

NUC970 Non-OS BSP 使用手冊

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

Nuvoton is providing this document only for reference purposes of NUC970 microcontroller based system design. Nuvoton assumes no responsibility for errors or omissions.

All data and specifications are subject to change without notice.

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

內容

1	NUC970 Non-OS BSP 簡介.....	4
1.1	開發環境	4
1.2	開發板設置	4
2	BSP 內容.....	5
2.1	BSP 目錄架構.....	5
2.2	Non-OS BSP 內容	5
3	Nu-Writer 使用說明	6
3.1	簡介.....	6
3.2	驅動程式安裝.....	6
3.3	USB ISP 模式設置.....	10
3.4	芯片設置	11
3.5	DDR/SRAM 模式	11
3.6	eMMC 模式	12
3.7	SPI 模式	14
3.8	NAND 模式	16
3.9	MTP 模式.....	Error! Bookmark not defined.
3.10	PACK 模式.....	Error! Bookmark not defined.
3.11	燒錄U-Boot	24
3.12	解決無法啟動Nu-Writer的問題	26
4	U-Boot 使用說明	27
4.1	配置.....	27
4.2	目錄架構	35
4.3	編譯 U-Boot	37
4.4	NAND AES secure boot 示範.....	39
4.5	U-Boot 命令	48
4.6	環境變數	78
4.7	mkimage 工具.....	80
4.8	安全性問題	84
4.9	Watchdog timer	85
4.10	U-Boot LCD	86
4.11	GPIO	86
4.12	網路測試環境.....	88
4.13	注意事項.....	94
5	版本歷史.....	96

1 NUC970 Non-OS 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 為核心的系統級單芯片. 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 Non-OS BSP 包含了以下內容:

- NUC970 Non-OS 驅動程式
- U-Boot 映像檔, 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

Non-OS BSP 是使用 Keil IDE 環境開發. 偵錯的 ICE 則是支援了 ULINK2. 開發環境的使用並不屬於本文件的介紹範圍. 若是需要使用說明, 可以至 Keil 的官方網站 <http://www.keil.com/> 查詢.

NUC970 支援 J-TAG 介面. 使用者可使用 ICE 透過 J-TAG 介面下載程式到 RAM 中進行偵錯. 但是 NUC970 在上電後, J-TAG 介面是關閉的, 除非讓 NUC970 新進入 USB 開機模式, 並使用 NuWriter 建立連線以後, 或者是 NUC970 不處於安全模式, 並且成功執行儲存於 SPI/NAND/eMMC 的程式. NUC970 的 J-TAG 介面才可使用.

NUC970 Non-OS BSP 的 loader 跟 Linux BSP 使用了一套相同的開源 loader, U-Boot. U-Boot 的開發環境是在 Linux 系統中. 若有開發需求, 可下載 NUC970 Linux BSP, 並參考 NUC970 Linux BSP 的使用手冊來架設開發環境. 或是可以直接使用 BSP 裡面已經預先編譯好的執行檔. 若是系統是從 SPI/eMMC 開機, 可以不使用 loader, 直接執行主程式. 但在 NAND boot 時, 需考慮壞塊, 的處理, 建議使用 U-Boot 開機.

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 內容

2.1 BSP 目錄架構

Non-OS BSP 包含了四個目錄, 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 Non-OS 驅動程式, 第三方軟體, 以及範例程式.
Documents	BSP 相關文件
Loader	預先編譯好的 U-Boot 映像檔.
Tools	Windows 上的燒錄工具以及驅動程式

2.2 Non-OS BSP 內容

BSP 下的壓縮包解開後, 會有以下內容:

目錄名稱	內容
Driver	NUC970 各個周邊的驅動程式. 個驅動程式的 API 說明請參考在 Document 目錄下的 NUC970 Non-OS BSP Driver Reference Guide.chm 文件.
Library	NUC970 使用的函數庫. 例如USB Host.
SampleCode	驅動相關範例程式
Script	包含了Keil 連結時的 link script. 以及要進入偵錯模式使用的腳本.
ThirdParty	第三方軟體, 包含了FATFS 文件系統, 以及 LwIP 開源TCP/IP協議棧.

3 Nu-Writer 使用說明

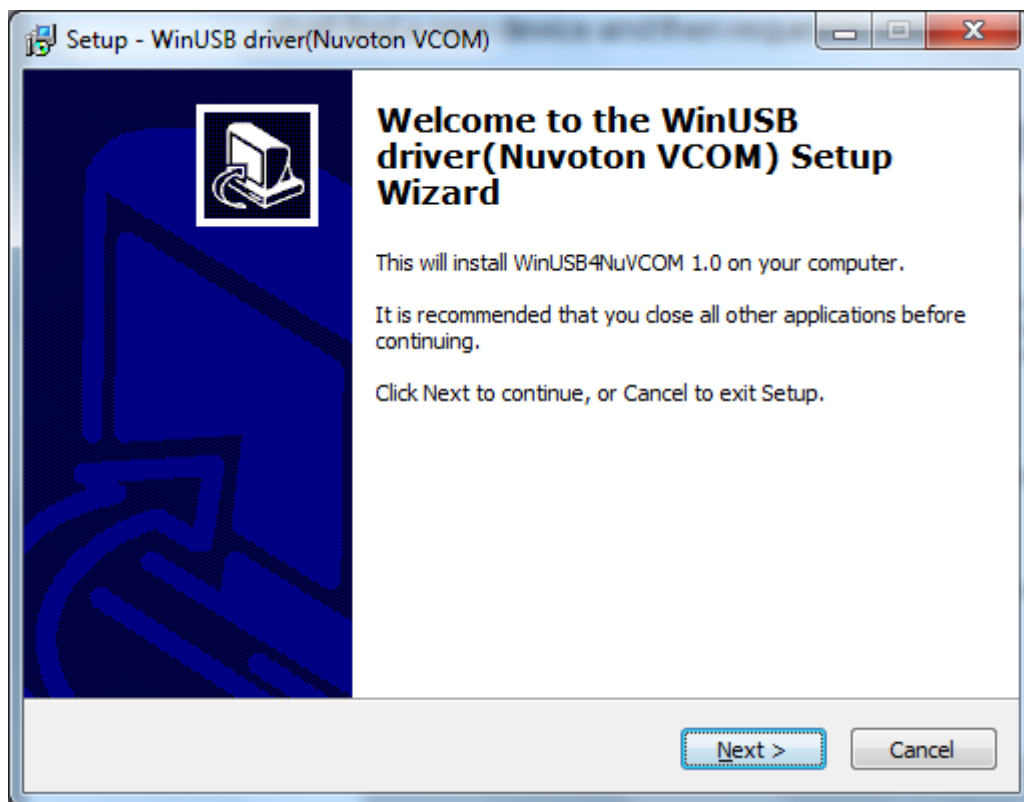
3.1 簡介

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。

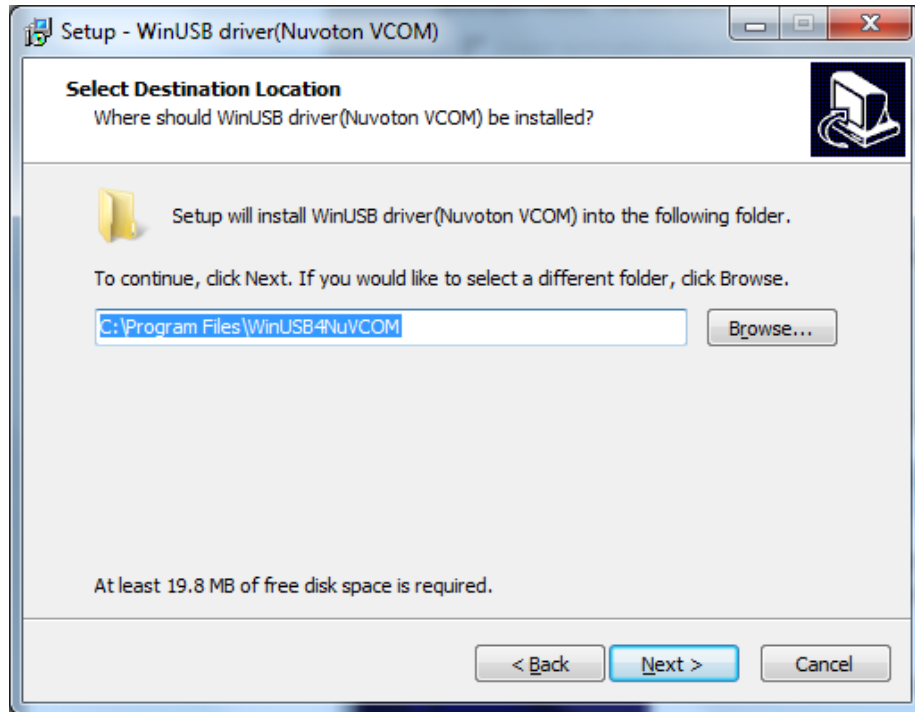
3.2 驅動程式安裝

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：

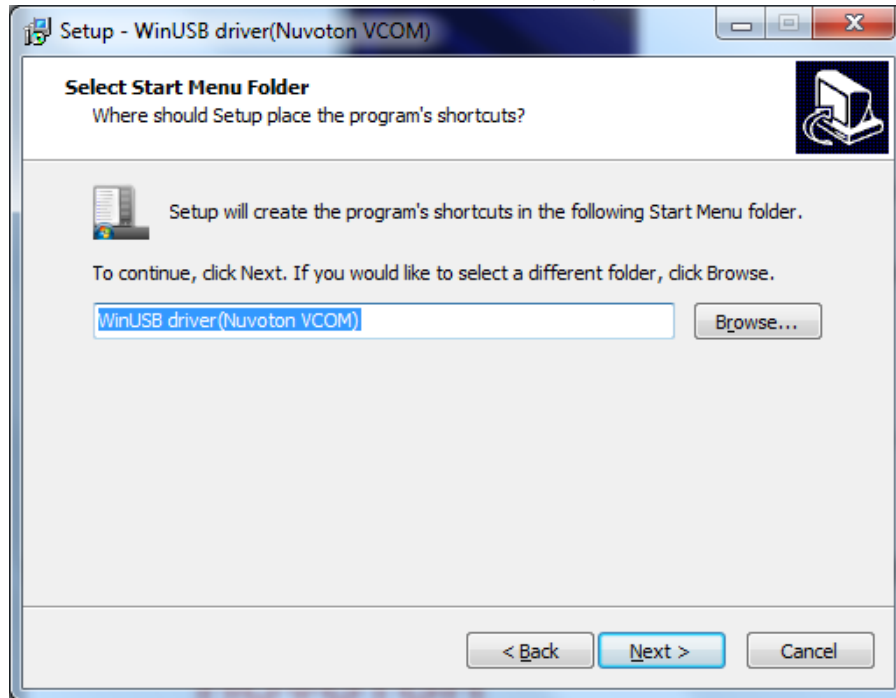
1. 將電腦與NUC970系列晶片開發板透過USB cable連接起來後，在電腦中執行WinUSB4NuVCOM.exe 開始安裝驅動程式。(在Linux BSP的Tools目錄中)
2. 開啟NUC970系列晶片開發板的電源之後，Windows 會發現新的設備，然後會要求你安裝驅動程式。



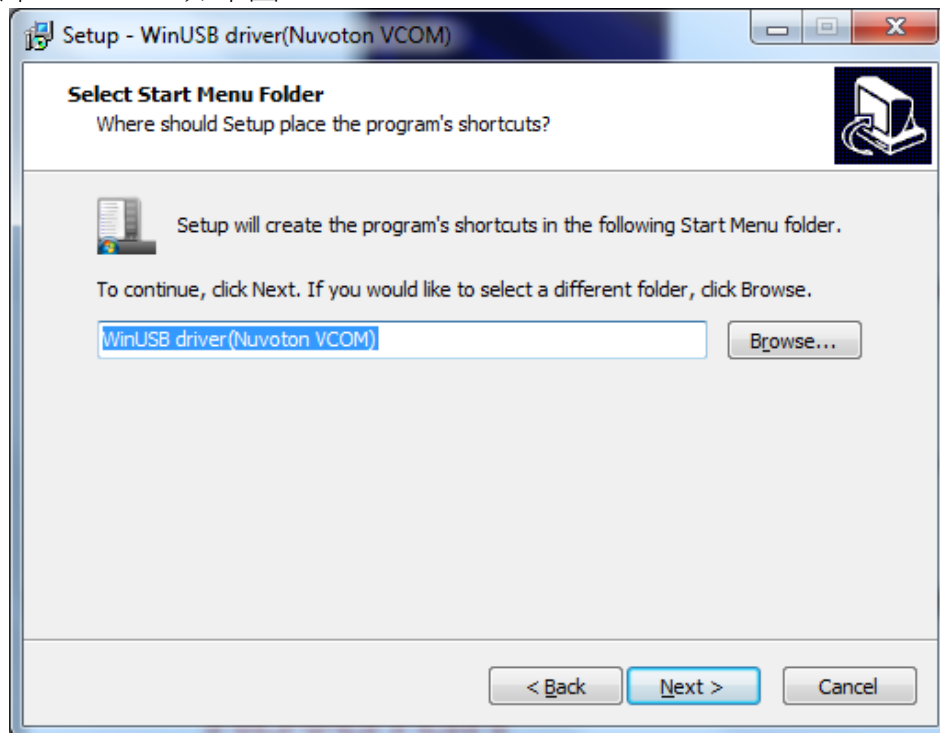
3. 按下 “Next” . 這個畫面告訴你即將要安裝WinUSB4NuVCOM 1.0 驅動程式. 如下圖：



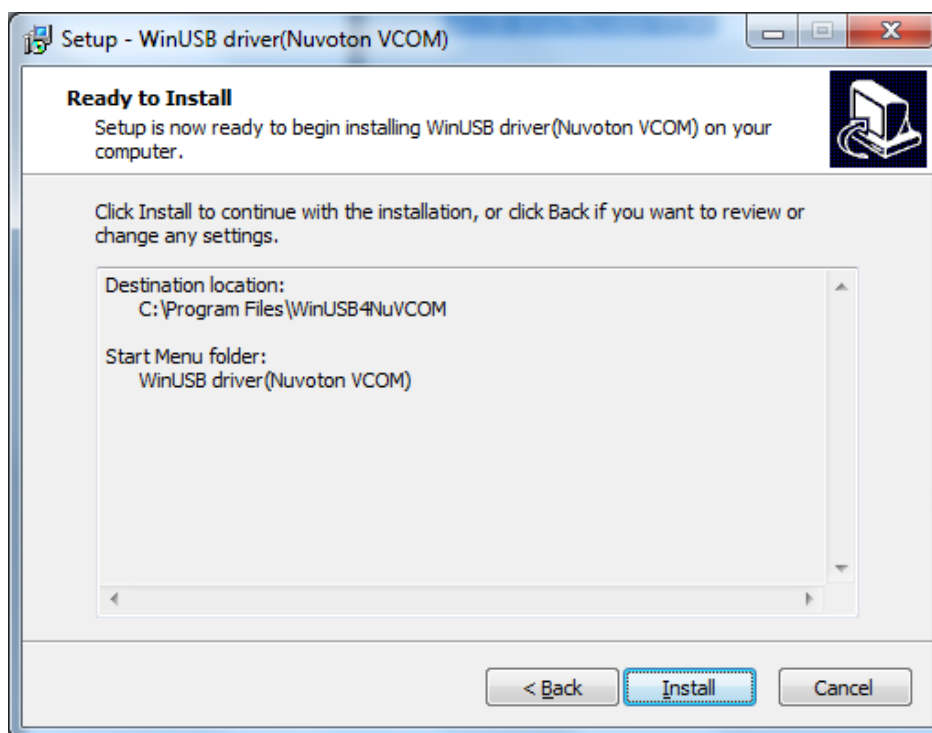
4. 選擇使用者想要安裝的路徑或使用預設的路徑, 確定以後按下 “Next” . 如下圖：



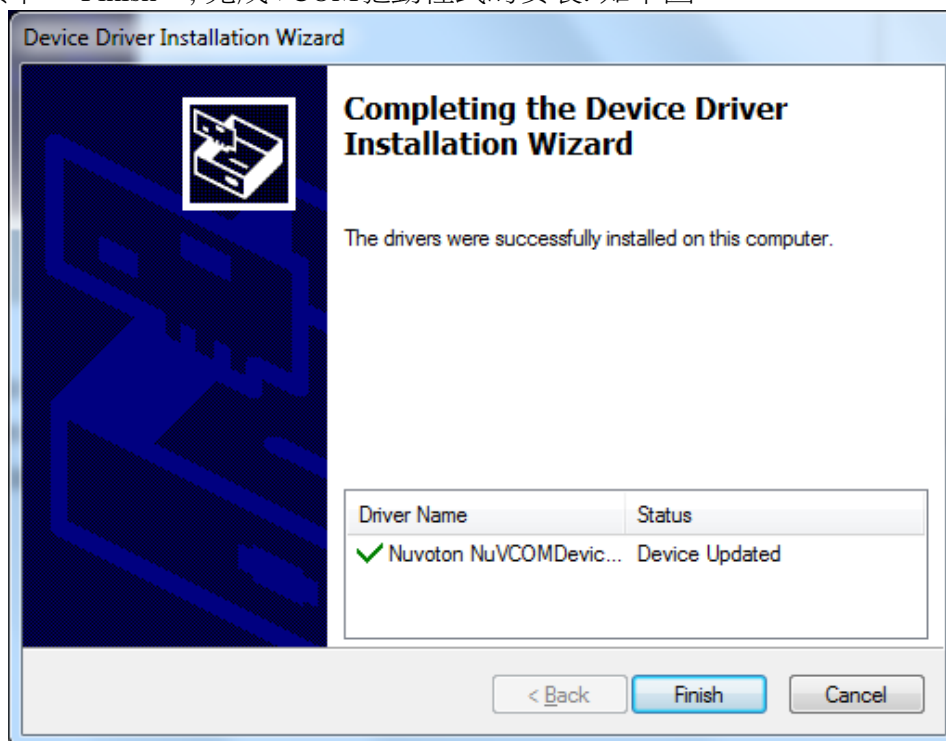
5. 按下 “Next” . 如下圖：



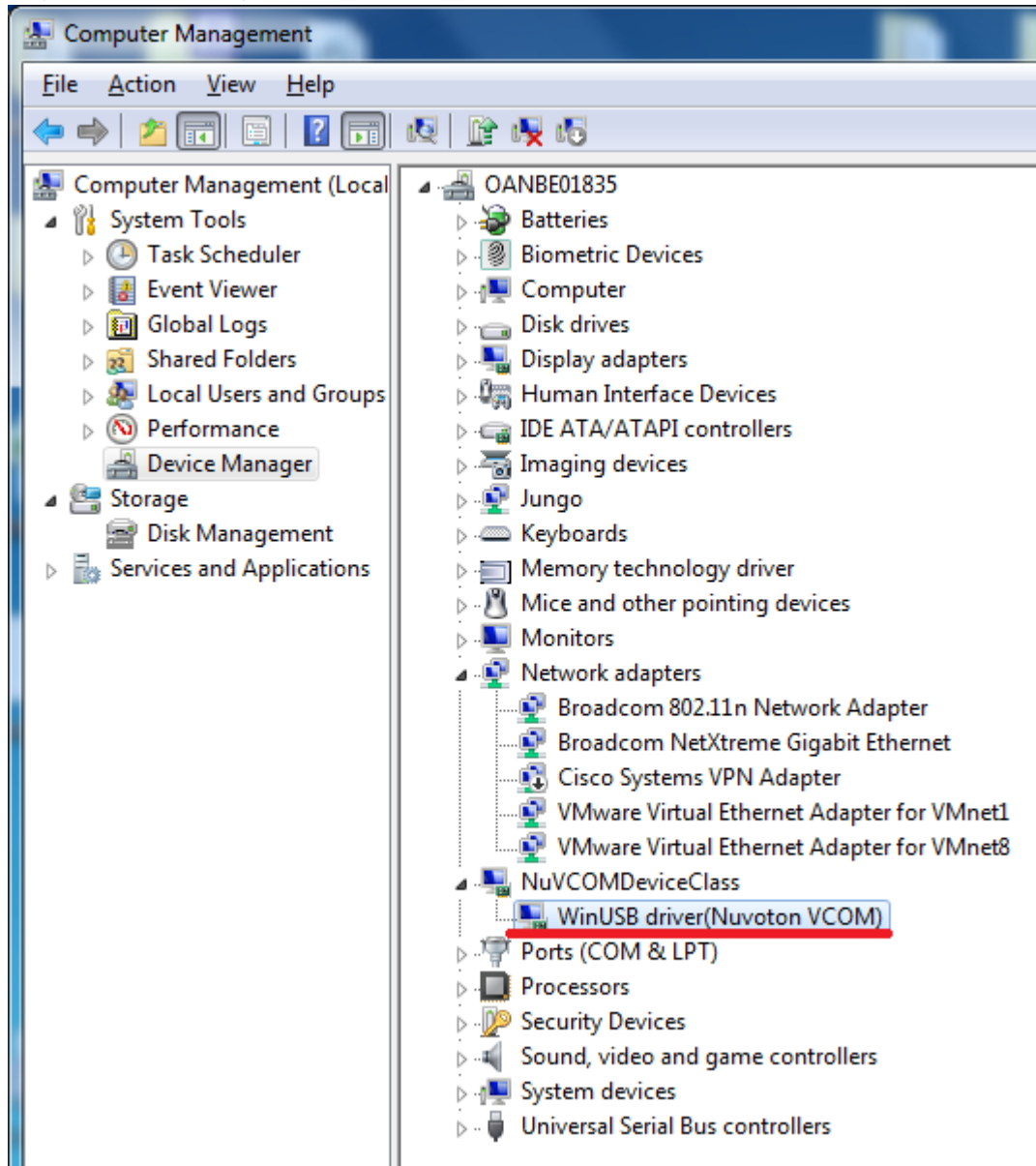
6. 按下 “Install” . 如下圖：



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

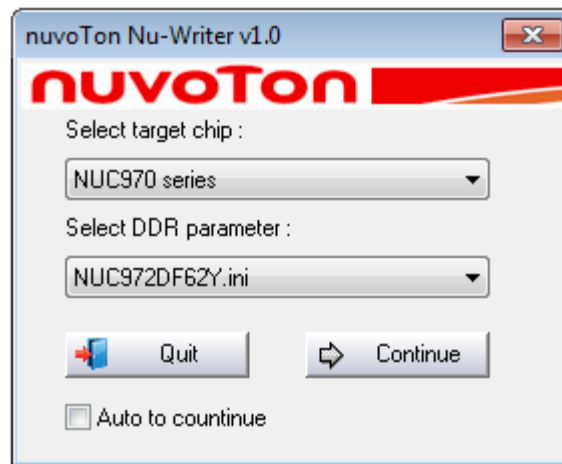
開啟NUC970系列晶片開發板的電源並且設定為USB ISP模式和開啟電腦上的Nu-Writer 工具, 即可開始使用.

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-xxxxxxx.7z(在BSP/Tools目錄下)壓縮包, 執行 “nuwriter.exe”, 第一個畫面如下.

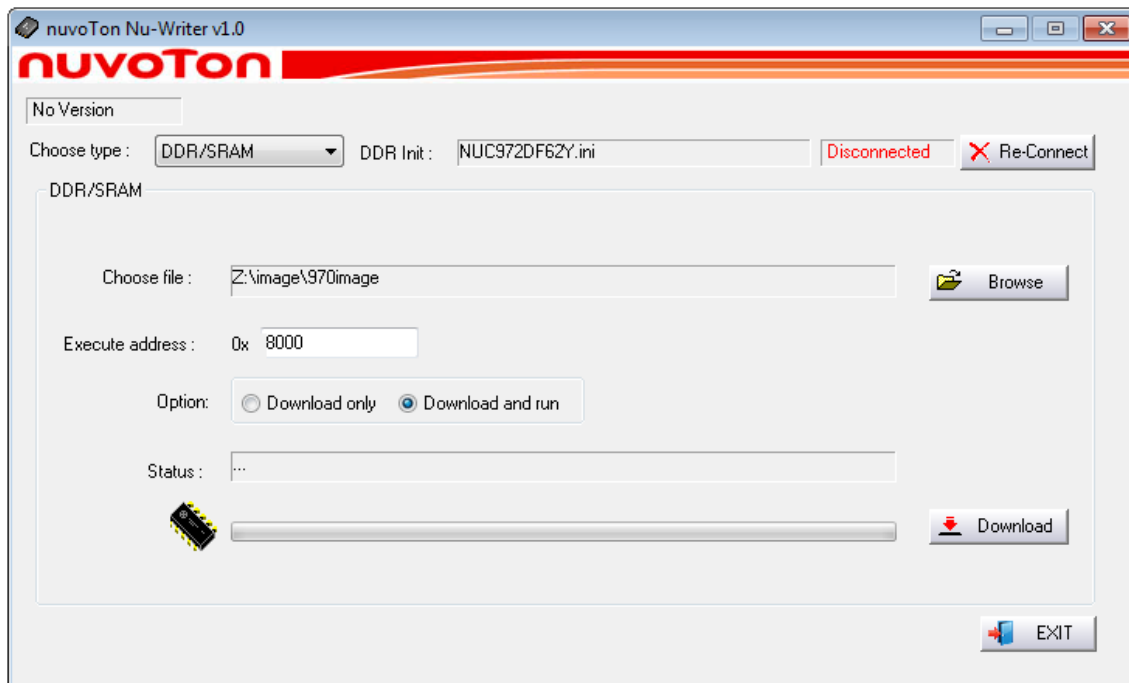
選擇目前晶片, 目前支援NUC970系列 (NUC972, NUC976... 等)晶片. 如果選擇NUC970系列晶片, 則必須選擇DDR參數, DDR參數依據NUC970系列芯片的PID來選擇. 選擇完成後按下 “Continue”, 即可開始使用Nu-Writer工具.



3.5 DDR/SRAM 模式

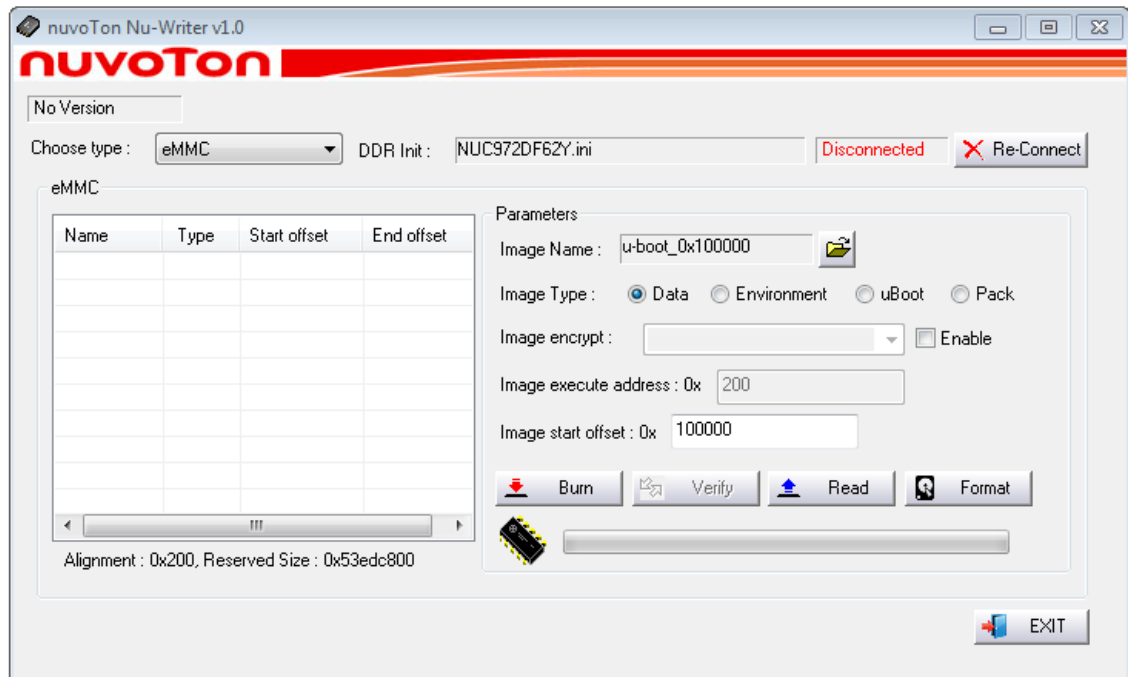
DDR/SRAM模式可以將Image檔案直接下載到DDR 或 SRAM 記憶體中. 操作步驟如下：

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

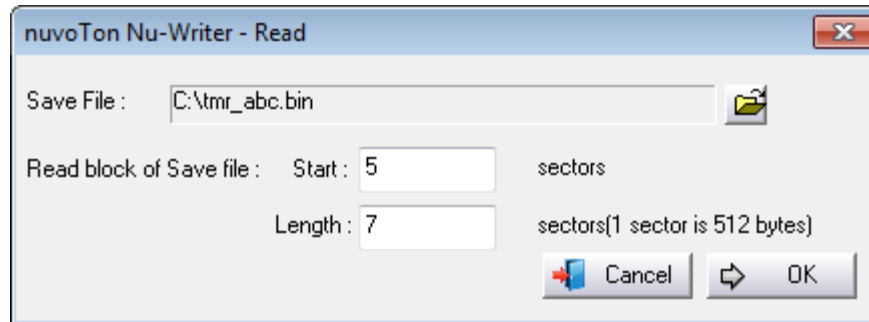
1. 選擇 “eMMC” 模式，表格只會紀錄當次燒錄的Image檔案,並不會讀取eMMC Flash中Image的資料。
2. 輸入 image檔案資料：
 - Image Name
 - Image Type
 - Image encrypt
 - Image execute address
 - Image start offset
3. 按下 “Burn”。
4. 等待進度表完成後，表格將會顯示這次燒錄完成的Image檔案。在完成以後，如果按下 “Verify” 即可確認燒入資料是否正確。

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。

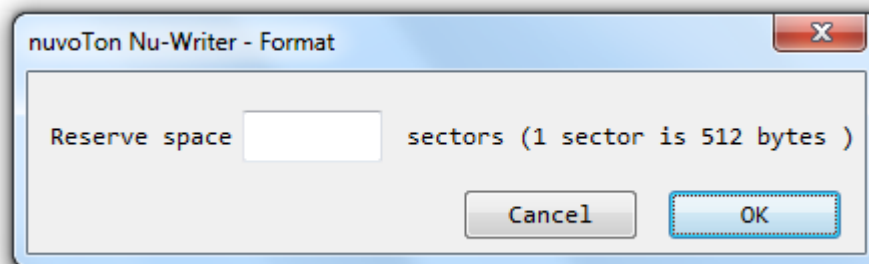


格式化 (FAT32)

3.6.3

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

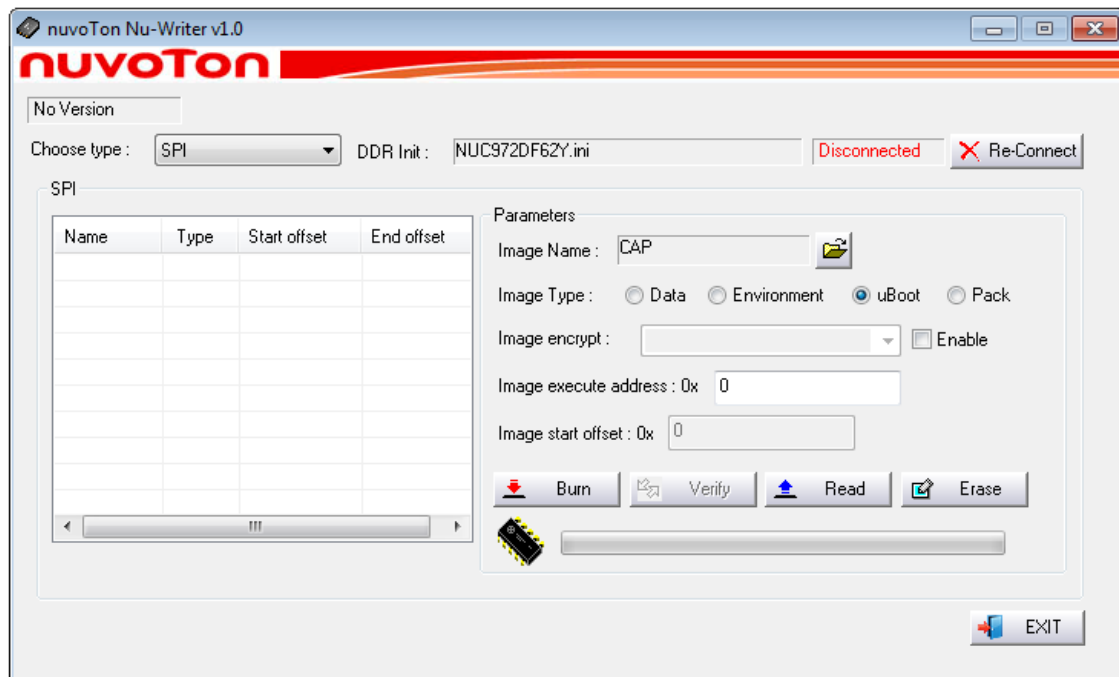
- 3.7.1 SPI 模式 可以將Image檔案燒入到SPI Flash中，並且將Image檔案型態設定為Data、Environment、uBoot、Pack, 四種型態中的其中一種。

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.

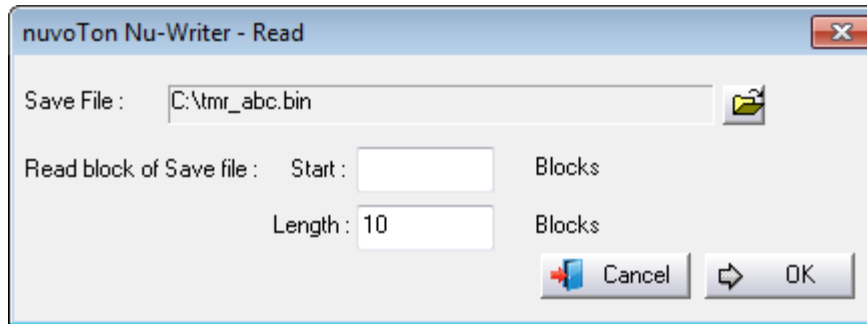


3.7.2

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。



移除 Image

3.7.3

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

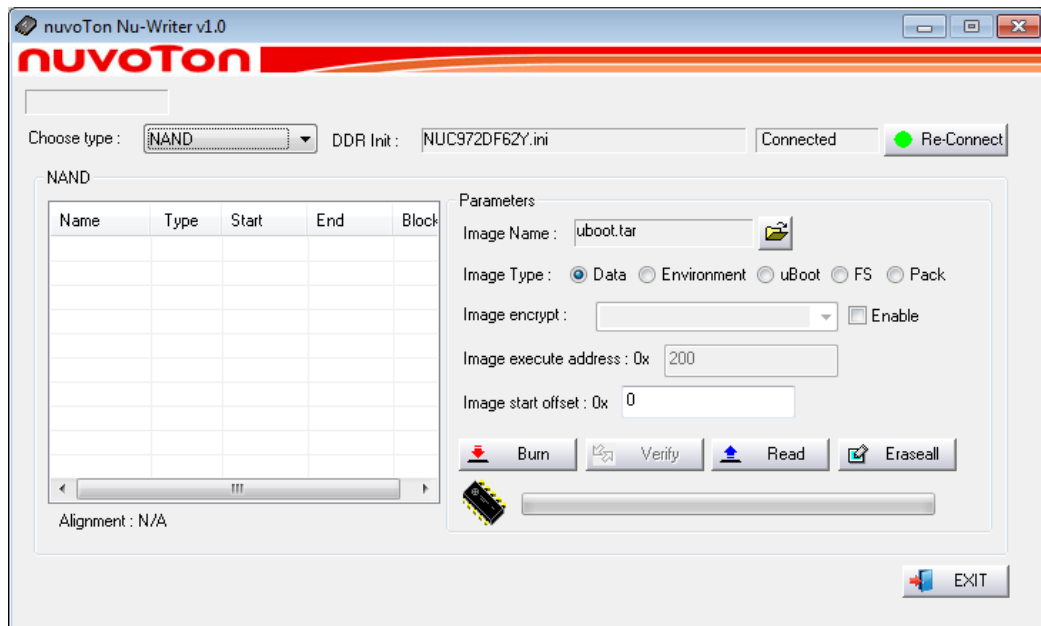
NAND模式 可以將Image檔案燒入到NAND Flash中，並且將Image檔案型態設定為Data、Environment、uBoot、FS、Pack, 五種型態中的其中一種。FS型態目前支援YAFFS2與UBIFS兩種檔案系統格式。這兩種格式都可以選擇FS型態，將做好的Image存放到NAND Flash對應的位址。讓使用者可以透過uBoot或Linux來讀取檔案系統。YAFFS2與UBIFS的Image檔的製作可以參考3.8.4章節

3.8.1

新增Image

依照下列步驟即可以完成新增Image檔案：

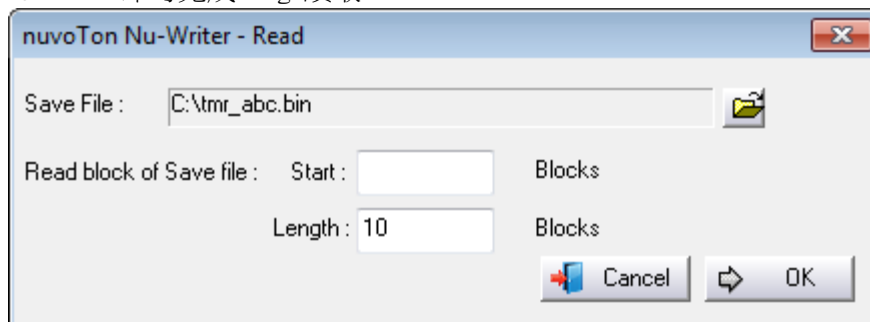
1. 選擇 “NAND” 模式, 表格只會紀錄當次燒錄的Image檔案, 並不會讀取NAND Flash中Image的資料。
2. 按下 “Add new” . (當選擇NAND模式時, 預設為 “Add new” 頁面)
3. 輸入 image檔案資料：
 - Image Name 選擇要燒錄的 loader 檔案
 - Image Type 選擇燒錄 Image的型態
 - Image encrypt 設置是否需AES加密, 若是, 設置秘鑰文件
 - Image execute address 設置 loader 執行位置, 依編譯設定輸入。
 - Image start offset 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。



移除 Image

依照下列步驟即可以完成移除Image檔案：

- 3.8.3. 選擇 “NAND” 模式。
2. 按下 “Erase all” ，即可完成移除Image。

製作FS型態Image

- 3.8.4. YAFFS2製作Image命令如下：

```
# mkyaffs2 -nuc970-ecclayout -p 2048 rootfs rootfs_yaffs2.img
```

--nuc970-ecclayout：使用NUC970硬體ECC編碼。

-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的地址。輸入下列命令即可將YAFFS2檔案系統掛在flash資料夾中：

```
# mount -t yaffs2 /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**Error! Reference source not found.**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 rootfs_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。

rootfs_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
```

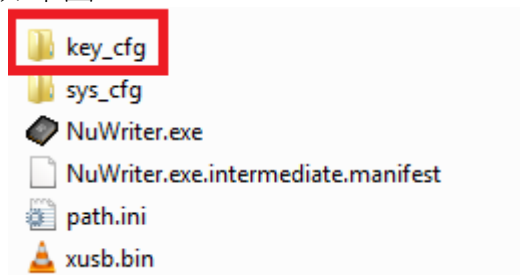
UBIFS相關指令可以在mtd-utils套件中找到。UBIFS文件系統設置可以參考Error! Reference source not found.章節。

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
3.9.1 NUC970系列晶片使用到存儲體(eMMC, NAND, SPI FLASH)中的程式碼。

新增key檔案

1. 進入資料夾key_cfg如下圖.

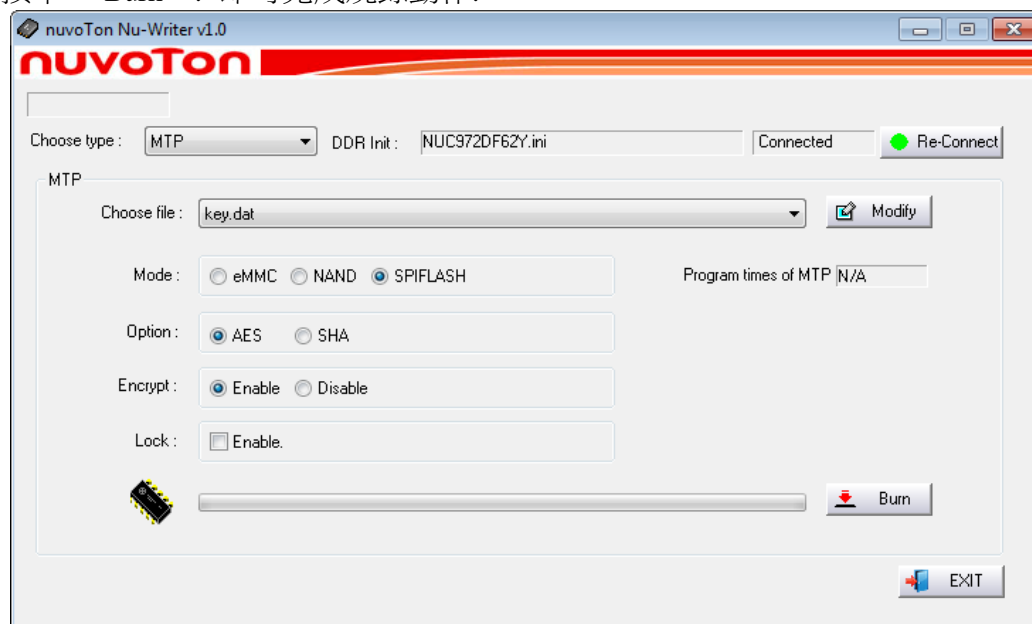


2. 建立文字檔和輸入密碼, 密碼格式如下, 第一行一定是 256. 之後連續 8 行大端模式密鑰.

256

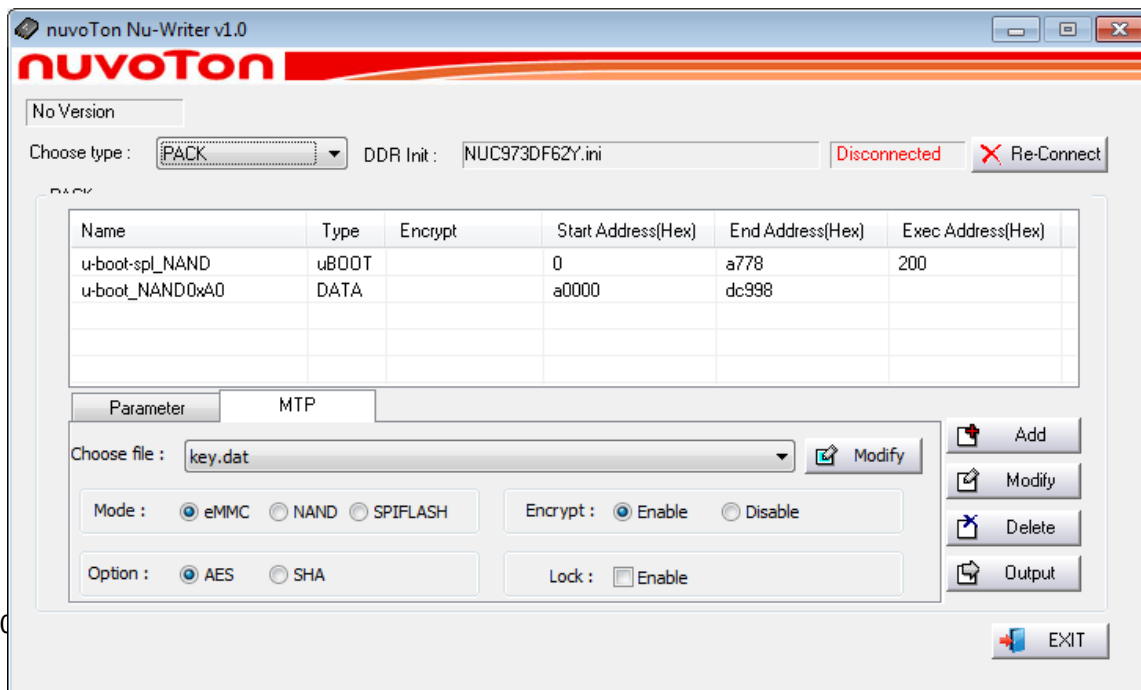
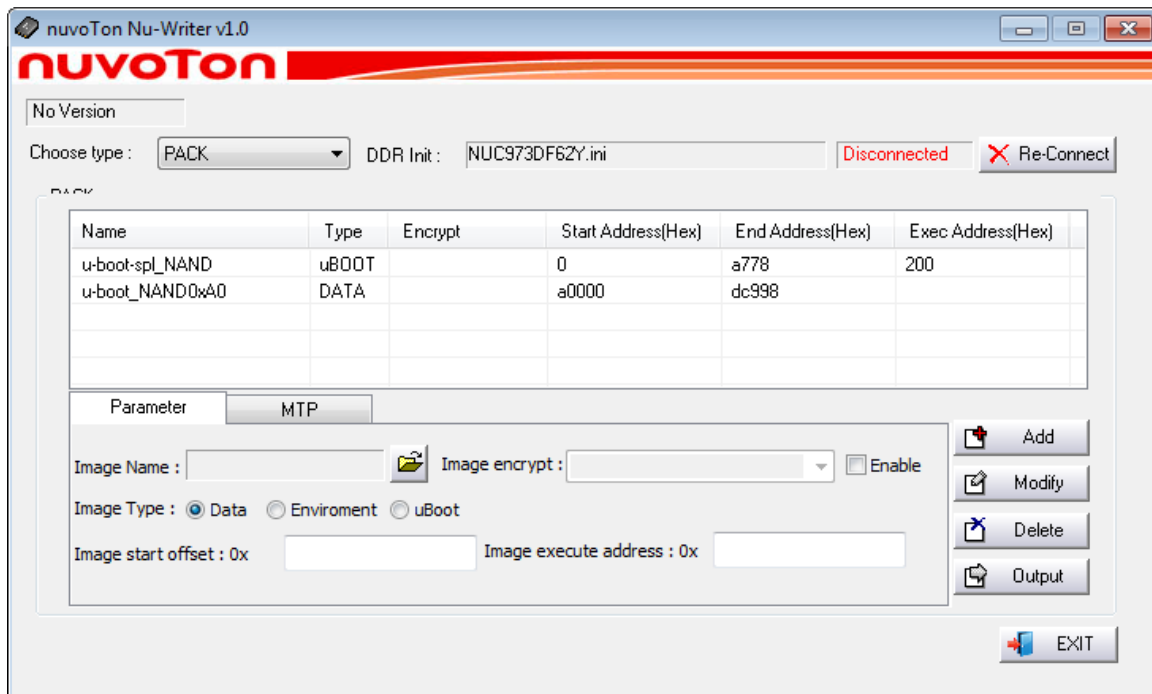
```
0x12345678
0x23456789
0x3456789a
0x456789ab
0x56789abc
0x6789abcd
0x789abcde
0x89abcdef
```

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

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” 。

輸出pack檔案

依照下列步驟即可以完成輸出pack檔案：

- 3.10.4 選擇 “Pack” 模式。
2. 按下 “Output” 。
3. 選擇儲存的檔案，按下open即可。

燒錄pack檔案

3.10.5 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” 。

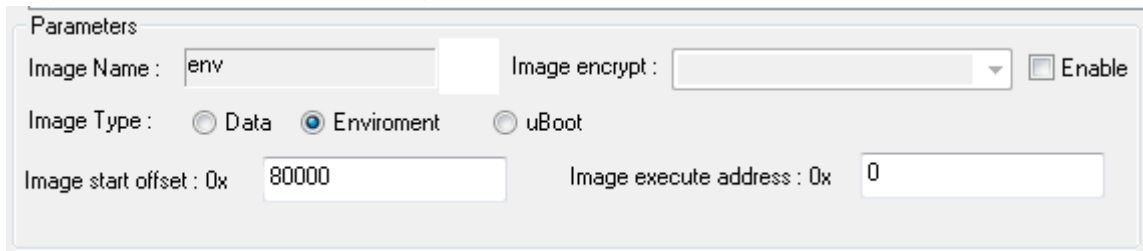
3.10.6 製作和燒錄pack範例

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：





Parameters

Image Name : env Image encrypt : ▼ ☐ Enable

Image Type : ☐ Data ☒ Enviroment ☐ uBoot

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

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

Parameters

Image Name :

Image encrypt : ☐ Enable

Image Type : ☐ Data ☐ Enviroment ☒ uBoot

Image start offset : 0x

Image execute address : 0x

5. 按下 Add 。

6. 按下 選u-boot.bin，並設定如下：

Parameters

Image Name :

Image encrypt : ☐ Enable

Image Type : ☒ Data ☐ Enviroment ☐ uBoot

Image start offset : 0x

Image execute address : 0x

7. 按下 Add 。

8. 按下 Output ，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：

Parameters

Image Name :

Image Type : ☐ Data ☐ Environment ☐ uBoot ☒ Pack

Image encrypt : ☐ Enable

Image execute address : 0x

Image start offset : 0x

3. 按下 Burn ，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。

燒錄所需檔案

4. u-boot.bin (預設 offset 為 0xA0000，執行位置為 0xE00000)
 - 3.11.1 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。

U-Boot 環境變數檔案(env.txt)內容說明

- 3.11.2 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。

燒錄至SPI Flash

1. 選擇 “SPI” 模式。
- 3.11.4 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。

燒錄至eMMC

- 3.11.5 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 Redistributable” 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_ETH0
//#define CONFIG_NUC970_ETH1
#define CONFIG_CMD_NET
#define CONFIG_NUC970_ETH
#define CONFIG_NUC970_PHY_ADDR    1
#define CONFIG_ETHADDR            00:00:00:11:66:88
```

```
#define CONFIG_SYS_RX_ETH_BUFFER      16 // default is 4, set to 16 here.
#define CONFIG_NUC970_CONSOLE

#define CONFIG_SYS_ICACHE_OFF
#define CONFIG_SYS_DCACHE_OFF
```

- CONFIG_SYS_USE_SPIFLASH: 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_JFFS2: 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_SYS_MAX_NAND_DEVICE 1
#define CONFIG_SYS_NAND_BASE      0xB000D000
#ifdef CONFIG_ENV_IS_IN_NAND
#define CONFIG_ENV_OFFSET          0x80000
#define CONFIG_ENV_SIZE            0x10000
#define CONFIG_ENV_SECT_SIZE      0x20000
#define CONFIG_ENV_RANGE           (4 * CONFIG_ENV_SECT_SIZE) /* Env range :
0x80000 ~ 0x100000 */
#define CONFIG_ENV_OVERWRITE
#endif

```

```
#endif

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

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

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

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

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

#endif //CONFIG_NAND_SPL
```

- CONFIG_NAND_NUC970: 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_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_WINBOND1
#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 512
#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)

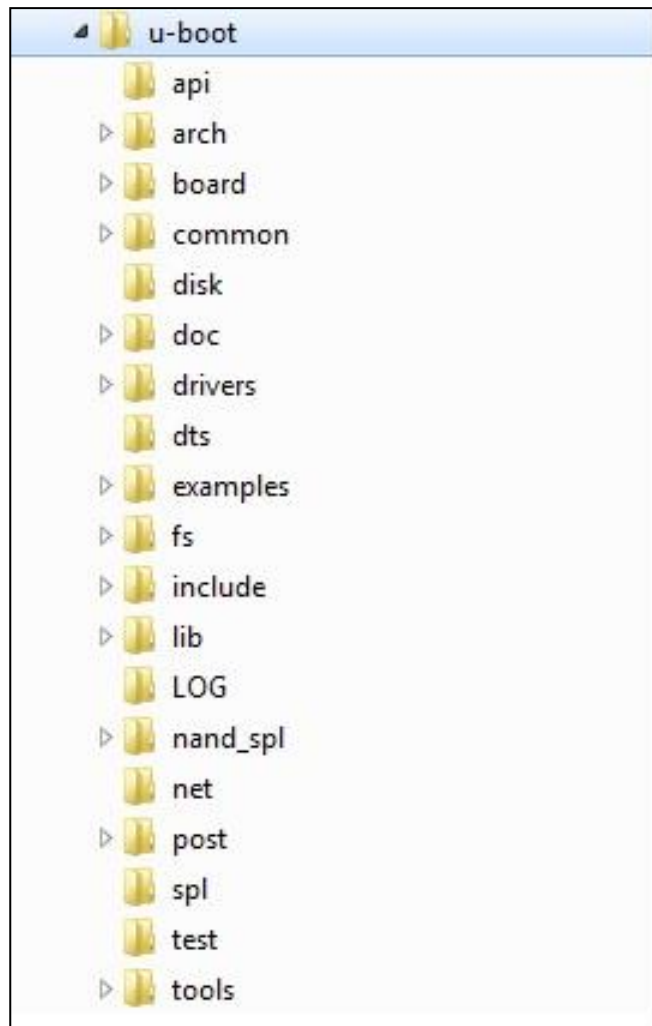
#define CONFIG_STACKSIZE (32*1024) /* regular stack */

#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

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

編譯命令

4.3.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

Main U-Boot 鏈結位址

- 4.3.3 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。

編譯 Main U-Boot 以及 SPL U-Boot

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

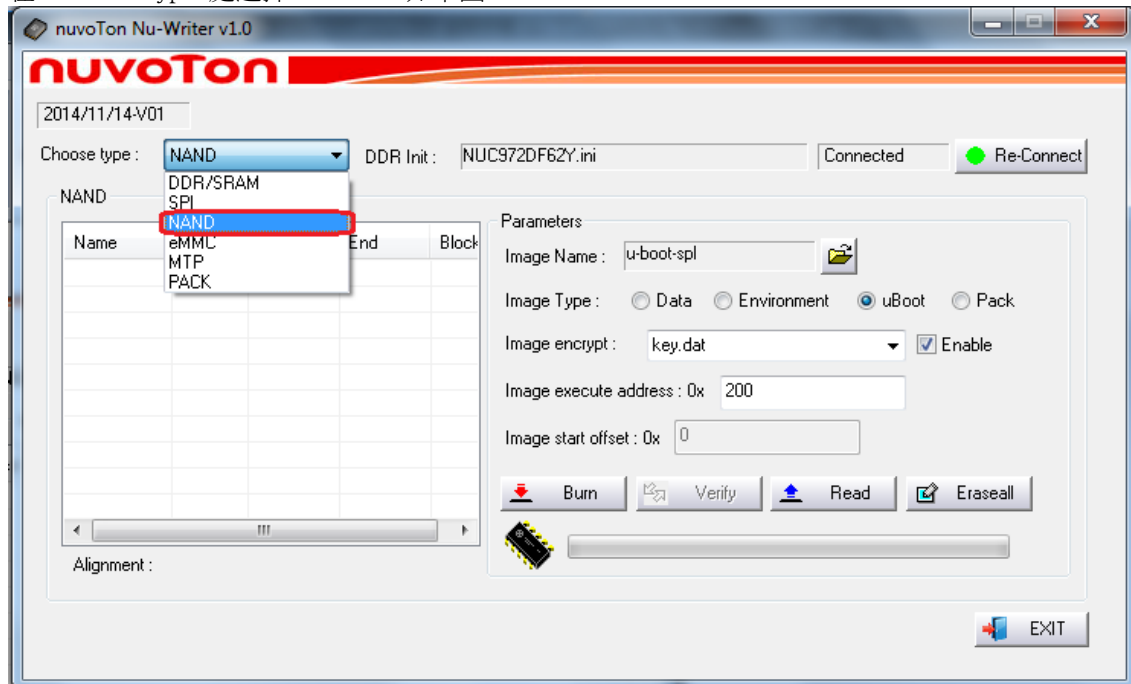
(make 的參數 O 是指定編譯產生的檔案要放到哪個目錄)

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

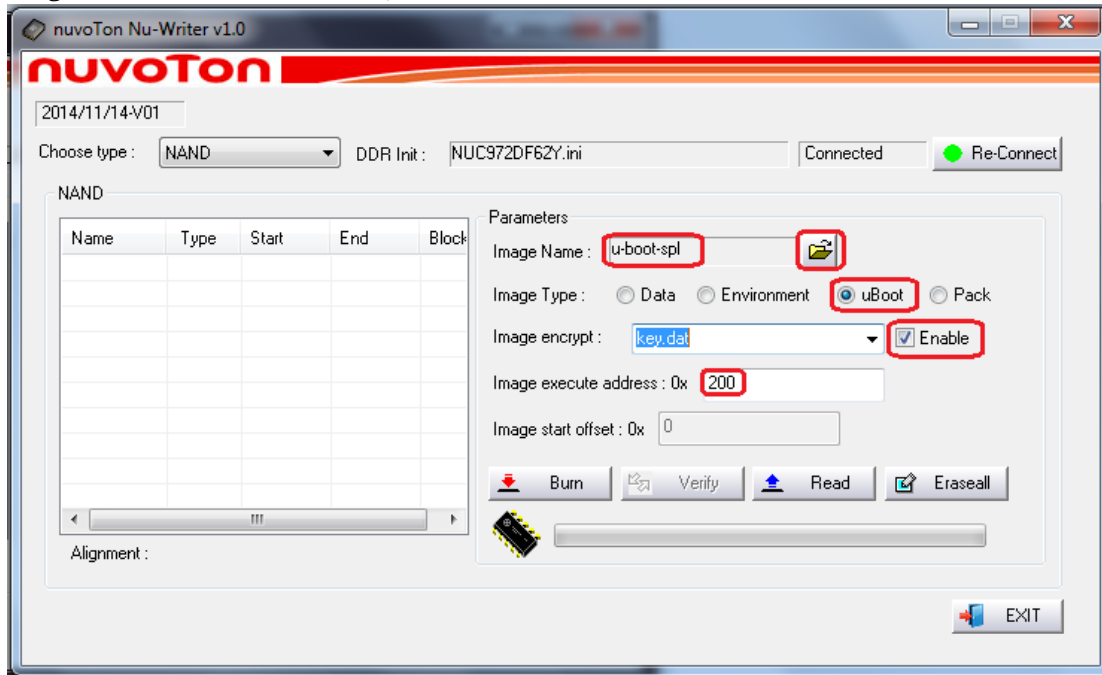
4.4.2 燒錄 SPL U-Boot

在 Choose type 處選擇 “NAND”，如下圖。

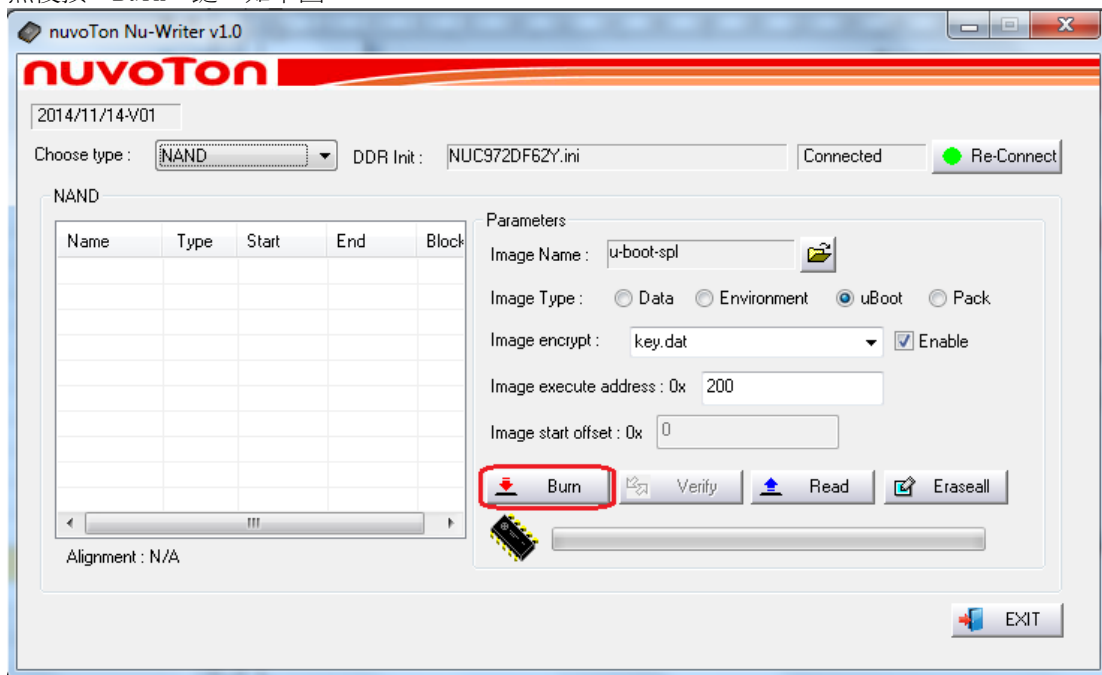


接著設定 Parameters，如下圖，

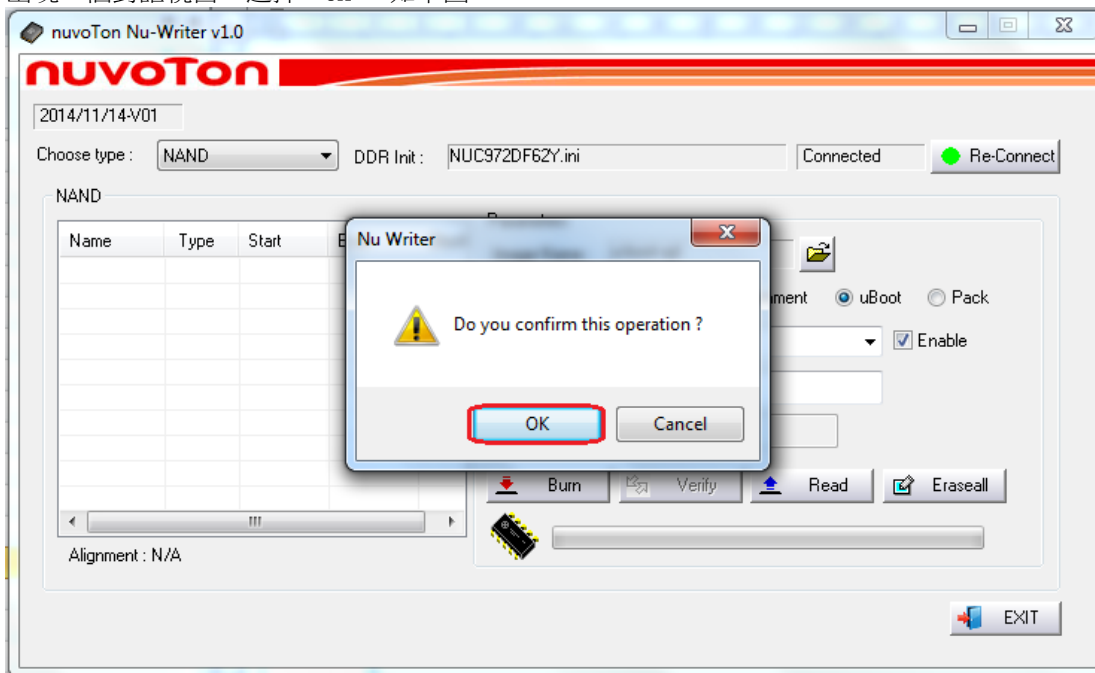
Image Name: 選取 u-boot-spl.bin ,
 Image Type: 選取 uBoot ,
 Image encrypt: 勾選 Enable
 Image execute address: 0x 填寫 200



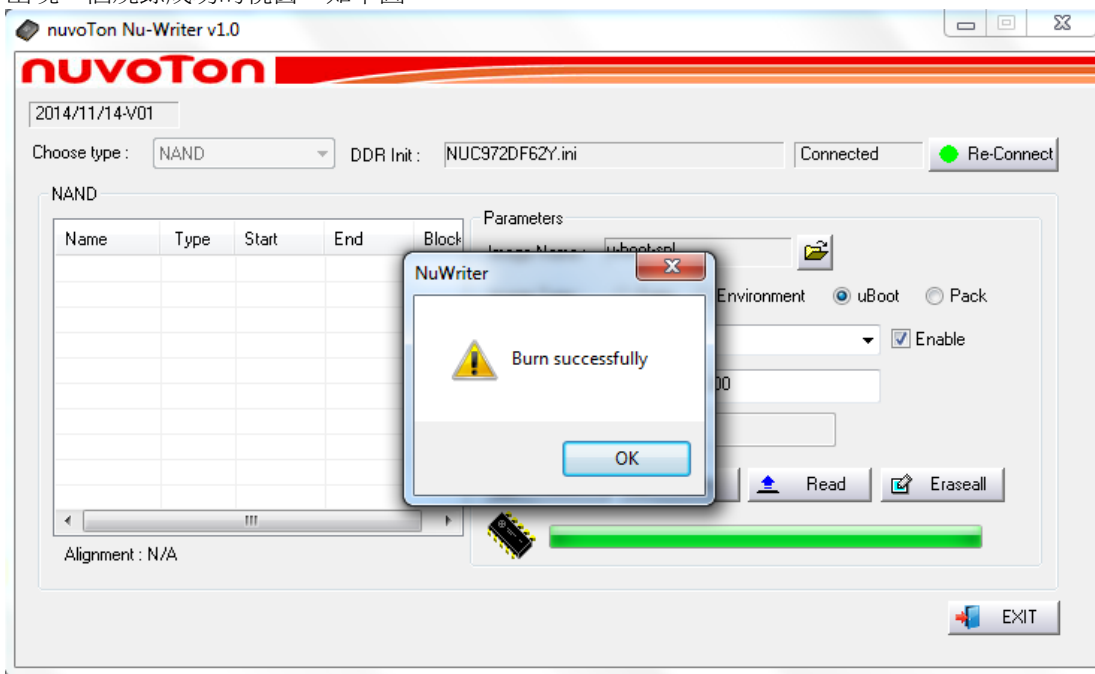
然後按 “Burn” 鍵，如下圖



出現一個對話視窗，選擇“OK”，如下圖，

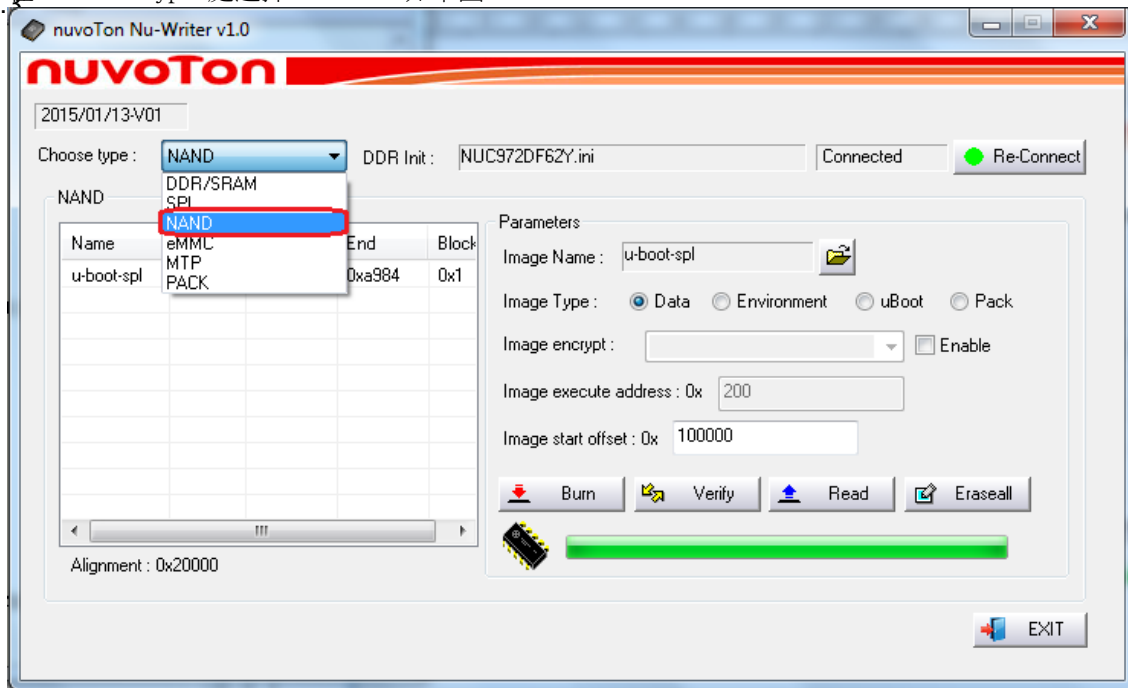


出現一個燒錄成功的視窗，如下圖，

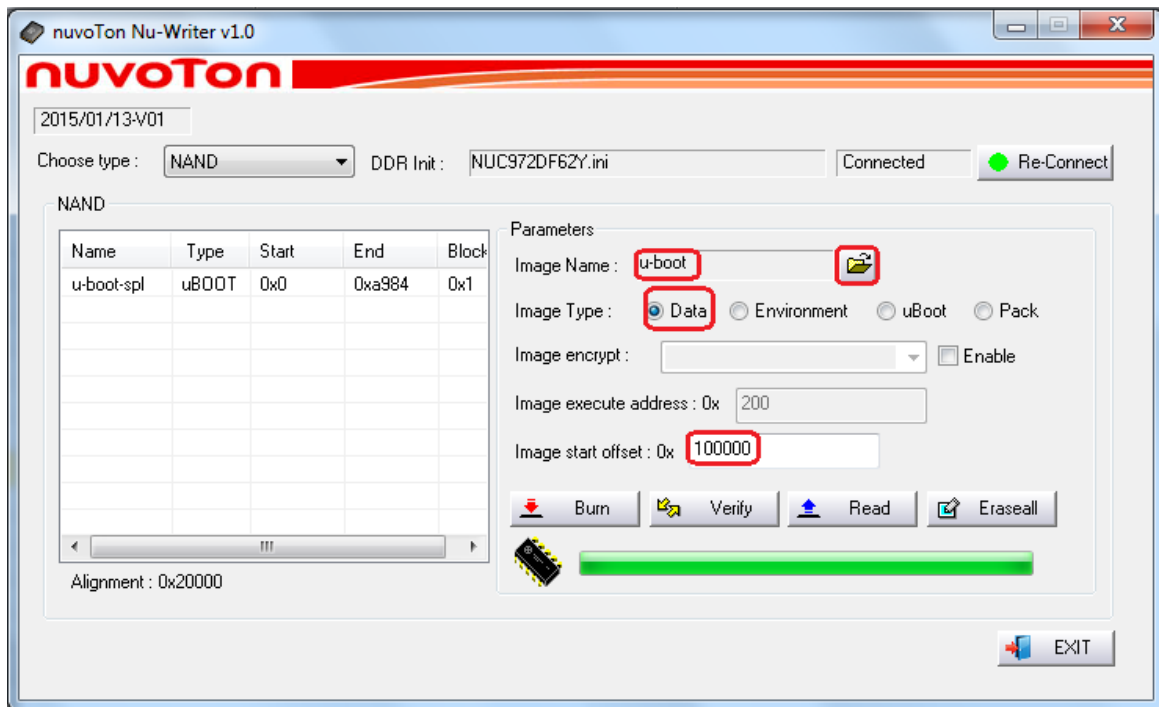


燒錄 Main U-Boot

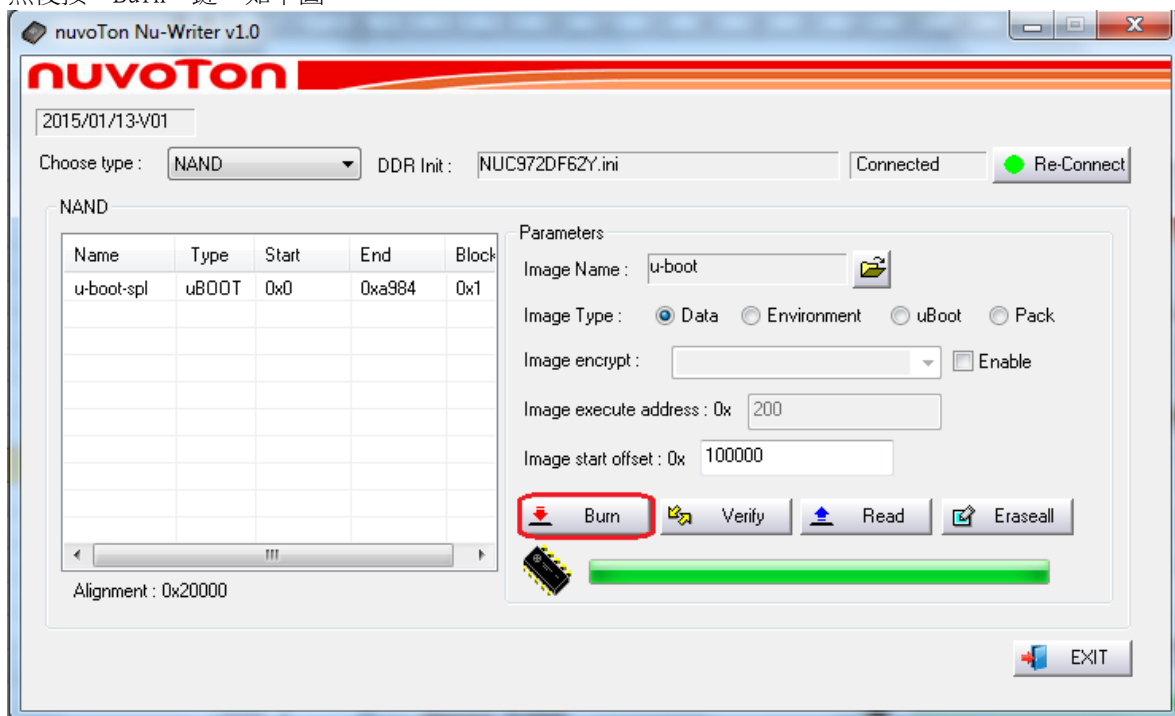
4.4. 在 Choose type 處選擇 “NAND”，如下圖。



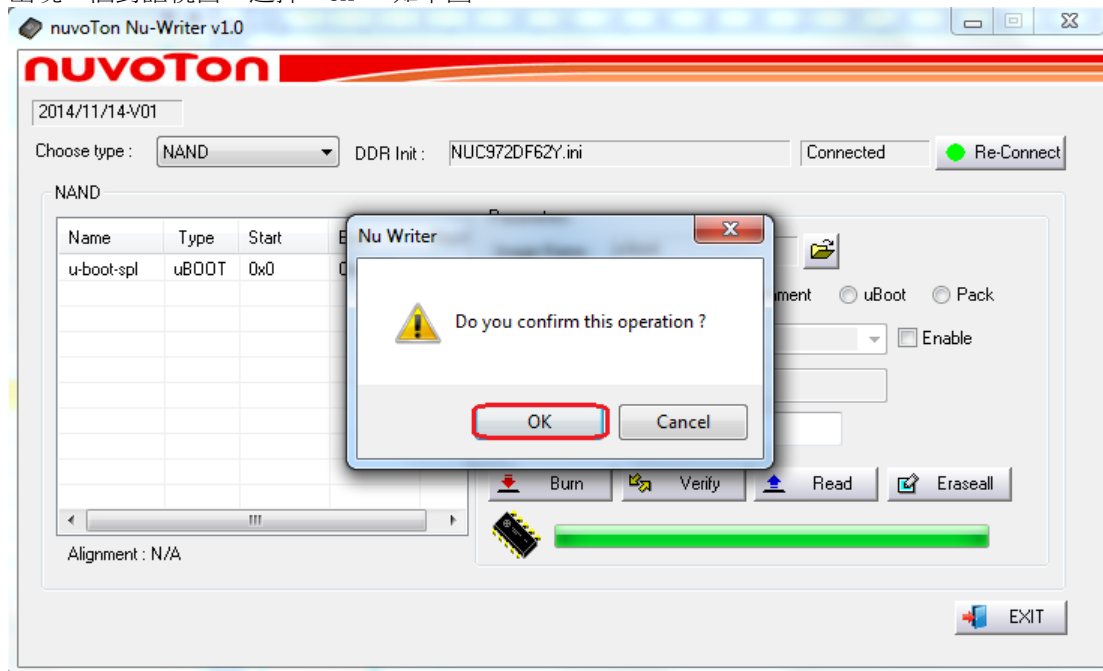
接著設定 Parameters，如下圖，
 Image Name: 選取 u-boot.bin，
 Image Type: 選取 Data，
 Image execute address:0x 填寫 100000



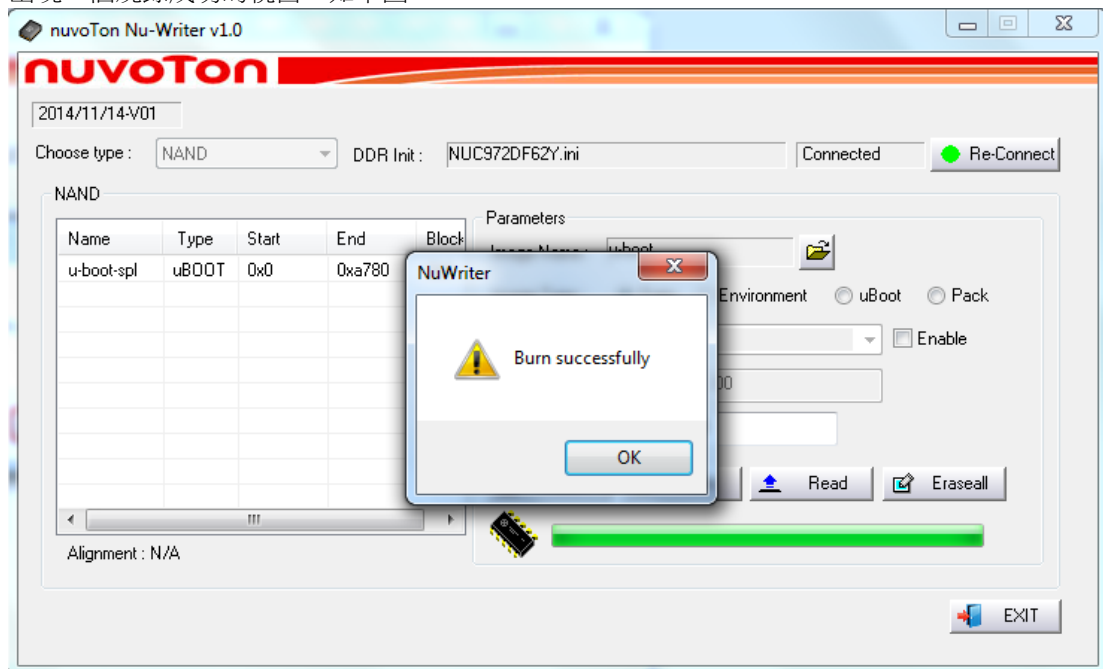
然後按“Burn”鍵，如下圖



出現一個對話視窗，選擇“OK”，如下圖，

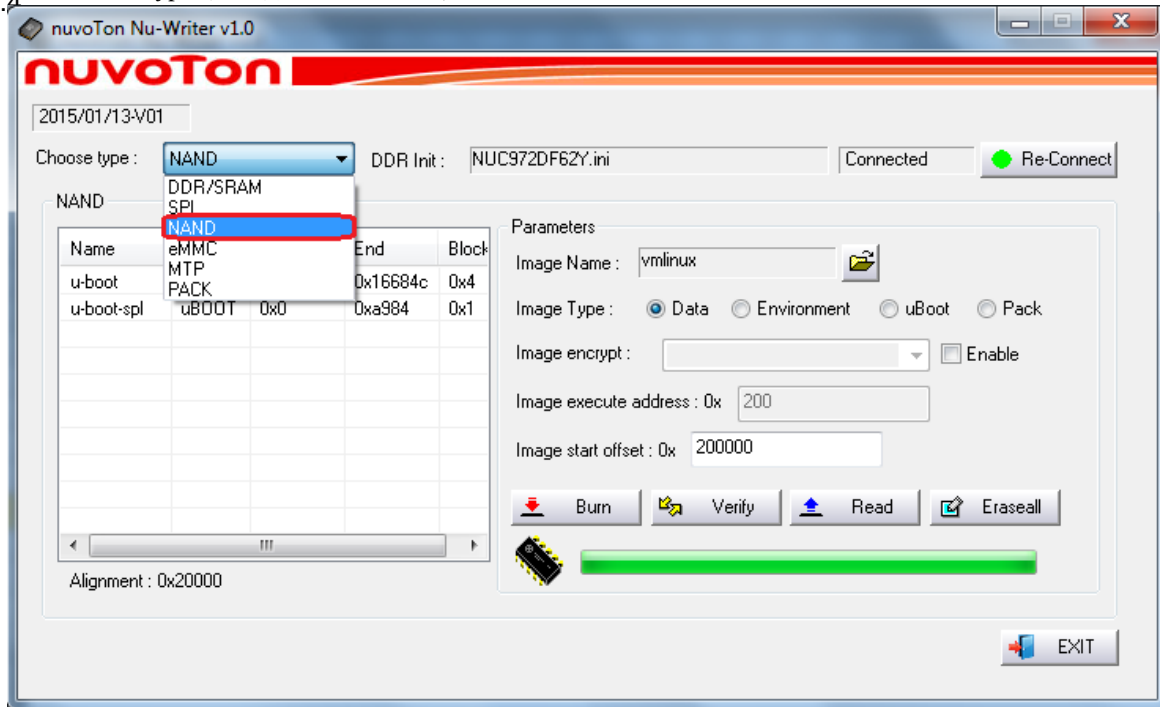


出現一個燒錄成功的視窗，如下圖，



燒錄 Linux 內核

4.4.4 在 Choose type 處選擇 “NAND”，如下圖。

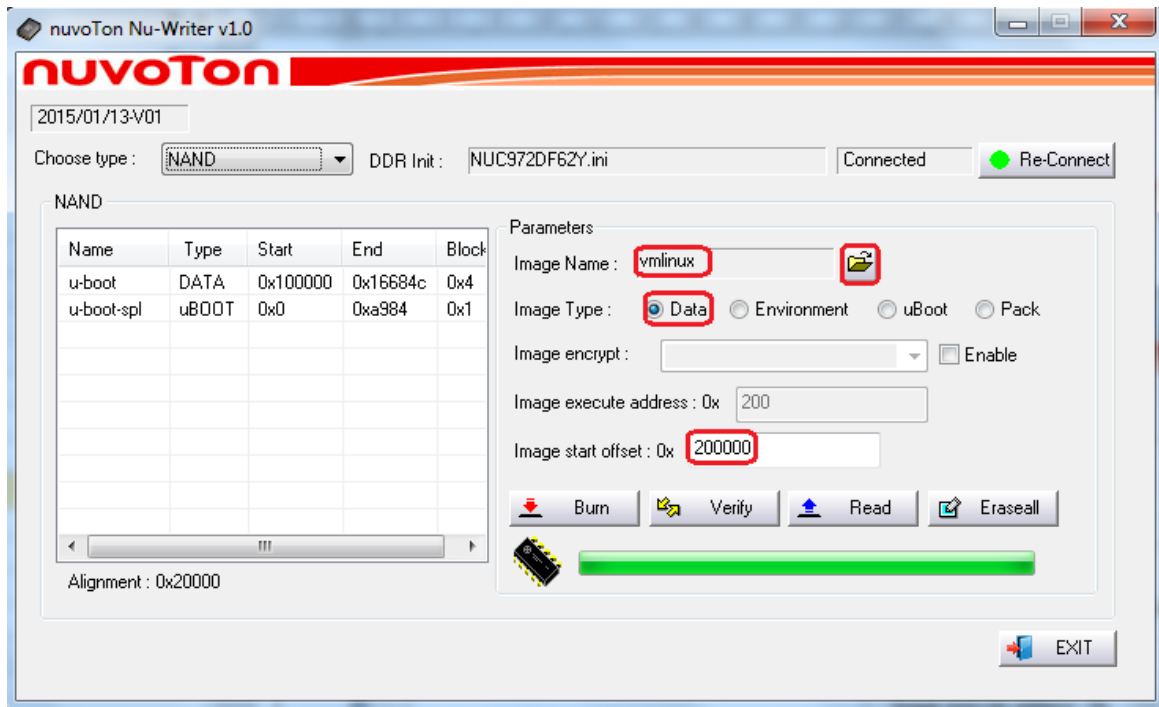


接著設定 Parameters，如下圖，

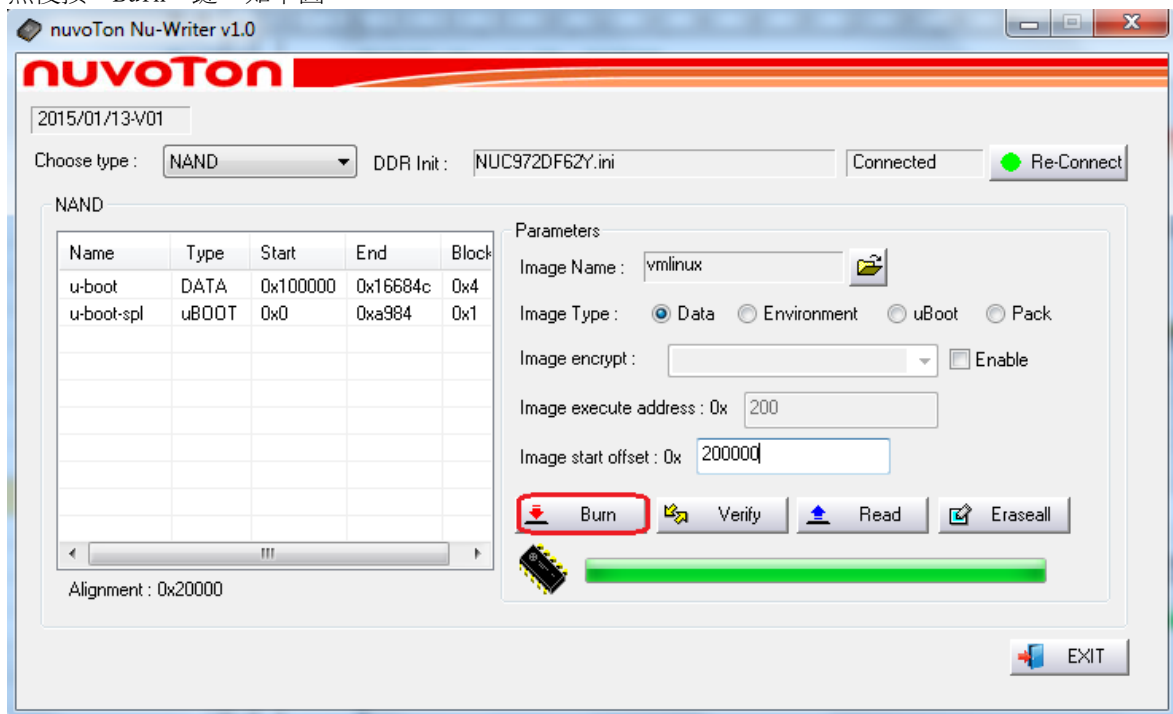
Image Name: 選取 vmlinux.ub (vmlinux.ub 的產生，請參考 4.7 章節)，

Image Type: 選取 Data，

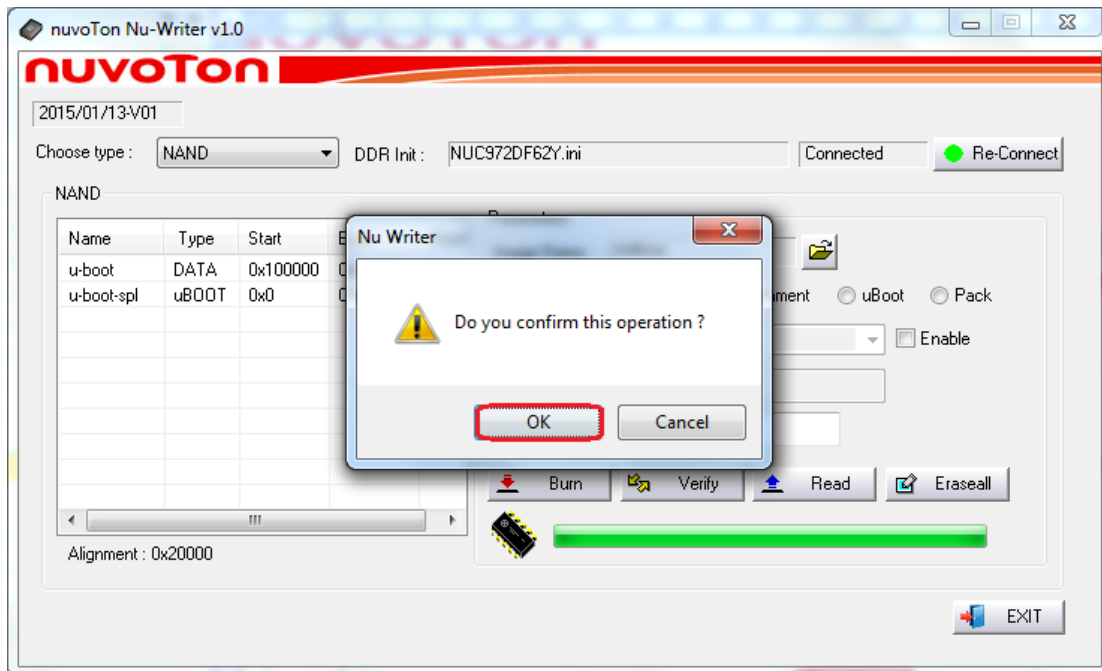
Image execute address: 0x 填寫 200000



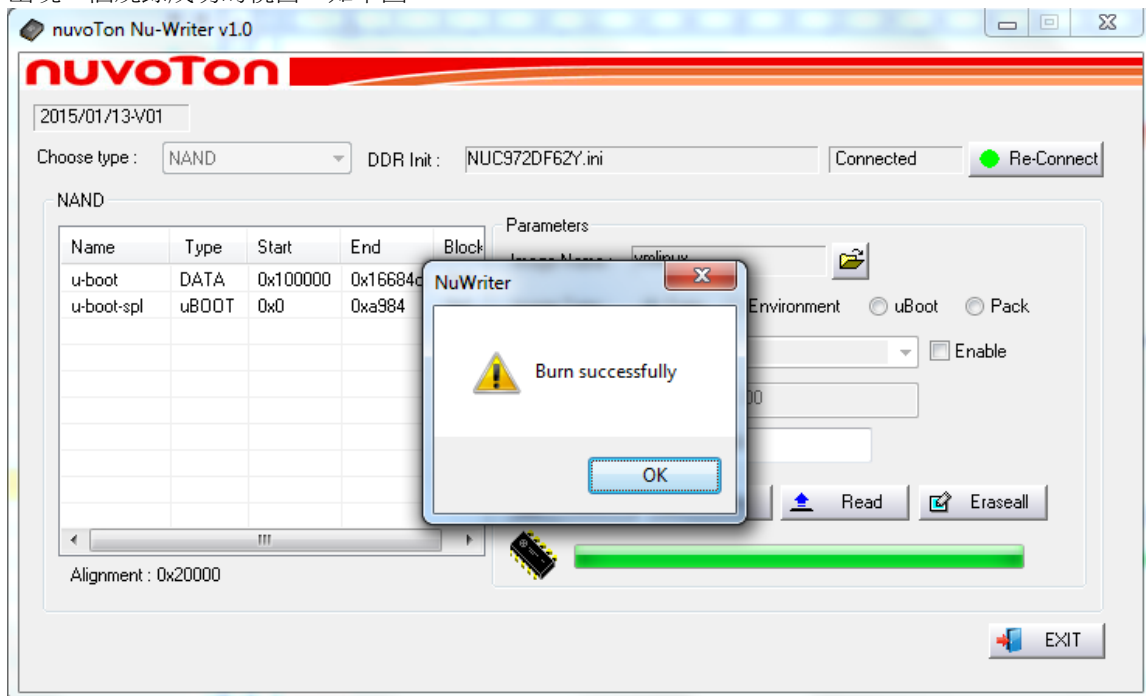
然後按“Burn”鍵，如下圖



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，



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)

Bootm 命令

4.5.1 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 ...
```

Go 命令

4.5.2

- 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
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 ...
```

記憶體命令

4.5.6 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: ffb3b319 effe9d7d bfbeeb09 ff7b4f31  ....}.....10{.
00001040: f7bf3973 eaff296c e6fce35e 6fffc7d7  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 `mmcinfo` 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 `fatload` 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

UBI 命令

- **mtddparts** : 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
mtddparts - define flash/nand partitions

Usage:
mtddparts
    - list partition table
mtddparts delall
    - delete all partitions
mtddparts del part-id
    - delete partition (e.g. part-id = nand0,1)
mtddparts 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 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>
```

- 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":
0x0000001500000-0x0000008000000 : "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 1
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: LZ0
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>
```

4.6 環境變數

環境變數的配置

4.6.1

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

命令腳本

4.6.3

下列是和命令腳本相關的环境變數

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

4.6.4

新增的环境變數

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

- 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".

使用 mkimage 產生影像檔

mkimage 工具放在 tools/mkimage, 下面的範例是將 ARM Linux 內核的 ELF 檔案 (970image) 4.7. 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

AES 加密功能

- 4.7.3 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 vmlinux.bin vmlinux.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

Watchdog timer 配置

4.9.1 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。

Watchdog timer 環境變數

4.9.2

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

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。

4.11.1 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。

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。

GPIO 驅動程式介面

4.1.2 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。

4.1.3

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

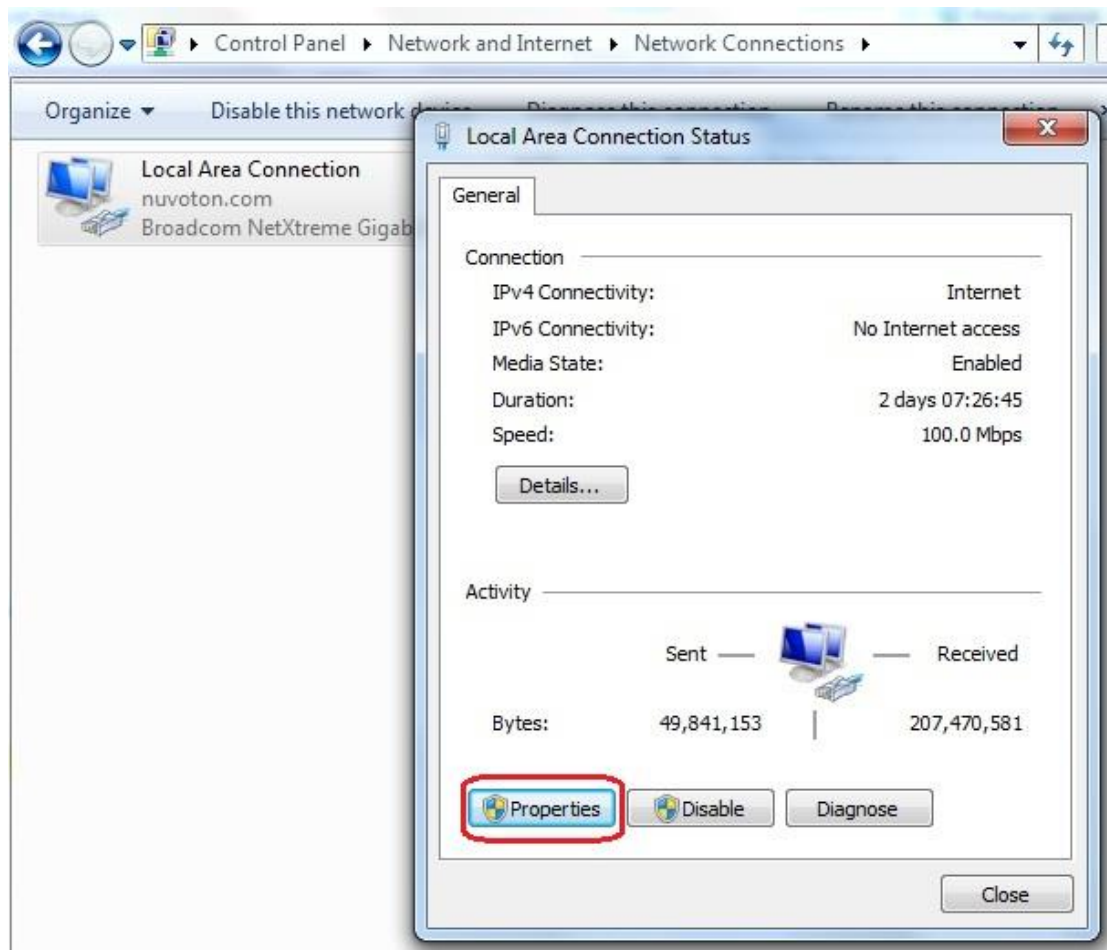
我們需要一部 PC, 該PC 有固定 IP 位址並且安裝TFTP/DHCP伺服器應用程式.

4.12.1 設置固定 IP 位址

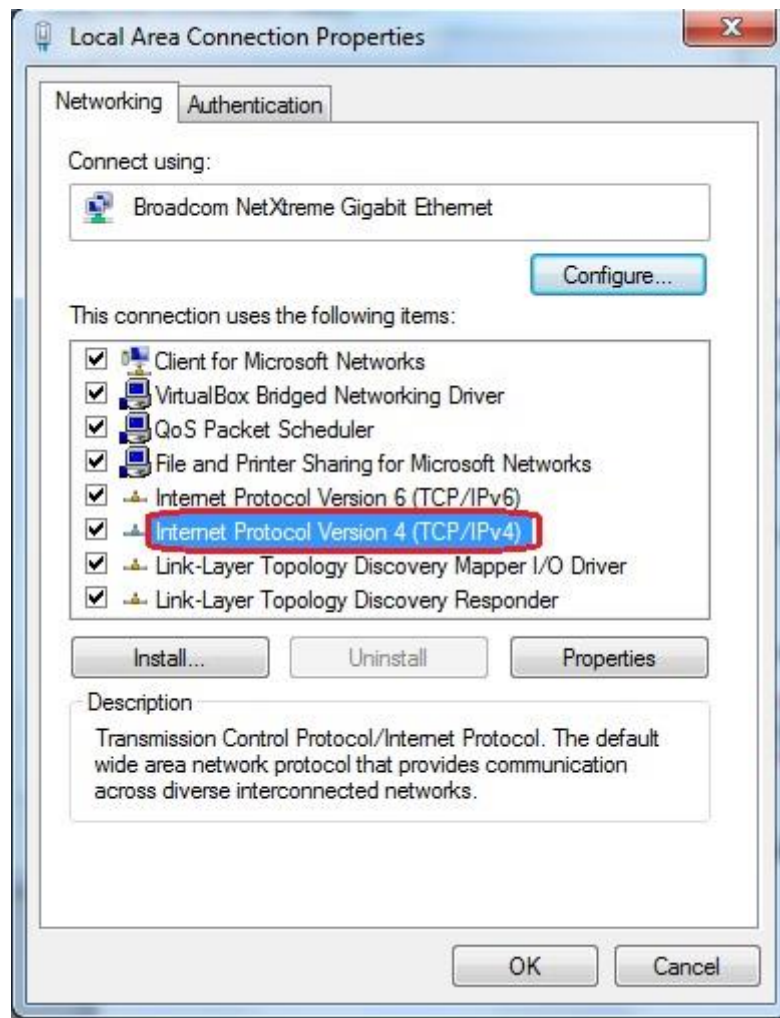
1. 點選Local Area Connection (Control Panel -> Network and Internet -> Network Connections



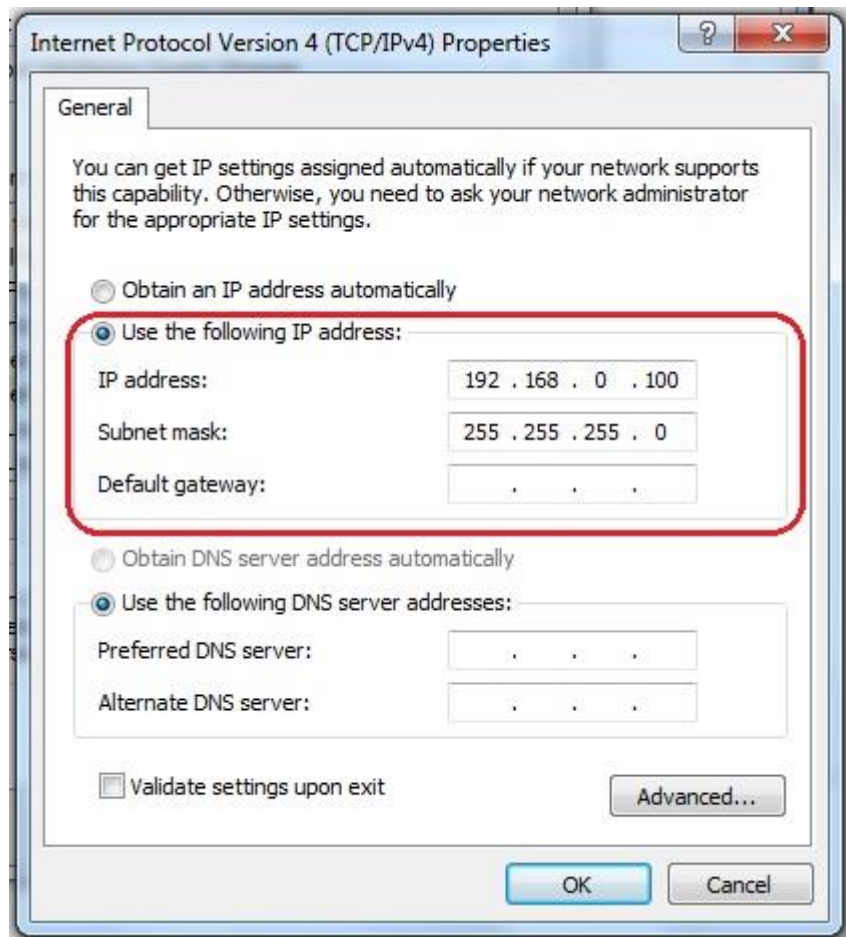
2. 點選Properties



- 點選 Internet Protocol Version 4 (TCP/IPv4)



4. 設置固定 IP 位址

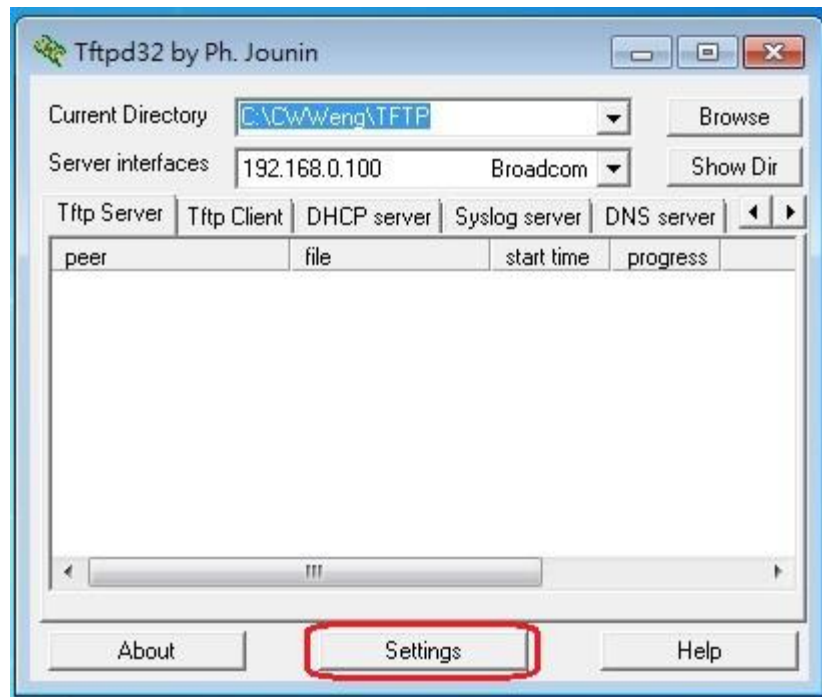


4.12.2

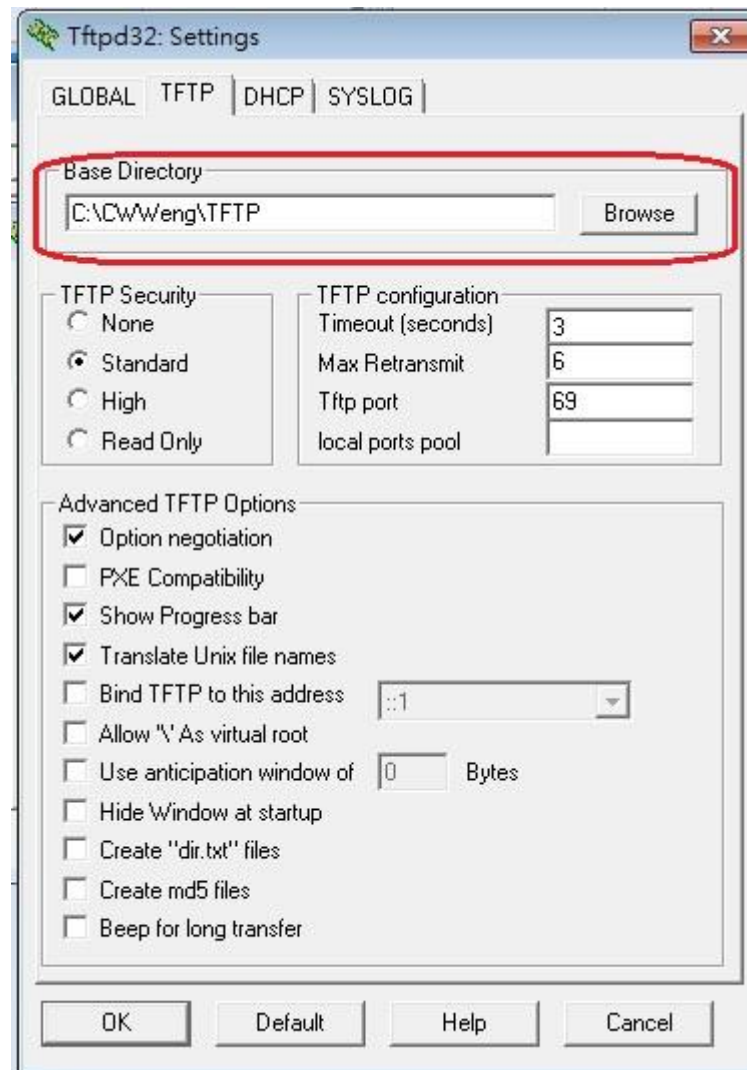
TFTP 和 DHCP 伺服器

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

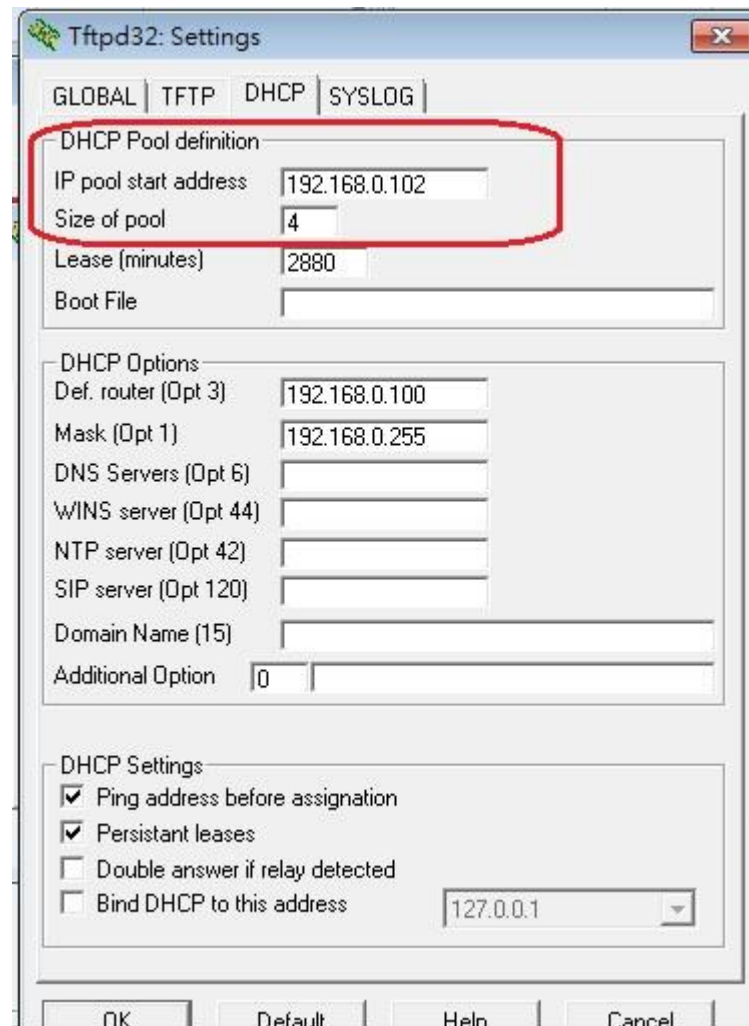
<http://www.jounin.net/tftp-server-for-windows.html>



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。

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>
```

採用 SHA-1 硬體計算 checksum

4.13.2

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 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 版本歷史

版本號	日期	描述
0.9	May. 19, 2014	初版發布

Important Notice

Nuvoton Products are neither intended nor warranted for usage in systems or equipment, any malfunction or failure of which may cause loss of human life, bodily injury or severe property damage. Such applications are deemed, “Insecure Usage”.

Insecure usage includes, but is not limited to: equipment for surgical implementation, atomic energy control instruments, airplane or spaceship instruments, the control or operation of dynamic, brake or safety systems designed for vehicular use, traffic signal instruments, all types of safety devices, and other applications intended to support or sustain life.

All Insecure Usage shall be made at customer’s risk, and in the event that third parties lay claims to Nuvoton as a result of customer’s Insecure Usage, customer shall indemnify the damages and liabilities thus incurred by Nuvoton.

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