列晶片開發板的電源並且設定為USB ISP模式和開啟電腦上的Nu-Writer 工具, 即可開始使用.
注意：如果電腦沒有找到VCOM驅動程式則Nu-Writer工具無法使用.

3.4 芯片設置

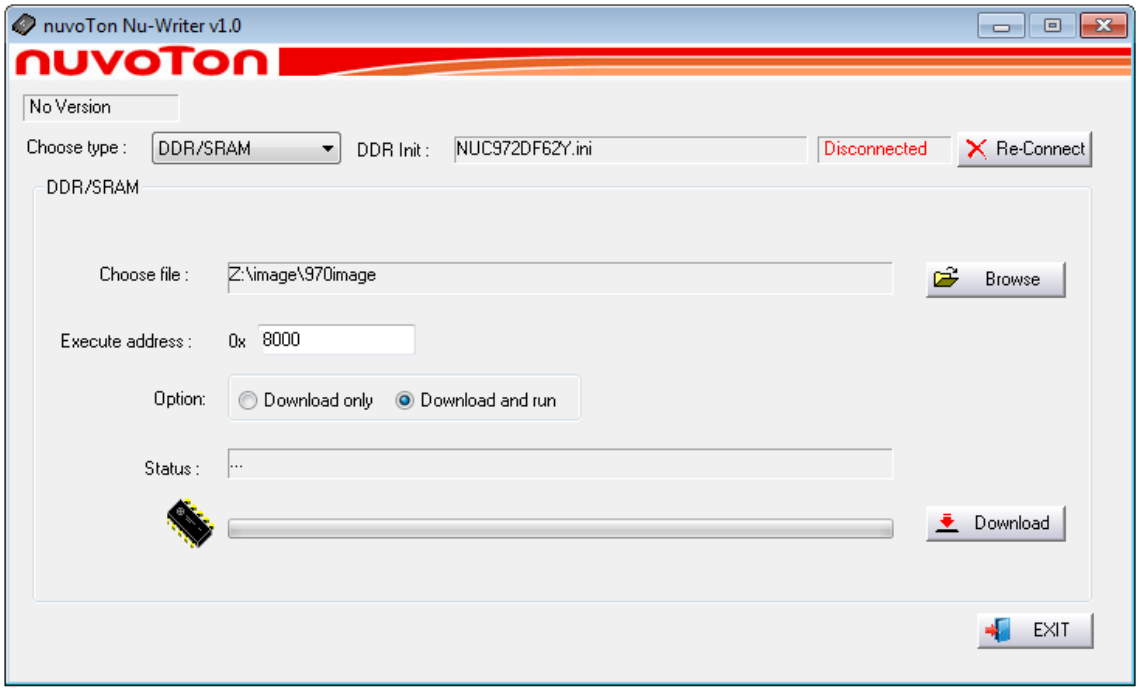
解開NuWriter-xxxxxxxx.7z(在BSP/Tools目錄下)壓縮包, 執行 “nuwriter.exe”, 第一個畫面如下.
選擇目前晶片, 目前支援NUC970系列 (NUC972, NUC976… 等)芯片. 如果選擇NUC970系列芯片, 則必須選擇DDR參數, DDR參數依據NUC970系列芯片的PID來選擇. 選擇完成後按下“Continue”, 即可開始使用Nu-Writer工具.



3.5 DDR/SRAM 模式

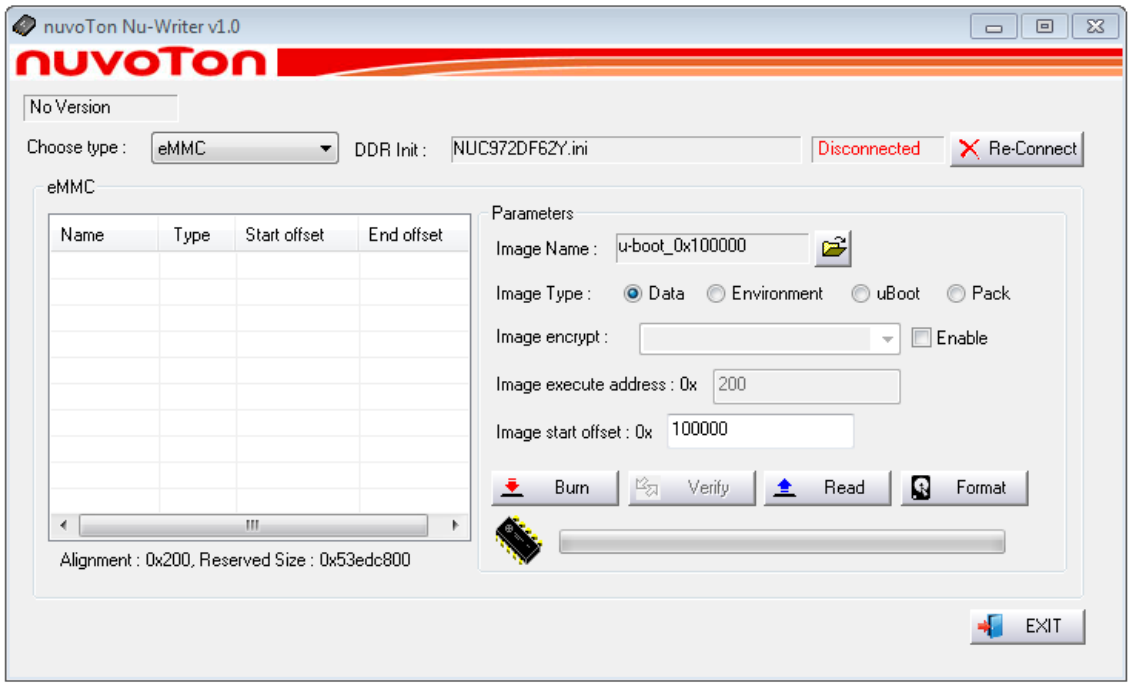
DDR/SRAM模式可以將Image檔案直接下載到DDR 或 SRAM 記憶體中. 操作步驟如下：

1. 選擇 “DDR/SRAM” 模式.
2. 選擇Image檔案.
3. 輸入Image檔案放在DDR/SRAM的位址. 注意：若要傳輸到 DDR 中, 位址必須介於0x00~0x1F00000(31MB).
4. 選擇” Download only” 或是選擇” Download and run”
5. 按下 “Download.



3.6 eMMC 模式

eMMC 模式 可以將Image檔案燒入到eMMC中，並且將Image檔案型態設定為Data、Environment、uBoot 、 Pack，四種型態中的其中一種 。



3.6.1 新增Image檔案

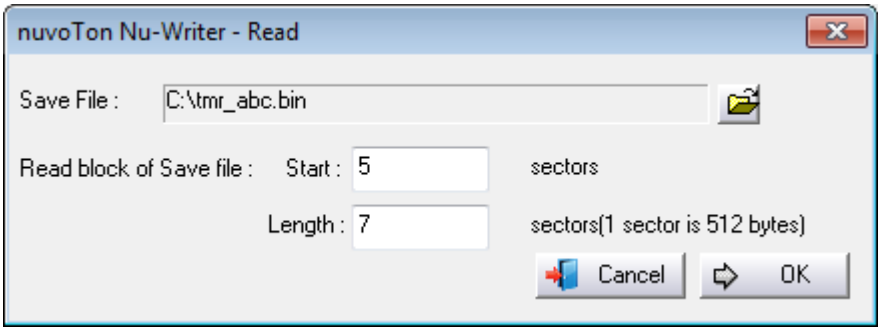
1. 選擇 “eMMC” 模式， 表格只會紀錄當次燒錄的Image檔案,並不會讀取eMMC Flash中Image的資料。
2. 輸入 image檔案資料：
   - Image Name
   - Image Type
   - Image encrypt
   - Image execute address
   - Image start offset
3. 按下 “Burn” 。
4. 等待進度表完成後，表格將會顯示這次燒錄完成的Image檔案。在完成以後，如果按下“Verify”即可確認燒入資料是否正確。

3.6.2 讀取eMMC

依照下列步驟即可以完成新增Image檔案：

1. 選擇 “eMMC” 模式。
2. 按下 “Read” 。
3. 輸入儲存的檔案。
4. 輸入讀回來的sectors(1 sector is 512 bytes) 。
   - Start : Sector 起始位置
   - Length : Sector 長度

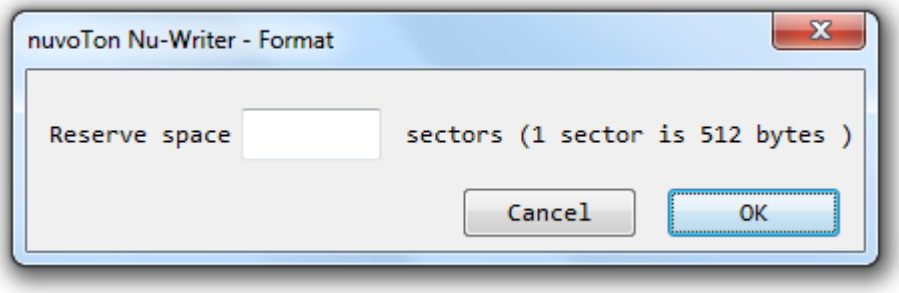
5. 按下 “OK” 。即可完成。



3.6.3 格式化 (FAT32)

依照下列步驟即可以完成eMMC格式化：

1. 選擇 “eMMC” 模式。
2. 按下 “Format” 。
3. 輸入保留空間(單位為512bytes)。注意：修改此參數可能造成Image或FAT32格式損毀。
4. 按下 “OK” 。即可完成。



3.7 SPI 模式

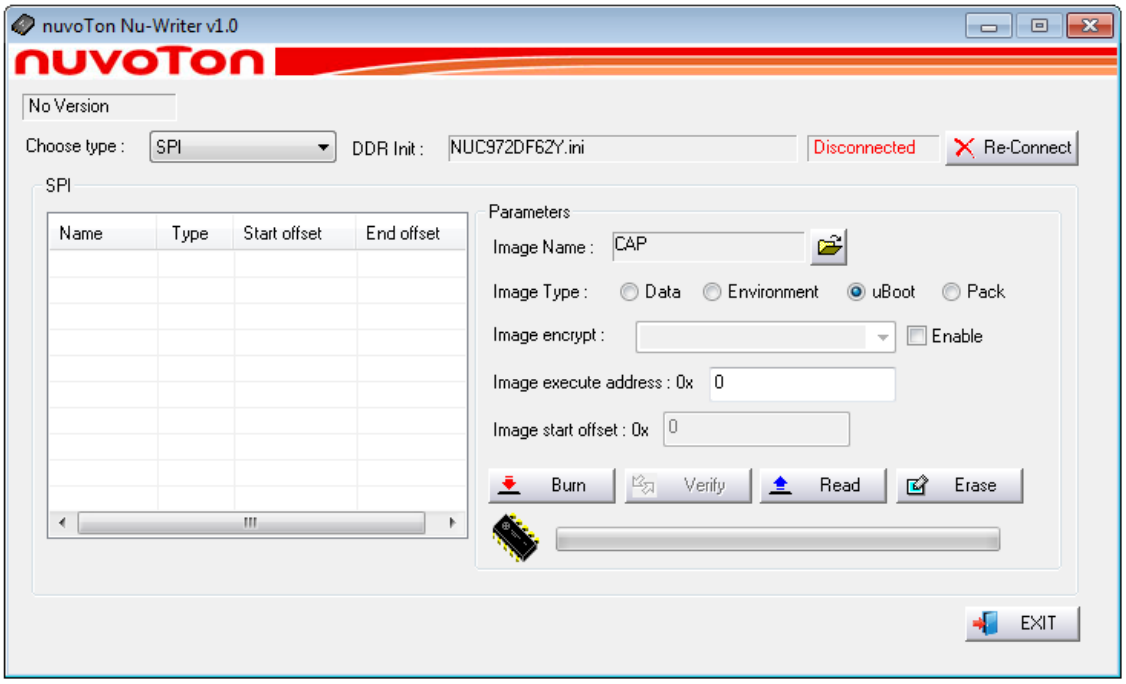
SPI 模式 可以將Image檔案燒入到SPI Flash中，並且將Image檔案型態設定為Data、Environment、uBoot 、 Pack, 四種型態中的其中一種 。

3.7.1 新增Image

依照下列步驟即可以完成新增Image檔案：

1. 選擇 “SPI” 模式, 表格只會紀錄當次燒錄的Image檔案，並不會讀取SPI Flash中Image的資料。
2. 輸入 image檔案資料：
   - Image Name 選擇要燒錄的 loader 檔案
   - Image Type 選擇燒錄 Image的型態
   - Image encrypt 設置是否需AES加密, 若是, 設置秘鑰文件
   - Image execute address 設置 loader 執行位置, 依編譯設定而輸入.
   - Image start offset 燒錄起始塊位置

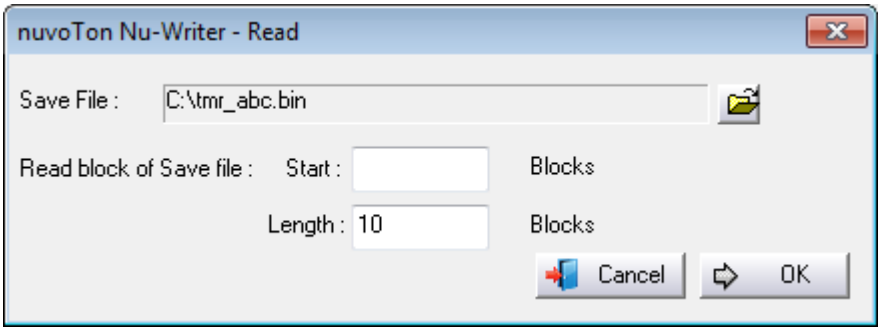
3. 按下 “Burn”.
4. 等待進度表完成. 在完成以後如果按下“Verify”即可確認燒入資料是否正確.



3.7.2 讀取Image

依照下列步驟即可以完成讀取Image：

1. 選擇 “SPI” 模式。
2. 按下 “Read” 。
3. 選擇要儲存檔案的位置。
4. 輸入讀回來的blocks，Block 大小是依據SPI FLASH規格所決定。
   - Start : Block 起始位置
   - Length : Block 長度
5. 按下 “OK” ， 即可完成Image讀取。



3.7.3 移除 Image

依照下列步驟即可以完成移除Image檔案：

1. 選擇 “SPI” 模式。
2. 按下 “Erase all” ，即可完成移除Image。

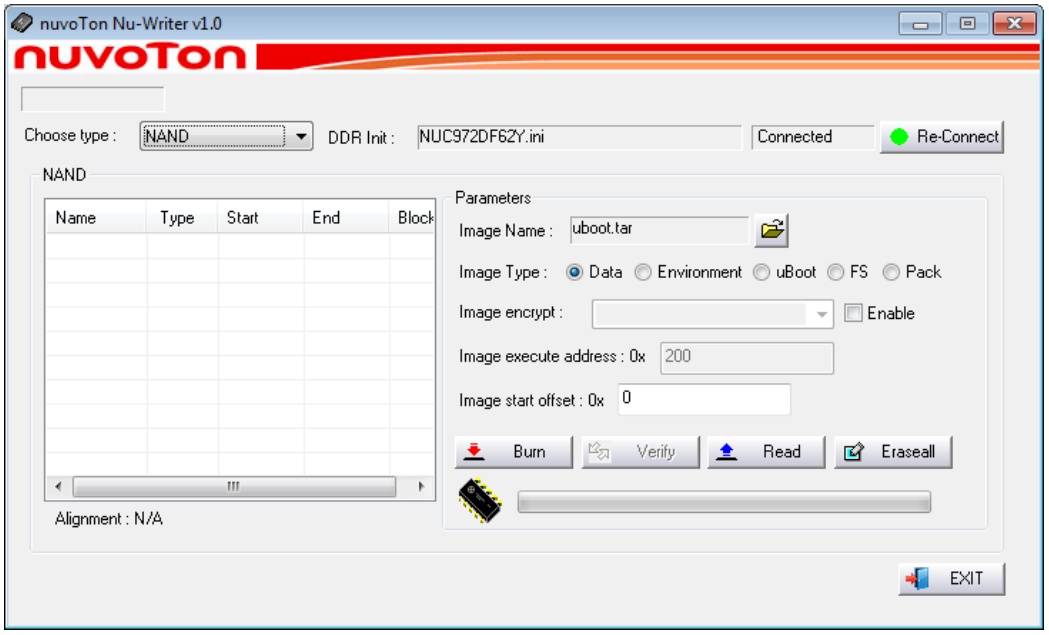
3.8 NAND 模式

NAND模式 可以將Image檔案燒入到NAND Flash中, 並且將Image檔案型態設定為Data、Environment、uBoot 、 FS、Pack, 五種型態中的其中一種 . FS型態目前支援YAFFS2與UBIFS兩種檔案系統格式。這兩種格式都可以選擇FS型態，將做好的Image存放到NAND Flash對應的位址。讓使用者可以透過uBoot或Linux來讀取檔案系統。YAFFS2與UBIFS的Image檔的製作可以參考3.8.4章節

3.8.1 新增Image

依照下列步驟即可以完成新增Image檔案：

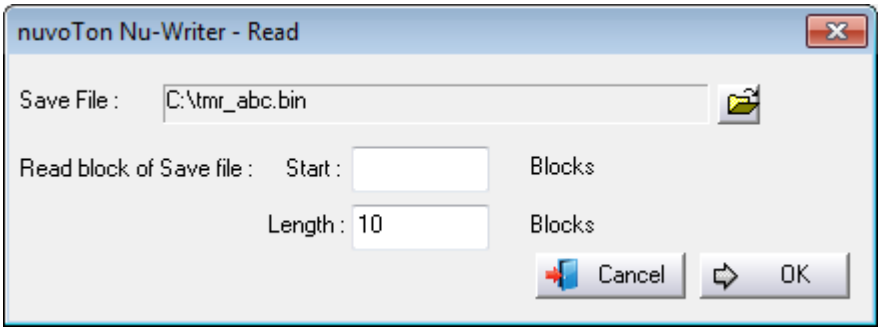
1. 選擇 “NAND” 模式, 表格只會紀錄當次燒錄的Image檔案,並不會讀取NAND Flash中Image的資料.
2. 按下 “Add new” . (當選擇NAND模式時, 預設為 “Add new” 頁面)
3. 輸入 image檔案資料：
   - Image Name 選擇要燒錄的 loader 檔案
   - Image Type 選擇燒錄 Image的型態
   - Image encrypt 設置是否需AES加密, 若是, 設置秘鑰文件
   - Image execute address 設置 loader 執行位置, 依編譯設定輸入.
   - Image start offset 燒錄起始塊位置
4. 按下 “Burn” .
5. 等待進度表完成. 在完成以後如果按下 “Verify” 即可確認燒入資料是否正確.



3.8.2 讀取Image

依照下列步驟即可以完成讀取Image：

1. 選擇 “NAND” 模式。
2. 按下 “Read” 。
3. 選擇要儲存檔案的位置。
4. 輸入讀回來的blocks，Block 大小是依據NAND FLASH規格所決定。
   - Start : Block 起始位置
   - Length : Block 長度
5. 按下 “OK” ， 即可完成Image讀取。



3.8.3 移除 Image

依照下列步驟即可以完成移除Image檔案：

1. 選擇 “NAND” 模式。

2. 按下 “Erase all” ，即可完成移除Image。

3.8.4 製作File System Image

這裡說明如何製作YAFFS2(yaffs2的tag儲存在DATA區塊中)和UBIFS，這兩種File System都可以使用DATA型態燒入到 NAND flash中，依據下列步驟即可完成製作File System Image。

1. YAFFS2製作In-band tags Image命令如下：(yaffs2的tag儲存在DATA區塊中)

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

--inband-tags：yaffs2的tag儲存在DATA區塊。
-p：設定NAND Flash頁的大小(Page Size)。

即可將rootfs資料夾壓縮成rootfs_yaffs2.img，再透過NuWriter放到相對應NAND Flash的位址。輸入下列命令即可將YAFFS2 inband-tags 檔案系統掛在flash資料夾中：

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

YAFFS2的指令可以在yaffs2utils套件中找到。YAFFS2文件系統設置可以參考5.3.4章節。

2. UBIFS製作Image命令如下：

```
# 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 ubinize.cfg
```

mkfs.ubifs 使用的參數說明如下：
-F ：設定檔案系統未使用的空間優先mount.
-x： 壓縮的格式，"lzo", "favor_lzo", "zlib" 或 "none" (預設："lzo")
-m：最小的I/O操作的大小，也就是NAND Flash一個頁的大小。
-e：邏輯擦除塊的大小(logical erase block size)。因為實體擦除塊(PEB)為128KiB，所以邏輯擦除塊設定為124KiB=126976。
-c：最大的擦除塊的號碼(maximum logical erase block count)。
-o：輸出檔案。
ubinize使用的參數說明名如下：
-o：輸出檔案。
-m：最小輸入/輸出的大小，也就是NAND Flash一個頁的大小。
-p：實體擦除塊大小，128KiB=131072。
-O：VID檔頭位移位置。
-s：使用最小輸入/輸出的大小，存放UBI檔頭。
ubinize.cfg 內容如下：

```
[rootfs-volume]
```

```
mode=ubi
image=rootfs_ubifs.img
vol_id=0
vol_size=92946432
vol_type=dynamic
vol_name=system
vol_flags=autoresize
```

即可將rootfs資料夾壓縮成ubi.img，再透過NuWriter放到相對應NAND Flash的位址。
輸入下列命令即可將UBIFS檔案系統掛在flash資料夾中：

需要參考/sys/class/misc/ubi_ctrl/dev內容，假設內容為 10：56，則設定如下：

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

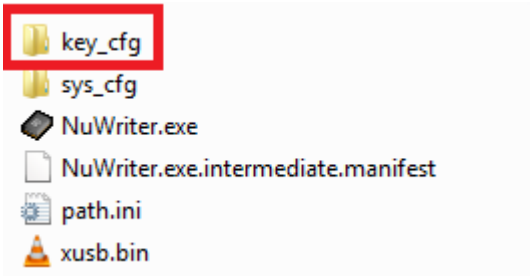
UBIFS相關指令可以在mtd-utils套件中找到。UBIFS文件系統設置可以參考5.3.4章節。

3.9 MTP 模式

MTP模式可以將你選擇的鑰匙檔案燒入到NUC970系列晶片的 MTP中, 藉由此鑰匙來保護NUC970系列晶片使用到存儲體(eMMC, NAND, SPI FLASH)中的程式碼.

3.9.1 新增key檔案

1. 進入資料夾key_cfg如下圖.

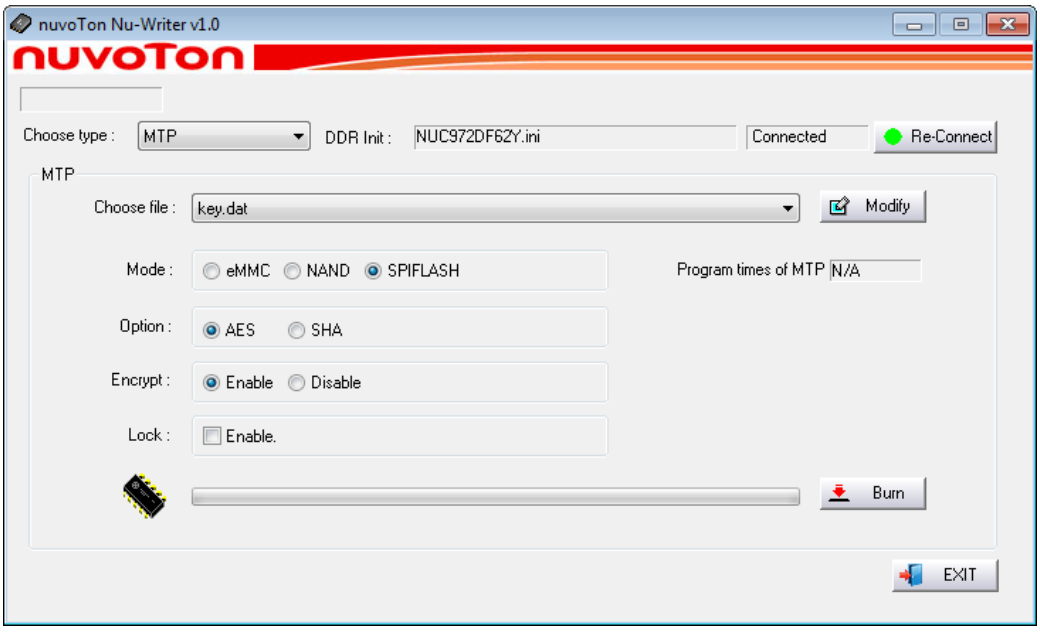


2. 建立文字檔和輸入密碼, 密碼格式如下, 第一行一定是 256. 之後連續 8 行大端模式密鑰.

```
256
0x12345678
0x23456789
```

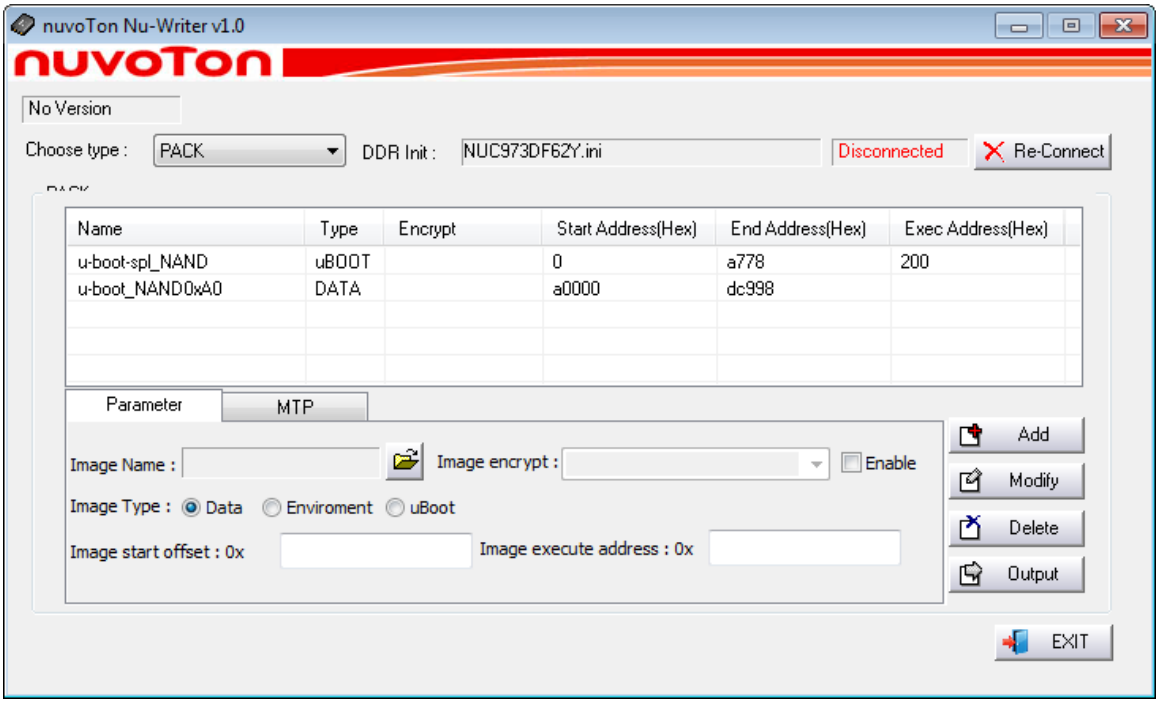
```
0x3456789a

0x456789ab

0x56789abc

0x6789abcd

0x789abcde

0x89abcdef
```

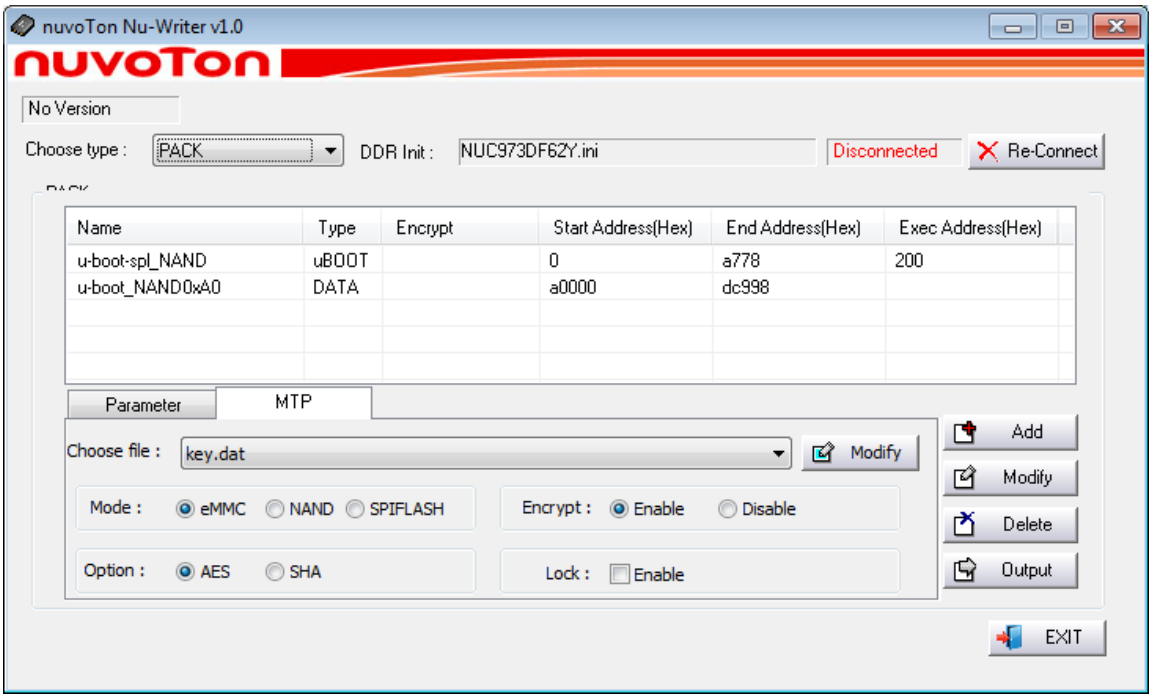
3. 重新開啟Nu-Writer工具, 並選擇 “MTP” 模式.
4. 選擇剛剛建立的文字檔.
5. 選擇燒入的方式
   - 開機模式選擇: eMMC, NAND, 或 SPIFLASH
   - 保護模式選擇: SHA 或 AES
   - 啟動模式選擇: Enable 或 Disable
   - 上鎖模式選擇: 如果此模式開啟時，則永久無法修改MTP相關的設定，使用此模式時請小心
   -
6. 按下 “Burn”. 即可完成燒錄動作.



3.10 PACK 模式

PACK模式可以將多個image檔案合併成一個pack image檔案，往後就可利用Nu-Writer將這個pack image直接快速的燒入到對應的存儲體中。（如 eMMC，NAND，SPI FLASH）





3.10.1 新增Image

依照下列步驟即可以完成新增image檔案：

1. 選擇 “Pack” 模式。

2. 選擇Parameter 或 MTP
3. Parameter：
   - Image Name 選擇要燒錄的檔案
   - Image Type 選擇燒錄 Image的型態
   - Image encrypt 設置是否需AES加密, 若是, 設置秘鑰文件
   - Image execute address 設置 loader 執行位置, 依編譯設定輸入.
   - Image start offset 燒錄起始塊位置
4. MTP：
   - 開機模式 :選擇模式
   - 保護模式 :選擇AES或SHA，如果選擇AES則需要選擇Key，反之如果選擇 SHA則需要選擇需要計算SHA的檔案
   - 啟動模式選擇： Enable或Disable
   - 上鎖模式 : 如果此模式開啟時，則永久無法修改MTP相關的設定，使用此模式時請注意
5. 按下 “Add” 。

3.10.2 修改Image

依照下列步驟即可以完成修改image：

1. 選擇 “Pack” 模式。
2. 輸入 image檔案資料：
   - Image Name 選擇要燒錄的檔案
   - Image Type 選擇燒錄 Image的型態
   - Image encrypt 設置是否需AES加密, 若是, 設置秘鑰文件
   - Image execute address 設置 loader 執行位置, 依編譯設定輸入.
   - Image start offset 燒錄起始塊位置
3. 按下 “Modify” 。

3.10.3 移除Image

依照下列步驟即可以完成移除image：

1. 選擇 “Pack” 模式。
2. 在image list上點選要刪除的image。
3. 按下 “Delete” 。

3.10.4 輸出pack檔案

依照下列步驟即可以完成輸出pack檔案：

1. 選擇 “Pack” 模式。

2. 按下“Output”。
3. 選擇儲存的檔案，按下open即可。

3.10.5 燒錄pack檔案

依照下列步驟即可以完成燒錄pack檔案：

1. 選擇需要燒錄的存儲體(eMMc/NAND flash/ SPI flash)。
2. 在Image Name 選擇要之前所產生的pack檔案。
3. 按下“Burn”。

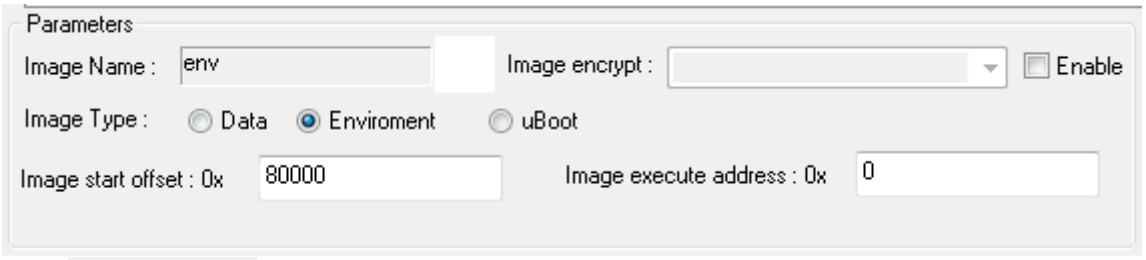
3.10.6 製作和燒錄pack範例

準備相關的檔案：

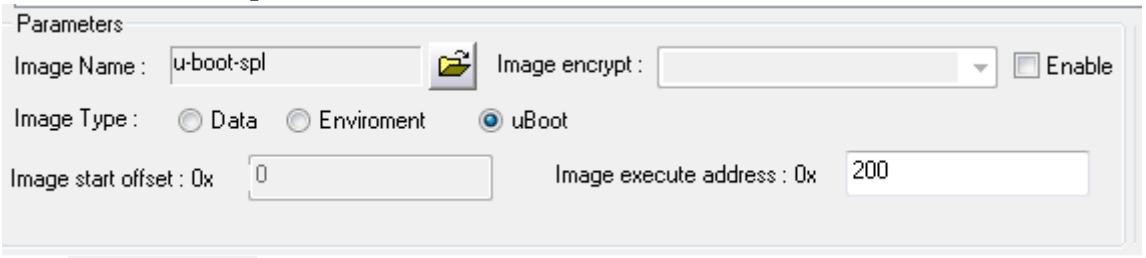
1. u-boot.bin (預設 offset 為 0x100000，執行位置為 0xE00000)
2. u-boot-spl.bin (預設DDR執行位置為 0x200)
3. env.txt (預設 offset為 0x80000)

目前假設需要將env.txt、u-boot.bin和u-boot-spl.bin 製作成一個pack檔案並燒錄到NAND
依照下列步驟即可以完成製作pack檔案：

1. 選擇PACK模式。
2. 按下 選 env.txt，並設定如下：

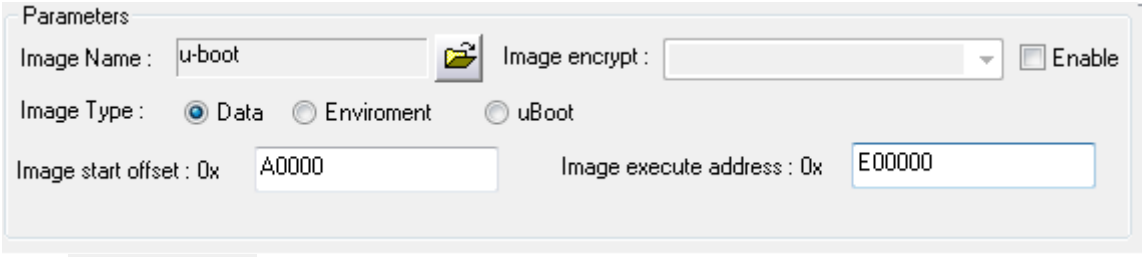


3. 按下 Add 。
4. 按下 選u-boot-spl.bin，並設定如下：



5. 按下 Add 。

6. 按下  選u-boot.bin，並設定如下：



7. 按下 

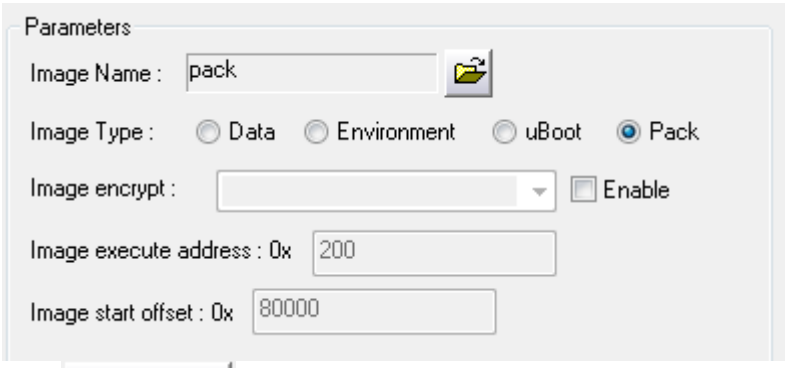
。

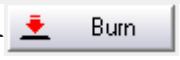
8. 按下 

，可產生一個pack 檔案。

依照下列步驟即可以完成燒錄pack檔案：

1. 選擇NAND模式。
2. 按下  選 剛剛產生出的檔案，並設定如下：



3. 按下 

，即可燒錄pack檔案。

3.11 燒錄U-Boot

依照下列步驟將編譯完成的U-Boot燒錄至NAND Flash/SPI Flash/eMMC 中。
U-Boot的編譯方法請參考4.3章節。

3.11.1 燒錄所需檔案

4. u-boot.bin (預設 offset 為 0x100000，執行位置為 0xE00000)
5. u-boot-spl.bin (預設DDR執行位置為 0x200)
6. env.txt (預設 offset為 0x80000)

各檔案中配置的offset 位置及執行位置可參考4.1章節內容說明。

3.11.2 U-Boot 環境變數檔案(env.txt)內容說明

env.txt 存放的是 U-Boot 的環境變數及其數值，內容舉例如下:

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

每一行表示一個 U-Boot 環境變數，格式為:
變數=數值
變數、=(等號) 和數值之間不要有空白，換行符號為 (0x0d, 0x0a)。
env.txt 當中的變數為 U-Boot 預設環境變數，變數意義請參考4.6.2預設的環境變數。

3.11.3 燒錄至NAND Flash

1. 選擇 “NAND” 模式。
2. 選擇 u-boot-spl.bin 檔案， 設定 **image type**為 **uBoot** 模式 ，設定 **image execute address** 為 0x200 ，按burn 燒錄 u-boot-spl.bin.。
3. 選擇 u-boot.bin檔案，設定 **image type**為 **Data** 模式，設定 **image start offset** 為0x100000 ，按**burn** 燒錄 u-boot.bin 。
4. 選擇 env.txt檔案，設定 **image type**為 **Environment** 模式，設定 **image start offset** 為0x80000，按 **burn** 燒錄env.txt 。

3.11.4 燒錄至SPI Flash

1. 選擇 “SPI” 模式。
2. 選擇 u-boot.bin 檔案， 設定 **image type**為 **uBoot** 模式，設定 **image execute address** 為 0xE00000，按burn 燒錄 u-boot.bin.。
   （**image execute address** 位址可以調整，請參考 4.3.3章節說明。）
3. 選擇 env.txt檔案，設定 **image type**為 **Environment** 模式，設定 **image start offset** 為0x80000，按 **burn** 燒錄env.txt 。

3.11.5 燒錄至eMMC

1. 選擇 “eMMC” 模式。

2. 選擇 u-boot.bin 檔案， 設定 **image type**為 **uBoot** 模式，設定 **image execute address** 為 0xE00000，按burn 燒錄 u-boot.bin.。
   (**image execute address** 位址可以調整，請參考 4.3.3章節說明。)
3. 選擇 env.txt檔案，設定 **image type**為 **Environment** 模式，設定 **image start offset** 為0x80000，按 **burn** 燒錄env.txt 。

3.12 解決無法啟動Nu-Writer的問題

目前Nu-Writer 是基於microsoft visual C++ 2008平台所編寫的一套工具,所以在執行此工具時,如遇到無法啟動的現象時,很有可能是由於在PC上缺少 “Microsoft Visual C++ 2008 Redistributalbe” 元件的關係。
如果缺少此元件，請至microsoft網站下載並安裝。
簡體中文版可至此下載 –
http://www.microsoft.com/en-us/download/details.aspx?id=29

4 U-Boot 使用說明

U-Boot 是一個主要用於嵌入式系統的開機載入程式, 可以支援多種不同的計算機系統結構, 包括ARM、MIPS、x86與 68K. 這也是一套在GNU通用公共許可證之下發布的自由軟體. 他支援下列功能:

- 網路下載: TFTP, BOOTP, DHCP
- 串口下載: s-record, binary (via Kermit)
- Flash 管理: 抹除, 讀, 寫
- Flash 型別: SPI flash, NAND flash
- 記憶體工具: 讀, 寫, 複製, 比對
- 交互式 shell: 命令, 腳本

NUC970 U-Boot 的版本是 201304RC2. 從下面連結下載
http://www.denx.de/wiki/U-Boot/SourceCode
U-Boot 官網上對各項功能有更詳盡的介紹
http://www.denx.de/wiki/view/DULG/UBoot

4.1 配置

U-Boot 是可配置的, 修改配置檔中的各項定義來產生不同的配置.
NUC970 配置檔位於 include/configs/nuc970_evb.h
以下分段介紹配置檔 nuc970_evb.h 中的各項定義.

```
#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: 影像檔所要下載位址
- CONFIG_EXT_CLK: 外部晶振頻率
- CONFIG_TMR_DIV: timer 除頻倍率
- CONFIG_SYS_HZ: timer 頻率
- CONFIG_SYS_MEMTEST_START: 記憶體測試的起始位址
- CONFIG_SYS_MEMTEST_END: 記憶體測試的結束位址
- CONFIG_NUC970_HW_CHECKSUM: 使用 SHA-1 計算 Linux 內核的 checksum (若屏蔽此定義，則採用 crc32 來計算 checksum)，必須與 mkimage 搭配使用，請參考 4.7.2 章節。
- CONFIG_CMD_TIMER: 使用timer 相關的命令

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

#define CONFIG_BOARD_EARLY_INIT_F
#define CONFIG_BOARD_LATE_INIT

#define CONFIG_NUC970_WATCHDOG
#define CONFIG_HW_WATCHDOG

#define CONFIG_DISPLAY_CPUINFO
#define CONFIG_BOOTDELAY            3
#define CONFIG_SYS_SDRAM_BASE       0
```

```
#define CONFIG_NR_DRAM_BANKS          2
#define CONFIG_SYS_INIT_SP_ADDR       0xBC008000
#define CONFIG_BAUDRATE               115200
#define CONFIG_SYS_BAUDRATE_TABLE     {115200, 57600, 38400}

#define CONFIG_NUC970_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: 使用 SPI flash
- CONFIG_SYS_USE_NANDFLASH: 使用 NAND flash
- CONFIG_ENV_IS_IN_NAND: 環境變數儲存在 NAND flash 中
- CONFIG_ENV_IS_IN_SPI_FLASH: 環境變數儲存在 NAND flash 中
- CONFIG_ENV_IS_IN_MMC: 環境變數儲存在 eMMC 中
- CONFIG_NUC970_WATCHDOG: 編譯 NUC970 watchdog timer 驅動程式
- CONFIG_HW_WATCHDOG: 打開 watchdog timer 功能 (CONFIG_NUC970_WATCHDOG 需同時打開)
- CONFIG_DISPLAY_CPUINFO: 顯示 CPU 相關資訊
- CONFIG_BOOTDELAY: 開機時的延遲秒數
- CONFIG_SYS_INIT_SP_ADDR: 系統初始化時的堆棧指針
- CONFIG_BAUDRATE: 串口波特率
- CONFIG_NUC970_EMAC0: 使用 NUC970 EMAC0

- CONFIG_NUC970_EMAC1: 使用 NUC970 EMAC1
- CONFIG_NUC970_ETH: 支援 NUC970 Ethernet
- CONFIG_NUC970_PHY_ADDR: PHY 位址
- CONFIG_CMD_NET: 支援網路相關命令
- CONFIG_ETHADDR: MAC 位址
- CONFIG_SYS_RX_ETH_BUFFER: Rx Frame Descriptors 的個數

```
/*
 * 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 伺服器的 IP 不會被改成 DHCP 伺服器的 IP
- CONFIG_CMD_PING: 使用網路的 ping 命令功能
- CONFIG_CMD_DHCP: 使用網路的DHCP 命令功能
- CONFIG_CMD_DHCP: 支持 JFFS2 命令功能

```
#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 */



#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: 開啟 NUC970 NAND 功能
- CONFIG_CMD_NAND: 使用 nand 命令功能
- CONFIG_MTD_DEVICE: 啟動 MTD 裝置
- CONFIG_MTD_PARTITIONS: 啟動 MTD 分區
- CONFIG_CMD_UBI: 啟動 UBI
- CONFIG_CMD_UBIFS: 啟動 UBIFS 文件系統

- CONFIG_CMD_MTDPARTS: MTD 分區命令
- CONFIG_RBTREE: 啟動 UBI 需要的配置
- CONFIG_LZO: 啟動 UBI 需要的配置
- MTDIDS_DEFAULT: 設定 MTD 名稱, 需要和內核中的設定一致
- MTDPARTS_DEFAULT: 分區配置
- CONFIG_CMD_NAND_YAFFS2: 啟動YAFFS2的命令
- CONFIG_YAFFS2: 啟動YAFFS2檔案系統
- CONFIG_SYS_MAX_NAND_DEVICE: 定義NAND 裝置個數
- CONFIG_SYS_NAND_BASE: 定義NAND controller base 位址
- CONFIG_ENV_OFFSET: 環境變數在 flash 中的偏移位址
- CONFIG_ENV_SIZE: 保留給環境變數的空間大小
- CONFIG_ENV_SECT_SIZE: 保留給環境變數的空間的 sector 大小
- CONFIG_ENV_RANGE: 定義環境變數的儲存範圍，範圍是 CONFIG_ENV_OFFSET 到 CONFIG_ENV_OFFSET + CONFIG_ENV_RANGE. (當遇到儲存環境變數的 block 是壞塊時，U-Boot 會將環境變數存到下一個 block)
- CONFIG_SYS_NAND_U_BOOT_OFFS: U-Boot 放在 NAND 中的偏移位址
- CONFIG_SYS_UBOOT_SIZE: U-Boot 使用的總空間 (code + data + heap)
- CONFIG_SYS_PHY_UBOOT_BASE: U-Boot 實際跑起來的位址
- CONFIG_SYS_NAND_U_BOOT_SIZE: U-Boot 影像檔大小
- CONFIG_SYS_NAND_PAGE_SIZE: NAND flash 一個 page 的大小
- CONFIG_SYS_NAND_BLOCK_SIZE: NAND flash 一個 block 的大小
- CONFIG_SYS_NAND_PAGE_COUNT: NAND flash 一個 block 有幾個 page

```
/* 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: 使用 SPI flash 的 sf 命令功能
- CONFIG_SPI_FLASH_MACRONIX: 使用 MACRONIX SPI flash
- CONFIG_SPI_FLASH_WINBOND: 使用 Winbond SPI flash
- CONFIG_SPI_FLASH_EON: 使用 EON SPI flash
- CONFIG_ENV_OFFSET: 環境變數在 flash 中的偏移位址
- CONFIG_ENV_SIZE: 保留給環境變數的空間大小

```
#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: 提示列字串
- CONFIG_SYS_LONGHELP: 顯示完整幫助選單
- CONFIG_CMDLINE_EDITING: 允許編輯命令

```
/* 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: 開啟 LCD 功能
- CONFIG_NUC970_LCD: 編譯 NUC970 驅動程式
- LCD_BPP: 輸出到 LCD 上的一個 pixel 用幾個 bit 來表示
- CONFIG_LCD_LOGO: 將 LOGO 輸出到 LCD 上
- CONFIG_LCD_INFO: 將 U-Boot 版本以及 NUC970 相關訊息輸出到 LCD 上
- CONFIG_LCD_INFO_BELOW_LOGO: 將 NUC970 相關訊息的輸出位置放在 LOGO 底下
- CONFIG_SYS_CONSOLE_IS_IN_ENV: stdin/stdout/stderr 採用環境變數的設定
- CONFIG_SYS_CONSOLE_OVERWRITE_ROUTINE: stdin/stdout/stderr 切換到 serial port

```
/* Following block is for MMC support */

#define CONFIG_NUC970_MMC

#define CONFIG_CMD_MMC

#define CONFIG_CMD_FAT

#define CONFIG_MMC

#define CONFIG_GENERIC_MMC

#define CONFIG_DOS_PARTITION

#define CONFIG_NUC970_SD_PORT0

//#define CONFIG_NUC970_SD_PORT1

//#define CONFIG_NUC970_EMMC    /* Don't enable eMMC(CONFIG_NUC970_EMMC) and
NAND(CONFIG_NAND_NUC970) at the same time! */
```

```
#ifdef CONFIG_ENV_IS_IN_MMC
#define CONFIG_SYS_MMC_ENV_DEV  2
#define CONFIG_ENV_OFFSET       0x80000
#define CONFIG_ENV_SIZE         0x10000
#define CONFIG_ENV_SECT_SIZE    512
#define CONFIG_ENV_OVERWRITE
#endif
```

- CONFIG_NUC970_MMC: 編譯 NUC970 驅動程式
- CONFIG_CMD_MMC: 支持 MMC 相關命令
- CONFIG_CMD_FAT: 支持 FAT 相關命令
- CONFIG_MMC: 支持 MMC
- CONFIG_GENERIC_MMC: 支持通用的 MMC
- CONFIG_DOS_PARTITION: 支持 DOS 分區
- CONFIG_NUC970_SD_PORT0: 支持 SD port 0
- CONFIG_NUC970_SD_PORT1: 支持 SD port 1
- CONFIG_NUC970_EMMC: 支持 eMMC
- CONFIG_SYS_MMC_ENV_DEV: 存放環境變數的 MMC 設備編號
- CONFIG_ENV_OFFSET: 環境變數存放位址
- CONFIG_ENV_SIZE: 環境變數大小
- CONFIG_ENV_SECT_SIZE: 存放環境變數的 eMMC 區塊大小

```
/* 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: 支持 USB 命令
- CONFIG_CMD_FAT: 支持 FAT 命令
- CONFIG_USB_STORAGE: 支持 USB 儲存系統
- CONFIG_USB_EHCI: 支持 USB 2.0
- CONFIG_USB_EHCI_NUC970: 支持 NUC970 芯片 USB 2.0
- CONFIG_DOS_PARTITION: 支持 DOS 分區

```
#define CONFIG_NUC970_GPIO


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


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


#endif
```

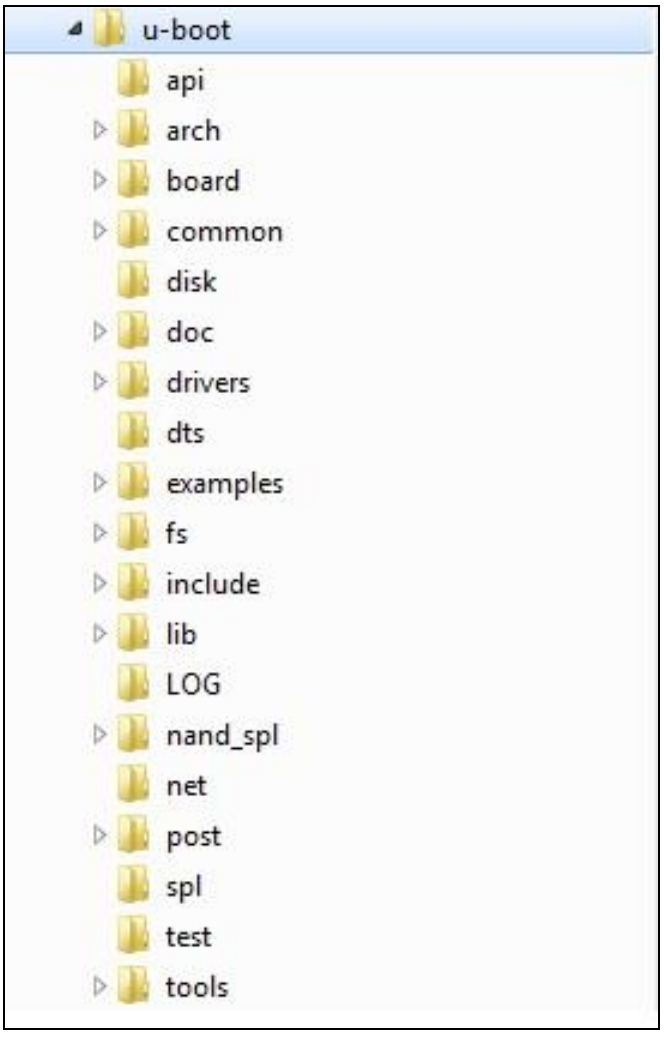
- CONFIG_NUC970_GPIO: 開啟 GPIO 功能
- CONFIG_SYS_MALLOC_LEN: 設置動態配置記憶體大小
- CONFIG_STACKSIZE: 設置堆棧大小

```
#define CONFIG_KPI_NUC970


#ifdef CONFIG_KPI_NUC970
//#define CONFIG_KPI_PA_PORT   1    // KPI select PA port
#define CONFIG_KPI_PH_PORT   1  // KPI select PH port
#define CONFIG_KPI_ROW_NUM   3
#define CONFIG_KPI_COL_NUM   3
#define CONFIG_KPI_DEBOUNCE  8  // debounce length setting: 0~13
#endif
```

- CONFIG_KPI_NUC970: 開啟 GPIO 功能
- CONFIG_KPI_PA_PORT: 選擇PORT A為按鍵來源
- CONFIG_KPI_PH_PORT: 選擇PORT H為按鍵來源
  (使用者只能選擇 CONFIG_KPI_PA_PORT 或 CONFIG_KPI_PH_PORT其中之)
- CONFIG_KPI_ROW_NUM: 設置掃描按鍵列的數目
- CONFIG_KPI_COL_NUM: 設置掃描按鍵行的數目
- CONFIG_KPI_DEBOUNCE: 設置掃描按鍵de-bounce的長度

4.2 目錄架構

U-Boot 的目錄結構如下圖



- arch: 包含CPU 相關的源代碼

- NUC970 CPU 相關的源代碼放在 arch/arm/cpu/arm926ejs/nuc900.
- board: 包含板子相關的源代碼
- NUC970 板子相關的源代碼放在 board/nuvoton/nuc970_evb.
- common: 包含 U-Boot 命令以及一些各平台共同的源代碼.
- doc: 放置各式各樣的 README 文件.
- drivers: 放置驅動程式源代碼.
- NUC970 的驅動程式源代碼也是放在 drivers 目錄下, 例如 Ethernet 驅動程式就放在 drivers/net/nuc900_eth.c
- examples: 放置一些範例. 例如 mips.lds 就是 MIPS 的鏈結腳本
- fs: 存放各種檔案文件系統. 例如: FAT, yaffs2.
- include: 存放頭文件以及配置檔. NUC970 的配置檔就放在 include/configs/nuc970_evb.h
- lib: 放置各種函式庫.
- nand_spl: 存放 NAND 開機源代碼
- net: 存放網路相關的源代碼. 例如: tftp.c, ping.c,
- tools: 存放一些工具, 例如 mkimage 就是一個產生影像檔的工具.

4.3 編譯 U-Boot

4.3.1 編譯命令

清除所有的 object code.

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

編譯 U-Boot

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

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

(make 的參數 O 是指定編譯產生的檔案要放到哪個目錄)

如果不需要產生 SPL U-Boot (NAND boot 才會用到)，編譯命令可以改成

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

# make O=../build/nuc970_uboot/ nuc970_nonand_config

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

(make 的參數 O 是指定編譯產生的檔案要放到哪個目錄)

4.3.2 編譯產生的檔案

編譯成功後會產生 Main U-Boot 和 SPL U-Boot:
Main U-Boot : 完整功能的 U-Boot

SPL U-Boot : 將 Main U-Boot 從 NAND flash 搬到 DDR 執行
SPL U-Boot 只有 NAND boot 時，才會用到；如果是 SPI boot 或 eMMC boot 只需要 Main U-Boot

Main U-Boot 和 SPL U-Boot 會分別產生在根目錄以及子目錄 nand_spl 中:
Main U-Boot 的檔案會產生在根目錄
- u-boot - Elf 執行檔 (可透過 GDB 或 IDE 下載)
- u-boot.bin - binary file (可透過 Nu-Writer 燒錄到 NAND/SPI flash、eMMC中，請參考 3.11)
- u-boot.map – 鏈結對應檔

SPL U-Boot 的檔案會產生在根目錄底下的子目錄 nand_spl中
- u-boot-spl - Elf 執行檔 (可透過 GDB 或 IDE 下載)
- u-boot-spl.bin - binary file (可透過 Nu-Writer 燒錄到 NAND flash中，請參考 3.11.3)
- u-boot-spl.map – 鏈結對應檔

4.3.3 Main U-Boot 鏈結位址

Main U-Boot 的鏈結位址是定義在 Makefile.
請找到以下片段

```
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
```

當中 RAM_TEXT 定義 U-Boot 的鏈結位址，
上面的例子，“RAM_TEXT = 0xE00000" ，U-Boot 的鏈結位址就是 0xE00000

如果是 NAND Boot，請同時修改 include/configs/nuc970_evb.h 當中的定義

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

CONFIG_SYS_PHY_UBOOT_BASE 必須和 Makefile 當中的 RAM_TEXT 定義在相同位址。

4.3.4 SPL U-Boot 鏈結位址

SPL U-Boot 的鏈結位址定義在 board/nuvoton/nuc970evb/config.mk
預設位址是 0x200，若要修改到其他位址，請找到以下片段，將 0x200 置換成新的位址。

```
ifndef CONFIG_NAND_SPL

CONFIG_SYS_TEXT_BASE = $(RAM_TEXT)

else

CONFIG_SYS_TEXT_BASE = 0x200
```

4.4 NAND AES secure boot 示範

NAND AES secure boot 需要先編譯產生 Main U-Boot 和 SPL U-Boot，然後透過 Nu-Writer 將 SPL U-Boot 做 AES 加密並燒錄到 NAND flash.

4.4.1 編譯 Main U-Boot 以及 SPL U-Boot

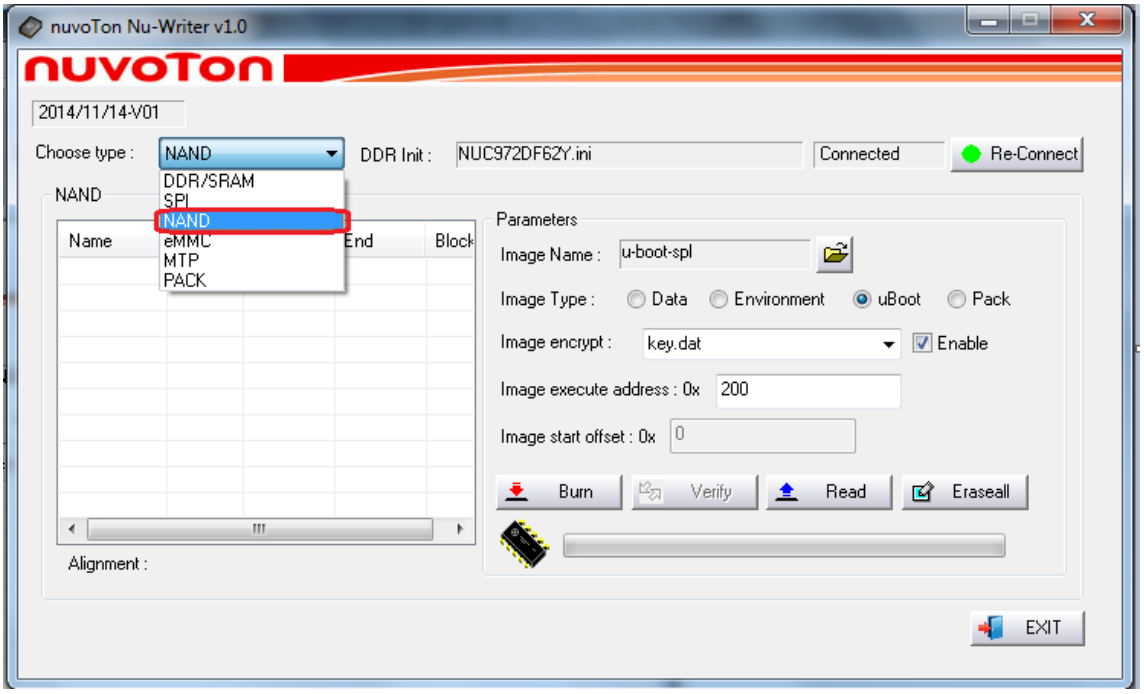
```
# make O=../build/nuc970_uboot/ distclean

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

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

(make 的參數 O 是指定編譯產生的檔案要放到哪個目錄)
編譯成功後，先找到 Main U-Boot 和 SPL U-Boot 的 binary file。

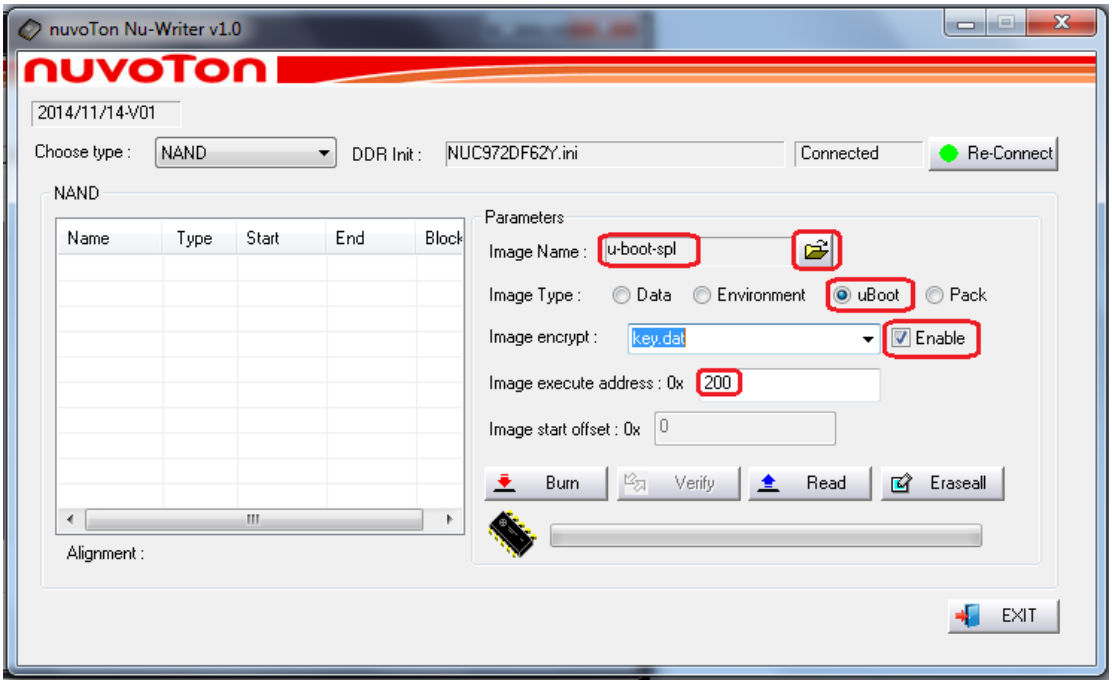
- Main U-Boot 的 binary file 會產生在根目錄下，檔名為 u-boot.bin
- SPL U-Boot 的 binary file 會產生在根目錄下的子目錄 nand_spl 中，檔名為 u-boot-spl.bin

4.4.2 燒錄 SPL U-Boot

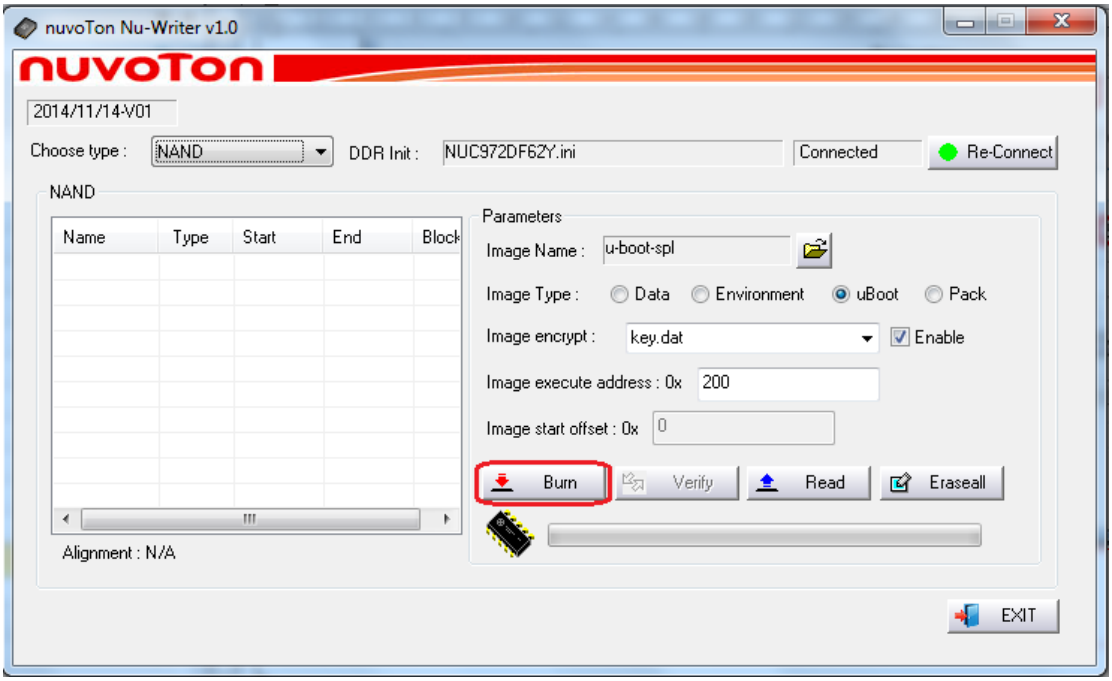
在 Choose type 處選擇 “NAND” ，如下圖。



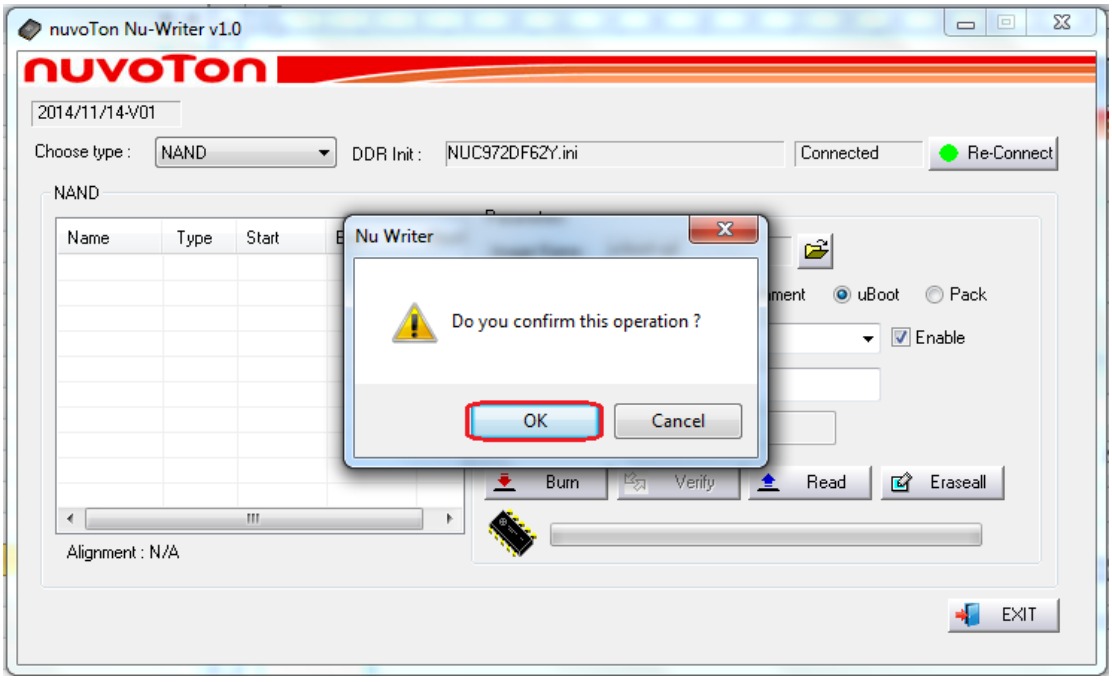
接著設定 Parameters，如下圖，
Image Name: 選取 u-boot-spl.bin，
Image Type: 選取 uBoot，
Image encrypt: 勾選 Enable
Image execute address:0x 填寫 200



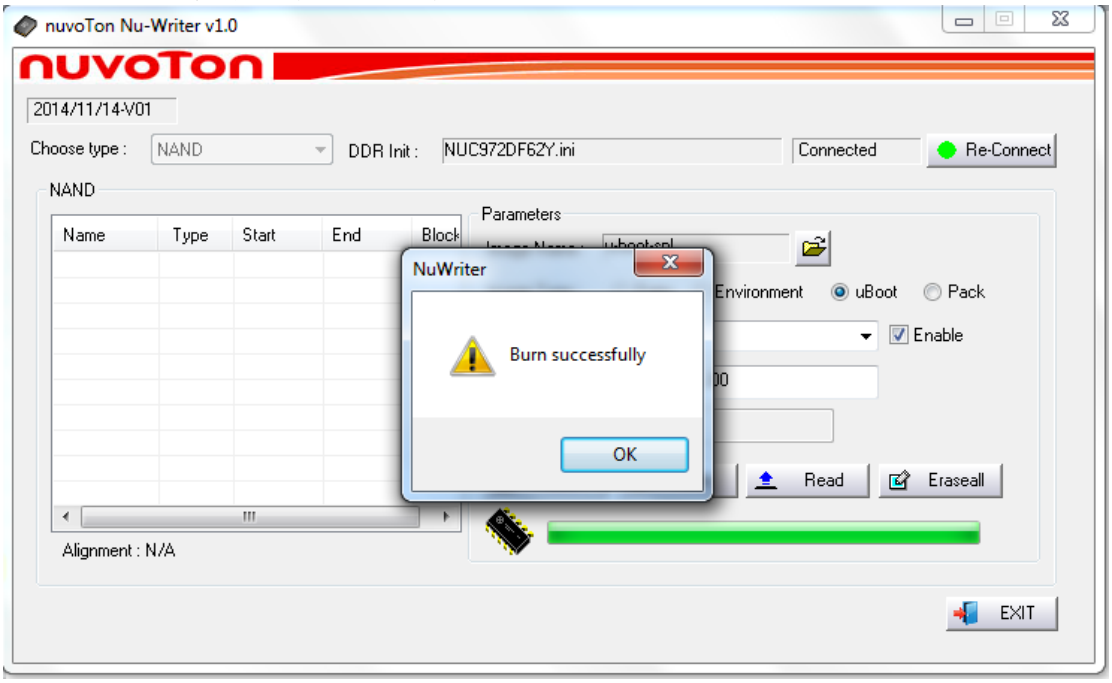
然後按 “Burn” 鍵，如下圖



出現一個對話視窗，選擇 “OK” ，如下圖，

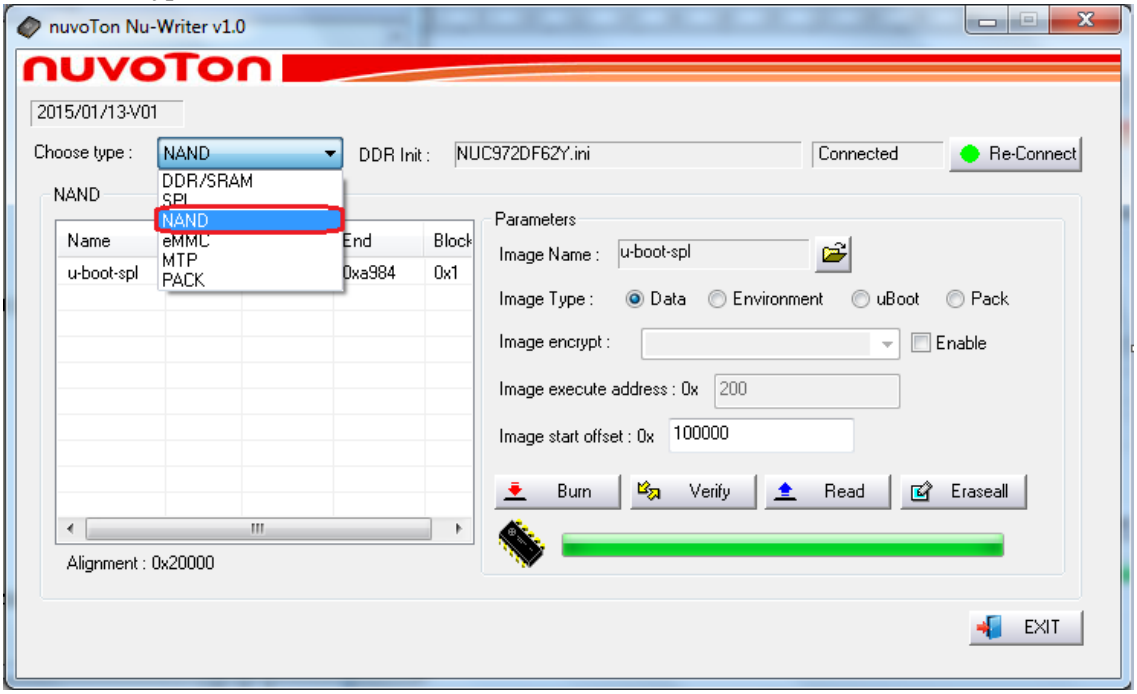


出現一個燒錄成功的視窗，如下圖，

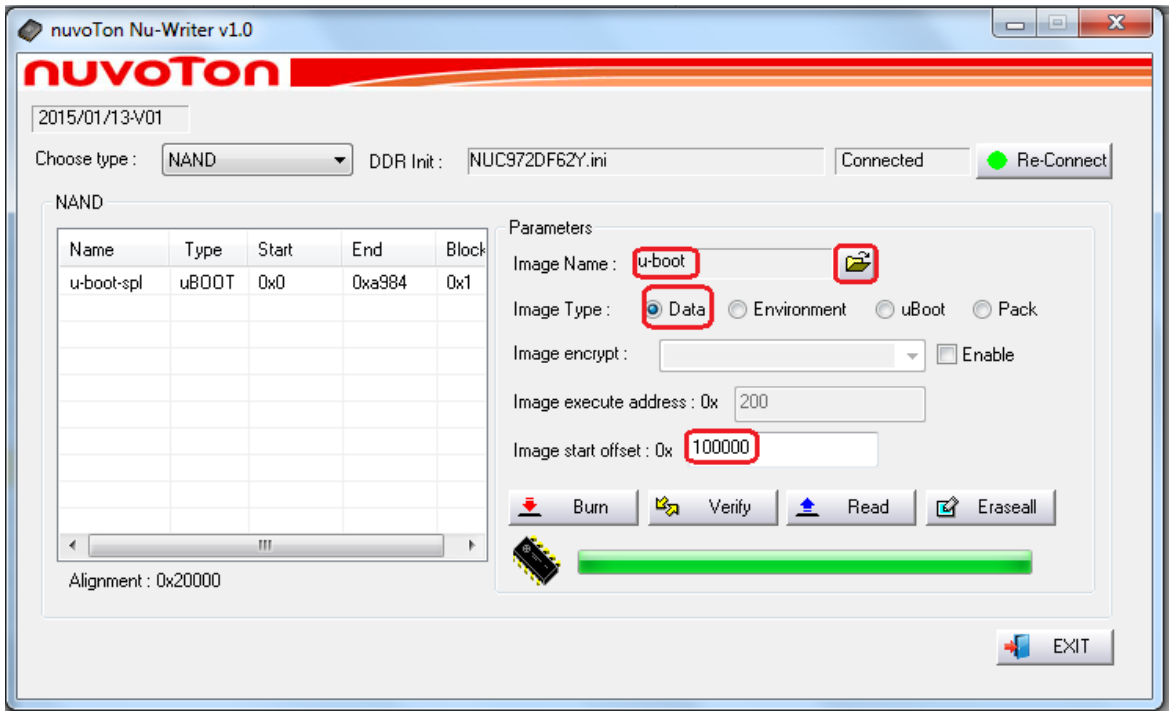


4.4.3 燒錄 Main U-Boot

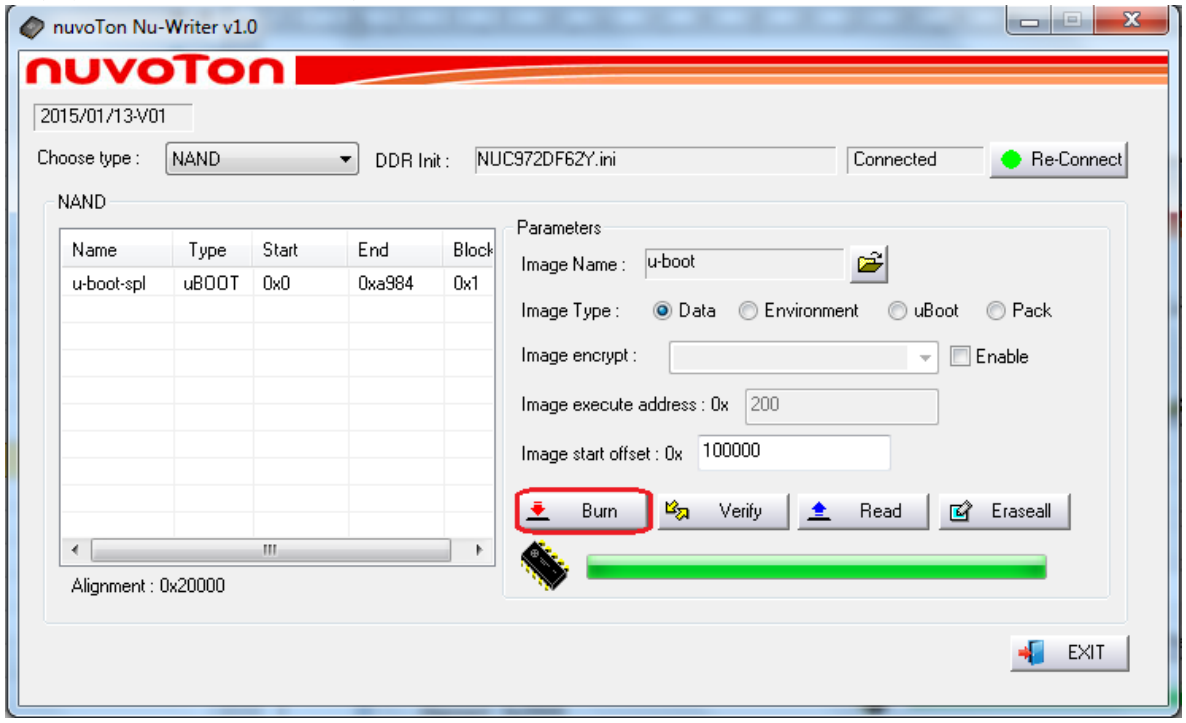
在 Choose type 處選擇 “NAND” ，如下圖。



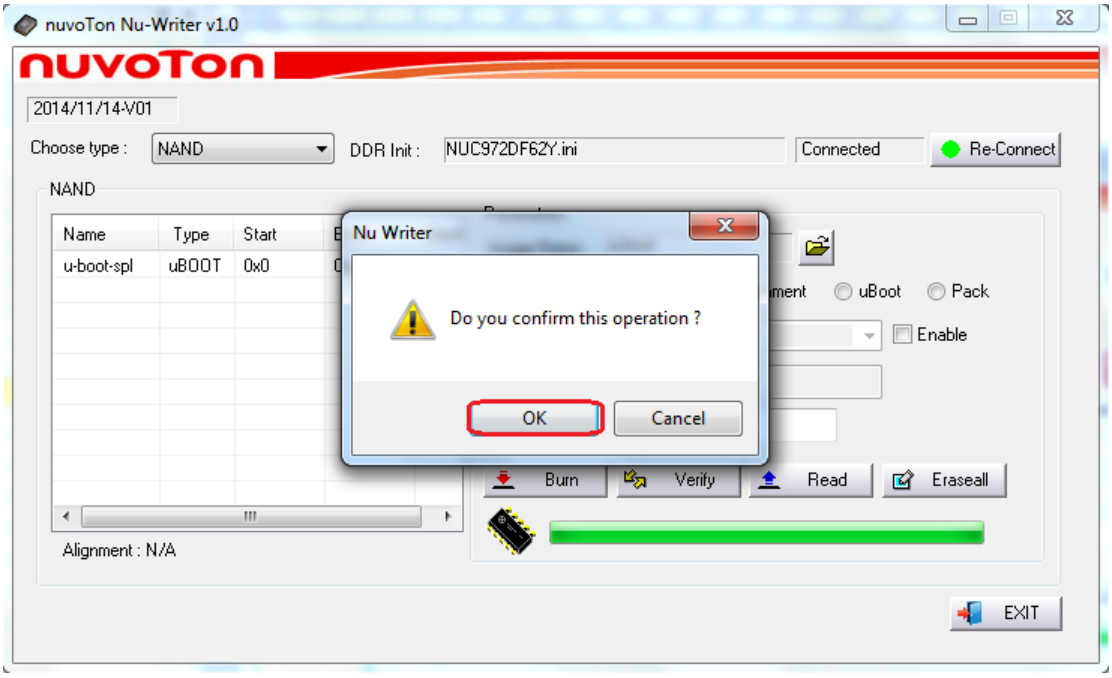
接著設定 Parameters，如下圖，
Image Name: 選取 u-boot.bin，
Image Type: 選取 Data，
Image execute address:0x 填寫 100000



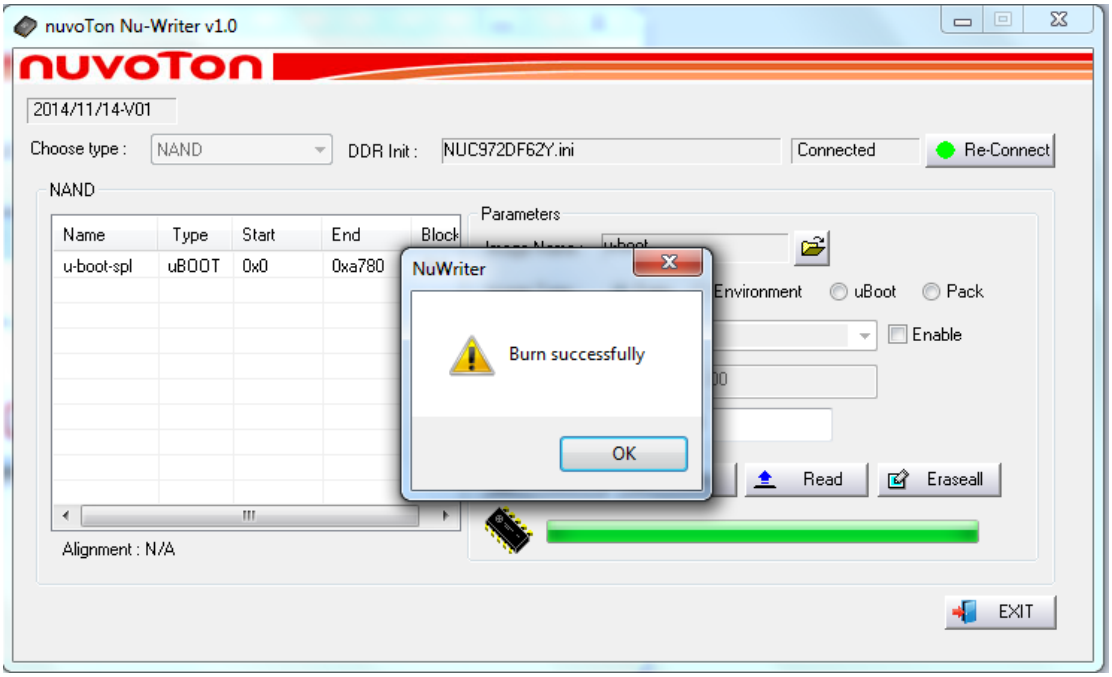
然後按 “Burn” 鍵，如下圖



出現一個對話視窗，選擇 “OK” ，如下圖，

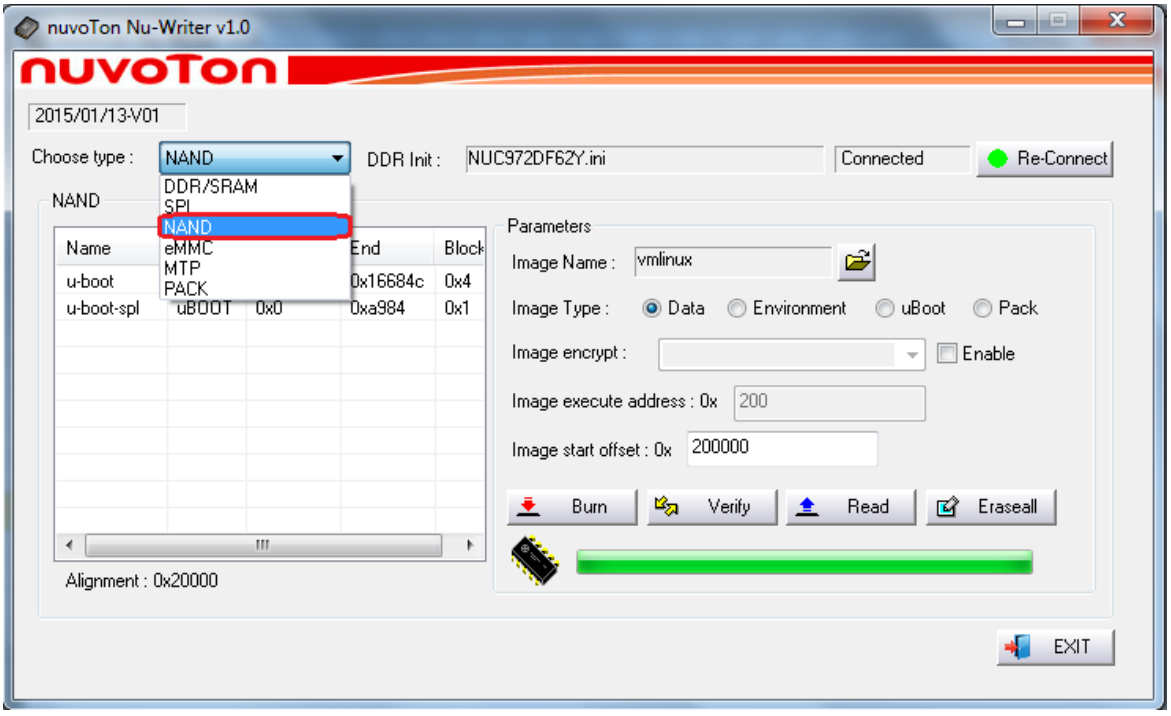


出現一個燒錄成功的視窗，如下圖，

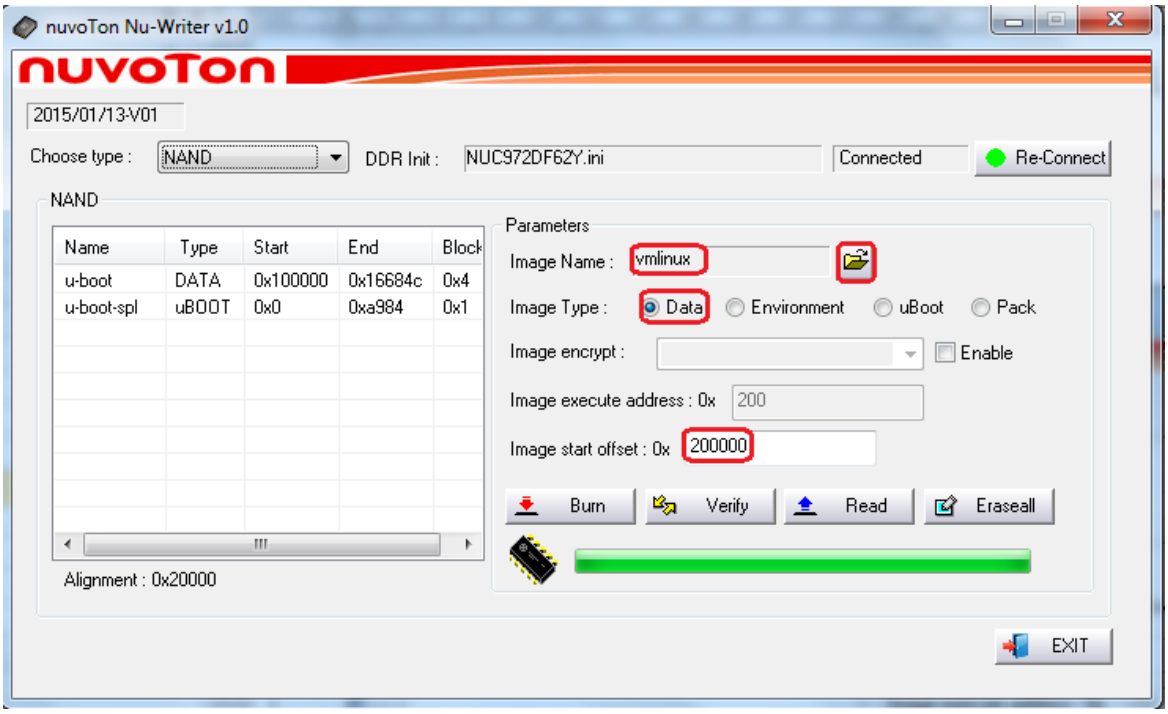


4.4.4 燒錄 Linux 內核

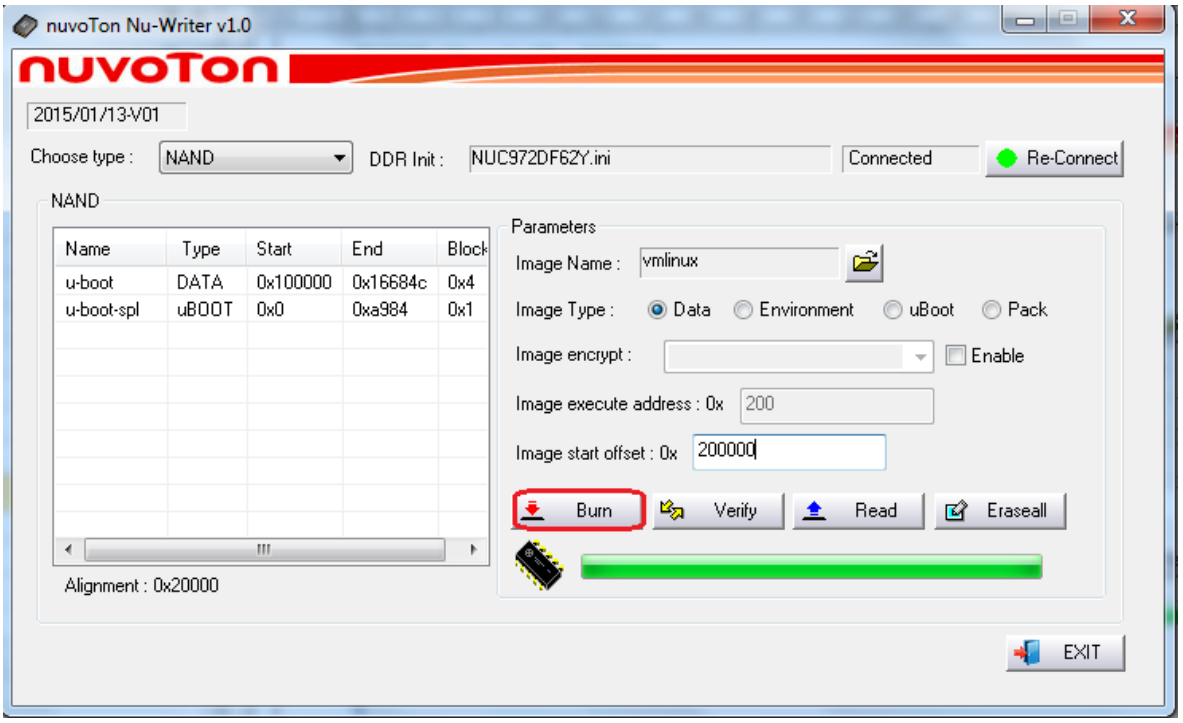
在 Choose type 處選擇 “NAND” ，如下圖。



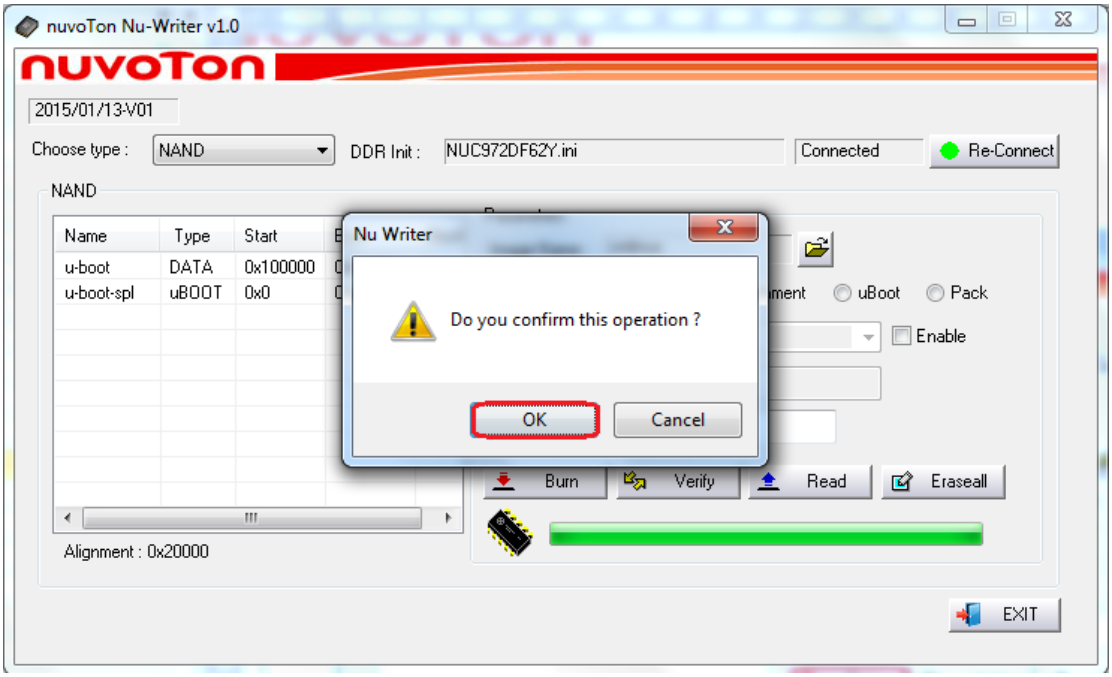
接著設定 Parameters，如下圖，
Image Name: 選取 vmlinux.ub (vmlinux.ub 的產生，請參考 4.7 章節)，
Image Type: 選取 Data，
Image execute address:0x 填寫 200000



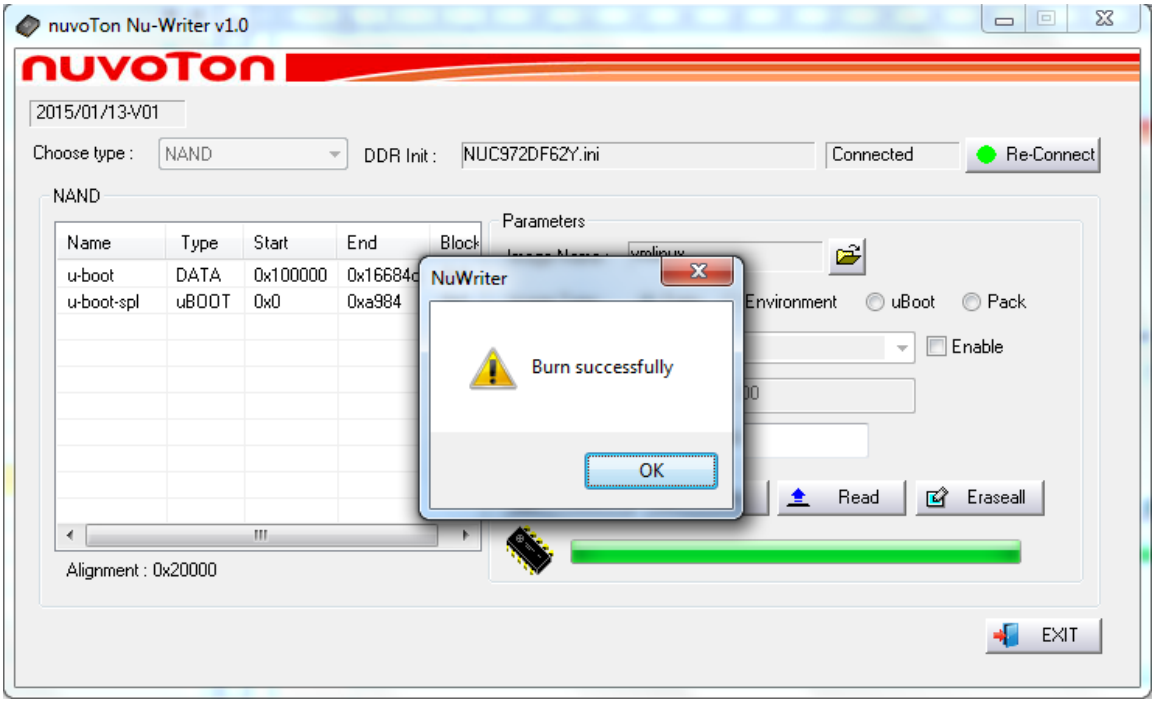
然後按 “Burn” 鍵，如下圖



出現一個對話視窗，選擇 “OK” ，如下圖，



出現一個燒錄成功的視窗，如下圖，



4.4.5 nboot 命令完成 NAND 開機

下面這個範例是用 nboot 命令將 Linux 內核影像檔從 NAND flash 偏移量 0x200000 這個位址讀取到 DDR 0x7fc0 的位址. 再透過 bootm 命令完成 Linux 內核的開機.

```
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 命令

U-boot 提供一個功能強大的命令列介面, 透過串口連接到 PC 端的終端機程式. 輸入 "help" 就會列出目前 U-Boot 支援的命令:

```
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'

…
```

大部分的命令不需要輸入完整的命令名稱, 只要命令前幾個字母和其他命令可區分即可, 例如 help 可以輸入 h 即可. 大部分的 U-Boot 命令中的參數是 16 進位 (例外: 因歷史包袱, sleep 命令的參數是 10 進位)

4.5.1 Bootm 命令

在介紹網路、NAND、SPI、USB、MMC 等相關命令之前，特別先介紹 bootm 命令。
因為 Linux 內核影像檔會儲存在網路、NAND、SPI、USB、MMC 等儲存媒介，透過這些儲存媒介相關的命令將 Linux 內核下載到 DDR 之後，再透過 bootm 命令完成 Linux 內核的開機。
因此，bootm 命令是用來啟動由 mkimage 工具產生的 Linux 內核或其他應用程式。
相較於 bootm 命令，go 命令 (4.5.2 會介紹) 是來啟動 “非” 經由 mkimage 工具產生的 Linux 內核或其他應用程式。

bootm 命令的格式如下:

```
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
```

下面的範例是假設已經將 Linux 內核下載到 DDR 0x7fc0 的位址，這時我們可以透過 bootm 命令來啟動 Linux 內核。

```
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 命令

- go: 執行應用程式

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

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

下面這個範例是執行一個已經下載到 DDR 0x100000 位址的程式。

```
U-Boot> go 0x100000
```

```
## Starting application at 0x00100000 ...



 Hello World!
```

4.5.3 網路相關命令

- ping

傳送 ICMP ECHO_REQUEST 封包給網路 host 端

```
U-Boot> help ping

ping - send ICMP ECHO_REQUEST to network host


Usage:

ping pingAddress

U-Boot>
```

在使用這個命令之前, 必須先將 IP 位址設定給環境變數 ipaddr
下面這個例子, 將環境變數 ipaddr 設為 192.168.0.101, 然後 ping 一台 IP 位址為 192.168.0.100 的 PC.

```
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

透過 TFTP 協定下載影像檔

```
U-Boot> help tftp

tftpboot - boot image via network using TFTP protocol

```

```
Usage:
tftpboot [loadAddress] [[hostIPaddr:]bootfilename]
U-Boot>
```

在使用這個命令之前, 必須先設定 IP 位址和 server IP 位址給環境變數.
下面這個範例是透過 TFTP 協定完成 Linux 內核開機. 首先, 將 NUC970 IP 位址設為 192.168.0.101, TFTP server 的 IP 位址設為 192.168.0.100. 然後透過 TFTP 協定將 Linux 內核影像檔下載到 0x7fc0, 最後以 bootm 命令完成 Linux 內核開機

```
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

透過 DHCP 協定從網路下載影像檔

```
U-Boot> help dhcp

dhcp - boot image via network using DHCP/TFTP protocol


Usage:

dhcp [loadAddress] [[hostIPaddr:]bootfilename]

U-Boot>
```

下面這個範例是透過DHCP 協定將 Linux 內核下載到 0x7fc0 這個位址. 然後再透過 bootm 命令完成 Linux 內核開機. 使用 dhcp 命令並不需要先設定 ipaddr 環境變數, 因為 DHCP server 會指定一個 IP 位址給你.

```
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

透過 BOOTP 協定從網路下載影像檔

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


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

下面這個範例是透過BOOTP 協定將 Linux 內核下載到 0x7fc0 這個位址. 然後再透過 bootm 命令完成 Linux 內核開機. 使用 dhcp 命令並不需要先設定 ipaddr 環境變數, 因為 DHCP server 會指定一個 IP 位址給你.

```
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 相關命令

- nand: NAND Sub-system

U-Boot 支援 NAND flash 相關的命令, 包括 nand info/device/erase/read/write.

命令的格式如下:

```
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>
```

下面的範例是透過 nand info/device 命令, 顯示出 Page size/OOB size/Erase size 等 NAND 裝置

的相關訊息.

```
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 可用來清除整個 NAND 裝置.

```
U-Boot> nand erase.chip


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

下面這個範例是將 Linux 內核的影像檔寫入 NAND flash. Linux 內核影像檔已事先放到 DDR 0x500000 這個位址, 大小為0x190580 bytes. 我們將把他寫到 NAND flash 偏移量 0x200000 的位址. 然後再把 Linux 內核影像檔從 NAND flash 讀回到 DDR 0x7FC0 的位址. 最後再透過 bootm 命令來完成 Linux 內核的開機.

```
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: 從 NAND裝置開機

命令格式如下:

```
U-Boot> help nboot
nboot - boot from NAND device

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

下面這個範例是用 nboot 命令將 Linux 內核影像檔從 NAND flash 偏移量 0x200000 這個位址讀取到 DDR 0x7fc0 的位址. 再透過 bootm 命令完成 Linux 內核的開機.

```
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 相關命令

U-Boot 支援 SPI flash 相關的命令, 包括 sf probe/read/write/erase/update. 命令的格式如下：

```
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>
```

要注意的一點是, 在使用 sf read/write/erase/update 之前, 必須先執行 sf probe 這個命令. sf 命令可以指定 SPI 的速度，下面這個範例是將 SPI 時鐘設為 18 MHz.

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

下面這個範例是將 Linux 內核的影像檔從 SPI flash 讀取到 DDR. 首先，透過 “sf probe” 命令設定 SPI 時鐘為 18 MHz. 然後用 “sf read” 命令將一個大小為 0x190580 位元的 Linux 內核影像檔 從 SPI flash 偏移量 0x200000 的位址讀取到 DDR 0x7FC0 的位址. 最後再透過 bootm 命令來完成 Linux 內核的開機.

```
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 記憶體命令

- md: 顯示記憶體內容.

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


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

下面這個範例是顯示位址0x10000 到 0x100ff 的記憶體內容.

```
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: 寫入記憶體

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


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

下面這個範例是將長度為4個word的0 寫到0x10000 這個位址.

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

透過 md 命令顯示記憶體位址0x10000位址的內容. 前4個 word 都是 0.

```
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: 比對記憶體.

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


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

下面這個範例是比對記憶體位址 0x8000 和 0x9000 的內容, 比對長度為64 個 word.

```
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: 記憶體讀寫測試.

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

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

下面這個範例是測試記憶體位址0xa00000 到 0xb00000, 寫入的內容是 0x5a5a5a5a, 測試次數為 0x20 (32) 次.

```
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 命令

- 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: 從 USB裝置的第 `blk#' 開始讀取 `cnt' 個block 到記憶體位址 `addr'.
- usb write: 將記憶體位址 `addr' 的內容寫到USB裝置的第 `blk#' block, 長度為 `cnt' 個block.
  下面這個範例對 USB 裝置編號 0 的第2個 block 做寫入動作, 寫入的內容是記憶體位址 0x10000 的內容, 長度為 1個 block. 然後再對USB 裝置編號 0 的第2個 block 做讀取動作, 讀取 1 個 block 到記憶體位址 0x20000. 最後用 cmp 命令來比對記憶體位址 0x10000 和 0x20000 的內容, 比對長度為 1 個 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: 從USB 裝置開機

```
U-Boot> help usb  boot

usbboot - boot from USB device


Usage:

usbboot loadAddr dev:part

U-Boot>
```

在使用usbboot 命令之前, 必須先透過 usb write 命令將Linux 內核影像檔寫到 USB 裝置. usb write 命令是以 block 為單位, 寫入的位址也是 block 編號. 而 usbboot會從 start block(sector) 開始讀取 Linux 內核影像檔, 因此, 我們必須知道 start(sector) 的編號. 這可透過

usb part 命令顯示出 USB 裝置編號 0 的分區表.

```
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>
```

由上圖可看出 start sector (block) 編號是 369 (0x171), 因此, 我們透過 usb write 命令將 Linux 內核影像檔寫到 USB 裝置編號 0 的第 # 369(0x171) 個 block. Linux 內核影像檔大小有幾個 block, 算法如下.
Linux 內核影像檔已事先透過 TFTP 或 ICE下載到記憶體位址 0x200000 的地方, 而 Linux 內核影像檔大小為 1639808 bytes, 1639808/512 = 3202.75, 因此, 總共需要3203 (0xc83) 個block 來存放Linux 內核影像檔.

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


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

現在, Linux 內核影像檔已存放在 USB 裝置編號 0 的第 #369(0x171) block, 因此, 我們可以透過 usbboot 命令將 Linux 內核影像檔從 USB 裝置中讀取到 DDR. 最後再透過 bootm 命令完成 Linux 內核開機.

```
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 ...
```

除了以 block 為單位的存取方式, U-Boot 還支援 fatls 和 fatload 命令, 可以透過文件系統 (file system) 來存取 USB 裝置中的檔案. 下面這個範例用 fatls 命令來列出 USB 裝置中有那些檔案, 在透過 fatload 命令將檔案從USB 裝置中讀取到 DDR, 最後再以 bootm 命令完成 Linux 內核開機.

```
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 環境變數相關的命令

- setenv: 設置環境變數

```
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>
```

下面這個範例是將環境變數 ipaddr 設置為192.168.0.101
然後使用 echo 命令顯示出 ipaddr 的設置.

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

```
192.168.0.101

U-Boot>
```

- saveenv: 將環境變數儲存到 flash 中.

```
U-Boot> help saveenv

saveenv - save environment variables to persistent storage


Usage:

saveenv

U-Boot>
```

- env: 環境變數處理命令

```
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 解密命令

除了原本U-Boot 支援的命令, NUC970 U-Boot 新增了一個解密的命令, 但目前只支援 AES 解密. 命令格式如下：

```
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!!!
```

例如, 要將一個加密過的 Linux 內核, 從記憶體位址 0x800000 解密到記憶體位址 0x7FC0, 大小為 0x190580, 命令如下

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

若要燒錄 AES key 到 MTP，可以透過 decrypt program 命令，同時並指定要不要進入 secure boot mode.
例如，只燒錄 AES key 到 MTP，但 ”不” 開啟 secure boot，命令如下:

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

若要燒錄 AES key 到 MTP，同時開啟 secure boot，命令如下:

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

注意的是，要開啟 secure boot 之前，必須確認 U-Boot 透過 Nu-Writer 燒錄時，也是使用同一把 AES key 加密，否則板子 reset 或重新上電之後，系統會鎖住，無法開機，務必要小心。

4.5.10 MMC 命令

- mmc : MMC sub-system

U-Boot 支持 MMC 相關命令，包括 read/write/erase/list/dev 等。
命令格式如下:

```
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 列出所有的 mmc device

```
U-Boot> mmc list

mmc: 0

 mmc: 1

 mmc: 2

U-Boot>
```

NUC970 支持的 mmc device 包括 SD port 0, SD port 1 和 eMMC.
用戶可以根據平台上實際支持的 mmc device 來修改配置檔 (nuc970_evb.h) 當中以下三個定義

```
#define CONFIG_NUC970_SD_PORT0

#define CONFIG_NUC970_SD_PORT1

#define CONFIG_NUC970_EMMC
```

默認的設定是打開 SD port 0 和 SD port 1，eMMC 因不能和 NAND 同時使用，默認是關掉的。
如果 SD port 0, SD port 1 和 eMMC 都打開，mmc device 編號如下:
device 編號 0 是 SD port 0
device 編號 1 是 SD port 1
device 編號 2 是 eMMC

假設用戶的平台只支持 SD port 0 和 eMMC (不支持 SD port 1)，必須修改配置檔 (nuc970_evb.h)，關掉 CONFIG_NUC970_SD_PORT1 的定義。
此時用命令 mmc list 看到的結果如下:

```
U-Boot> mmc list

mmc: 0

mmc: 1

U-Boot>
```

device 編號 0 是 SD port 0
device 編號 1 是 eMMC

假設用戶的平台只支持 eMMC (不支持 SD port 0 和SD port 1)，必須修改配置檔 (nuc970_evb.h)，關掉 CONFIG_NUC970_SD_PORT0 和CONFIG_NUC970_SD_PORT1 的定義。
此時用命令 mmc list 看到的結果如下:

```
U-Boot> mmc list

mmc: 0

U-Boot>
```

device 編號 0 是 eMMC

以下的範例是假設用戶將 SD port 0, SD port 1 和 eMMC 功能都打開。

下面的範例是透過 mmc dev 設定當前的 device 為 device 1 (SD port 1)，再透過 mmclist 命令顯示 SD 卡相關訊息。

```
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>
```

下面的範例是透過 mmc dev 設定當前的 device 為 device 0 (SD port 0)，
mmc erase 來抹除 SD 卡的 block 0x30 和 0x31, 然後從 DDR 0x8000 的位址拷貝資料到 SD 卡的 block 0x30 和 0x31, 之後再讀取 SD 卡的 block 0x30 和 0x31 到 DDR 0x500000, 最後比對 DDR 0x8000 和 0x500000 的資料來確認讀寫 SD 卡的正確性。

```
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>
```

下面的範例是透過 mmc dev 設定當前的 device 為 device 2 (eMMC)，
mmc erase 來抹除 eMMC 卡的 block 1024 到 2047, 然後從 DDR 0x8000 的位址拷貝資料到 eMMC 卡的 block 1024 到 2047, 之後再讀取 eMMC 卡的 block 1024 到 2047 到 DDR 0x500000, 最後比對 DDR 0x8000 和 0x500000 的資料來確認讀寫 eMMC 卡的正確性。

```
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>
```

除了透過 mmc 命令存取 SD/eMMC 卡之外，也可以透過 fatls 和 fatload 命令存取 SD/eMMC 卡中的檔案。
下面的範例是先以 fatls 命令列出 SD port 0 中的檔案，然後透過 flatload 命令將 SD 卡中的 Linux kernel 影像檔 (vmlinux.ub) 讀取到 DDR 0x7fc0，再經由 bootm 命令完成 Linux kernel 開機。

```
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 命令

- mtdparts : define flash/nand partitions

U-Boot 支持 MTD partition 相關命令，包括 add/del/list 等。
命令格式如下:

```
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 設定MTD分區預設值, 預設值定義在 nuc970_evb.h中. 這個設定是將MTD分區ID設為nand0, 預設三個分區u-boot, kernel 和user.
第一分區: 名稱為u-boot, 起始位置為0x0, 大小為0x200000.
第二分區: 名稱為kernel, 起始位置為0x200000, 大小為0x1400000.
第三分區: 名稱為user, 起始位置為0x1600000, 大小為剩餘空間.

```
#define MTDIDS_DEFAULT “nand0=nand0”

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

mtdparts 列出所有的 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 命令

- ubi : ubi commands

U-Boot 支持 UBI 相關命令，包括 info/create/read/write 等。
命令格式如下:

```
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 顯示或設置目前的分區

```
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 顯示容積及 ubi 信息

```
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 安裝ubifs文件系統

```
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 列出ubifs文件系統中的目錄或文件

```
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 卸載ubifs文件系統

```
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 命令

- yaffs : yaffs commands

U-Boot 支持 YAFFS 相關命令，包括 mount/list/mkdir/rmdir/rd/wr 等。
命令格式如下:

```
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 配置YAFFS掛載點

```
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 顯示YAFFS掛載點

```
U-Boot> ydevls

nand            0 0x000b0 0x003ff using inband tags, not mounted
```

ymount 掛載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 顯示YAFFS文件系統內容, 一個掛載點就是一個目錄區, 上述的範例 nand 就是一個目錄區

```
U-Boot> yls

Bad arguments: yls [-l] dir

U-Boot> yls -l nand

lost+found                          2032     2 directory
```

ymkdir 建立一個目錄

```
U-Boot> ymkdir nand/test

U-Boot> yls -l nand

test                                2032   257 directory

lost+found                          2032     2 directory
```

yrmdir 刪除一個目錄

```
U-Boot> yrmdir nand/test

U-Boot> yls -l nand

lost+found                          2032     2 directory
```

ywr / ywrm 寫一個檔案 / 將一塊memory存成檔案

```
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 讀一個檔案 / 將檔案讀到 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 刪除一個檔案

```
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 卸載YAFFS

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

4.6 環境變數

4.6.1 環境變數的配置

環境變數可以存放在 NAND flash、SPI flash 或 eMMC，可以透過修改配置檔 (nuc970_evb.h) 中以下三個定義:

- CONFIG_ENV_IS_IN_NAND: 將環境變數儲存在 NAND flash
- CONFIG_ENV_IS_IN_SPI_FLASH: 將環境變數儲存在 SPI flash
- CONFIG_ENV_IS_IN_MMC: 將環境變數儲存在 eMMC

注意的是，三者只能定義其中的一個。

環境變數儲存在 flash 的偏移量和保留給環境變數的空間大小則由配置檔 (nuc970_evb.h) 中以下兩個定義的數值來決定:

- CONFIG_ENV_OFFSET: 環境變數儲存在flash的偏移量。
- CONFIG_ENV_SIZE: 保留給環境變數的空間大小。
- 如果想要將環境變數存放到其他位址或調整空間大小，修改以上兩個定義的數值即可。

4.6.2 預設的環境變數

當 flash 中不存在環境變數時，U-Boot 會帶出預設的一組環境變數，以下是 U-Boot 預設的環境變數。

- baudrate

Console baudrate , 單位是 bps. baudrate 數值來自於 nuc970_evb.h 中的 CONFIG_BAUDRATE

- bootdelay

這是開機延遲的秒數. 在這段延遲時間內, 按下任何按鍵將會阻止 U-Boot 去執行 bootcmd 中的命令腳本. bootdelay 數值來自於 nuc970_evb.h 中的 CONFIG_BOOTDELAY

- ethact

控制目前哪一個 interface 的狀態為 active, 因為 nuc970 ethernet 驅動程式設定 device name 為 emac, 因此 ethact 只能設為 emac

- ethaddr

Ethernet mac address. ethaddr 數值來自於 nuc970_evb.h 中的 CONFIG_ETHADDR

- stderr

  設定 stderr，預設值是 serial
- stdin

  設定 stdin，預設值是 serial
- stdout

  設定 stdout，預設值是 serial

4.6.3 命令腳本

下列是和命令腳本相關的環境變數

- bootcmd

每當 U-Boot 開機後, U-Boot 會自動地執行bootcmd 中的命令腳本.
下面這個範例是將bootcmd 的命令腳本設為從SPI flash 偏移量0x200000 的地方讀取 Linux 內核影像檔到 DDR 0x7fc0 的位址, 並完成 Linux 內核開機. 最後, 記得將環境變數儲存到 SPI 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

這個參數將會傳遞給內核系統, 內核和文件系統的不同就會有不同的設置方法.
下面這個範例是將bootargs 的NAND MTD層分區傳遞至內核, 詳細的分區設定請參考4.5.10 MTD命令. 最後, 記得將環境變數儲存到 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>
```

注意:如果 U-boot 有設定環境變數, 那內核裡必須勾選 "Command line partition table parsing", 如此一來, 參數才會傳入內核裡.

```
Device Drivers  --->

  -*- Memory Technology Device (MTD) support  --->

        <*>   Command line partition table parsing
```

4.6.4 新增的環境變數

除了 U-Boot 支持的環境變數之外，NUC970 U-Boot 新增自行定義的環境變數

- spimode: 定義 SPI 傳送模式是one-bit 或 Quad mode. 設置 spimode 的命令格式如下：

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

參數 mode 數值可以是 1 或 4.
1: one-bit mode
4: Quad mode
例如, 要將 SPI 設為 Quad mode

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

如果你的 SPI flash 並不支援 Quad mode, 你可以將 spimode 設為 one-bit mode

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

記得將環境變數儲存到 flash 中.

```
U-Boot> saveenv
```

- watchdog: 定義 watchdog timer 功能是否打開. 設置 watchdog 的命令格式如下：

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

參數 mode 可以是 on 或 off.
on: watchdog timer 功能打開
off: watchdog timer 功能關掉
例如, 要將 watchdog 功能關掉

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

如果要重新將 watchdog 功能打開

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

記得將環境變數儲存到 flash 中.

```
U-Boot> saveenv
```

4.7 mkimage 工具

U-Boot 支援一些不同的影像檔格式, 可供下載、儲存到 flash、執行. 這些影像檔型別包括:

- Linux 內核
- 腳本文件

- Binaries
- RAM disk 影像檔

這些影像檔的副檔名通常都是 ".ub".

4.7.1 使用 mkimage 產生影像檔

mkimage 工具放在 tools/mkimage, 下面的範例是將 ARM Linux 內核 (970image) 透過 mkimage 打包成一個影像檔 (970image.ub). Linux 內核下載位址是 0x7fc0，執行位址是0x8000.

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

Image Name:

Created:      Fri Aug  8 14:38:39 2014

Image Type:   ARM Linux Kernel Image (uncompressed)

Data Size:    1639744 Bytes = 1601.31 kB = 1.56 MB

Load Address: 00007FC0

Entry Point:  00008000
```

-A: 設置 CPU 架構為 arm
-O: 設置operating system 為 linux
-T: 設置影像檔型別為內核
-a: 設置下載位址為 0x7fc0
-e: 設置程式進入點為只為 0x8000
-d: 設置影像檔的來源為 970image

4.7.2 Checksum 計算方式 (SHA-1 或 crc32)

mkimage tool 會計算影像檔的 checksum，並將此數值存放在影像檔頭。
透過 bootm 開機時，bootm 會計算影像檔的 checksum，並和影像檔頭的 checksum 做比較。如果兩邊的 checksum 數值不同，就會出現 “Verifying Checksum ... Bad Data CRC” 錯誤訊息，無法完成開機；如果兩邊 checksum 數值相同，則會出現 “Verifying Checksum ... OK” 的訊息，然後繼續完成開機。
原本 mkimage tool 計算影像檔的 checksum 方式是採用 crc32，是透過軟體運算，比較費時； NUC970 新增透過 SHA-1 來計算影像檔的 checksum，因為是採用硬體運算，所以可以加快開機速度。

NUC970 的 mkimage tool 新增加了一個參數 “-S” 來指定計算 Linux 內核 checksum 的計算方式。
原本 mkimage tool 計算影像檔的 checksum 方式是採用 crc32， NUC970 提供另一個選項，SHA-1，下面的範例就是採用 SHA-1 計算 Linux 內核的 checksum. 加上參數 “-S sha1”，請務必記得將 nuc970_evb.h 中的 CONFIG_NUC970_HW_CHECKSUM 選項打開。

```
u-boot/tools# ./mkimage -A arm -O linux  -T kernel -S sha1 -a 0x7fc0 -e
0x8000 -d 970image 970image.ub

Image Name:

Created:      Fri Aug  8 14:39:47 2014

Image Type:   ARM Linux Kernel Image (uncompressed)

Data Size:    1639744 Bytes = 1601.31 kB = 1.56 MB

Load Address: 00007FC0

Entry Point:  00008000
```

-A: 設置 CPU 架構為 arm
-O: 設置operating system 為 linux
-T: 設置影像檔型別為內核
-S: 設置計算內核 checksum 的方式為 sha1
-a: 設置下載位址為 0x7fc0
-e: 設置程式進入點為只為 0x8000
-d: 設置影像檔的來源為 970image

如果選擇使用 crc32 來計算內核的checksum，範例如下：加上參數 “-S crc32”，同時請記得關掉 nuc970_evb.h 中的 CONFIG_NUC970_HW_CHECKSUM 選項。

```
u-boot/tools# ./mkimage -A arm -O linux  -T kernel -S crc32 -a 0x7fc0 -e
0x8000 -d 970image 970image.ub

Image Name:

Created:      Fri Aug  8 14:39:47 2014

Image Type:   ARM Linux Kernel Image (uncompressed)

Data Size:    1639744 Bytes = 1601.31 kB = 1.56 MB

Load Address: 00007FC0
```

```
Entry Point:  00008000
```

-A: 設置 CPU 架構為 arm
-O: 設置operating system 為 linux
-T: 設置影像檔型別為內核
-S: 設置計算內核 checksum 的方式為 crc32
-a: 設置下載位址為 0x7fc0
-e: 設置程式進入點為只為 0x8000
-d: 設置影像檔的來源為 970image

4.7.3 AES 加密功能

另外，NUC970 的 mkimage 工具新增 AES 加密的功能，增加了兩個新的參數,
-E AES, 對影像檔進行加密； 以及 －K， 指定 AES 加密使用的 key 存在哪一個檔案。下面這個範例是對一個 ARM Linux 內核影像檔 (970image) 進行打包及加密動作，AES 加密的 key 放在 key.dat 中。 Linux 內核下載位址是 0x7fc0，執行位址是 0x8000.

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

Image Name:

Created:      Fri Aug  8 14:39:47 2014

Image Type:   ARM Linux Kernel Image (uncompressed)

Data Size:    1639744 Bytes = 1601.31 kB = 1.56 MB

Load Address: 00007FC0

Entry Point:  00008000
```

-A: 設置 CPU 架構為 arm
-O: 設置operating system 為 linux
-T: 設置影像檔型別為內核
-a: 設置下載位址為 0x7fc0
-e: 設置程式進入點為只為 0x8000
-d: 設置影像檔的來源為 970image
-E: 設置加密型別為AES
-K: 指定 AES 加密使用的 key 所存放的檔案

和 mkimage 同目錄 (tools/) 下有一個名叫 key.dat 的檔案，這個檔案就是存放 AES 加密所使用的 key。使用者可以編輯 key.dat 來修改 key

key.dat 的內容如下:

```
0x34125243

0xc41a9087

0xa14cae80

0xc4c38790

0x672ca187

0xd4f785a1

0x562fa1c5

0x78561198
```

每一列有 4 的 bytes，一共有 8 列。
每一列的開頭是 0x，請不要修改，若要修改 key，直接修改 0x 後面的數字即可。
這個檔案請直接在 Linux 開發環境下修改， 如果在 Windows 下編輯 key.dat 再拷貝到 Linux 底下時，請用 dos2unix 工具轉換 key.dat 的格式，否則加密時會讀錯 key。

```
u-boot/tools$ dos2unix key.dat

dos2unix: converting file key.dat to Unix format ...
```

4.8 安全性問題

4.8.1 加密

為了保護影像檔 (例如: Linux kernel) 的安全性，避免被人竊取影像檔的內容，我們透過 mkimage 工具對影像檔進行加密保護，下面是對 Linux kernel 影像檔做 AES 加密的範例:

```
u-boot# ./mkimage -A arm -O linux  -T kernel -a 0x8000 -e 0x8000 -E AES -K
key.dat -d 970image 970image.ub
```

4.8.2 解密

我們也可以在命令列中透過 decrypt 這個命令對影像進行解密，下面這個範例是
將一個加密過的 Linux kernel， 從記憶體位址 0x200000 解密到記憶體位址 0x400000，大小為 0x190580，命令如下

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

4.8.3 風險

將影像解密後，再透過命令列中的 md 這個命令，顯示出記憶體內容，如此影像檔的內容就完全攤在陽光下。

針對這個安全性的問題，解決辦法如下：

- 移除 md 命令的支持。

修改 include/config_cmd_default.h
將 #define CONFIG_CMD_MEMORY 這行注釋掉。

4.9 Watchdog timer

4.9.1 Watchdog timer 配置

若要將 NUC970 的 watchdog timer 功能打開，請將 nuc970_evb.h 當中以下兩個定義打開。

```
#define CONFIG_NUC970_WATCHDOG

#define CONFIG_HW_WATCHDOG
```

反之，若要將 NUC970 的 watchdog timer 功能關掉，請將上面兩個定義注釋掉。

4.9.2 Watchdog timer 環境變數

當 NUC970 的配置 watchdog timer 功能是打開時，用戶可以將環境變數 “watchdog” 設為 “off” 或 “0” ，即可將 watchdog timer 功能關掉，而不必修改配置檔及重新編譯。

```
U-Boot> set watchdog off

U-Boot> saveenv
```

用戶若要重新將 watchdog timer 功能打開，將環境變數 “watchdog” 設為 “on” 即可。

```
U-Boot> set watchdog on

U-Boot> saveenv
```

注意的是，修改環境變數 “watchdog” 後，記得用 saveenv 命令將環境變數 “watchdog” 儲存到 flash。
當 NUC970 的配置 watchdog timer 功能是關掉時，修改環境變數“watchdog”是沒有意義的。

4.9.3 Watchdog timer 的時間長度

當 Watchdog timer 功能打開後，系統閒置超過 14 秒之後，Watchdog timer 會重置系統，U-Boot 重新開機；每當用戶在 U-Boot 命令列下完成一個命令 (輸入 Enter 鍵) 之後，會重新計數 14 秒。

4.10 U-Boot LCD

4.10.1 NUC970 LCD 顯示內容

U-Boot 開機過程當中，會將 LOGO 及 U_Boot 相關訊息顯示在 LCD 面板上。 NUC970 U-Boot 在 LCD 面板顯示的內容如下：

4.10.2 如何置換 LOGO

LOGO 的部分是顯示 Nuvoton LOGO，如果要置換為其他公司的 LOGO，例如公司名稱為 abc，LOGO 圖檔為 abc.bmp，步驟如下:

- 將 abc.bmp 檔案放置到 tools/logos 目錄下
- 修改 tools/Makefile

先找到以下三行，

```
ifeq ($(VENDOR),nuvoton)
LOGO_BMP= logos/nuvoton.bmp
endif
```

在這三行之後加上以下三行

```
ifeq ($(VENDOR),abc)
LOGO_BMP= logos/abc.bmp
endif
```

4.11 GPIO

U-Boot GPIO 最常見的應用是拿來點亮 LED。
NUC970 U-Boot 提供設定 GPIO 的功能，用戶可以透過 GPIO 驅動程式介面來存取 GPIO。

4.11.1 NUC970 GPIO

NUC970 的 GPIO port 包括 port A ~ port J，每個 port 有 16 根 pin 腳。
但 GPIO port C 的 pin 15 和 GPIO port J 的 pin 5~15 是保留的，請勿使用。
NUC970 U-Boot 對每一根 pin 腳定義一個 GPIO 編號，例如 GPIO port A 的 pin 0 編號就是 GPIO_PA0，GPIO port B 的 pin 2 編號就是 GPIO_PB2，以此類推。
用戶在使用 NUC970 GPIO 驅動程式時，都必須傳入 GPIO 編號。

4.11.2 GPIO 驅動程式介面

NUC970 提供以下的 GPIO APIs

```
int gpio_request(unsigned gpio, const char *label);

int gpio_direction_input(unsigned gpio);

int gpio_direction_output(unsigned gpio, int value);

int gpio_get_value(unsigned gpio);

int gpio_set_value(unsigned gpio, int value);
```

每個 API 的第一個參數都是 GPIO 編號。

- `gpio_request`

確認GPIO是否正在使用，第二個參數用不到，傳 0 進去即可。
如果指定的 GPIO pin 已被切換到其他功能 (非 GPIO)，會出現錯誤訊息。
例如，當我們想要使用 GPIO port D0 時，做了以下調用:

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

假設 port D0 已被切換到其他功能，會出現下列錯誤訊息。

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

- `gpio_direction_input`

將 GPIO pin 設為輸入模式。

- `gpio_direction_output`

將 GPIO pin 設為輸出模式，並指定輸出值。

- `gpio_get_value`

讀取 GPIO pin 的數值。

- `gpio_set_value`

設定 GPIO pin 的輸出值。

4.11.3 使用範例

下面這個範例是將 GPIO port G0 ~ port G5 的數值設為 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);
```

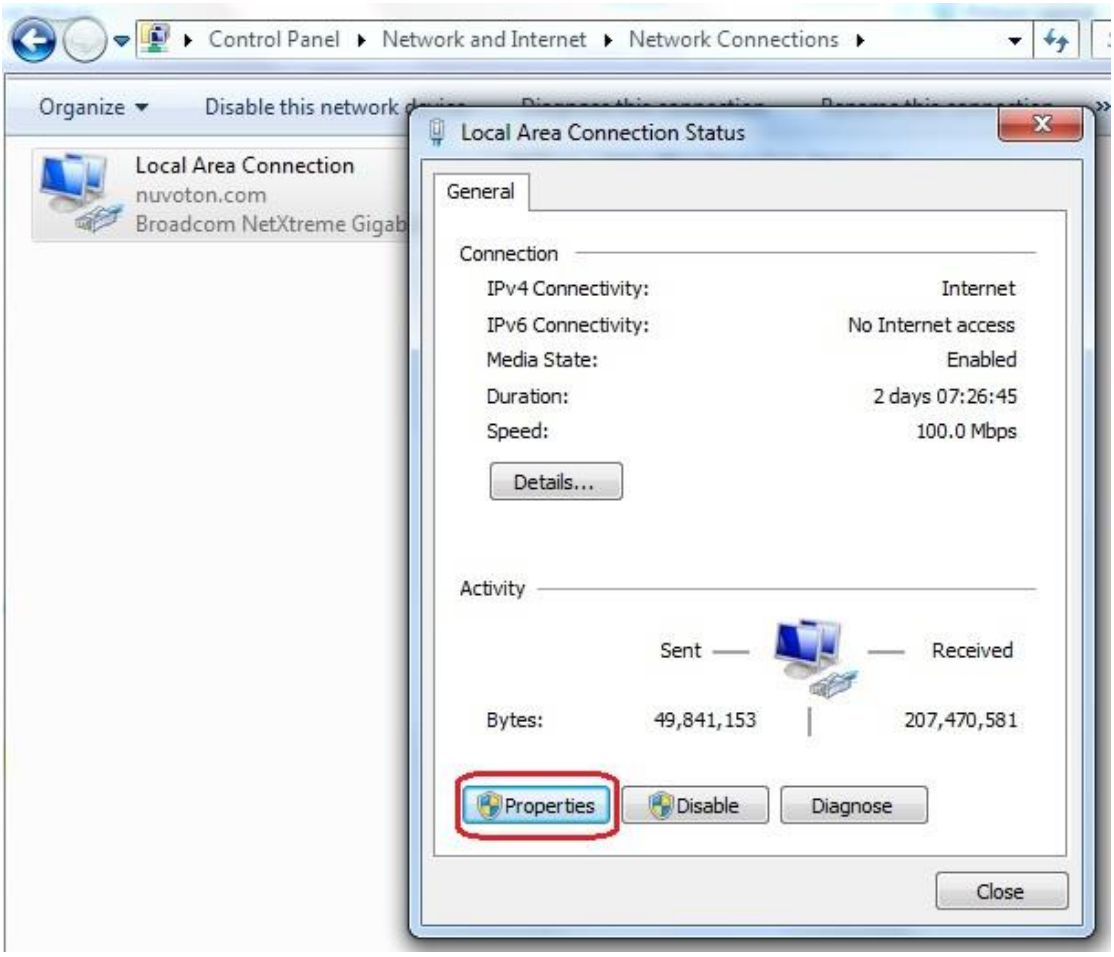
4.12 網路測試環境

我們需要一部 PC, 該PC 有固定 IP 位址並且安裝TFTP/DHCP伺服器應用程式.

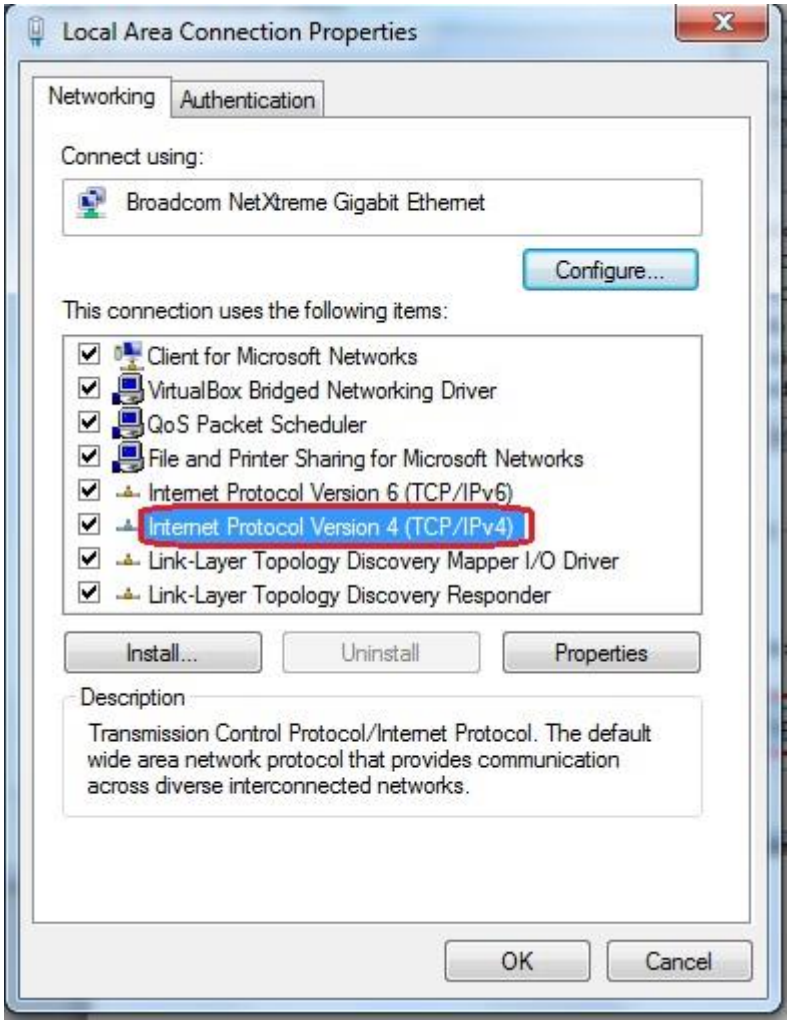
4.12.1 設置固定 IP 位址

1. 點選Local Area Connection (Control Panel -> Network and Internet -> Network Connections

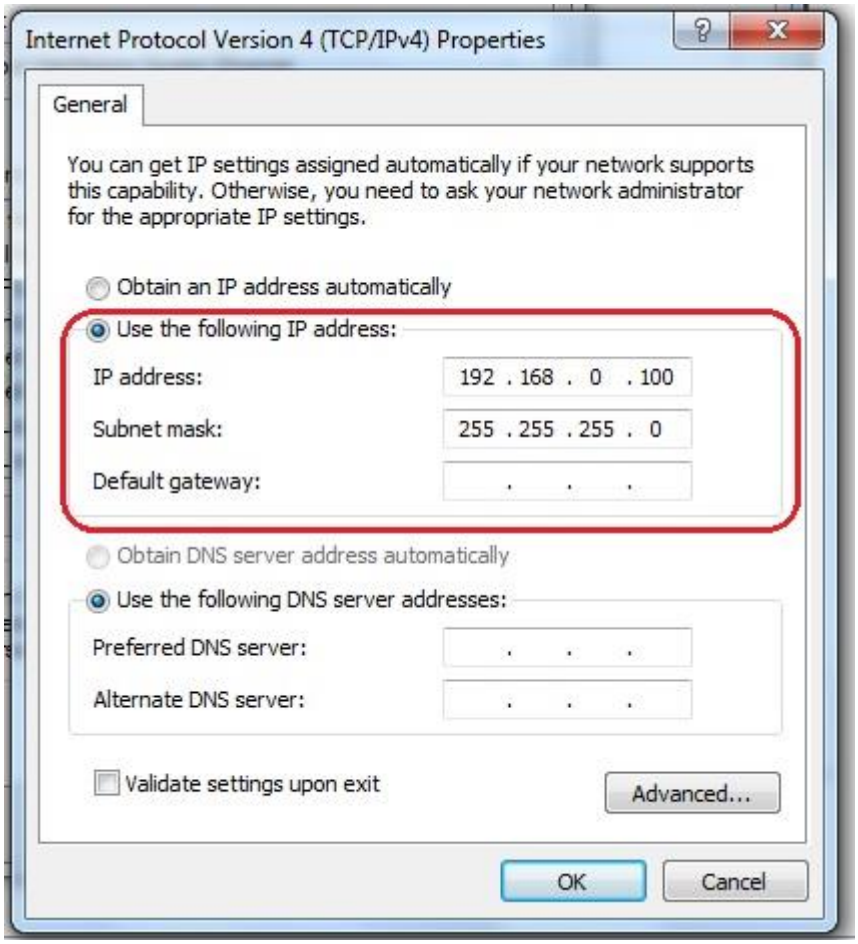
2. 點選Properties



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


Local Area Connection Properties
Networking
Authentication
Connect using:
Broadcom NetXtreme Gigabit Ethernet
Configure...
This connection uses the following items:
Client for Microsoft Networks
VirtualBox Bridged Networking Driver
QoS Packet Scheduler
File and Printer Sharing for Microsoft Networks
Internet Protocol Version 6 (TCP/IPv6)
Internet Protocol Version 4 (TCP/IPv4)
Link-Layer Topology Discovery Mapper I/O Driver
Link-Layer Topology Discovery Responder
Install...
Uninstall
Properties
Description
Transmission Control Protocol/Internet Protocol. The default wide area network protocol that provides communication across diverse interconnected networks.
OK
Cancel

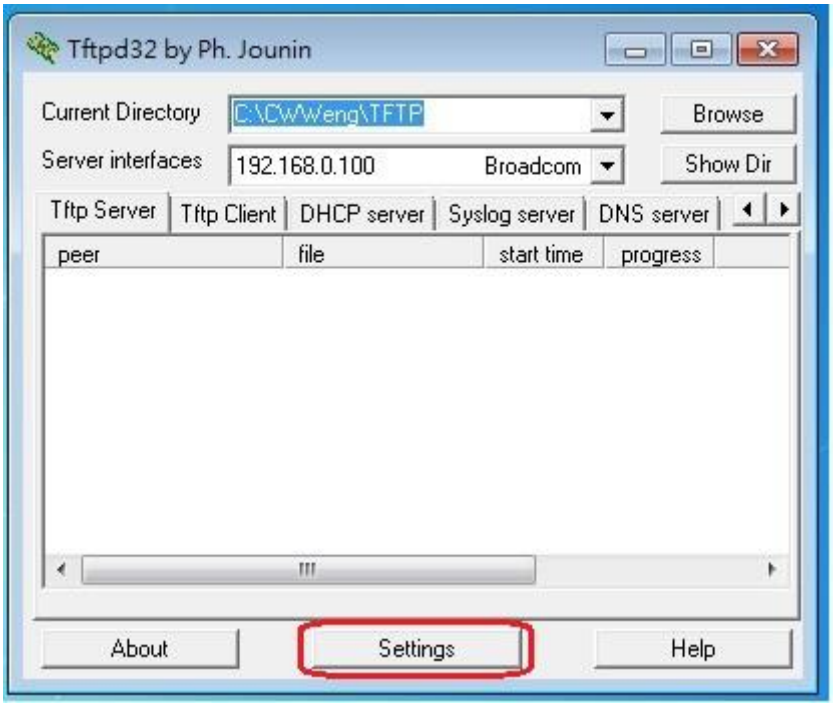
4. 設置固定 IP 位址



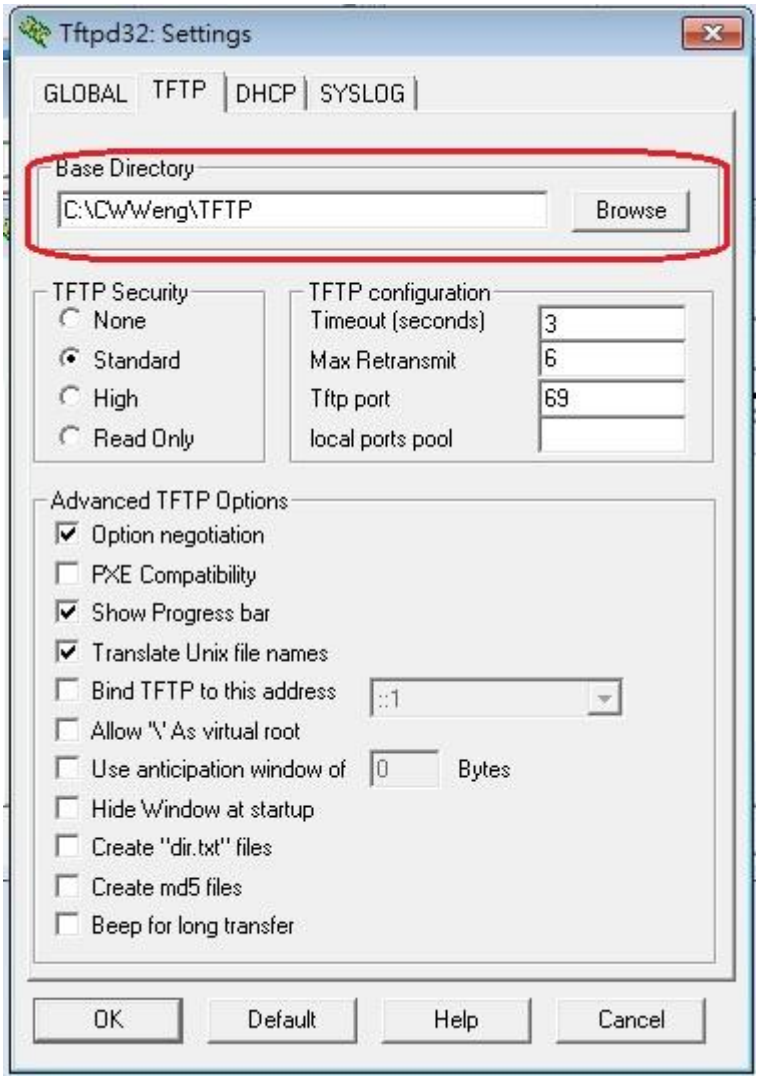
4.12.2 TFTP 和 DHCP 伺服器

TFTPD32 是一個免費、源代碼公開的應用程式, 它包含TFTP 和 DHCP 伺服器等功能. 可以從下面網址進行下載

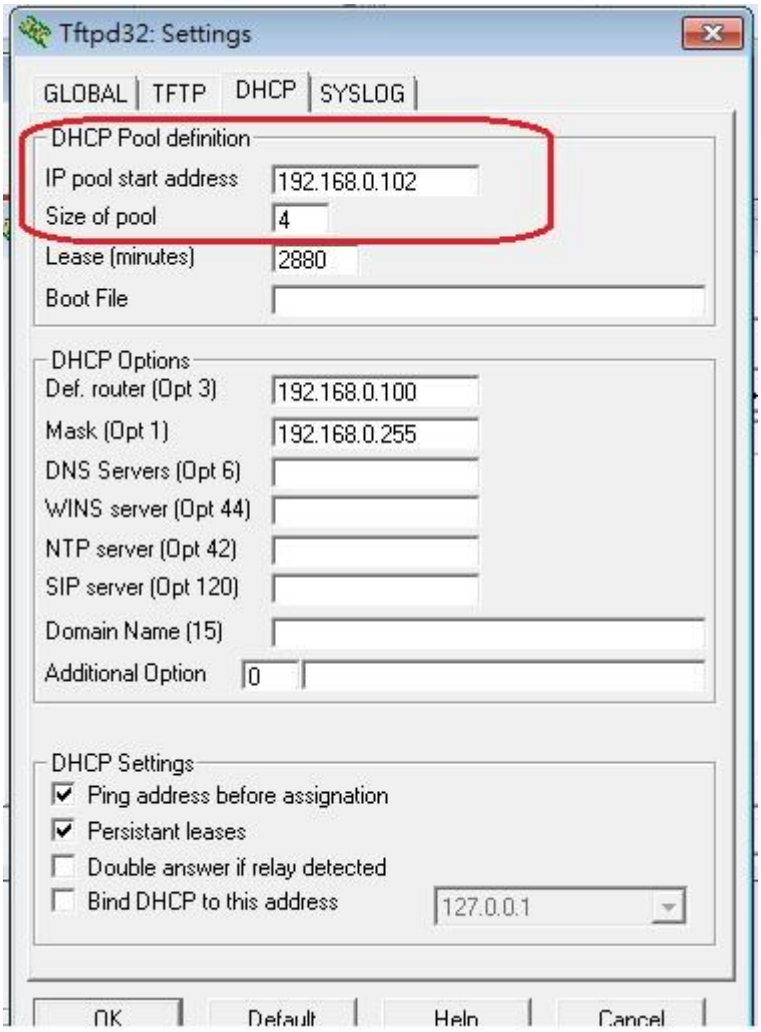
http://www.jounin.net/tftp-server-for-windows.html



點選 Settings 按鍵來設置TFTP server 和 DHCP server.
設置 TFTP server 的檔案存放目錄



設置 DHCP pool definitions. 下面這個範例將 IP pool start address 設為 192.168.0.102



4.13 加快 SPI flash 開機速度

當開機模式是 SPI 開機時，U-Boot 透過 “sf read” 命令，從 SPI flash 將 Linux 內核讀取到 DDR，再透過 bootm 命令完成開機。如果要加快 SPI flash 開機速度，可從提高 SPI 運作速度，以及 bootm 開機過程中採用硬體計算 checksum 等方法。

4.13.1 提高 SPI 運作速度

提高 SPI 運作速度可以從下列方法著手。

- 讓 SPI 運作在 Quad 模式
- 調高 SPI 時鐘

以下的範例是將環境變數 spimode 設為 4，讓 SPI 運作在 Quad 模式，記得用 saveenv 命令將環境變數 spimode 儲存到 flash。再透過 sf probe 命令將 SPI 時鐘設在最高速 (18 MHz)。

```
U-Boot> set spimode 4

U-Boot> saveenv

Saving Environment to SPI Flash...

Erasing SPI flash...Writing to SPI flash...done

U-Boot> sf probe 0 18000000

SF: Detected W25Q128 with page size 4 KiB, total 16 MiB

U-Boot>
```

4.13.2 採用 SHA-1 硬體計算 checksum

bootm 開機過程中採用硬體計算 checksum，也可以縮短開機時間。請參考 4.7.2 章節採用 SHA-1 硬體計算 checksum，來縮短 bootm 開機時間。

4.14 Tomato

針對 Tomato，U-Boot 提供一個 Tomato config，支持 SPI 開機，然後從 SD card 中開啟 Linux 內核。

4.14.1 Tomato U-Boot 編譯命令

清除所有的 object code.

```
# make O=../build/nuc970_tomato/ distclean
```

編譯 U-Boot

```
# make O=../build/nuc970_tomato/ nuc970_tomato_config

# make O=../build/nuc970_tomato/ all
```

(make 的參數 O 是指定編譯產生的檔案要放到哪個目錄)

4.14.2 Tomato U-Boot 鏈結位址

Tomato U-Boot 的鏈結位址是定義在 Makefile.
請找到以下片段

```
nuc970_tomato_config:     unconfig
```

```
        @mkdir -p $(obj)include $(obj)board/nuvoton/nuc970evb

        @echo "RAM_TEXT = 0x2000000" >>
$(obj)board/nuvoton/nuc970evb/config.tmp

        @$(MKCONFIG) nuc970_tomato arm arm926ejs nuc970evb nuvoton nuc970
```

當中 RAM_TEXT 定義 U-Boot 的鏈結位址，
上面的例子，“RAM_TEXT = 0x2000000" ，U-Boot 的鏈結位址就是 0x2000000

4.14.3 使用 mkimage 將 Linux 內核打包

Tomato U-Boot 從 SPI flash 開機後會自動從 SD card 根目錄讀取名叫 970image.sha 的 Linux 內核並完成開機。為了縮短開機時間，Tomato U-Boot 進行 Linux 內核開機過程中採用硬體計算 checksum，因此。請照以下方式將 Linux 內核打包並命名為 970image.sha。

```
u-boot/tools# ./mkimage -A arm -O linux  -T kernel -S sha1 -a 0x7fc0 -e
0x8000 -d 970image 970image.sha

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
```

4.15 注意事項

U-Boot SPI sub-system 目前只支持 NUC970 SPI0. 它所使用的腳位分別是 PB.6, PB.7, PB.8, PB.9, PB.10, 以及 PB.11. 因此, 你必須檢查這些 pin 是否有正確地連接.

5 Linux 內核

5.1 內核設置介面

Linux 內核支持各式不同的設置. 可將不需要的功能移除, 只保留需要的功能. 減少內核所消耗的資源.

要進入內核設置頁面, 請在 linux-3.10.x 目錄下輸入 “make menuconfig” 的命令. 即可進入內核設置頁面.

內核設置選單是多層次的. 在本頁面內, 可透過上, 下, 左, 右, 四個方向鍵控制選單的位置. 上, 下鍵選擇要控制的內核功能. 左, 右鍵則是選擇了下排的功能按鈕. 要進入更深一層的選單時, 按下 “Enter” 鍵即可.

下一排的按鈕有五個, 停留在 “Select” 時, 可透過空白鍵始能及失能內核功能. 顯示[ ] 代表此功能失能. [*] 代表始能. [M] 代表此功能邊一層模塊, 可動態加載. 下排按鈕停留在 “Exit” 時, 按空白鍵可帶往上層選單, 若已在最上層選單, 回出現提示是否要儲存並退出. 下排按鈕停留在 “Help” 時, 按下空白鍵可顯示幫助畫面. 下排按鈕停留在 “Save” 以及 “Load” 則是用來儲存當前設置, 或是加載設置.

在設置完成後, 內核的設置檔案名稱為 “.config”, 放置在 linux-3.10.x 目錄下.

5.2 默認設置

新唐為NUC970 系列芯片提供了默認設置. 建議使用這要更改內核設置前, 先加載默認設置, 再進行更改. 使用者可透過 “make <mcu name>_defconfig” 來加載內核設置, <mcu name>可為:nuc972,nuc973,nuc976, nuc977. 例如使用 “make nuc972_defconfig” 加載 NUC972 的默認設置. 有時因內核設置錯誤造成無法開機時, 也可使用此命令將內核設置還原到可開機的狀態.

.

5.3 Linux 內核設置

本節依據不同的驅動或功能, 介紹所需使能的內核設置.

5.3.1 基本系統設置

- 掛載模塊支持設置

  有些驅動程式只支持動態加載, 例如 USB WiFi 驅動程序, 或是 USB device 的驅動…等. 請依照下的設定使能掛載模塊支持. 在上電進入 shell 後, 可使用 “insmod <模塊名稱>“ 來加載模塊.

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

- 卸載模塊支持設置

  使能掛載模塊支持後, 若是系統需要支持動態卸載, 請依照下的設定使能掛載模塊支

持. 要卸載模塊時, 可使用 “rmmod <模塊名稱>“ 來卸載模塊.

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

  [*]   Module unloading
```

- 開機命令設置 – 以RAM為根文件系統的設定
  開機命令可以用來設置系統. 包含了根文件系統型態, 記憶體大小, console 波特率…等. 以下只是一個簡單的設置, 內核還另外支持了眾多的命令. 可參考 Documentation/kernel-parameters.txt 中的說明.

```
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)  ---
>
```

- 開機命令設置 – 以SPI Flash JFFS2文件系統為根文件系統的設定
  開機命令可以用來設置系統. 包含了根文件系統型態, 記憶體大小, console 波特率…等. 使用這個設定需要啟動JFFS2文件系統 (請參考 5.3.4 文件系統設置), 並且取消RAM根文件系統設定.

```
General setup  --->

  [ ] Initial RAM filesystem and RAM disk (initramfs/initrd) support
```

下列範例是需要先將根文件系統使用mkfs.jffs2工具製作成JFFS2格式的鏡像文件再燒入至mtdblock1所在的位置, 當內核啟動會依據下列開機命令將根文件系統帶起.

```
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)  ---
>
```

- 開機命令設置 – 以NAND Flash YAFFS2文件系統為根文件系統的設定
  開機命令可以用來設置系統. 包含了根文件系統型態, 記憶體大小, console 波特率…等. 使用這個設定需要啟動YAFFS2文件系統 (請參考 5.3.4 文件系統設置), 並且取消RAM根文件系統設定.

```
General setup  --->

  [ ] Initial RAM filesystem and RAM disk (initramfs/initrd) support
```

下列範例是需要先將根文件系統製作成YAFFS2格式的鏡像文件(請參考3.8.4製作FS型態Image), 再燒入至mtdblock2所在的位置, 當內核啟動會依據下列開機命令將根文件系統帶起.

```
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)  ---
>
```

- 開機命令設置 – 以NAND Flash UBIFS文件系統為根文件系統的設定
  開機命令可以用來設置系統. 包含了根文件系統型態, 記憶體大小, console 波特率…等. 使用這個設定需要啟動UBIFS文件系統 (請參考 5.3.4 文件系統設置), 並且取消RAM根文件系統設定.

```
General setup  --->

  [ ] Initial RAM filesystem and RAM disk (initramfs/initrd) support
```

下列範例是需要先將根文件系統製作成UBIFS格式的鏡像文件(請參考3.8.4製作FS型態Image), 再燒入至mtd2所在的位置, 當內核啟動會依據下列開機命令將根文件系統帶起.

```
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)  ---
>
```

- 開機命令設置 – 以NFS文件系統為根文件系統的設定
  在開發初期，開發者可能需要經常更改或更換rootfs，把rootfs用NFS的方式掛載，這樣可以大幅減少開發的時程。

```
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
```

其中，x.x.x.x和z.z.z.z均代表NFS伺服器的ip位置，y.y.y.y代表本機的ip位置，g.g.g.g代表gateway的ip位置，m.m.m.m代表netmask的ip位置。
除了設置boot options之後，還需要啟動網路功能(可參考5.3.2)，另外以下功能也需一並啟動。

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

當然還需要啟動NFS的功能。

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

5.3.2 網絡功能設置

- TCP/IP 設置

  要使能基本的網絡功能, 請依照以下設置即可.

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

- WiFi無線網路設置

  若是要使用無線網路設備, 除了以上的 TCP/IP 設置需使能外, 以下的設置也需一併使能

```
[*] 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 驅動設置

- Audio 音效接口設置
  I²S 音效接口的設置如下:

```
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 支持主模式或從模式, 使用者可透過選單決定運作的模式.
I²S選定時, NAU8822 codec驅動將被自行被選中, 另外,使用時必須同時啟動I²C功能, 系統才能正確偵測到NAU8822裝置.

如果audio應用程式使用舊式OSS架構編寫, 可使能以下兩個選項, 使ALSA 相容 OSS 架構. 可參考音效範例程式alsa_audio. (源碼位於BSP/applications/demos/alsa_audio 目錄下)

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

- Cryptographic Accelerator設置
  要支持Cryptographic Accelerator, 請先勾選打開 Netowrking support裡的PF_KEY socket.

```
[*]  Networking support  --->
  Networking options --->
```

```
        <*> PF_KEY sockets
```

然後再打開 Cryptographic API 裡面的相關設定

```
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 硬件支持 AES, DES, 及 3-DES crypto 演算法, 及支持 SHA 及 HMAC hash 演算法. 可參考範例程式 crpyto. (源碼位於 BSP/applications/demos/crypto 目錄下)

- DMA 設置

  NUC970 系列芯片支援DMA功能. 要在內核中使能 DMA, 必須進入”DMA Engine support”選單中, 開啟”NUC970 DMA support”
  如果要在內核中使用DMA, 可以參考 linux-3.10.x/drivers/dma/dmatest.c中的使用方法, 如果需要實際了解dmatest.c的運作流程, 可以開啟”DMA Test Client”. DMA test client將會被自行被帶起.
  如下圖：

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

- 使用者空間記憶體管理功能設置

  如需要在使用者空間獲取一塊記憶體的虛擬和實體位置，就可啟動這功能來達成這樣的目的。
  請啟動如下的配置：

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

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

使用方法可參考2D 範例程式。

- Ethernet 網口設置

  NUC970 系列支持了兩個網口, 可分別開起或同時開啟 要支持網口, 除了網口驅動外, PHY驅動需要一併使能. NUC970 開發版上使用的是 ICPlus 的 PHY, 若是使用了不同的 PHY, 設置需做相應的修改. NUC970 可支援透過 Magic Packet 喚醒, 但須透過 ethtool 開啟此功能. Magic Packet 的格式依循 AMD 的 Magic Packet Technology 白皮書規範. ethtool 工具的使用請參考 6.1 節 ethtool 的說明.

```
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 設置

  NUC970 內核執行時, 使用基本型時鐘來計時. 另外還提供了四通道的加強型時鐘可輸出 50% 佔空比的輸出, 或是支持捕獲的功能. 四個通道可以獨立控制功能. 以下是相關的設置畫面. 不須使用的功能選擇 “No output” 或是 “No input” 即可. 以下的範例 Etimer 通道 0 及 1 分別使用 PC.6, PC.8 當輸出. 而 通道 2 及通道 3 分別使用 PC.11 及 PC.13 當作捕獲管腳. 請使能”NUC970 Enhance Timer(ETIMER) wake-up support”來支持加強型時鐘喚醒系統的功能.

```
Device Drivers  --->
  Misc devices  --->
        <*> NUC970 Enhance Timer (ETIMER) support
        <*> NUC970 Enhance Timer (ETIMER) wake-up 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)  --
->
```

應用程序可透過 ioctl() 來控制加強型時鐘的功能, 目前支持了喚醒系統, periodic, toggle out, trigger counting mode, 以及 free counting mode 等五個功能. 喚醒系統功能, 可以設定不同的叫醒時間來喚醒系統. 在捕獲模式下 (trigger counting mode), 所捕獲到的值可經由 read() 讀回. 單位是us, 代表了兩次觸發條件的間隔. free counting mode會測量外部引腳的輸入頻率, 所捕獲到的值可經由 read() 讀回, 單位是Hz, 代表外部引腳的輸入頻率. 所有的功能可以參考 BSP 中所附的範例程序(源碼位於 BSP/applications/demos/etimer目錄)來開發所需要的功能.

- 智能卡 (Smartcard) 設置

  NUC970 擁有兩個智能卡接口, 符合 ISO-7816 以及 EMV 2000 規範. 若是系統中有使用智能卡的需求, 可以參考以下的設定, 使能智能卡的驅動. 這份智能卡驅動支持了 T = 0 以及 T = 1 的智能卡. 兩個接口可以分別設置卡插拔偵測的准位, 以及上電時需要給高電位或是低電位. 這個設定跟外部線路以及使用的卡槽有關. 另外接口 0 可以選擇使用 Port I 或是 Port G 的管腳. 當打開 Perform EMV check 的選項時, 驅動會執行較為嚴格, 符合 EMV 規範的檢查. 有些智能卡可能會被判別為無法操作的卡.

```
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
```

應用程序可透過 ioctl() 來控制智能卡接口. 以下列出支持的命令以及功用. 實際使用可以參考BSP 中所附的範例程序(源碼位於BSP/applications/demos/sc目錄).
SC_IOC_GETSTATUS: 判斷目前卡槽的狀態, 例如是否插入還是拔出.
SC_IOC_ACTIVATE: 激活智能卡. 若是成功, ioctl() 會返回 ATR 的長度.
SC_IOC_READATR: 讀取智能卡回覆的 ATR (Answer to reset)
SC_IOC_DEACTIVATE: 關閉智能卡.
SC_IOC_TRANSACT: 透過 sc_transact 這個結構體對智能卡下命令並讀取回應.
請注意, 進入省電模式前, 智能卡會被關閉. 系統喚醒後, 若是應用程序要讀寫智能卡, 必須重新激活智能卡.

- GPIO 設置

NUC970系列芯片支援GPIO介面控制, 要讓內核支持 GPIO 的控制, 請使能 “NUC970 GPIO support” 以及 “/sys/class/gpio/…” .

```
Device Drivers  --->

  [*] GPIO Support  --->

        [*]   /sys/class/gpio/... (sysfs interface)

              <*>   NUC970 GPIO support

               <*>     NUC970 external I/O wake-up support
```

GPIO 驅動程序將NUC970系列芯片的 GPIO口, 從GPIOA~GPIOJ 每組IO都保留32個號碼, 所以GPIOA編號0x000~0x01F, GPIOB編號0x020~0x03F, GPIOC編號0x040~0x05F, GPIOD編號0x060~0x07F, GPIOE編號0x080~0x09F, GPIOF編號0x0A0~0x0BF, GPIOG編號 0x0C0~0x0DF, GPIOH 編號 0x0E0~0x0FF, GPIOI 編號 0x100~0x11F, GPIOJ 編號 0x120~0x13F.
用戶程序可透過 sysfs 來控制各 GPIO 口. /sys/class/gpio/…” 是通用的GPIO操作接口, 可以透過下列方法來控制GPIO

- /sys/class/gpio/export :來告訴系統需要控制哪個GPIO
- /sys/class/gpio/unexport: 則可以取消哪個GPIO控制
- /sys/class/gpio/gpio0/direction : 針對GPIOA00控制 in 或 out
- /sys/class/gpio/gpio0/value : 針對GPIOA00控制輸出1 或 0

下面範例可將GPIOA0設定成輸出High：

```
$ echo 0 > /sys/class/gpio/export

$ echo out >/sys/class/gpio/gpio0/direction

$ echo 1 >/sys/class/gpio/gpio0/value
```

也可以參考範例程式中的 gpio_demo(源碼位於 BSP/applications/demos/gpio目錄下)

在其他的驅動程序中可以透過下步驟控制GPIO：

- 在驅動程序中加入#inlcude <linux/gpio.h>
- 依據 arch\arm\mach-nuc970\inlcude\mach\gpio.h 決定使用哪個GPIO
- 以NUC970_PC7為例子如下:

  設定輸入模式： gpio_direction_input(NUC970_PC7);
  設定輸出模式和輸出值：gpio_direction_output(NUC970_PC7,1);
  設定輸出high： gpio_set_value(NUC970_PC7, 1);
  設定輸出low： gpio_set_value(NUC970_PC7, 0);
  取值： gpio_get_value(NUC970_PC7);
  確認GPIO是否正在使用: gpio_request(NUC970_PC7, "NUC970_PC7");
  取得GPIO 中斷號碼 ： gpio_to_irq(NUC970_PC7);

  使用GPIO中斷範例：

```
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);
    }
```

需要讓GPIO從power down模式中叫醒 ，可以開啟”NUC970 external I/O wake-up support”功能. 並且在linux-3.10.x/drivers/gpio/gpio-nuc970.c 中可以找到EINT0, EINT1, EINT2, EINT3, EINT4, EINT5, EINT6, EINT7的設定. 以EINT0為例子 如下:

```
/*
 * @brief       External Interrupt 0 Handler
 * @details     This function will be used by EINT0,
 *              when enable IRQ_EXT0_H0 or IRQ_EXT0_F11 in eint0
 */
/*
static irqreturn_t nuc970_eint0_interrupt(int irq, void *dev_id){
        printk("@0\n");
        return IRQ_HANDLED;
}
*/


/* If enable IRQ_EXT0_H0 or IRQ_EXT0_F11 , linux will enable EINT0
 * User can modify trigger tiypes as below :
 * IRQF_TRIGGER_FALLING / IRQF_TRIGGER_RISING / IRQF_TRIGGER_HIGH / IRQF_TRIGGER_LOW
 */
struct nuc970_eint_pins eint0[]={
//{IRQ_EXT0_H0, nuc970_eint0_interrupt,IRQF_TRIGGER_FALLING | IRQF_TRIGGER_RISING,"eint0"},
//{IRQ_EXT0_F11,nuc970_eint0_interrupt,IRQF_TRIGGER_FALLING | IRQF_TRIGGER_RISING,"eint0"},
{0,0,0,0}
};
```

由上面可以開啟 nuc970_eint0_interrupt 和設定 nuc970_eint_pins eint0 從那一個GPIO pin 和觸發的型態,假設需要使用EINT0中斷從 PH0 , 觸發的型態設定成 上升和下降觸發, 可以參考如下面設定即可.

```
static irqreturn_t nuc970_eint0_interrupt(int irq, void *dev_id){
        printk("@0\n");
        return IRQ_HANDLED;
}
struct nuc970_eint_pins eint0[]={
{IRQ_EXT0_H0, nuc970_eint0_interrupt,IRQF_TRIGGER_FALLING |
IRQF_TRIGGER_RISING,"eint0"},
{0,0,0,0}
};
```

- GPIO 模似I²C 接口設置

  除了使用硬體I²C, 尚可使用IO模擬的方式來實現I²C控制, 透過修改 arch/arm/mach-nuc970/dev.c 中i2c_gpio_adapter_data結構來選擇使用的腳位. 如設定 .sda_pin = NUC970_PG1, .scl_pin = NUC970_PG0 即是使用PG0為SCL腳, PG1為SDA 腳.

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

- I²C 接口設置

  I²C 接口的設置如下:

```
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)  --->
```

I²C port 1 有多組的管腳可選擇使用, 如 Port-B, Port-G, Port-H或者Port-I

選擇硬體I²C, 系統將使用NUC970內建I²C硬體模組, 進行I²C控制.

內建5組I²C port 0 client, 分別是 OV7725, OV5640, NT99050, NT99141及NAU8822, 使用者可自行修改/新增或修改至port 1, 修改檔案位置為：arch/arm/mach-nuc970/dev.c 中的 nuc970_i2c_clients0 結構.

- LCD 接口設置

  要使能 LCD 屏幕支持的話，請依照以下的選項來設置內核：

```
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
```

在開發板上已接上800x480 5吋的LCD屏，使用18bit的資料匯流排與屏連接，顏色數為RGB888(24-bit)，顏色數可視應用層狀況由使用者自行選擇。

如果要將Linux 的企鵝logo在屏上顯示出來，則需選中下列選項：

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

如此開機時，就可以在屏上看到Linux 的企鵝logo。

可參考範例程式 lcm，了解如何操作 frame buffer。(源碼位於 BSP/applications/demos/lcm 目錄下)

- 2D 圖形加速器設置

  NUC970支持2D圖形加速功能，來實現畫線、畫矩形、旋轉、放大/縮小及BitBlt功能。

  配置方法如下：

```
Device Drivers  --->
  Generic Driver Options  --->
        Misc devices  --->
```

```
            <*> NUC970 2D support
```

使用方式可參考2D範例程式。(源碼位於BSP/applications/demos/2d 目錄下)

- MTD NAND flash 設置

  NAND flash 是使用的驅動是掛在 MTD 子系統之下. 可按照以下設置使能. NAND flash 接口有兩組管腳可選擇, Port C 以及 Port I, 需視硬件接線設置.

```
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)  --->
```

  驅動中的基本設置如果需要由U-boot環境變數傳入就必須將”Command line partition table parsing”選上, 否則會使用驅動程式裡的默認配置, 主要會將 MTD 分為三塊空間. 上電進入 shell 後, 分別是/dev/mtdblock0, /dev/mtdblock1, 以及 /dev/mtdblock2. 第一塊是放置 U-Boot 的空間, 第二塊放置內核文件, 第三塊則是用來掛載 YAFFS2 或 UBIFS 文件系統的空間. 若是配置有需要更改, 例如增加或減少分區, 改變分區大小. 請直接編輯 uboot/include/nuc970_evb.h 或 drivers/mtd/nand/nuc970_nand.c.

- PWM 設置

  NUC 970 BSP 中的內核版本為 3.10, 但 3.10 的 PWM 驅動程序不支持用戶程序透過 sysfs 控制 PWM. 所以目前的PWM 子系統是另外由 3.11 移植回 3.10, 方便用戶程序控制 PWM.

  要支持 PWM, 請依以下的設定使能 PWM, 使用的管腳可能須依系統改變. 不須使用的通道選擇 “No output” 即可.

```
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)  --->
```

要透過sysfs控制 PWM, 首先在系統開機後,進入 /sys/class/pwm/, 裡面可看到四個 PWM (pwmchip0~3), 每一組代表著一個 PWM通道. 要使用前, 先進入要控制的通道目錄執行 echo 0 > export, 此時會多生成一個 pwm0 目錄. 接著就可以開始控制這個通道了. 在新生成的目錄中有幾個文件可用來控制PWM, 它們的意義列在下表.

文件名稱	目的
period	控制週期, 單位是 ns. 目前驅動支持的最小單位是 us, 以打出 20us 之週期波為例, 使用 echo 20000 > period 命令
duty_cycle	設置佔空比. 單位是 ns. 目前驅動支持的最小單位是 us, 以打出 15us 之佔空比為例, 使用 echo 15000 > period 命令
polarity	設置輸出相位. 可支持 echo normal > polarity 正向輸出, 或是 echo normal > polarity 反向輸出
enable	使能及失能控制. 要使能, 使用 echo 1 > enable 命令, 要使能則使用 echo 0 > enable 命令

以下就是一個開啟 PWM0, 並輸出週期 300us, 佔空比 33% 的例子.

```
$ 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 支持

  要支持 RT3070 USB WiFi 功能的話, 除了內核本身無線網絡功能要使能, USB host 驅動要使能外, 以及掛載模塊支持要使能外, 還需開啟以下的功能:

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

並且在 Boot command 加上以下命令.

```
coherent_pool=2M
```

RT3070 的驅動並沒有放在內核源碼之中, 而是單獨放在 BSP/applications/DPO_RT3070_LinuxSTA_V2.3.0.2_20100412/ 目錄下. 所以編譯產生的是內核模塊, 可以動態加載.

目錄下的 Makefile 需要手動編輯, 設定正確的內核源碼路徑, 之後輸入 “make” 命令進行編譯.

```
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
```

編譯完成後, 所產生的模塊叫做 rt3070sta.ko. 要使用本驅動的話, 除了這個模塊需要複製到根文件系統中, RT2870STA.dat 這隻檔案也需複製到根文件系統的 etc/Wireless/RT2870STA/ 目錄下.

```
$ 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
```

這個驅動的使用說明如下.

1. 上電進入shell後, 使用 insmod 命令加載模塊.

```
$ insmod rt3070.ko
```

2. 啟動無線網口

```
$ ifconfig ra0 up
```

3. 使用 BSP 中的 wireless_tool.29版本連接無線基地台

3.1 使用 WEP 方式連線

```
$ iwconfig ra0 essid "基地台名稱"
$ iwconfig ra0 key open
$ iwconfig ra0 key 密鑰
```

3.2 使用 WPAPSK 方式連線

```
$ iwpriv ra0 set NetworkType=Infra
$ iwpriv ra0 set AuthMode=WPAPSK
$ iwpriv ra0 set EncrypType=TKIP
$ iwpriv ra0 set WPAPSK=密鑰
$ iwpriv ra0 set SSID="基地台名稱"
```

3.3 使用 WPA2PSK 方式連線

```
$ iwpriv ra0 set NetworkType=Infra

$ iwpriv ra0 set AuthMode=WPA2PSK

$ iwpriv ra0 set EncrypType=AES

$ iwpriv ra0 set WPAPSK=密鑰

$ iwpriv ra0 set SSID="基地台名稱"
```

連線成功後, 設置固定 IP, 或使用以下命令向 DHCP 服務器取得動態 IP.

```
$ udhcpc -i ra0
```

- Realtek RTL8188 802.11 WiFi 支持

  要支持RTL8188 USB WiFi 功能的話, 除了內核本身無線網絡功能要使能, USB host 驅動要使能外, 以及掛載模塊支持要使能外, 還需開啟以下的功能:

```
[*] Networking support --->

-*- Wireless --->

  <*>  Nuvoton external WiFi driver support
```

RTL8188 的源碼並沒有包含在 BSP 中. 若是需要源碼, 請跟模組廠索取, 新唐科技無法提供.

這個驅動的使用說明如下.

1. 上電進入shell後, 使用 insmod 命令加載模塊.

```
$ insmod rtl8188eu.ko
```

2. 啟動無線網口

```
$ ifconfig lo up

$ ifconfig wlan0 up
```

3. 使用 wpa_supplicant 連接無線基地台

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

其中需指定基地台的連線設定檔, 以下是基地台設置成 WPA AES 連線時的範例設置:

```
network={

  ssid="TESTTEST"

  proto=WPA
```

```
  key_mgmt=WPA-PSK
  pairwise=CCMP
  psk="ZZZZZZZZ"
}
```

- Realtek RTL8192 802.11 WiFi 支持

  要支持RTL8192 SDIO WiFi 功能的話, 除了內核本身無線網絡功能要使能, SDIO host 驅動要使能外, 以及掛載模塊支持要使能外, 還需開啟以下的功能:

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

  RTL8192 的源碼並沒有包含在 BSP 中. 若是需要源碼, 請跟模組廠索取, 新唐科技無法提供.

  這個驅動的使用說明如下.

  4. 上電進入shell後, 使用 insmod 命令加載模塊.

```
$ insmod 8192es.ko
```

  5. 啟動無線網口

```
$ ifconfig wlan0 up
```

  6. 使用 wpa_supplicant 連接無線基地台

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

  其中需指定基地台的連線設定檔, 以下是基地台設置成 WPA AES 連線時的範例設置:

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

- RS232, RS485, IrDA 串口設置
  NUC970 系列帶有11個串口, 可以分別獨立設置. 請依以下的說明來使能串口功能. 每個串口可以單獨的開關. 除 UART0, UART3, UART5 外, 有多組管腳可選擇, 需要一併設置. 其中, UART0 這組串口是保留給 console 使用的. 不須特別使能.
  除UART0, UART3, UART5, UART7, UART9外,使用者可以開關喚醒功能.

```
Device Drivers  --->
  Character devices  --->
        Serial drivers  --->
              [*]  NUC970 serial support
              [*]    NUC970 UART1 support
              [*]      Enable UART1 CTS wake-up function
                       NUC970 UART1 pin selection (Tx:PE2, Rx:PE3)  --->
              [*]    NUC970 UART2 support
              [*]      Enable UART2 CTS wake-up function
                       NUC970 UART2 pin selection (Tx:PF11, Rx:PF12)  --->
              [*]    NUC970 UART3 support
              [*]    NUC970 UART4 support
              [*]      Enable UART4 CTS wake-up function
                       NUC970 UART4 pin selection (Tx:PC10, Rx:PC11)  --->
              [*]    NUC970 UART5 support
              [*]    NUC970 UART6 support
              [*]      Enable UART6 CTS wake-up function
                       NUC970 UART6 pin selection (Tx:PB2, Rx:PB3)  --->
              [*]    NUC970 UART7 support
                       NUC970 UART7 pin selection (Tx:PG4, Rx:PG5)  --->
              [*]    NUC970 UART8 support
              [*]      Enable UART8 CTS wake-up function
                       NUC970 UART8 pin selection (Tx:PE10, Rx:PE11)  --->
```

```
                [*]    NUC970 UART9 suppor
                         NUC970 UART9 pin selection (Tx:PH2, Rx:PH3)  --->
                [*]    NUC970 UART10 support
                [*]      Enable UART10 CTS wake-up function
                         NUC970 UART10 pin selection (Tx:PB10, Rx:PB11)  --->
                [*]     Console on NUC970 serial port
```

若是串口需要作為紅外控制使用, 則除了要使用的串口需要使能外, 紅外模塊的驅動也需依以下的範例始能.

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

- SD 卡設置

  以下是SD卡接口使能的設置. NUC970支持兩個 SD 卡接口, 目前的驅動只能選擇其中一個支持, 而不能同時對兩個介面進行操作. 若是選擇 SD1 的話, 需要選擇使用的管腳, 可為 Port E, Port H, 或是 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)  --->
```

上電進入 shell 後, 若是偵測到有卡插入, 會在 /dev 下出現一個 mmcblk0 裝置. 若卡上有分區, 會依分區方式另外出現 mmcblk0, mmcblk1… 等設備.

- SPI 接口設置

  NUC970 系列蕊片支持了兩個 SPI 接口, 可以單獨使能或是同時使能. 以下是同時使能兩個接口的說明:

```
Device Drivers  --->
  [*] SPI support  --->
        <*>  Nuvoton NUC970 Series SPI Port 0
              SPI0 pin selection (Normal mode)  --->
        <*>  SPI0 enable pin for the second chip select
              Pin selection (Use SS1 (PB0))  --->
        <*>  Nuvoton NUC970 Series SPI Port 1
              SPI1 transfer mode selection (Normal mode)  --->
              SPI1 IO port selection (Port B)  --->
        < >   SPI1 enable pin for the second chip select
```

SPI0/1都可藉由選擇normal或quad模式來決定使用SPI的腳位數量，Normal為4個腳或者 Quad為(6個腳。 另外有第二組片選腳可供選擇。

SPI port 1 則可選擇 Port B/Port I (Normal mode) 或者 Port B/Port I (Quad mode). 使用者也可以選擇開啟第二組片選腳，如果是選擇Port B，第二組片選腳固定為PB.1，如果是選擇Port I的話，第二組片選腳則固定為PH.13。

如要使用SPI flash, 需要開啟MTD功能, 使能如下選項：

```
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, ...)
```

另外, JFFS2檔案系統的功能也要一併選上, 如此才能正確使用SPI Flash裝置.
JFFS2 文件系統的設置方式請參考文件系統設置的章節.
Linux 內核需正確的判斷 SPI flash 才能對其讀寫. 要讓內核識別新的 SPI flash型號, 請修改/新增 drivers/mtd/devices/m25p80.c中的 m25p_ids結構

```
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) },
  { },
};
```

以及 arch/arm/mach-nuc970/dev.c 中的nuc970_spi_flash_data結構.
其中 .type的字串需與 m25p_ids結構中的其中一個字串相同, 不然無法比對到正確裝置.

```
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",
};
```

如欲修改SPI Flash partition數, 則可修改 arch/arm/mach-nuc970/dev.c 中的nuc970_spi_flash_partitions結構.

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

- USB Host 設置

  要支持 USB Host, 請先勾選打開 USB Host 端支持。 NUC970 USB Host包含EHCI(USB 2.0)及 OHCI(USB1.1)兩個USB Host控制器，必須同時打開。 以下所列之項目必須全部勾選：

```
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)
--->

        [*]     NUC970 turn off Usb Hots VBUS power while power down

        <*>     OHCI HCD support

        [*]     NUC970 OHCI (USB 1.1) support
```

  其中在 “NUC970 EHCI (USB 2.0) support”底下，依據所使用的芯片型號, 選擇USB Host port power pin. 選單如下:

```
[ ] 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
```

  如果 USB Host 電路設計上使用了 Power Switch IC, 那麼可選擇USB Host port0 及 port1 分別由PE.14及 PE.15控制; 或是選擇統一由 PF.10控制。當選擇使用 USBH_PPWRx 的情況下，USBH_OVRCUR就會被用來當作過電流保護指示腳來使用。如果電路設計沒有採用 Power Switch IC，而是直接供電給 USB port，那麼 USBH_PPWRx 就可以挪作 GPIO 使用，可選擇第三個或第四個選項。如果過電流保護指示腳也不需要，那麼可以選擇第四個選項。

  NUC970 USB Host 驅動，提供讓使用者選擇在 power down (休眠) 模式下，選擇是否關閉 VBUS 供電。

```
 [*]     NUC970 turn off Usb Hots VBUS power while power down
```

  如果使用者不勾選這項設定，那麼在系統休眠的時候，NUC970 USB Host 僅會關閉

USB PHY 電源，而不關閉 VBUS 電源。由於 VBUS 仍然保持供電，所以 USB 裝置盡管處於 suspend (暫停) 狀態，它仍然可以獲得 NUC970 USB Host 的供電，以維持在不斷電狀態。
如果使用者勾選了這項設定，那麼在系統休眠的時候，NUC970 USB Host會同時關閉 USB PHY 及 VBUS 電源。對於完全由 USB bus 供電的 USB device 來說，這就像當於斷電了。當系統喚醒時，NUC970 USB Host 重新對 USB PHY 及 VBUS 供電，這相當於所有的 USB 裝置斷開之後的重新連接，因此，所有連接的 USB 裝置都需要重新枚舉及初始化。

- USB Host 與 USB Mass storage 裝置設置
  除了NUC970 的 USB host 驅動需要設置, 另外還需選擇所要支持的裝置類別, 例如 Mass Storage. 若是要選擇 Mass Storage, 則需先開啟 SCSI 設備支持, 才會出現 Mass Storage 的選項

```
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 與 USB Video Class 裝置設置
  如果要支援 USB Video Class (UVC) 視訊裝置，必須打開下列的設置選項：

```
Device Drivers  --->
  Multimedia support  --->
        <*>  Cameras/video grabbers support
        <*>  Media Controller API
        <*>  V4L2 sub-device userspace API
```

```
          <*>  V4L2 int device
        <*>  Media USB Apapters  --->
              <*>  USB Video Class (UVC)
              [*]    UVC input events device support
        <*>  V4L platform device
  Graphics support  --->
        <*>  Support for frame buffer devices  --->
              [*]  NUC970 LCD framebuffer support
```

- USB Host 與 HID 裝置設置
  如欲使用USB HID (如USB mouse, keyboard …)等裝置，除了需要USB host的功能外，另外需開啟HID及input的功能，請參考以下的設置。

```
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設置 – Mass Storage Device

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

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

編譯完內核後，會產生三個 module 文件， fs/configfs/configfs.ko, drivers/usb/gadget/libcomposite.ko, drivers/usb/gadget/g_mass_storage.ko, 需要將此三個文件複製到rootfs或是系統可以存取到的地方。

USB Mass Storage Gaget的使用方法如下(以mmcblk0p1 裝置為例)：

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

- USB Device設置 – Serial Device

```
Device Drivers --->
  [*] USB support --->
        <*> USB Gadget Support --->
              USB Peripheral Controller --->
                    <*> NUC970 USB Device Controller
              <M> USB Gadget Driver
              <M> Serial Gadget (with CDC ACM and CDC OBEX support)
```

編譯完內核後，會產生七個 module 文件， fs/configfs/configfs.ko, drivers/usb/gadget/libcomposite.ko, drivers/usb/gadget/g_serial.ko, drivers/usb/gadget/u_serial.ko, drivers/usb/gadget/usb_f_serial.ko, drivers/usb/gadget/usb_f_acm.ko, drivers/usb/gadget/usb_f_obex.ko, 需要將此七個文件複製到rootfs或是系統可以存取到的地方。

USB Serial Gaget的使用方法如下：

```
$ insmod configfs.ko
$ insmod libcomposite.ko
```

```
$ insmod u_serial.ko
$ insmod usb_f_acm.ko
$ insmod usb_f_serial.ko
$ insmod usb_f_obex.ko
$ insmod g_serial.ko
g_serial gadget: Gadget Serial v2.4
g_serial gadget: g_serial ready
```

- VCAP 影像擷取接口設置
  NUC970系列芯片支援Video-Capture介面，在內核中使能此功能，首先必須開啟“cameras/video grabbers support”，然後進入 “Encoders, decoders, sensors and other helper chips”選項中, 開啟 “NUC970 Video-in support”並且選擇攝影機的型號, 目前支援OV7725, OV5640, NT99050 以及 NT99141.

```
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)  --->
```

Video-Capture介面必須使用I²C介面來控制攝影機, 所以在開啟Video-Capture介面之前

必須設定先開啟I²C介面, I²C的部分請參考I²C章節.
目前Video-Capture驅動支援V4l2 API, 範例部分請參考範例程式中的cap2lcm. (BSP/applications/demos/cap 目錄) , 從cap2lcm 的源碼中可以看見API如下:
xioctl(fd, VIDIOC_S_FMT, &fmt) : 設定影像的長寬以及格式
xioctl(fd, VIDIOC_DQBUF, &buf) : 接收影像
xioctl(fd, VIDIOC_QBUF, &buf) : 將接收完成的影像釋放出來
xioctl(fd, VIDIOC_STREAMON, &type) : 啟動CAP
xioctl(fd, VIDIOC_STREAMOFF, &type) : 停止CAP

- Watchdog Timer 看門狗設置
  要支持看門狗, 請參考以下的設置. 目前默認的超時為 2.03 秒. 用戶程序可透過 ioctl() 下 WDIOC_SETTIMEOUT 命令更改超時時間. 驅動中支持三個不同週期, 輸入小於2的話, 是0.53 秒. 介於 2~8 之間的話是 2.03 秒, 而超過 8 的話是設置成 8 秒. 可參考wdt範例程式. (BSP/applications/demos/wdt目錄)
  如果要支持看門狗喚醒系統的功能，請使能 “NUC970 WDT wake-up support”

```
Device Drivers  --->

  [*] Watchdog Timer Support  --->

        <*>   Nuvoton NUC970 Watchdog Timer

        <*>     NUC970 WDT wake-up support
```

- Window Watchdog Timer 窗口看門狗設置
  要支持窗口看門狗, 請參考以下的設置.

```
Device Drivers  --->

  [*] Watchdog Timer Support  --->

        <*>   Nuvoton NUC970 Window Watchdog Timer
```

NUC970 窗口看門狗與看門狗主要有三個差異, 第一, 設置始能之後, 無法更改設置, 無法停止. 第二, 只能在特定的時間內讓窗口看門狗復位, 而不是如同看門狗只要還沒超時, 隨時可復位. 所以應用程序在使用時, 一定要先使用 ioctl() 的 WDIOC_GETTIMELEFT 命令取得可復位的時間. 只有當返回時間為 0, 才可使用 WDIOC_KEEPALIVE 命令窗口看門狗復位, 否則在不正確的時間下達 WDIOC_KEEPALIVE, 會馬上造成系統復位. 第三, 窗口看門狗的設計是 CPU 在掉電以及閒置模式時, 是不數的. 當系統不忙碌時, 而Linux 內核會在每次系統時鐘中斷之間, 讓 CPU 進入閒置模式. 所以窗口看門狗可復位的時間會依據系統的忙碌程度而

有所改變.可參考wwdt範例程式. (BSP/applications/demos/wwdt目錄)

- Keypad 設置

```
Device Drivers  --->
  Input device support --->
        [*] Keyboards  --->
              <*>  NUC970 Matrix Keypad support
              <*>  NUC970 KEYPAD wake-up 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
```

如要使用開發板上的keypad，則需選擇”Keypad pins are 4x2 matrix PH pin”，可搭配keypad範例程式(BSP/applications/demos/keypad)來使用。當芯片進入掉電模式, 使能”NUC970 KEYPAD wake-up support”，按任意鍵可以將系統喚醒.

- RTC 設置

使用者使用RTC功能時,還可以設置RTC的喚醒功能

```
Device Drivers --->
  [*] Real Time Clock --->
        <*> NUC970 RTC driver
        [*]   Enable RTC wake-up function
```

- CAN 設置

NUC970 系列帶有2個CAN(Controller Area Network), 可以分別獨立設置. 請依以下的說明來使能CAN功能. 每個CAN可以單獨的開關. CAN0有多組管腳可選擇, 需要一併設

置.
使用者也可以設置CAN的喚醒功能

```
-*- 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
                              [*]    Enable CAN0 wake-up function
                                     NUC970 CAN0 pin selection (Tx:PB11, Rx:PB10)
--->
                                          (X) Tx:PB11, Rx:PB10
                                          ( ) Tx:PH3, Rx:PH2
                                          ( ) Tx:PI4, Rx:PI32
                              [*]   NUC970 CAN1 support
                              [*]    Enable CAN1 wake-up function
                                     NUC970 CAN1 pin selection (Tx:PH15, Rx:PH14)
--->
                                          (X) Tx:PH15, Rx:PH14
```

可搭配CAN範例程式(BSP/applications/demos/CAN)來使用。

- JPEG 設置

  要支持JPEG CODEC, 請參考以下的設置.

```
Device Drivers  --->
     <*> Multimedia support  --->
          [*]   Cameras/video grabbers support
```

```
            NUC970 JPEG CODEC  --->
                *** NUC970 JPEG codec ***
                    <*> NUC970 JPEG codec support
                     (0xC8000) Max Raw data size - w*h*byteperpixel(hex) +
                     Max Bistream Size
```

- ADC 電池設置

  NUC970系列晶片支援ADC Battery介面，必須開啟Power supply class support中的”NUC970 ADC battery driver”。如下圖：

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

在shell模式下可以進入”sys/class/power_supply”中找到NUC970 Bettery(ADC)進入後即可查看目前的狀態，如下所示：

電池的電壓 – cat voltage_now

電池百分比 – cat present

```
# 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設置

  NUC970系列晶片支援ADC keypad/touchscreen介面，必須進入”Input device support”中，然後開啟並進入”Input NUC970 ADC”，即可看見”Keypad support” 和“Touchscreen support”，選擇需要的功能開啟即可，如果需要支援wake-up的功能，開啟即可。

```
Device Drivers  --->
  [*] Input device support  --->
```

```
        <*> Event interface
        <*> Input NUC970 ADC  --->
              <*>Keypad support
              <*>  Keypad wake-up support
              <*>Touchscreen support
              <*>  Touchscreen wake-up support
              (0) ADC Sample Counter
```

當使用keypad 時，可以透過修改 drivers/input/nuvoton/nuc970adc.c 中的nuc970_keycode 以修改按鈕回報值及nuc970_key_th 來設定ADC 的門檻值，假設把ADC得到的電壓需要分成八個key則nuc970_keycode[0]設定第一個key回報的key值和nuc970_key_th[0]設定第一個key相對應ADC得到電壓的範圍 ，nuc970_keycode[1]設定第二個key回報的key值和nuc970_key_th[1]設定第二個key相對應ADC得到電壓的範圍，以此類推，如下：

```
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},
};
```

- IIO ADC 設置

  使用者可以透過 IIO的架構來使用普通模式的ADC，可參考以下的設置來開啟IIO功能。

```
Device Drivers  --->
  <*> Industrial I/O support  --->
  [*]   Enable buffer support within IIO
  [*]   Enable triggered sampling support
  Analog to digital converters  --->
    <*> Nuvoton NUC970 Normal ADC driver
        Reference voltage selection (Internal bandgap, 2.5V)  --->
```

  其中，參考電壓有三種可以選擇，分別是”Internal bandgap 2.5V”，”Internal AVDD 3.3V”及”VREF input”。

  應用層使用方式如下：

```
$ cat /sys/bus/iio/devices/iio:device0/in_voltageX_raw
```

  其中，X為頻道(X=0~7)。

- Touch screen與tslib設置

  當使用touch screen時，可以透過修改 drivers/input/nuvoton/nuc970adc.c 中#define Z_TH的值，以防止誤判的情況發生，Z_TH可以根據實際得到的壓力值來調整，ADC Sample Counter需要根據不同的touch screen來調整，還可以透過tslib1.1來校正touch screen。

  在applications.tar.gz 解壓縮後，進入tslib-1.1資料夾後執行下列步驟：

  1. 編譯tslib-1.1
     - ./configure --prefix=$(pwd)/install --enable-static --enable-shared --host=arm-linux
     - make
     - make install
  2. 將$(pwd)/install中所有檔案複製到 $(rootfs)目錄下。

3. 修改 $(rootfs)/etc/profile 並增加下列幾行。

```
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. 修改 $(rootfs)/etc/ts.conf，如下：

```
# 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. 執行ts_calibrate校正程式，請依據屏的指示去點選畫面，完成以後執行ts_test測試程式，即可去測試touch screen。如果測試效果不好(無法點選特定的地方或畫線有很大的偏差)，建議再執行一次校正程式。

```
# 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. 執行ts_test，即可看到屏顯示下圖

```
# ts_test
```

- SCUART 智能卡串口模式設置

  NUC970 系列帶有2個智能卡接口. 這兩個接口除了智能卡功能外, 也能拿來當成基本的串口使用. 當系統自帶的串口不敷使用時, 可再擴充出兩個串口. 在智能卡串口模式下, SC_CLK 會當成傳送端使用, 而SC_DAT 則是當成接收端使用. 這兩個接口可以分別使能. 請依以下的說明來使能串口功能. 每個串口可以單獨的開關. 其中 SCUART0 有兩組管腳可選擇, 需要一併設置.

```
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
```

  在智能卡串口模式下, 設備文件會是 /dev/ttySCU0 以及 /dev/ttySCU1. 基本的操作與源生串口相同. 但是限制比原生的串口多, 例如傳送以及接收各只有四級的 FIFO, 不支援流量控制, 也無法支援 RS485, IrDA 等傳輸模式. 當系統串口足夠使用時, 應優先考慮使用原生的串口.

- Loop back裝置設置

Loop back裝置可讓系統讀寫一個文件就像讀寫block裝置一樣，這個文件可以是任何系統可以讀寫的文件系統，然後加載至指定的loop back裝置上即可使用。使能設置方式如下：

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

使用方式如下：
1. 產生文件檔

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

2. 格式化文件檔 (以FAT文件系統為例)

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

3. 加載文件檔

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

5.3.4 文件系統設置

- FAT 文件系統設置

  FAT 是 SD 卡以及 U 盤上常見的文件格式. 可以照以下的設置使能.

```
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
```

以 SD 卡的第一分區為例的話, 加載文件系統的命令是:

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

- JFFS2 文件系統設置

  JFFS2是 NAND flash 上使用的文件系統之一. 可依照以下的設置使能.

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

```
        [*]   JFFS2 write-buffering support
```

- ROMFS 文件系統設置
  ROMFS 文件系統是根文件系統使用的格式之一, 可以照以下的設置使能.

```
File systems  --->

  [*] Miscellaneous filesystems  --->

        <*>   ROM file system support

              RomFS backing stores (Block device-backed ROM file system
support)  --->
```

- YAFFS2 文件系統設置
  YAFFS2 是 NAND flash 上使用的文件系統之一， 需先使能 MTD 的 “Caching block device access to MTD devices Device drivers” 方可勾選。 可以照以下的設置使能：

```
File systems  --->

  [*] Miscellaneous filesystems  --->

        <*>   yaffs2 file system support

        <*>   Autoselect yaffs2 format

        <*>   Enable yaffs2 xattr support
```

  加載 YSFFS2 文件系統的命令是：

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

- exFAT 文件系統設置
  exFat為微軟開發出來新一代的文件系統，exFAT可補足FAT在單一文件大小及裝置總容量上的限制。可以依照以下的設置使能：

```
File systems  --->

  DOS/FAT/NT Filesystems  --->

        <*> exFAT fs support
```

  以 SD 卡的第一分區為例的話，加載 exFAT 文件系統的命令是：

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

- FUSE 文件系統設置及支援NTFS文件系統
  FUSE (Filesystem in Userspace)，是指完全在用戶態實現的檔案系統。使用者可以透過FUSE系統實現各種自訂的文件系統，較常見使用FUSE來實現用戶文件系統的有

NTFS-3G、SSHFS等等。以下將介紹如何透過FUSE實現微軟的NTFS系統(NTFS-3G)。
可以依照以下的設置使能FUSE文件系統：

NTFS-3G 是一個由Tuxera公司開發並維護的開源項目，目的是為 Linux 提供 NTFS 分區的的驅動程式。能夠安全快速的對 Windows NT （包括 Windows 2000、Windows XP、Windows Server 2003 和 Windows Vista）的檔案系統進行讀寫。目前NTFS-3G的最新版本是 Tuxera 公司於 2014年2月23日發佈的 ntfs-3g_ntfsprogs-2014.2.15。
下載位置為：http://www.tuxera.com/community/ntfs-3g-download/。
下載完後依照ntfs-3g上的使用手冊進行編譯，完成後使用下面命令即可使用。

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

- UBIFS文件系統設置

  UBIFS 是用於固態硬碟儲存裝置上，並與LogFS相互競爭，作為JFFS2的後繼檔案系統之一。UBIFS 在設計與效能上均較YAFFS2、JFFS2更適合 MLC NAND FLASH。例如：UBIFS 支援 write-back, 其寫入的資料會被 cache, 直到有必要寫入時才寫到 flash, 大大地降低分散小區塊數量並提高 I/O 效率。

```
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 使用FIQ

一般的中斷在某些時候有可能會被系統鎖住，進而影響此中斷的即時性。此時，就可使用此章節內所提的FIQ，來確保中斷的即時性。

- 內核設置

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

- 使用方式範例

初始化流程如下(以timer2為例)：

```
/*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);

}
```

其中，regs.ARM_r8必須為fiq handler的位址及regs.ARM_r10必須為REG_AIC_IPER的位置。另外需要再對timer2進行必要配置即可使用FIQ。

5.3.6 電源管理

Linux 內核有支持電源管理的功能, 讓系統進入掉電模式, 降低電力的消耗, 等到接收到喚醒信號, 再喚醒系統. 要支持電源管理功能, 請使能以下內核配置選項, 再重新編譯內核.

```
Power management options  --->

  [*] Suspend to RAM and standby
```

使用支持電源管理的內核時, 可以使用以下的命令讓系統進入掉電模式. 在這個模式中, 不必要的時鐘會被關閉, DDR 也會進入自刷新模式(self-refresh mode). 只有被使能地喚醒源能將系統自掉電模式中喚醒.

```
$ echo mem > /sys/power/state
```

請注意, 要降低系統功耗至最低, 進入掉電模式前, GPIO 的上拉/下拉也需要設置. 這部分與硬件設計相關. 要依所接的裝置決定. 請在 arch/arm/mach-nuc970/pm.c 中, nuc970_suspend_enter() 這隻函數的前頭加上相關的設定.

5.4 Linux 內核編譯

內核設置完成後, 在 linux-3.10.x 目錄下執行 “make” 命令, 即可編譯內核. 若是編譯過程沒有錯誤, 則產生的內核影像檔, 以及使用 zip 壓縮的內核影像檔會放置在源碼上一層的 image 目錄中.

```
$ 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 使用者程序

6.1 範例程序

NUC970 BSP 提供了一些範例程式在 applications 中. 下表列出了這些範例程序, 以及其內容

目錄	描述
alsa-utils-1.0.23	ALSA 命令列工具.* 交叉編譯命令如下: $ ./configure -host=arm-linux -disable-nls --disable-xmlto PKG_CONFIG_LIBDIR=/usr/local/arm_linux_4.8/usr/lib -disable-alsamixer $ make 播放混音設定範例如下: ● Fiel $./amixer set PCM 85% ● $./amixer set Headphone 90% 錄音混音設定範例如下: 來源為Mic時： ● $./amixer set “Mic Bias” on ● $./amixer set “Input PGA” 100% ● $./axmier set ADC 90% 來源為Line In 時： ● $./axmier set "Right Input Mixer R2" on ● $./axmier set "Left Input Mixer L2" on ● $./axmier set "L2/R2 Boost" 100%

	● $./axmier set ADC 90% 播放的命令: ● $./aplay <檔案名稱> 如要播放 BSP 內之音效範例檔，可使用下列命令： $ cd usr $./aplay –c 2 –f S16_LE alsa/8k2ch.pcm 錄音的命令: ● $./arecord –d 10 –f S16_LE –c2 –r8000 –t wav –D plughw:0,0 <檔案路徑及名稱> 同時錄放的命令： ● $./arecord -f S16_LE -r 8000 -c 2 -D plughw:0,0 | ./aplay
benchmark/netperf-2.6.0	網絡效能測試工具. 交叉編譯命令如下: $./configure --host=arm-linux $ make
busybox-1.22.1/	Busybox 源碼. 交叉編譯命令如下: 1. $ make menuconfig 2. Select applets to be build 3. $ make
lighttpd-1.4.39	lighttpd 源碼. 交叉編譯命令如下: 如果使用 Toolchain gcc 4.8: $ ./configure --host arm-linux --build pentium-pc-linux --

	without-zlib --without-bzip2 --without-pcre --target arm-linux
	
	$ make
	
	$ sudo make install
	
	如果使用 Toolchain gcc 4.3:
	
	$ ./configure --host arm-linux --build pentium-pc-linux --without-zlib --without-bzip2 --without-pcre --disable-ipv6 --target arm-linux
	
	$ make
	
	$ sudo make install
	
	lighttpd 的 lib 和 sbin 目錄會安裝在 /usr/local 底下，使用時必須將他們拷貝到 root file system.
	
	config_html_sample/ 目錄下有 lighttpd configuration file (lighttpd.conf) 以及 homepage html (index.html) 範例. 請將 lighttpd.conf 放在 sbin/ 目錄下，並在 sbin/ 目錄下新增子目錄 www，將 index.html 放在 www/ 目錄下.
	
	另外，請新增以下兩個檔案，分別是 lighttpd 的錯誤以及存取紀錄.
	
	/var/log/lighttpd/error.log
	
	/var/log/lighttpd/access.log
	
	開啟 lighttpd 步驟如下:
	
	● $ ifconfig eth0 192.168.0.100

	● $ cd /usr/local/sbin ● $ ./arm-linux-lighttpd start -f lighttpd.conf
demos/alsa_audio	音效範例程序 *
demos/cap	影像擷取範例程序 *
demos/can	CAN bus 範例程序 *
demos/crypto	加解密範例程序 *
demos/etimer	加強型時鐘範例程序 *
demos/gpio	GPIO 範例程序 *
demos/irda	紅外範例程序*
demos/lcm/	LCD 範例程序*
demos/sc	智能卡範例程序 *
demos/thread	Thread sample applications. *
demos/rtc	RTC 範例程序 *
demos/uart	UART 範例程序 *
demos/wdt	看門狗範例程序 *
demos/wwdt	窗口看門狗範例程序 *
demos/dma	DMA 範例程序 *
i2c-tools	i2c-tools工具 $ make
DPO_RT3070_LinuxSTA_V2.3.0.2_20100412	RT3070 無線 dongle開源驅動程序
wireless_tools.29	WiFi 設定工具包. 包含了 iwconfig, iwlist, iwpriv…

	等工具.
tslib-1.1	Touch screen庫，含校正及測試程序 *
yaffs2utils.tar.gz	yaffs2命令工具 $ make

lzo-2.09.tar.gz	壓縮/解壓縮工具.
	交叉編譯命令如下:
	$ cd lizo-2.09
	$ ./configure --host=arm-linux --prefix=$PWD/../install
	$ make
	$ make install

libuuid-1.0.3.tar.gz	產生獨立的序號工具. 交叉編譯命令如下:
	$ cd libuuid-1.0.3
	$ ./configure --host=arm-linux --prefix=$PWD/../install
	$ make
	$ make install

mtd-utils.tar.gz	mtd-utils源碼. 交叉編譯命令如下:
	需要使用到lzo-2.09.tar.gz套件和libuuid-1.0.3.tar.gz套件
	$ 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

Python-2.7.9	Python-2.7.9源碼. 交叉編譯命令如下:
	$ ./make_python.sh
	Note : 電腦必須安裝Python-2.7.9
	在rootfs環境變數中需要設定如下:
	$ export PYTHONHOME=/lib/python2.7
	$ export PYTHONPATH =/lib/python2.7

qt-everywhere-opensource-src-4.8.5	QT gui 源碼

	1. 如果有使用 tslib 來支持觸摸屏，請先修改如下文件： **qt-everywhere-opensource-src-4.8.5/mkspecs/qws/linux-nuc970-g++/qmake.conf** **指定 QMAKE_INCDIR 及 QMAKE_LIBDIR，如下：** **QMAKE_INCDIR = path to /tslib-1.1/install/include** **QMAKE_LIBDIR = path to /tslib-1.1/install/lib** 2.設定環境變數： **$ 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.交叉編譯命令如下： **$ make** 5. 編譯 example **$ cd /path/to/qt-everywhere-opensource-src-4.8.5/examples/dialogs/trivialwizard** **$ ../../../bin/qmake** **$ make** 6. Run QT a.Copy trivialwizard 到系統可以存取到的地方 b.設定 tslib 環境變數 **$ export QWS_MOUSE_PROTO=Tslib:/dev/input/event0** c.執行 **$ trivialzardwizard –qws**

minigui-3.0.12	交叉編譯步驟及使用方法如下:
	
	1. 編譯 libminigui-gpl-3.0.12 庫
	
	在libminigui-gpl-3.0.12目錄下，輸入下面的命令:
	
	$ ./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. 編譯 minigui-res-be-3.0.12
	
	在minigui-res-be-3.0.12目錄下，輸入下面的命令:

	$ ./configure --prefix=$PWD/../build
	
	$ make
	
	$ make install
	
	3. 編譯mg-samples-3.0.12 範例代碼
	
	在mg-samples-3.0.12目錄下，輸入下面的命令:
	
	$ 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. 所產生的範例執行檔將會放至../build目錄下
	
	5. 再將build目錄下所有的檔案拷貝至rootfs根目錄
	
	6. 依照下面所述，修改/etc/MiniGUI.cfg設置檔，並拷貝至與minigui執行檔同一目錄下。
	
	[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/
	7. 必需開啟內核中ps2 mouse功能。(請參考5.3.3章節)
	8. 執行執行檔

ethtool-4.6	ethtool 是一個標準的應用程序可以控制有線網卡的硬件及驅動. 例如控制是否系統要透過網口Magic Packet 喚醒. 要交叉編譯請使用以下命令: ./configure CC=arm-linux-gcc CFLAGS=-mach=armv5te --host=arm-linux;make 讓網口支援喚醒功能, 請使用 ./ethtool –s eth0 wol g 命令. 要關閉喚醒功能請使用 ./ethtool –s eth0 wol d 命令.

*. 若是內核沒有正確的設置驅動程序, 這些範例程序的執行結果會不正確.

6.2 交叉編譯

有時在一些專案中需要移植軟件到ARM 平台. 許多開源軟件都已支持交叉編譯, 此時只要依據這些軟件的說明文檔實行交叉編譯的設定即可.

但有時遇到不支援編譯的軟件時, 就需要手動來更改 Makefile. 一般來說更改過後的 Makefile 與

原始版本會相近, 只需要做小幅度的更動即可支援交叉編譯. 以下列了需要修改的部分

- 編譯工具的前贅字需要設置. 例如原本的 Makefile 設定 gcc 為編譯工具, 則在新的 Makefile 需要改成交叉編譯使用的 arm-linux-gcc. 其他例如 as, ld 等工具也須分別更改為 arm-linux-as 及 arm-linux-ld
- 庫文件以及頭文件所在的路經必須作相對應修改. 交叉編譯不使用 x86 系統下的 glibc. 而是使用工具鏈中所提供佔系統資源較少的 uClibc.

以下是一個簡單的交叉編譯 Makefile 可供參考.

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


ROOT = /usr/local/arm_linux_4.8/usr

LIB =$(ROOT)/lib

INC :=$(ROOT)/include


CC=arm-linux-gcc -O2 -I$(INC)

WEC_LDFLAGS=-L$(LIB)

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 版本歷史

版本號	日期	描述
0.08	Aug. 15, 2014	初版發布
0.09	Oct. 8, 2014	1. 修改內文文字敘述 2. 新增內核設置：USB Device, keypad, RTC, ADC/tslib
0.10	Oct. 24, 2014	1. 新增U-Boot SPL相關配置說明 2. 新增/修改Nu-Writer相關配置說明
0.11	Oct. 28, 2014	1. 新增Nu-Write燒錄U-Boot章節 2. 新增Linux 內核SCUART驅動設置
0.12	Nov. 26, 2014	1. 新增U-Boot MMC 相關配置說明 2. 新增U-Boot LCD 相關配置說明 3. 新增 Loop back裝置相關配置說明 4. 新增 exFAT/NTFS 相關配置說明 5. 新增U-Boot eMMC相關配置說明 6. 新增QT application 7. 修改tslib相關配置說明
0.13	Mar. 6, 2015	1. 新增 U-Boot bootm 命令說明 2. 新增 U-Boot UBI 命令說明 3. 修改 U-Boot mkimage 工具相關說明 4. 新增 U-Boot GPIO 相關配置說明 5. 新增 U-Boot Watchdog timer 相關配置說明

0.14	Apr. 8, 2015	1. 修改範例程式章節中 alsa-utils-1.0.23 使用描述 2. 修改內核設置LCD說明章節 3. 修改 U-Boot SPI 相關配置說明 4. 修改 U-Boot NAND 相關配置說明 5. 修改 U-Boot CONFIG_SYS_NAND_U_BOOT_OFFS 配置說明 6. 修改 U-Boot NAND AES secure boot 示範 7. 修改 U-Boot SPI 相關命令範例 8. 修改 U-Boot NAND 相關命令範例 9. 修改 U-Boot 環境變數命令腳本範例 10. 新增 U-Boot CONFIG_BOOTP_SERVERIP 配置說明 11. 新增 U-Boot CONFIG_ENV_RANGE 配置說明
0.15	May. 6, 2015	1. 修改默認設置描述 2. 修改NuWriter NAND/eMMC/SPI Read mode配置說明 3. 範例程序增加lzo-2.09/libuuid-1.03/mtd-utils/yaffs2utils配置說明 4. 新增UBIFS文件系統設置說明 5. 增加3.8.4章節製作FS型態Image 6. 增加內核FIQ使用的說明
0.16	May. 29, 2015	1. 增加YAFFS2 inband-tags Image 使用說明 2. 增加YAFFS2命令使用 3. 增加YAFFS2為根文件系統的開機設定 4. 增加UBIFS為根文件系統的開機設定 5. 增加 RTL8188 USB dongle 使用說明

0.17	Nov.06, 2015	1. 增加NFS開機功能說明 2. 增加SPI flash開機功能說明 3. 增加2D驅動配置說明 4. 增加使用者空間記憶體管理配置說明 5. 增加可選擇LCD驅動源顏色數說明 6. 增加使用aplay說明 7. 增加U-Boot go命令使用說明 8. 修改USB host選單設定說明 9. 增加RTL8188使用說明 10. 增加CAN驅動配置說明 11. 其它錯誤字修改
0.18	Apr.22,2016	1. 增加RTL8192使用說明 2. 新增 RTL8192 WIFI 說明。 3. 新增在U-Boot中如何使SPI開機更迅速的說明。 4. 更新 U-Boot中mkimage工具的說明。 5. 新增i2c-tools工具的說明。 6. 新增Python-2.9.9使用說明。 7. 新增如何使用SPI第二個SS的說明。 8. 新增在U-Boot中EMMC相關的環境變數說明。 9. 更新U-Boot中環境變數的說明。 10. 新增 U-Boot bootarg的說明。 11. 更新 BSP安裝路徑說明。 12. 新增 EBI and JPEG功能說明。 13. 更名ubinize.cfg. 14. 新增lighttpd功能說明。 15. 新增在U-Boot中搭配Tomato板子相關說明。 16. 新增使用UVC相關說明。 17. 新增使用arm_linux_4.8編繹工具說明。 18. 修改arm_linux_4.8 取代arm_linux_4.3. 19. 增加IIO ADC說明章節。

1.00	Nov. 8, 2016	1. 增加capture範例程式說明。 2. 增加系統睡眠/喚醒說明。 3. 修正MiniGUI章節。 4. 增加ethtool說明。 5. 修改EMAC Wol說明。 6. 增加DMA範例程式說明。 7. 增加WWDT說明。

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.