



NUC970/N9H30 U-Boot v2016_11 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/N9H30 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	U-Boot 使用說明	3
1.1	配置	3
1.2	目錄架構	13
1.3	編譯 U-Boot	15
1.4	NAND AES secure boot 示範	16
1.5	U-Boot 命令	28
1.6	環境變數	67
1.7	mkimage 工具	70
1.8	安全性問題	74
1.9	Watchdog timer	74
1.10	U-Boot LCD	75
1.11	GPIO	76
1.12	網路測試環境	78
1.13	加快 SPI flash 開機速度	84
1.14	利用 Power-on Setting 來改變 NAND flash page size 和 ECC type	85
1.15	Tomato	85
1.16	注意事項	86

1 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。U-Boot 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/N9H30 U-Boot 的版本是 v2016.11 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>

配置

U-Boot 是可配置的，U-Boot v2016.11 支持兩種配置方式，第一個方式是修改配置檔中的各項定義來產生不同的配置，第二個方式是透過選單來配置。

1.1.1 編輯配置檔

NUC970/N9H30 配置檔位於 include/configs/ 目錄下，以下分段介紹配置檔 nuc970_evb.h 中的各項定義。

```
#define EXT_CLK          12000000          /* 12 MHz crystal */

#define CONFIG_SYS_TEXT_BASE          0xE00000

#define CONFIG_SYS_LOAD_ADDR          0x8000

#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
```

- EXT_CLK: 外部晶振頻率，timer 驅動程式初始化會參考此設定
- CONFIG_SYS_TEXT_BASE: 定義 U-Boot 鏈結位址
- CONFIG_SYS_LOAD_ADDR: 影像檔所要下載位址
- 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

```
#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_HW_WATCHDOG

#define CONFIG_SYS_BOOTM_LEN      0x1000000 /* 16MB max kernel size */

#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_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_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_HW_WATCHDOG: 打開 watchdog timer 功能 (CONFIG_NUC970_WATCHDOG 需同時打開)
- CONFIG_SYS_BOOTM_LEN: 定義 boom 命令在解壓縮內核的最大長度
- 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/N9H30 EMAC0
- CONFIG_NUC970_EMAC1: 使用 NUC970/N9H30 EMAC1
- CONFIG_NUC970_ETH: 支援 NUC970/N9H30 Ethernet
- CONFIG_NUC970_PHY_ADDR: PHY 位址
- CONFIG_ETHADDR: MAC 位址
- CONFIG_SYS_RX_ETH_BUFFER: Rx Frame Descriptors 的個數
- CONFIG_SYS_DCACHE_OFF: 關掉 D cache

```
/*
 * 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 */
```

- CONFIG_BOOTP_SERVERIP: TFTP 伺服器的 IP 不會被改成 DHCP 伺服器的 IP

```
#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 */
```

```
#define CONFIG_SPL_TEXT_BASE      0x200
```

```
#define CONFIG_SPL_STACK          0xBC008000
```

```
#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/N9H30 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_NUC970_SPI 1
```



```
#ifndef 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_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: 環境變數的 sector 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/N9H30 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/N9H30 相關訊息輸出到 LCD 上
- CONFIG_LCD_INFO_BELOW_LOGO: 將 NUC970/N9H30 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_DOS_PARTITION
/*#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 1
#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/N9H30 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_DOS_PARTITION: 支持 DOS 分區
- 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/N9H30 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

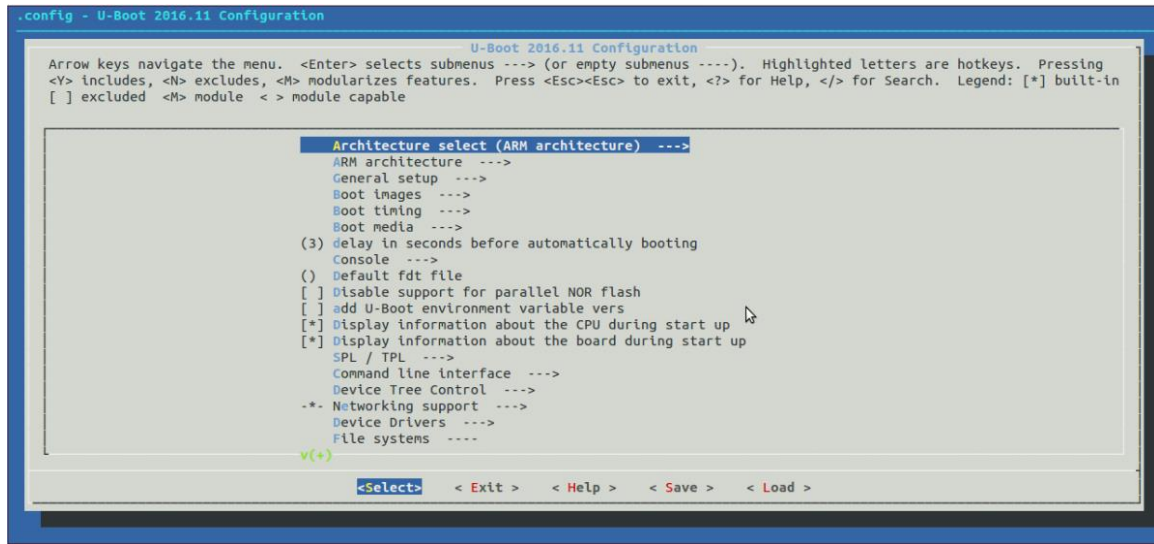
#ifdef CONFIG_KPI_NUC970
// #define CONFIG_KPI_PA_PORT 1 // KPI select PA port
#define CONFIG_KPI_PH_PORT 1 // KPI select PH port
#define CONFIG_KPI_ROW_NUM 3
#define CONFIG_KPI_COL_NUM 3
#define CONFIG_KPI_DEBOUNCE 8 // debounce length setting: 0~13
#endif
```

- 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

1.1.2 選單配置

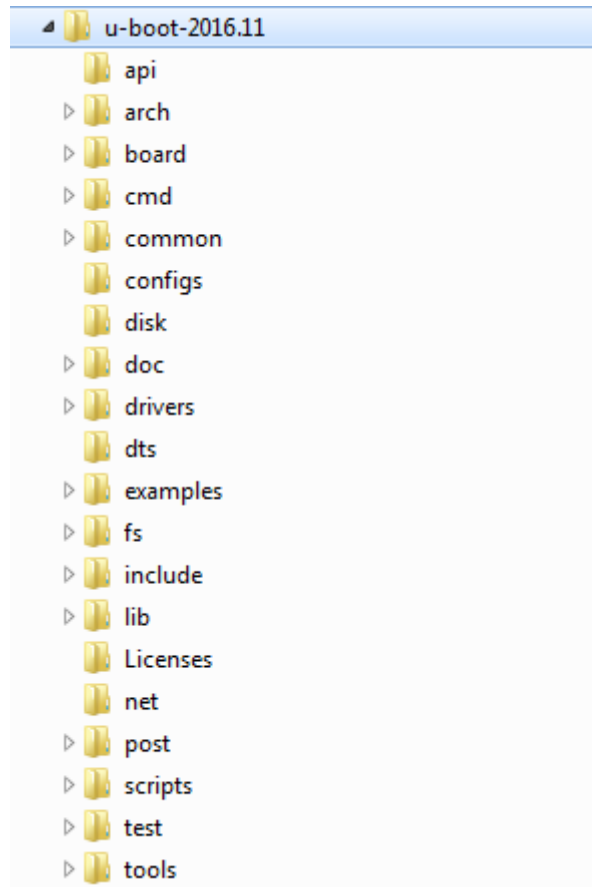
U-Boot v2016.11 支持透過命令 “make menuconfig” 來進行選單配置

```
# make menuconfig
```



目錄架構

U-Boot 的目錄結構如下圖



- api: 提供開發應用程式所需的 API.
- arch: 包含CPU 相關的源代碼
- NUC970/N9H30 CPU 相關的源代碼放在 arch/arm/cpu/arm926ejs/nuc970.
- board: 包含板子相關的源代碼
NUC970/N9H30 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.
- configs: 各廠商提供的默認配置檔
- disk: 磁碟分割相關源代碼
- doc: 放置各式各樣的 README 文件.
- drivers: 放置驅動程式源代碼.
NUC970/N9H30 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/N9H30 的配置檔就放在 include/configs/nuc970_evb.h



- lib: 放置各種函式庫.
- Licenses: 存放 GPL 相關文件
- net: 存放網路相關的源代碼. 例如: tftp.c, ping.c,
- post: 針對沒有提供 hotkey 的平台, 提供 post_hotkeys_pressed() 的默認實作方法
- scripts: 提供編譯與配置時可用到的腳本
- test: 存放一些測試程式, 細節請參考 test/README 這份文件
- tools: 存放一些工具, 例如 mkimage 就是一個產生影像檔的工具.

編譯 U-Boot

1.1.3 編譯命令

清除所有的 object code.

```
# make distclean
```

產生默認配置 (選擇 NUC970 或 N9H30 默認配置, 二擇一)

```
# make nuc970_defconfig (for NUC970)
```

```
# make n9h30_defconfig (for N9H30)
```

編譯 U-Boot

```
# make
```

1.1.4 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 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)
- 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 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)
- u-boot-spl.map - 鏈結對應檔

1.1.5 Main U-Boot 鏈結位址

Main U-Boot 的鏈結位址是定義在 include/configs/nuc970_evb.h

```
#define CONFIG_SYS_TEXT_BASE 0xE00000
```

上面的例子，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 CONFIG_SYS_TEXT_BASE 定義在相同位址。

1.1.6 SPL U-Boot 鏈結位址

SPL U-Boot 的鏈結位址定義在 include/configs/nuc970_evb.h

預設位址是 0x200，若要修改到其他位址，請找到以下定義，將 0x200 置換成新的位址。

```
#define CONFIG_SPL_TEXT_BASE 0x200
```

新增 SPI 配置

NUC970/N9H30 默認的配置是不支持 SPI。如果要開啟 SPI 配置，請修改 include/configs/nuc970_evb.h，然後透過 “make menuconfig” 命令新增相關配置。

1.1.7 修改 include/configs/nuc970_evb.h

修改 include/configs/nuc970_evb.h，打開定義 “CONFIG_SYS_USE_SPIFLASH”，並將環境變數改為存放在 SPI flash。

```
#define CONFIG_SYS_USE_SPIFLASH
```



```
/* #define CONFIG_ENV_IS_IN_NAND */  
#define CONFIG_ENV_IS_IN_SPI_FLASH
```

1.1.8 使能 NUC970/N9H30 SPI 驅動

透過 “make menuconfig” ，使能 NUC970/N9H30 SPI 驅動並選擇 SPI 運作在 Quad 或正常模式。

```
-> Device Drivers  
    -> SPI Support  
        [*] NUC970/N9H30 SPI driver  
            Select NUC970/N9H30 SPI in Quad mode or Normal mode (Quad mode) --->
```

1.1.9 使能 sf/spi 命令

透過 “make menuconfig” ，使能 sf/spi 命令。

```
-> Command line interface  
    -> Device access commands  
        [*] sf  
        [*] sspi
```

1.1.10 使能 SPI Flash 介面支持

透過 “make menuconfig” ，使能 SPI Flash 介面支持，以及存取超過 SPI Flash 16 Mbytes 位址。同時請根據開發板上的 SPI Flash 廠牌開啟相關的支持，下面的範例是開發板上的 SPI flash 廠牌是華邦。

```
-> Device Drivers  
    -> SPI Flash Support  
        [*] Legacy SPI Flash Interface support  
        [*] SPI flash Bank/Extended address register support  
        [*] winbond SPI flash support
```

NAND AES secure boot 示範 (僅限 NUC970)

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.

1.1.11 編譯 Main U-Boot 以及 SPL U-Boot

```
# make distclean
# make nuc970_defconfig (for NUC970)
# make
```

1.1.12 燒錄 key 到 MTP

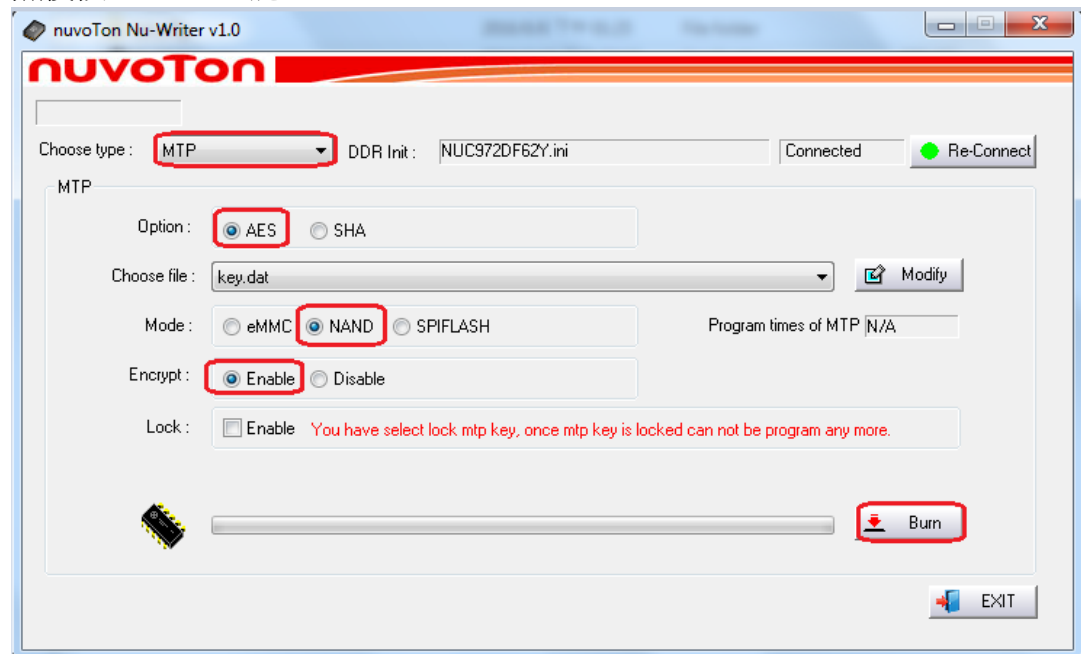
在 Choose type 處選擇 “MTP”

Option: 選擇 AES

Mode: 選擇 NAND

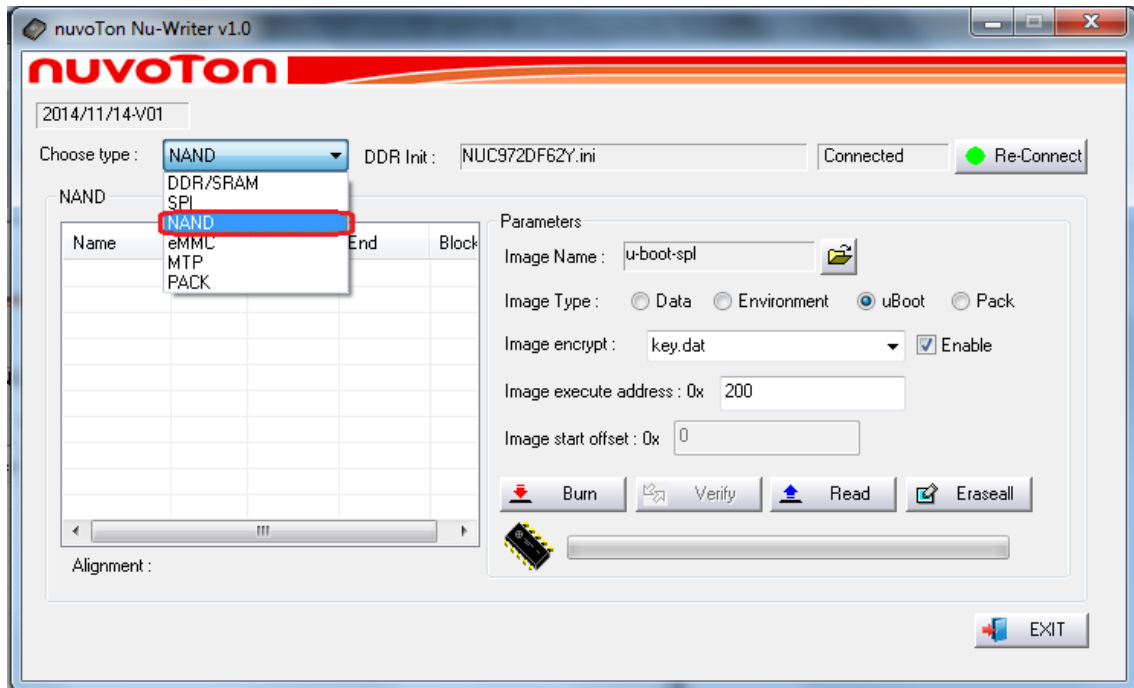
Encrypt: 選擇 Enable

然後按 “Burn” 鍵

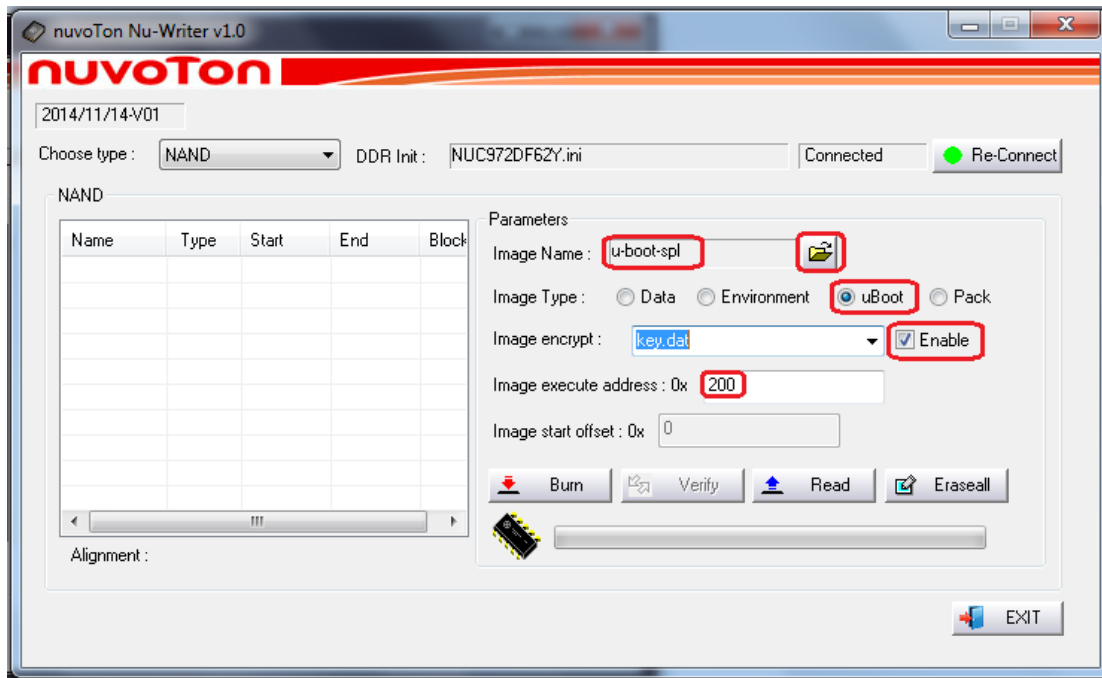


1.1.13 燒錄 SPL U-Boot

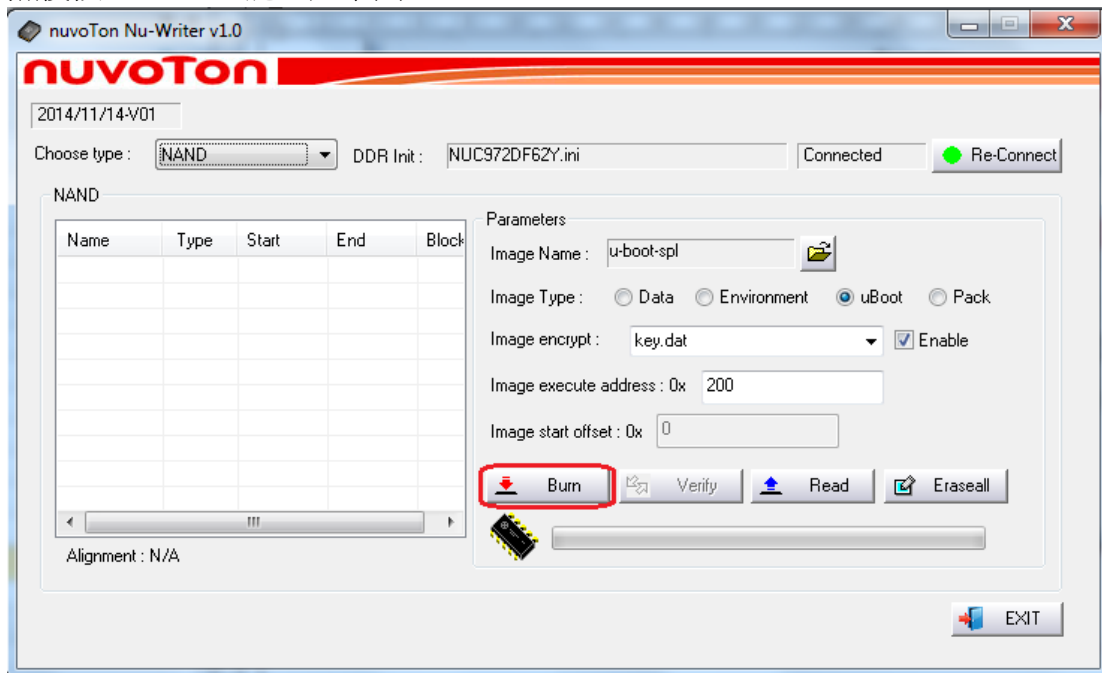
在 Choose type 處選擇 “NAND”，如下圖。



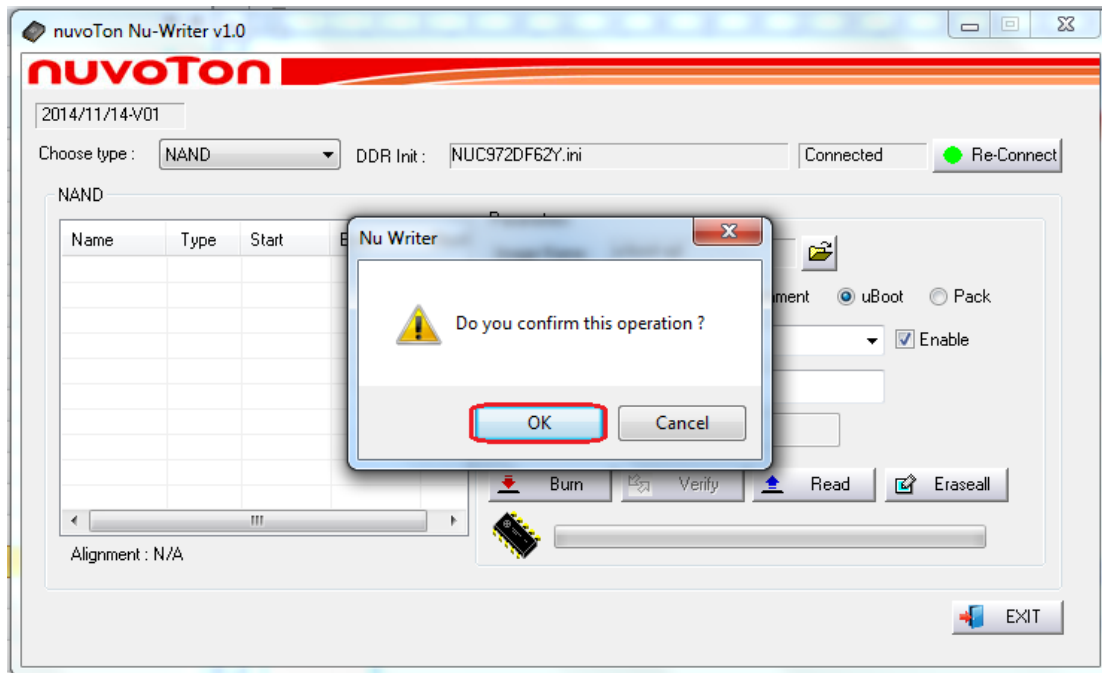
接著設定 Parameters，如下圖，
 Image Name: 選取 u-boot-spl.bin，
 Image Type: 選取 uBoot，
 Image encrypt: 勾選 Enable
 Image execute address: 0x 填寫 200



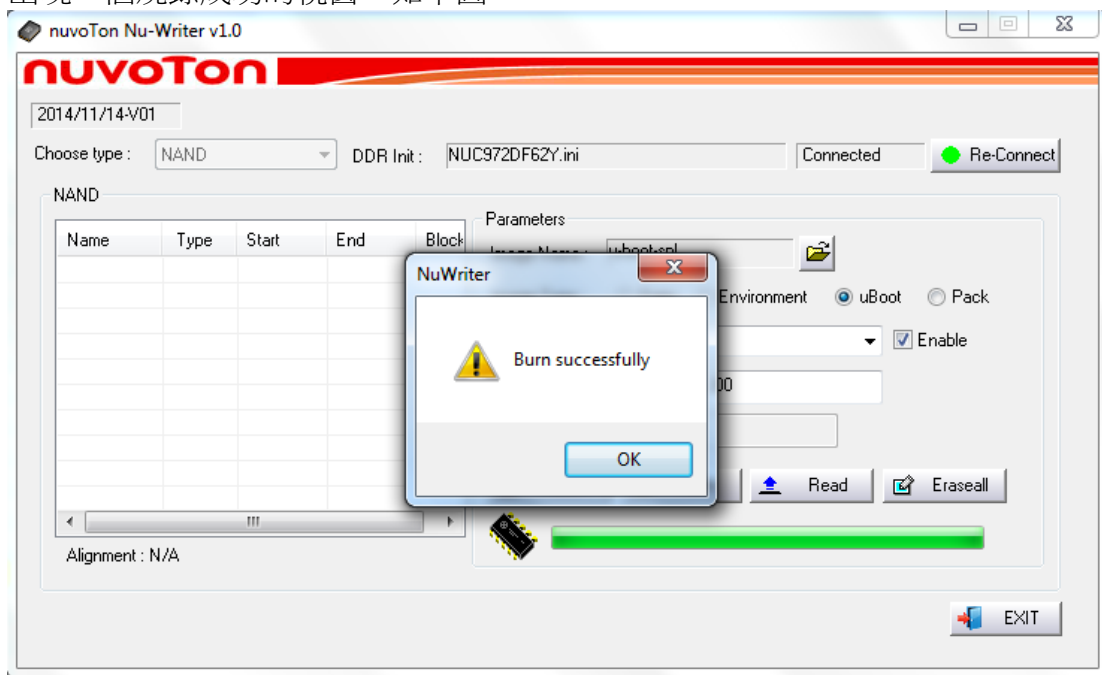
然後按 “Burn” 鍵，如下圖



出現一個對話視窗，選擇 “OK” ，如下圖，

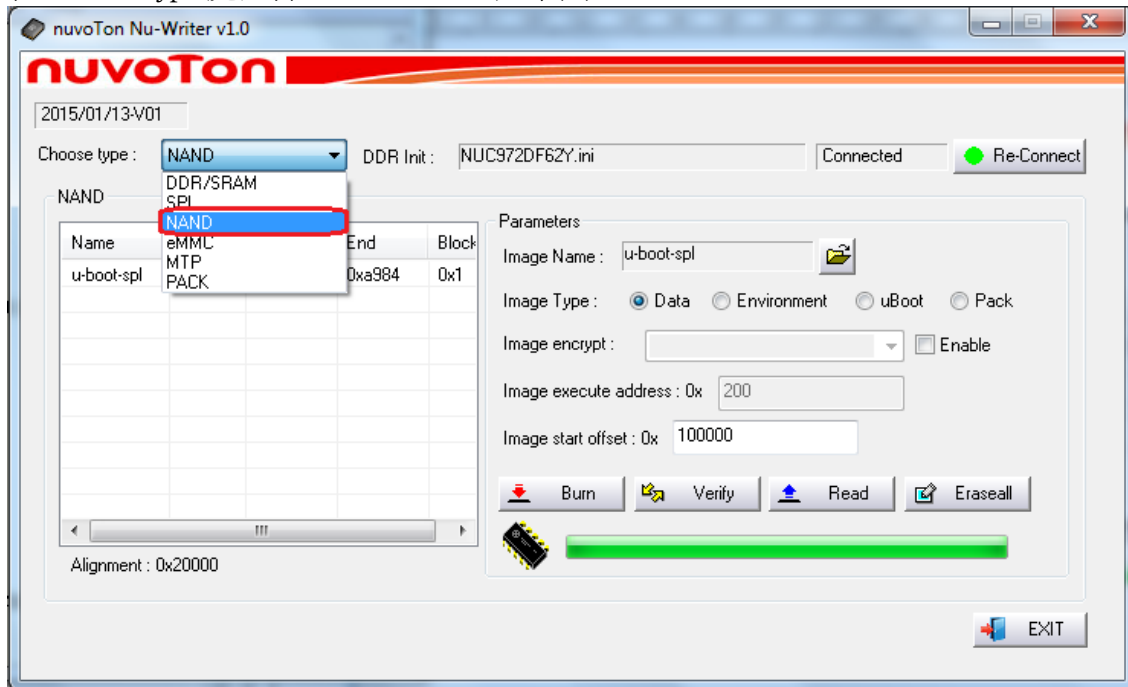


出現一個燒錄成功的視窗，如下圖，



1.1.14 燒錄 Main U-Boot

在 Choose type 處選擇 “NAND”，如下圖。

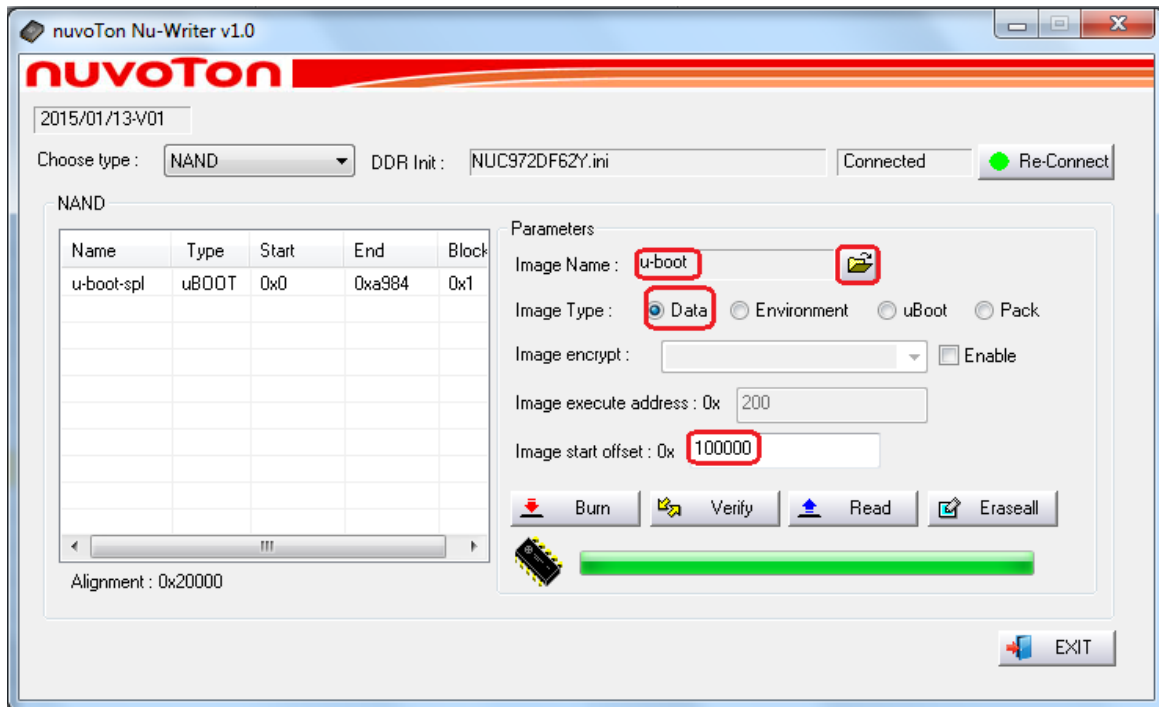


接著設定 Parameters，如下圖，

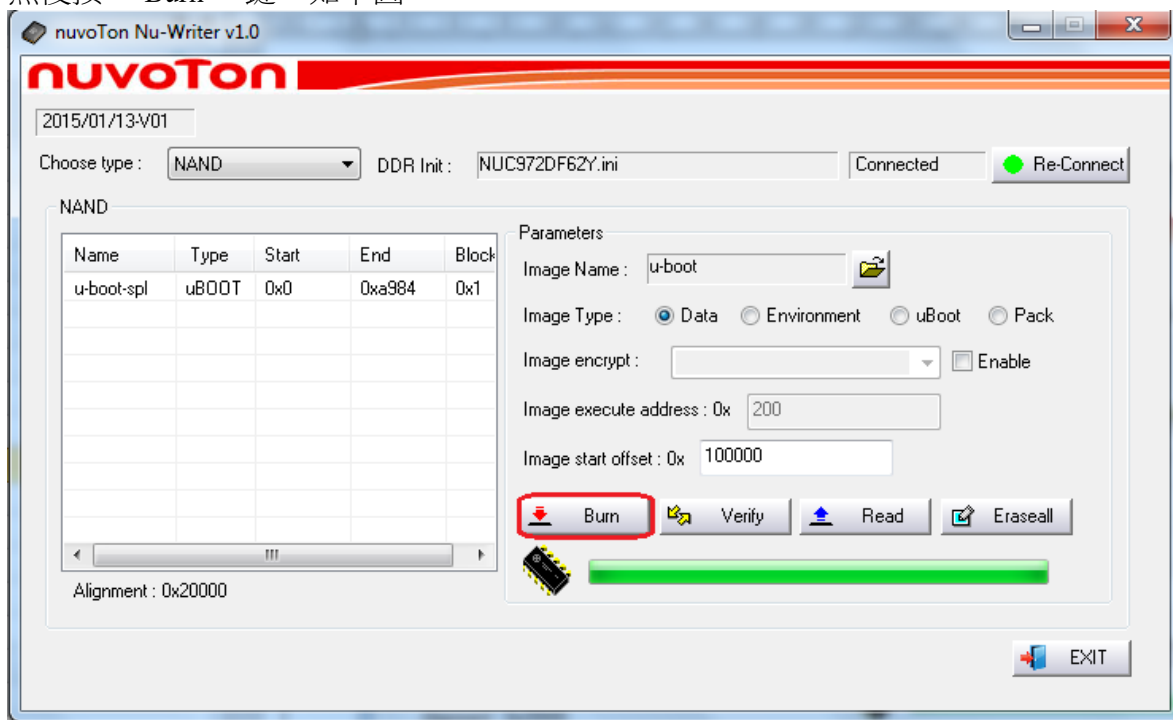
Image Name: 選取 u-boot.bin，

Image Type: 選取 Data，

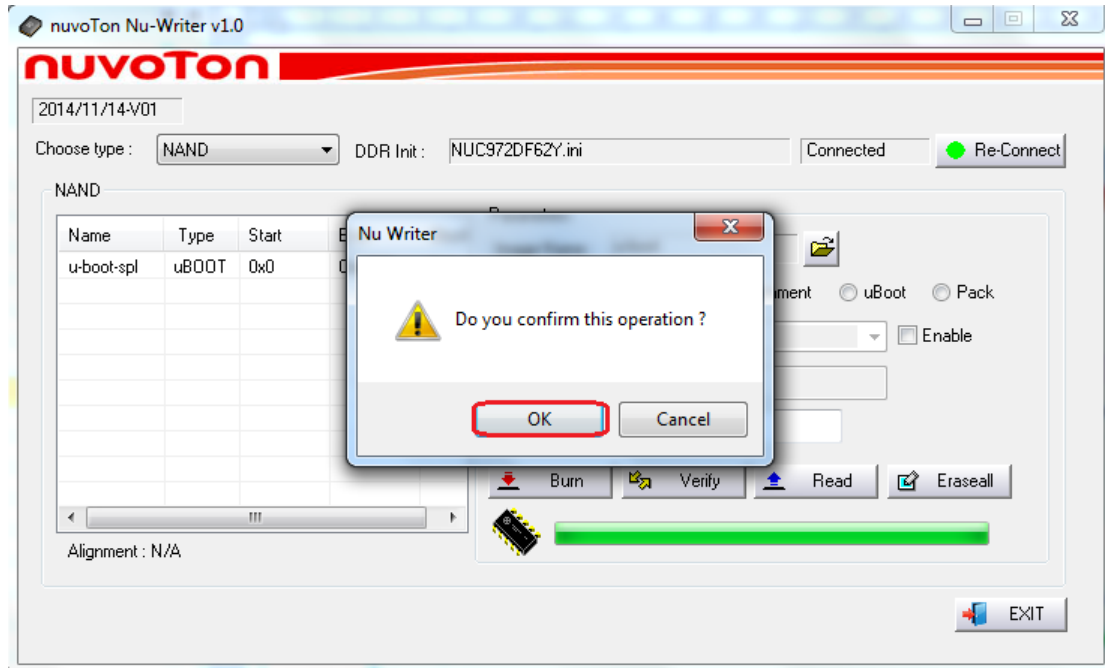
Image execute address: 0x 填寫 100000



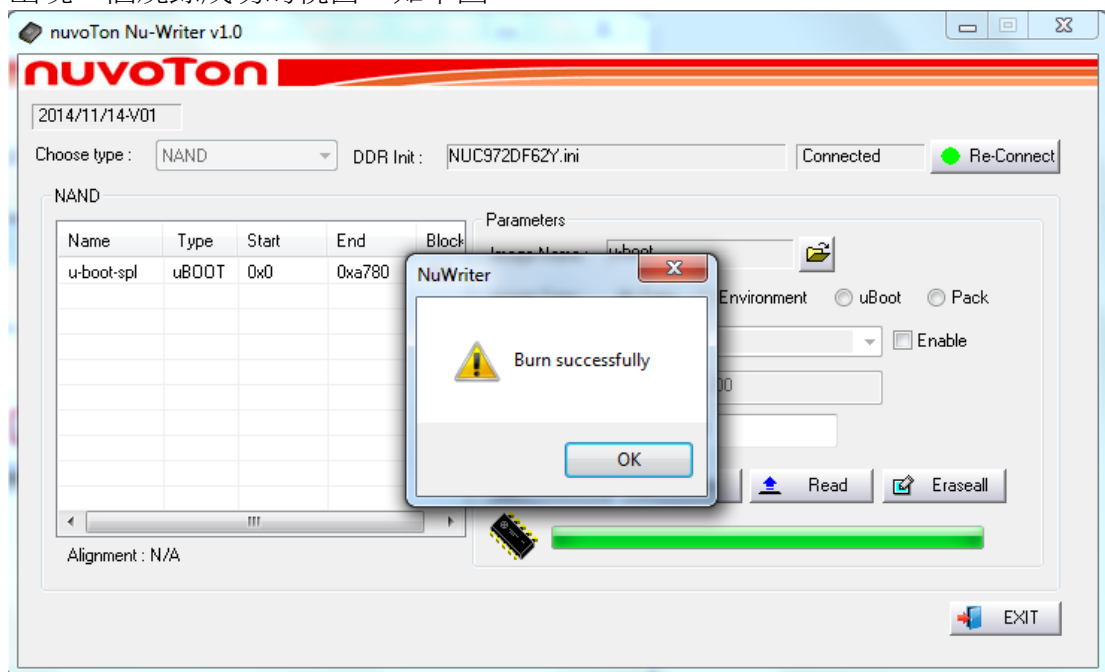
然後按 “Burn” 鍵，如下圖



出現一個對話視窗，選擇“OK”，如下圖，

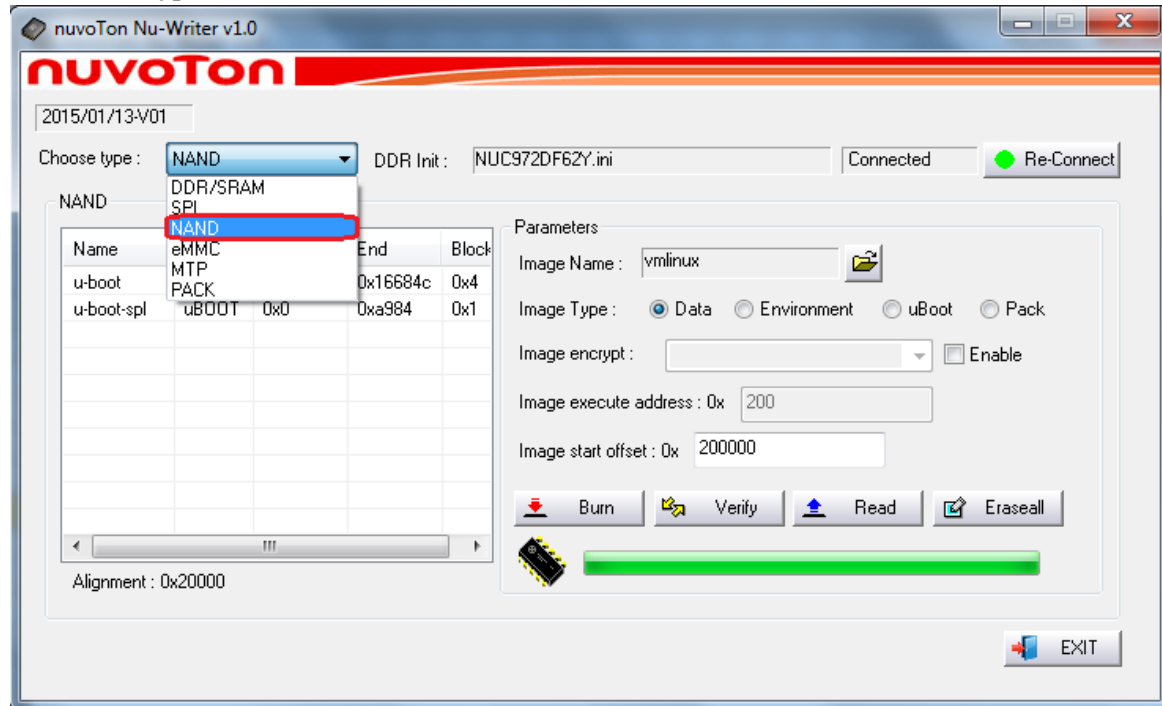


出現一個燒錄成功的視窗，如下圖，



1.1.15 燒錄 Linux 內核

在 Choose type 處選擇 “NAND” ，如下圖。

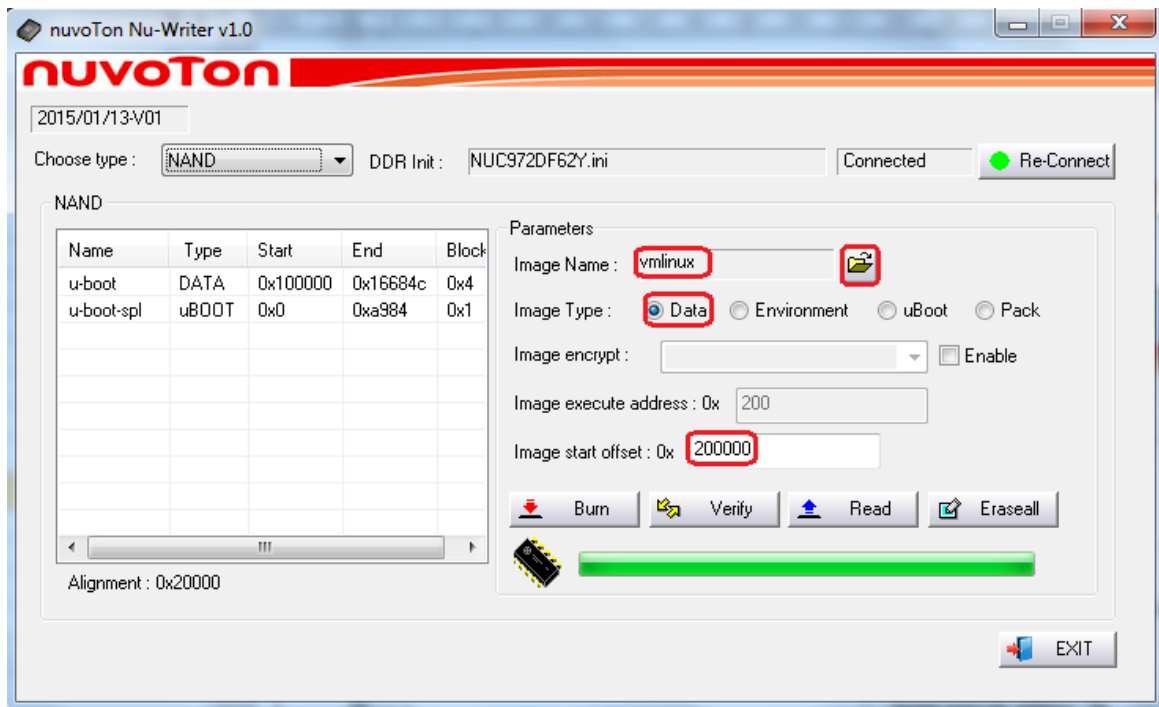


接著設定 Parameters ，如下圖，

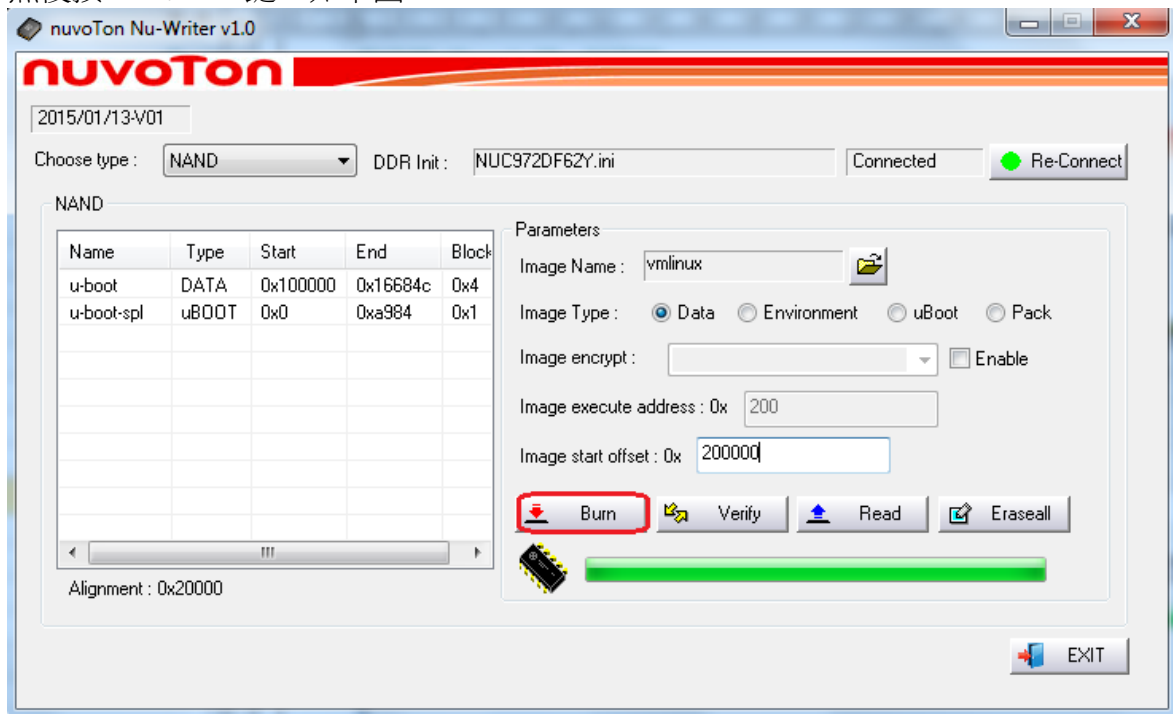
Image Name: 選取 vmlinux.ub (vmlinux.ub 的產生，請參考章節 0) ，

Image Type: 選取 Data ，

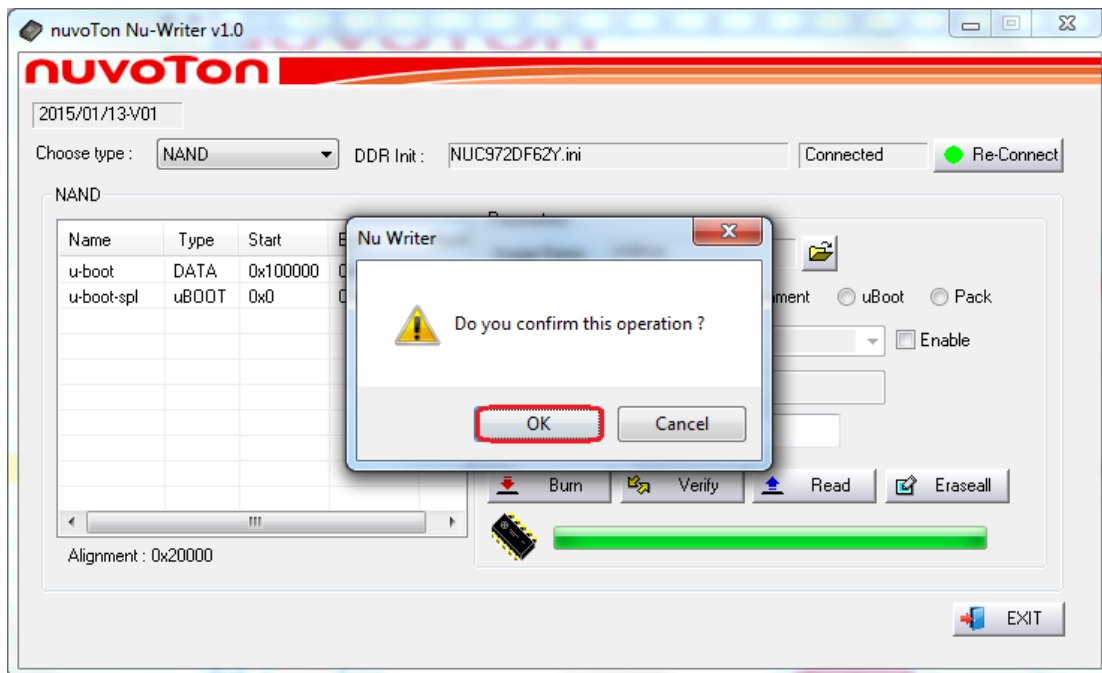
Image execute address:0x 填寫 200000



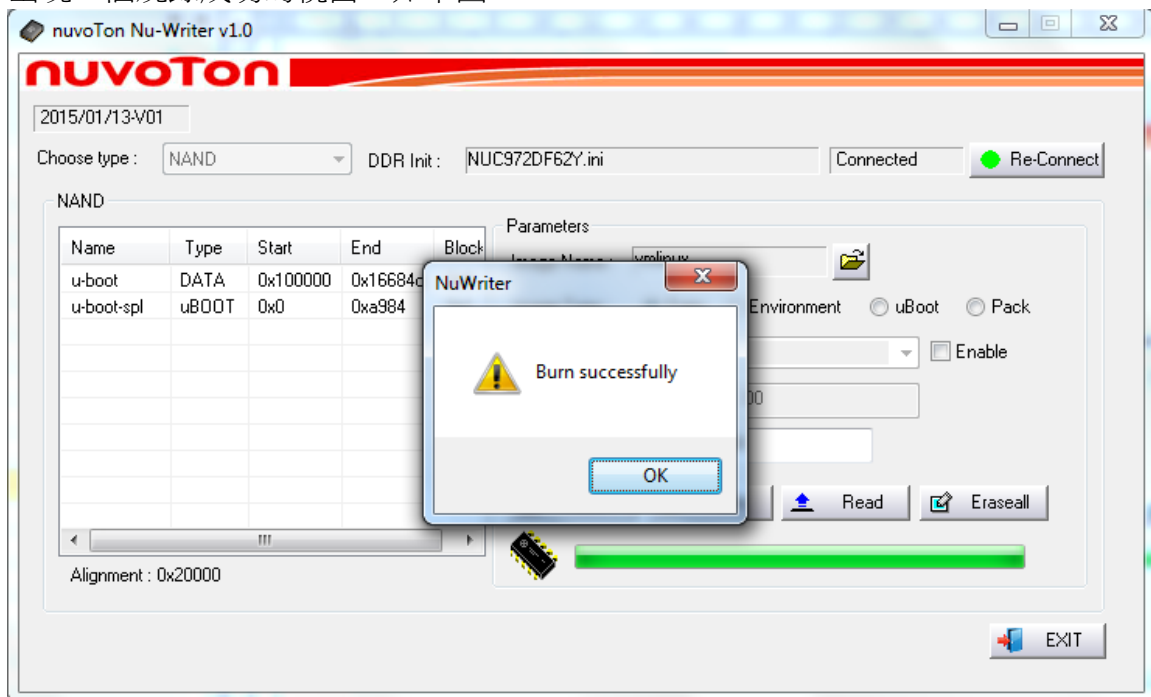
然後按“Burn”鍵，如下圖



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，



1.1.16 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: 00008000
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: 00008000
Entry Point:  00008000
Verifying Checksum ... OK
XIP Kernel Image ... OK

OK

Starting kernel ...
```

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)

1.1.17 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 (1.1.18 章節會介紹) 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: 00008000
Entry Point:  00008000
Verifying Checksum ... OK
XIP Kernel Image ... OK

OK

Starting kernel ...
```

1.1.18 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!
```

1.1.19 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> setenv 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/N9H30 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> setenv ipaddr 192.168.0.101
```

```
U-Boot> setenv 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: 00008000

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: 00008000
   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: 00008000
   Entry Point:  00008000
   Verifying Checksum ... OK
   XIP Kernel Image ... OK
OK

Starting kernel ...

```

1.1.20 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: 00008000
   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: 00008000
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: 00008000
Entry Point:  00008000
Verifying Checksum ... OK
XIP Kernel Image ... OK

OK

Starting kernel ...
```

1.1.21 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: 00008000
```

```
Entry Point: 00008000
```

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



```

Loading kernel Image ... OK

OK

Starting kernel ...

```

1.1.22 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: bffbcb5c 5ffb56ff fcff5f41 ff67760b  \....V._A...vg.
00001010: fcd227e3 dffefeeb 70cf7cb3 dbefc7cb  .'.....|.p....
00001020: fbda3e3b eb3e9ebb aa3abc95 e5fbbb2f  ;>....>...:/...
00001030: ffbbb319 effe9d7d bfbeeb09 ff7b4f31  ....}.....10{.
00001040: f7bf3973 eaff296c e6fce35e 6fffc7df  s9..l)..^.....o
00001050: cfd28a65 8cd69f2b efecce87 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 cfbbecbc 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 1afffffe 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>
```

1.1.23 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: 00008000
```

```
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: 00008000
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: 00008000
Entry Point:  00008000
Verifying Checksum ... OK
XIP Kernel Image ... OK

OK

Starting kernel ...
```

1.1.24 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>
```

1.1.25 解密命令 (僅限 NUC970)

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，務必小心。

1.1.26 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/N9H30 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 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: 00008000
   Entry Point:  00008000
   Verifying Checksum ... OK
   XIP Kernel Image ... OK

OK

Starting kernel ...
```

1.1.27 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
'mtddids' - linux kernel mtd device id <-> u-boot device id mapping
mtddids=<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>
    
```

1.1.28 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":
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>
```


1.1.29 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
yrmkdir  - YAFFS rmdir
ytrace   - show/set yaffs trace mask
yumount  - unmount yaffs
ywr      - write file to yaffs
yworm    - 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
```

y-mount 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 / ywrn 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> ywrn nand/wrn.bin 0xe00000 0x1000
U-Boot> yls -l nand
wrn.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]
```

ym 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
```

環境變數

1.1.30 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。

1.1.31 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/N9H30 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

1.1.32 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> setenv 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 1.1.27 MTD 命令章節. 最後, 記得將環境變數儲存到 NAND flash.

```
U-Boot> setenv 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
```

1.1.33 新增的環境變數

除了 U-Boot 支援的環境變數之外，NUC970/N9H30 U-Boot 新增自行定義的環境變數

- 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
```

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".

1.1.34 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 0x8000, 執行位址是 0x8000.

```
u-boot/tools# ./mkimage -A arm -O linux -T kernel -a 0x8000 -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: 00008000

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 0x8000
- e: 設置程式進入點為只為 0x8000
- d: 設置影像檔的來源為 970image

1.1.35 Checksum 計算方式 (SHA-1 或 crc32, 僅限 NUC970)

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/N9H30 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/N9H30 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/N9H30 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 0x8000 -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: 00008000

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 0x8000
- 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 0x8000 -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: 00008000

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 0x8000
- e: 設置程式進入點為只為 0x8000
- d: 設置影像檔的來源為 970image

1.1.36 AES 加密功能（僅限 NUC970）

另外，NUC970/N9H30 的 mkimage 工具新增 AES 加密的功能，增加了兩個新的參數，

-G AES, 對影像檔進行加密；以及 -H，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 0x8000，執行位址是 0x8000。


```
u-boot/tools# ./mkimage -A arm -O linux -T kernel -a 0x8000 -e 0x8000 -G AES
-H 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: 00008000

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 0x8000
- e: 設置程式進入點為只為 0x8000
- d: 設置影像檔的來源為 970image
- G: 設置加密型別為AES
- H: 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 ...
```

安全性問題 (僅限 NUC970)

1.1.37 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 -G AES -H
key.dat -d 970image 970image.ub
```

1.1.38 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
```

1.1.39 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。

Watchdog timer

1.1.40 Watchdog timer 配置

若要將 NUC970/N9H30 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/N9H30 的 watchdog timer 功能關掉，請將上面兩個定義注釋掉。

1.1.41 Watchdog timer 環境變數

當 NUC970/N9H30 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> setenv 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> setenv 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/N9H30 的配置 watchdog timer 功能是關掉時，修改環境變數 “watchdog” 是沒有意義的。

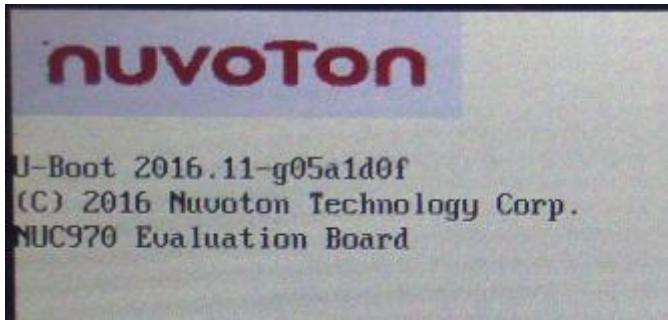
1.1.42 Watchdog timer 的時間長度

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。

U-Boot LCD

1.1.43 NUC970/N9H30 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/N9H30 U-Boot 在 LCD 面板顯示的內容如下：



1.1.44 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

找到以下七行

```
# Use board logo and fallback to vendor
ifneq ($(wildcard $(srctree)/$(src)/logos/$(BOARD).bmp),)
LOGO_BMP= $(srctree)/$(src)/logos/$(BOARD).bmp
else
ifneq ($(wildcard $(srctree)/$(src)/logos/$(VENDOR).bmp),)
LOGO_BMP= $(srctree)/$(src)/logos/$(VENDOR).bmp
endif
```

在這七行之後加入以下這個定義

```
LOGO_BMP= $(srctree)/$(src)/logos/abc.bmp
```

GPIO

U-Boot GPIO 最常見的應用是拿來點亮 LED。

NUC970/N9H30 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。

1.1.45 NUC970/N9H30 GPIO

NUC970/N9H30 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/N9H30 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/N9H30 GPIO 驅動程式時，都必須傳入 GPIO 編號。

1.1.46 GPIO 驅動程式介面

NUC970/N9H30 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。

1.1.47 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);
```

網路測試環境

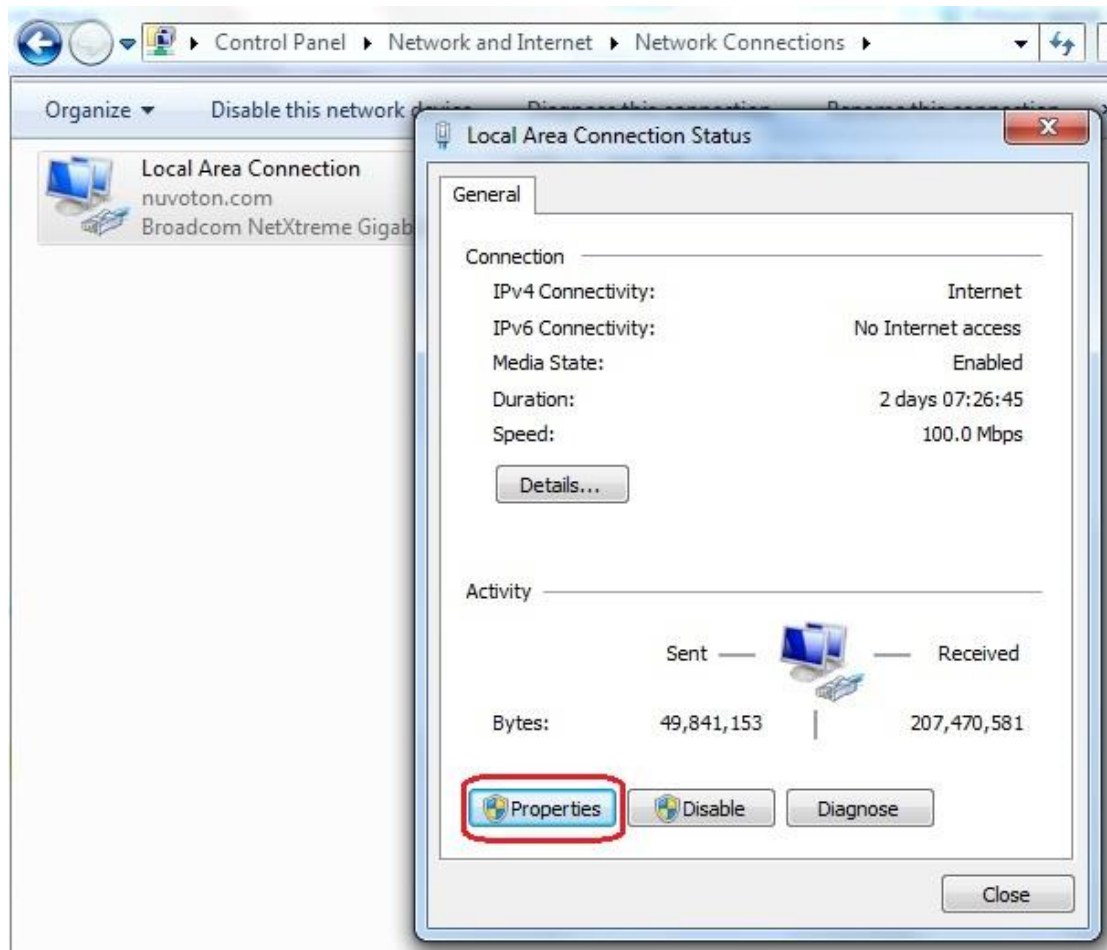
我們需要一部 PC, 該PC 有固定 IP 位址並且安裝TFTP/DHCP伺服器應用程式.

1.1.48 設置固定 IP 位址

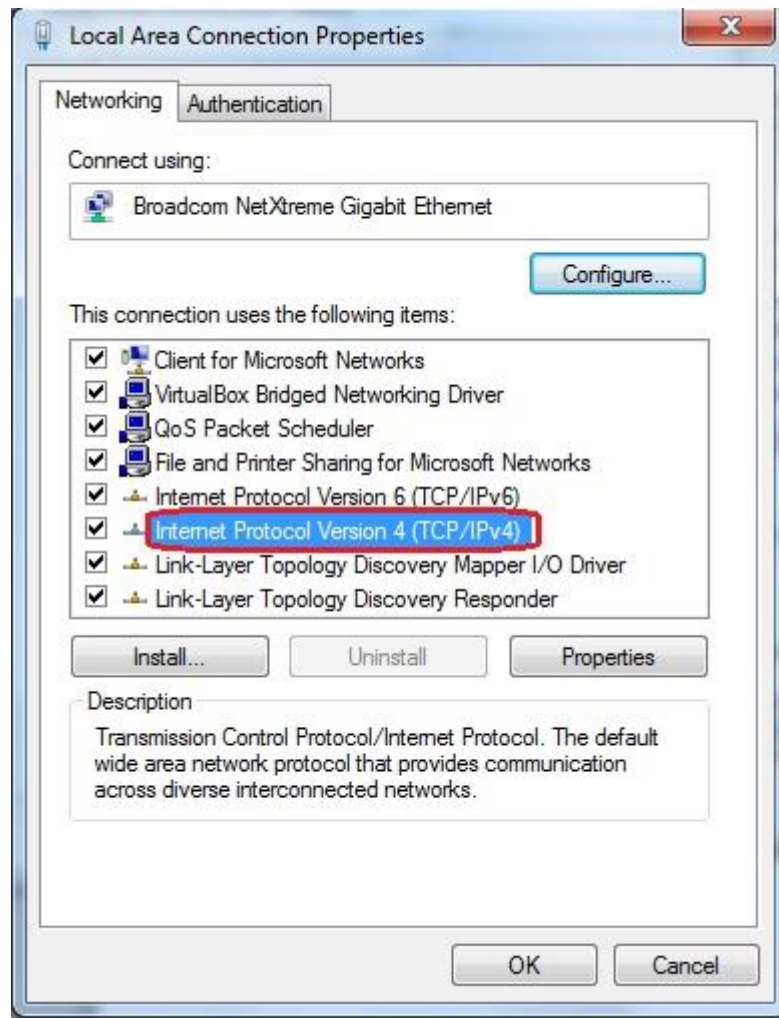
1. 點選Local Area Connection (Control Panel -> Network and Internet -> Network Connections



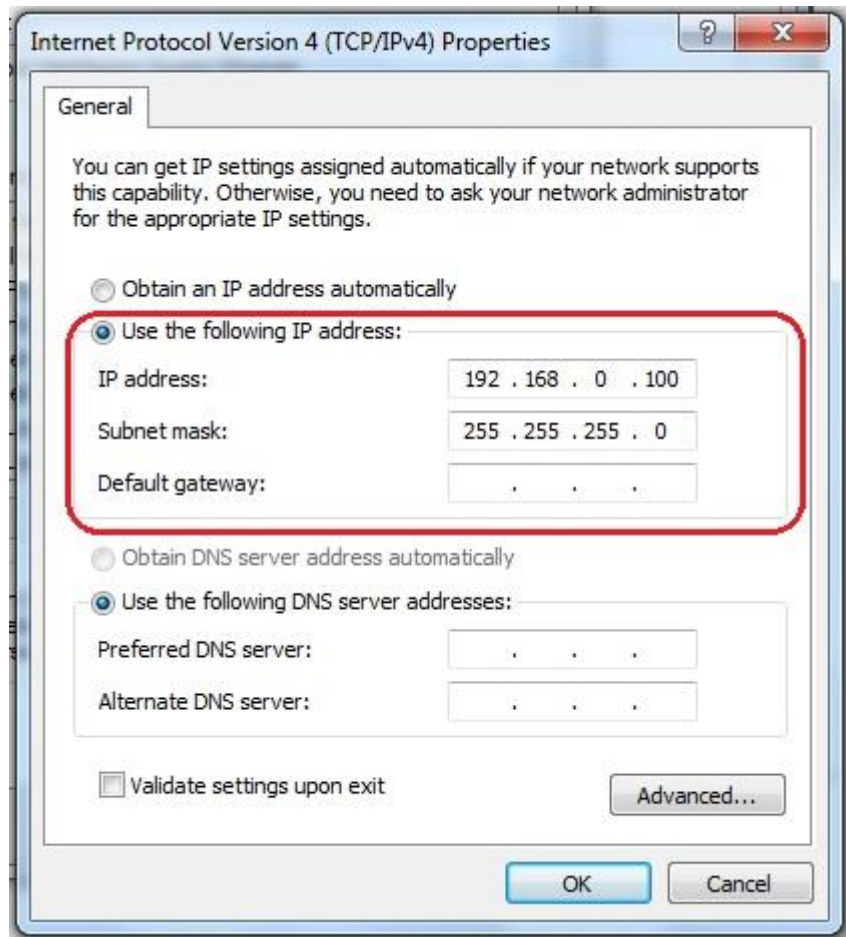
2. 點選Properties



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



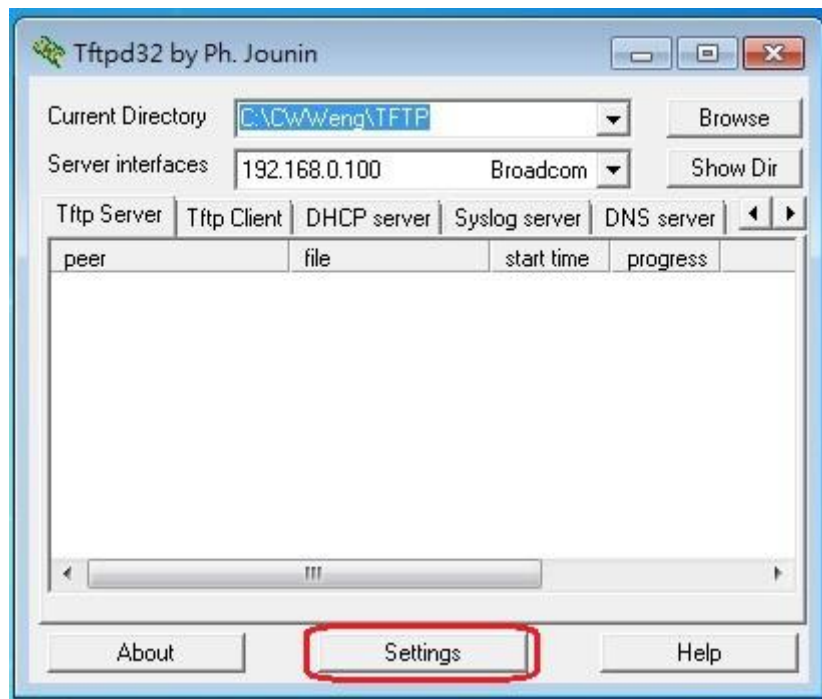
4. 設置固定 IP 位址



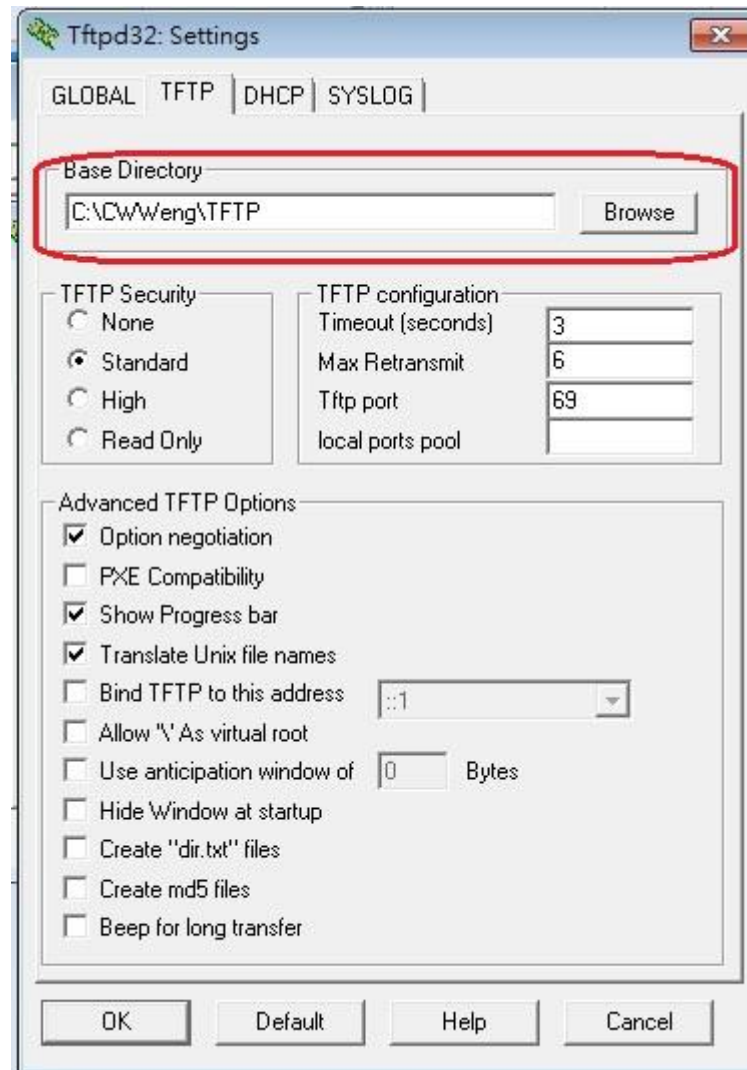
1.1.49 TFTP 和 DHCP 伺服器

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

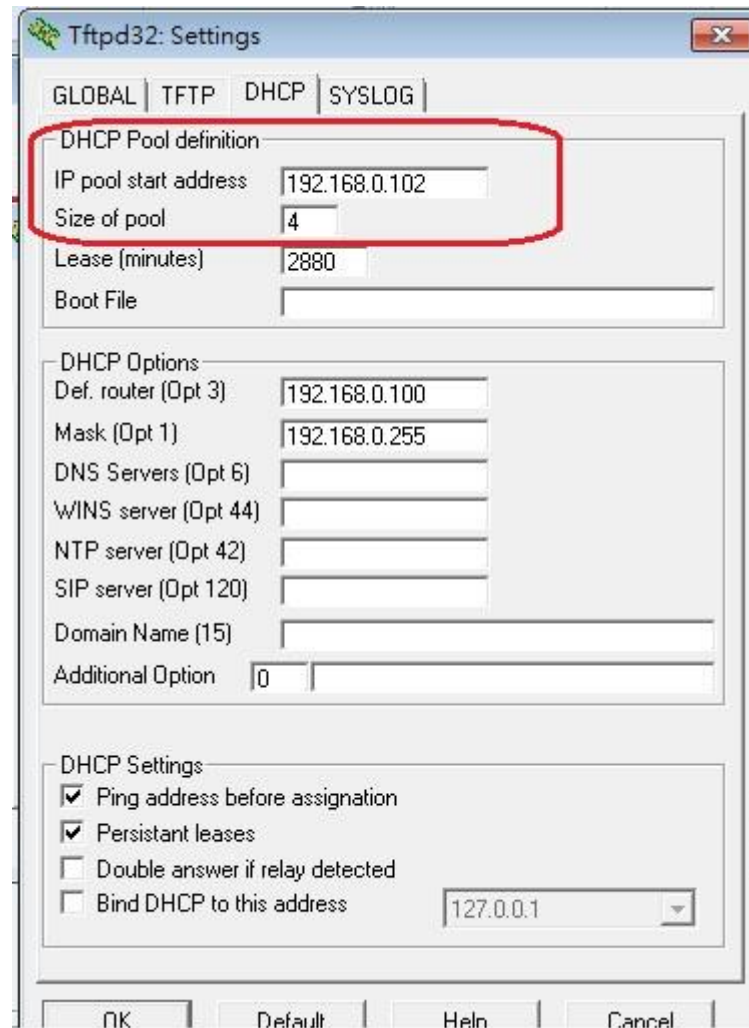
<http://www.jounin.net/tftp-server-for-windows.html>



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



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。

1.1.50 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

以下的範例是透過 sf probe 命令將 SPI 時鐘設在最高速 (18 MHz)。

```
U-Boot> sf probe 0 18000000
SF: Detected w25q128 with page size 4 KiB, total 16 MiB
U-Boot>
```

1.1.51 採用 SHA-1 硬體計算 checksum (僅限 NCU970)

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 1.1.35 章節採用 SHA-1 硬體計算 checksum，來縮短 bootm 開機時間。

利用 Power-on Setting 來改變 NAND flash page size 和 ECC type

U-Boot 在 NAND 初始化時，從 NAND ID 表中查出 NAND flash 的 page size 和 ECC type。如果你的 NAND flash 不存在 ID 表中，或者 page size 和 ECC type 與 ID table 中的數值不同，可以透過切換開發板上的 Power-on Setting 開關來改變 page size 和 ECC type 設定值。

1.1.52 切換 Config 6/7 開關改變 page size 設定值

Config 7 = 0, Config 6 = 0 將 NAND flash page size 設定成 2K Byte.

Config 7 = 0, Config 6 = 1 將 NAND flash page size 設定成 4K Byte.

Config 7 = 1, Config 6 = 0 將 NAND flash page size 設定成 8K Byte

1.1.53 切換 Config 6/7 開關改變 page size 設定值

Config 9 = 0, Config 8 = 0 將 NAND flash ECC type 設定成 BCH T12.

Config 9 = 0, Config 8 = 1 將 NAND flash ECC type 設定成 BCH T15.

Config 9 = 1, Config 8 = 0 將 NAND flash ECC type 設定成 BCH T24

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。

1.1.54 Tomato U-Boot 編譯命令

清除所有的 object code.

```
# make distclean
```

編譯 U-Boot

```
# make nuc970_tomato_defconfig
```

```
# make
```

1.1.55 Tomato U-Boot 鏈結位址

Tomato U-Boot 的鏈結位址是定義在 include/configs/nuc970_tomato.h
請找到以下定義

```
#define CONFIG_SYS_TEXT_BASE 0x2000000
```

當中定義 U-Boot 的鏈結位址是 0x2000000

1.1.56 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 0x8000 -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: 00008000

Entry Point: 00008000

注意事項

U-Boot SPI sub-system 目前只支持 NUC970/N9H30 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.

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.*