



**ARM926EJ-S™**  
**32-bit Microprocessor**

**NUC980 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 NUC980 microprocessor based system design. Nuvoton assumes no responsibility for errors or omissions.  
All data and specifications are subject to change without notice.*

*For additional information or questions, please contact: Nuvoton Technology Corporation.  
[www.nuvoton.com](http://www.nuvoton.com)*

## Table of Contents

1	概述 .....	5
2	配置 .....	6
2.1	編輯配置檔 .....	6
2.2	選單配置 .....	14
3	目錄架構 .....	15
4	編譯 U-Boot .....	17
4.1	編譯命令 .....	17
4.2	編譯產生的檔案 .....	17
4.3	Main U-Boot 鏈結位址 .....	17
4.4	SPL U-Boot 鏈結位址 .....	18
5	新增 SPI NOR 配置並移除 NAND 配置 .....	19
5.1	修改 include/configs/nuc980_evb.h .....	19
5.2	禁能 SPL .....	19
5.3	使能 NUC980 SPI 驅動 .....	19
5.4	使能 SPI Flash 介面支持 .....	19
5.5	使能 sf/spi 命令並禁能 nand 命令 .....	20
6	新增 SPI 配置 (NAND 也使能) .....	21
6.1	修改 include/configs/nuc980_evb.h .....	21
6.2	使能 NUC980 SPI 驅動 .....	21
6.3	使能 SPI Flash 介面支持 .....	21
7	新增 SPI NAND 配置 .....	22
7.1	修改 include/configs/nuc980_evb.h .....	22
7.2	修改環境變數大小 .....	22
7.3	使能 NUC980 SPI 驅動 .....	22
7.4	使能 SPI NAND Flash 介面支持 .....	22
8	新增 SD0 配置 .....	24

8.1	修改 include/configs/nuc980_evb.h .....	24
8.2	產生 NUC980 默認配置 .....	24
8.3	進入 menuconfig,調整配置 .....	24
9	新增 SD1 配置 .....	26
9.1	修改 include/configs/nuc980_evb.h .....	26
9.2	產生 NUC980 默認配置 .....	26
9.3	進入 menuconfig,調整配置 .....	26
10	U-Boot 命令 .....	28
10.1	Bootm 命令 .....	28
10.2	Go 命令 .....	29
10.3	網路相關命令 .....	30
10.4	Nand flash 相關命令 .....	34
10.5	SPI flash 相關命令 .....	38
10.6	SPI NAND Flash 相關命令 .....	40
10.7	記憶體命令 .....	40
10.8	USB 命令 .....	44
10.9	環境變數相關的命令 .....	51
10.10	MMC 命令 .....	52
10.11	MTD 命令 .....	56
10.12	UBI 命令 .....	58
10.13	YAFFS2 命令 .....	62
11	環境變數 .....	66
11.1	環境變數的配置 .....	66
11.2	預設的環境變數 .....	66
11.3	命令腳本 .....	67
11.4	新增的環境變數 .....	68
12	mkimage 工具 .....	69
12.1	使用 mkimage 產生影像檔 .....	69
12.2	Checksum 計算方式 (SHA-1 或 crc32) .....	69
13	Watchdog timer .....	72

13.1	Watchdog timer 配置 .....	72
13.2	Watchdog timer 環境變數 .....	72
13.3	Watchdog timer 的時間長度.....	72
14	GPIO.....	73
14.1	NUC980 GPIO .....	73
14.2	GPIO 驅動程式介面 .....	73
14.3	使用範例.....	74
15	網路測試環境 .....	75
15.1	設置固定 IP 位址 .....	75
15.2	TFTP 和 DHCP 伺服器 .....	77
16	加快 SPI flash 開機速度 .....	81
16.1	提高 SPI 運作速度 .....	81
16.2	採用 SHA-1 硬體計算 checksum .....	81
17	NAND flash 相關 Power-on Setting .....	82
17.1	切換 Config 6/7 開關改變 page size 設定值 .....	82
17.2	切換 Config 6/7 開關改變 page size 設定值 .....	82
18	修改歷史.....	83

## 1 概述

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

NUC980 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>.

## 2 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.

### 2.1 編輯配置檔

NUC980 配置檔位於 include/configs/nuc980\_evb.h  
以下分段介紹配置檔 nuc980\_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

```
efine 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_NUC980_ETH
#define CONFIG_NUC980_ETH
#define CONFIG_NUC980_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\_NUC980\_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\_NUC980\_ETH0: 使用 NUC980 EMAC0
- CONFIG\_NUC980\_ETH1: 使用 NUC980 EMAC1
- CONFIG\_NUC980\_ETH: 支援 NUC980 Ethernet
- CONFIG\_NUC980\_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

```
#ifndef CONFIG_SYS_USE_NANDFLASH
#define CONFIG_NAND_NUC980

#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\_NUC980: 開啟 NUC980 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_NUC980_SPI        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\_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 MMC support */

#define CONFIG_NUC980_MMC

#define CONFIG_CMD_FAT

#define CONFIG_DOS_PARTITION

/*#define CONFIG_NUC980_EMMC */ /* Don't enable eMMC(CONFIG_NUC980_EMMC)
and NAND(CONFIG_NAND_NUC980) 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\_NUC980\_MMC: 編譯 NUC980 驅動程式
- CONFIG\_CMD\_FAT: 支持 FAT 相關命令
- CONFIG\_DOS\_PARTITION: 支持 DOS 分區
- CONFIG\_NUC980\_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_NUC980  
  
#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\_NUC980: 支持 NUC980 芯片 USB 2.0
- CONFIG\_DOS\_PARTITION: 支持 DOS 分區

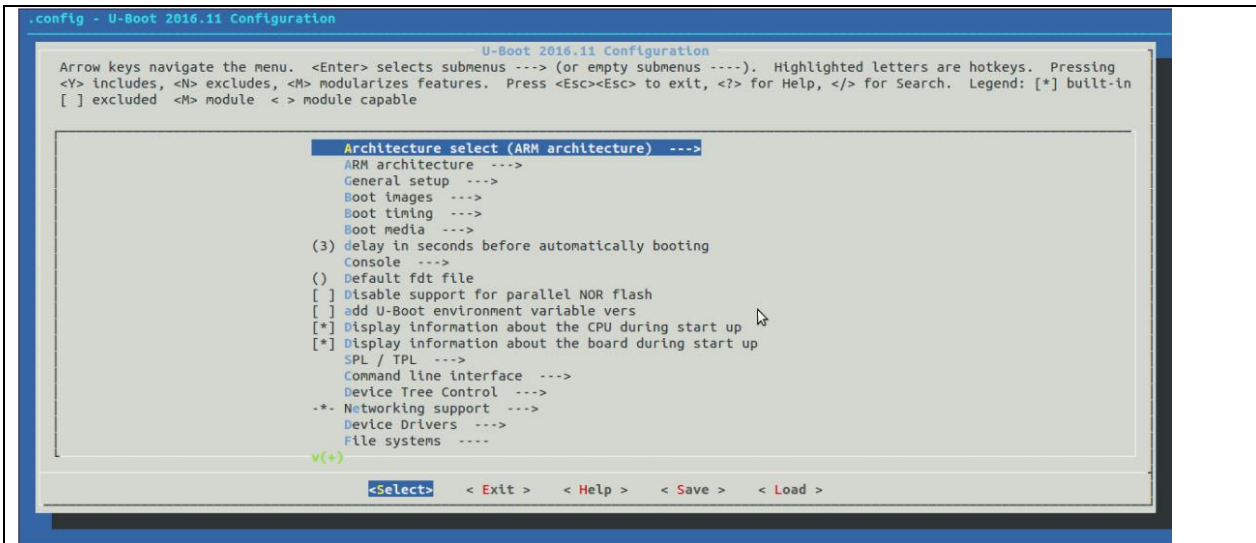
```
#define CONFIG_NUC980_GPIO  
  
/*  
 * Size of malloc() pool  
 */  
  
#define CONFIG_SYS_MALLOC_LEN    (1024*1024)  
  
#define CONFIG_STACKSIZE    (32*1024)    /* regular stack */  
  
#endif
```

- CONFIG\_NUC980\_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

## 2.2 選單配置

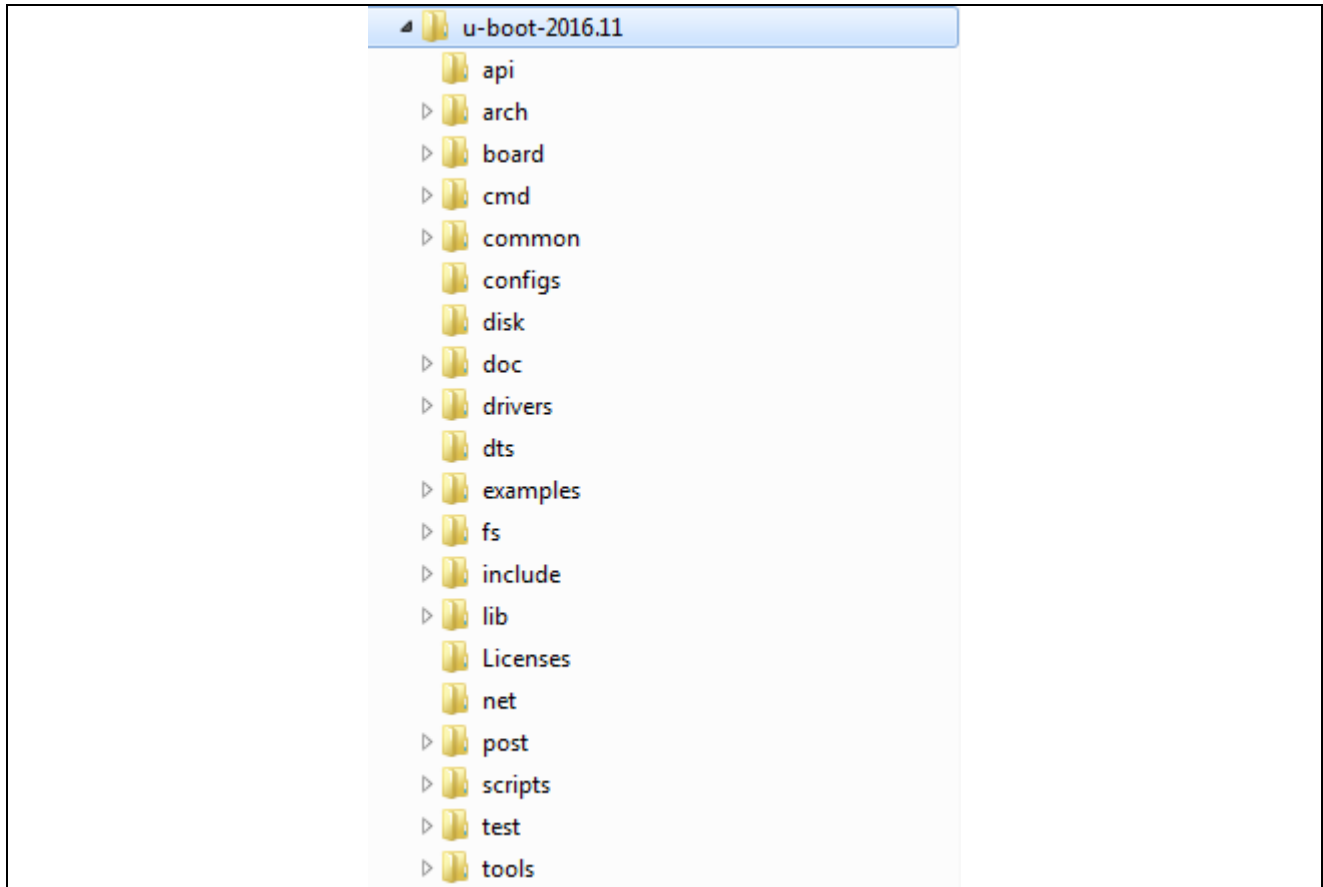
U-Boot v2016.11 支持透過命令 “make menuconfig” 來進行選單配置

```
# make menuconfig
```



### 3 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
- NUC980 CPU 相關的源代碼放在 arch/arm/cpu/arm926ejs/nuc980.
- board: 包含板子相關的源代碼
- NUC980 板子相關的源代碼放在 board/nuvoton/nuc980\_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.
- NUC980 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/nuc980\_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. NUC980 的配置檔就放在 include/configs/nuc980\_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.



## 4 編譯 U-Boot

### 4.1 編譯命令

清除所有的 object code.

```
# make distclean
```

編譯 U-Boot

```
# make nuc980_defconfig
```

```
# make
```

### 4.2 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 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

### 4.3 Main U-Boot 鏈結位址

Main U-Boot 的鏈結位址是定義在 include/configs/nuc980\_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/nuc980\_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.

#### 4.4 SPL U-Boot 鏈結位址

SPL U-Boot 的鏈結位址定義在 include/configs/nuc980\_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
```

## 5 新增 SPI NOR 配置並移除 NAND 配置

NUC980 默認的配置是支持 NAND 不支持 SPI.如果要開啟 SPI 配置並移除 NAND 配置,請修改 include/configs/nuc980\_evb.h ,然後透過 “make menuconfig” 命令新增相關配置.

### 5.1 修改 include/configs/nuc980\_evb.h

修改 include/configs/nuc980\_evb.h,打開定義 “CONFIG\_SYS\_USE\_SPIFLASH” 並關掉定義 “CONFIG\_SYS\_USE\_NANDFLASH” ,並將環境變數改為存放在 SPI flash.

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

### 5.2 禁能 SPL

透過” make menuconfig” ,禁能 SPL.

```
-> SPL / TPL ---->
[ ] SPL
```

### 5.3 使能 NUC980 SPI 驅動

透過” make menuconfig” ,使能 NUC980 SPI 驅動並選擇 SPI 運作在 Quad 或正常模式.

```
-> Device Drivers
-> SPI Support
[*] NUC980 SPI driver
Select NUC980 SPI in Quad mode or Normal mode (Quad mode) ---->
```

### 5.4 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
```

### 5.5 使能 sf/spi 命令並禁能 nand 命令

透過” make menuconfig” ,使能 sf/spi 命令,並禁能 nand 命令.

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

## 6 新增 SPI 配置 (NAND 也使能)

NUC980 默認的配置是不支持 SPI.如果要開啟 SPI 配置,請修改 include/configs/nuc980\_evb.h , 然後透過 “make menuconfig” 命令新增相關配置.

### 6.1 修改 include/configs/nuc980\_evb.h

修改 include/configs/nuc980\_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
```

### 6.2 使能 NUC980 SPI 驅動

透過” make menuconfig” ,使能 NUC980 SPI 驅動並選擇 SPI 運作在 Quad 或正常模式.

```
-> Device Drivers
    -> SPI Support
        [*] NUC980 SPI driver
            Select NUC980 SPI in Quad mode or Normal mode (Quad mode) --->
```

### 6.3 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
```

## 7 新增 SPI NAND 配置

NUC980 默認的配置是不支持 SPI.如果要開啟 SPI 配置,請修改 include/configs/nuc980\_evb.h , 然後透過 “make menuconfig” 命令新增相關配置.

### 7.1 修改 include/configs/nuc980\_evb.h

修改 include/configs/nuc980\_evb.h,打開定義 “CONFIG\_SYS\_USE\_SPIFLASH” .

```
#define CONFIG_SYS_USE_SPIFLASH
#define CONFIG_ENV_IS_IN_NAND
```

### 7.2 修改環境變數大小

修改 include/configs/nuc980\_evb.h 將” CONFIG\_ENV\_SIZE” 的定義改為 0x20000.

```
#ifdef CONFIG_ENV_IS_IN_NAND
#define CONFIG_ENV_OFFSET      0x80000
#define CONFIG_ENV_SIZE        0x20000
```

### 7.3 使能 NUC980 SPI 驅動

透過” make menuconfig” ,使能 NUC980 SPI 驅動並選擇 SPI 運作在 Quad 或正常模式.

```
-> Device Drivers
    -> SPI Support
        [*] NUC980 SPI driver
            select NUC980 SPI in Quad mode or Normal mode (Quad mode) --->
```

### 7.4 使能 SPI NAND 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. 同時始能 SPI NAND Flash 並根據開發板上的 SPI NAND Flash 廠牌開啟相關的支持,下面的範例是開發板上的 SPI NAND flash 廠牌是華邦.

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

Select SPI NAND Flash (winbond SPI NAND flash support) --->

## 8 新增 SD0/eMMC0 配置

NUC980 默認的配置是不支持 SD0/eMMC0. 如果要開啟 SD0/eMMC0 配置,請修改 include/configs/nuc980\_evb.h,然後透過 “make menuconfig” 命令新增相關配置.

### 8.1 修改 include/configs/nuc980\_evb.h

修改 include/configs/nuc980\_evb.h,關掉定義 “CONFIG\_SYS\_USE\_NANDFLASH” 和 “CONFIG\_ENV\_IS\_IN\_NAND”,並打開定義 “CONFIG\_ENV\_IS\_IN\_MMC”,將環境變數改為存放在 SD0/eMMC0.

```
/*#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
```

### 8.2 產生 NUC980 默認配置

```
# make distclean
# make nuc980_defconfig
```

### 8.3 進入 menuconfig,調整配置

透過 “make menuconfig”,修改配置.

```
# make menuconfig
```

#### 8.3.1 禁能 SPL

透過 “make menuconfig”,禁能 SPL.

```
-> SPL / TPL --->
[ ] SPL
```

#### 8.3.2 使能 MMC 命令並禁能 NAND 命令

透過 “make menuconfig”,使能 MMC 命令並禁能 NAND 命令.

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



☐ nand

### 8.3.3 使能 MMC 相關配置

透過” make menuconfig” ,使能 SD0 以及 MMC 相關配置.

```
-> Device Drivers
    -> MMC Host controller Support
        -> NUC980 MMC support (NUC980_MMC [=y])
            [*] Enable MMC support
            [*] Generic MMC support
            [*] NUC980 MMC support
            [ ] NUC980 SD1 support (SD Host Port F)
            [*] NUC980 SD0 support (FMI Port C)
```

### 8.3.4 禁能 NAND 驅動配置

透過” make menuconfig” ,禁能 NAND 驅動配置.

```
-> Device Drivers
    -> NAND Device Support
        [ ] NUC980 NAND support
```

## 9 新增 SD1/eMMC1 配置

NUC980 默認的配置是不支持 SD1/eMMC1. 如果要開啟 SD1/eMMC1 配置,請修改 include/configs/nuc980\_evb.h,然後透過 “make menuconfig” 命令新增相關配置.

### 9.1 修改 include/configs/nuc980\_evb.h

修改 include/configs/nuc980\_evb.h,關掉定義 “CONFIG\_SYS\_USE\_NANDFLASH” 和 “CONFIG\_ENV\_IS\_IN\_NAND”,並打開定義 “CONFIG\_ENV\_IS\_IN\_MMC”,將環境變數改為存放在 SD1/eMMC1.

```
/*#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
```

### 9.2 產生 NUC980 默認配置

```
# make distclean
# make nuc980_defconfig
```

### 9.3 進入 menuconfig,調整配置

透過 “make menuconfig”,修改配置.

```
# make menuconfig
```

#### 9.3.1 禁能 SPL

透過 “make menuconfig”,禁能 SPL.

```
-> SPL / TPL --->
[ ] SPL
```

#### 9.3.2 使能 MMC 命令並禁能 NAND 命令

透過 “make menuconfig”,使能 MMC 命令並禁能 NAND 命令.

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

☐ nand

### 9.3.3 使能 MMC 相關配置

透過” make menuconfig” ,使能 SD1 以及 MMC 相關配置.

```
-> Device Drivers
    -> MMC Host controller Support
        -> NUC980 MMC support (NUC980_MMC [=y])
            [*] Enable MMC support
            [*] Generic MMC support
            [*] NUC980 MMC support
            [*]   NUC980 SD1 support (SD Host Port F)
            [ ]   NUC980 SD0 support (FMI Port C)
```

### 9.3.4 禁能 NAND 驅動配置

透過” make menuconfig” ,禁能 NAND 驅動配置.

```
-> Device Drivers
    -> NAND Device Support
        [ ]   NUC980 NAND support
```

## 10 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)

### 10.1 Bootm 命令

在介紹網路、NAND、SPI、USB、MMC 等相關命令之前, 特別先介紹 bootm 命令. 因為 Linux 內核影像檔會儲存在網路、NAND、SPI、USB、MMC 等儲存媒介, 透過這些儲存媒介相關的命令將 Linux 內核下載到 DDR 之後, 再透過 bootm 命令完成 Linux 內核的開機. 因此, bootm 命令是用來啟動由 mkimage 工具產生的 Linux 內核或其他應用程式. 相較於 bootm 命令, go 命令 (10.2 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 ...
```

## 10.2 Go 命令

- go  
執行應用程式

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

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

下面這個範例是執行一個已經下載到 DDR 0x100000 位址的程式.

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

Hello world!
```

### 10.3 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> 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 NUC980 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 ...

```

#### 10.4 Nand flash 相關命令

- nand

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 ...
```

## 10.5 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 30 MHz.

```
U-Boot> sf probe 0 30000000
```

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 30000000

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 ...
```

## 10.6 SPI NAND Flash 相關命令

U-Boot 支援 SPI NAND Flash. 讀取 SPI NAND 中的內核時,記得用命令 `nand`,而非 `sf`.

下面這個範例是將 Linux 內核的影像檔從 SPI NAND Flash 讀取到 DDR. 首先,用 “`nand`” 命令將一個大小為 0x190580 位元的 Linux 內核影像檔 從 SPI NAND 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> 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 ...
```

## 10.7 記憶體命令



- md  
顯示記憶體內容.

```
U-Boot> help md

md - memory display

Usage:
md [.b, .w, .l] address [# of objects]

U-Boot>
```

下面這個範例是顯示位址 0x10000 到 0x100ff 的記憶體內容.

```
U-Boot> md 0x1000
00001000: bffbcf5c 5ffb56ff fcff5f41 ff67760b  \....V._A...vg.
00001010: fcd227e3 dffefeeb 70cf7cb3 dbefc7cb  .'.....|.p....
00001020: fbda3e3b eb3e9ebb aa3abc95 e5fbbb2f  ;>....>....:/...
00001030: ffb3b319 effe9d7d bfbbeb09 ff7b4f31  ....}.....10{.
00001040: f7bf3973 eaff296c e6fce35e 6fffc7d7  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 cfbbbecc 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>
```

## 10.8 USB 命令

- usb

```
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 ...
```

## 10.9 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>
```

## 10.10 MMC 命令

- **mmc**  
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

U-Boot>

NUC980 支持的 mmc device 包括 SD port 0 和 eMMC. 用戶可以根據平台上實際支持的 mmc device 來修改配置檔 (nuc980\_evb.h) 當中以下兩個定義

```
#define CONFIG_NUC980_SD_PORT0
```

```
#define CONFIG_NUC980_EMMC
```

默認的設定是打開 SD port 0, eMMC 因不能和 NAND 同時使用, 默認是關掉的.

如果 SD port 0 和 eMMC 都打開, mmc device 編號如下:

Device 編號 0 是 SD port 0

Device 編號 2 是 eMMC

假設用戶的平台只支持 eMMC (不支持 SD port 0), 必須修改配置檔 (nuc980\_evb.h), 關掉 CONFIG\_NUC980\_SD\_PORT0 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

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 1 (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 1
mmc1(part 0) is current device
U-Boot> mmc erase 0x400 0x400

MMC erase: dev # 1, block # 1024, count 1024 ... 1024 blocks erase: OK
U-Boot> mmc write 0x8000 0x400 0x400
```

```
MMC write: dev # 1, block # 1024, count 1024 ... 1024 blocks write: OK
U-Boot> mmc read 0x500000 0x400 0x400

MMC read: dev # 1, 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 ...
```

### 10.11 MTD 命令

- **mtdparts**

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
'mtids' - linux kernel mtd device id <-> u-boot device id mapping
mtids=<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 nuc980\_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 MTIDS_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>
```

## 10.12 UBI 命令

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

```
U-Boot> help ubi

ubi - ubi commands

Usage:
ubi part [part] [offset]
    - Show or set current partition (with optional VID header offset)
ubi info [l[ayout]] - Display volume and ubi layout information
ubi create[vol] volume [size] [type] - create volume name with size
ubi write[vol] address volume size - write volume from address with size
ubi read[vol] address volume [size] - Read volume to address with size
ubi remove[vol] volume - Remove volume

[Legends]
volume: character name
size: specified in bytes
```

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

```
U-Boot>
```

ubi part 顯示或設置目前的分區

```
U-Boot> ubi part user
```

```
Creating 1 MTD partitions on "nand0":
```

```
0x000001500000-0x000008000000 : "mtd=2"
```

```
UBI: attaching mtd1 to ubi0
```

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

```
UBI: logical eraseblock size: 126976 bytes
```

```
UBI: smallest flash I/O unit: 2048
```

```
UBI: VID header offset: 2048 (aligned 2048)
```

```
UBI: data offset: 4096
```

```
UBI: attached mtd1 to ubi0
```

```
UBI: MTD device name: "mtd=2"
```

```
UBI: MTD device size: 107 MiB
```

```
UBI: number of good PEBs: 855
```

```
UBI: number of bad PEBs: 1
```

```
UBI: max. allowed volumes: 128
```

```
UBI: wear-leveling threshold: 4096
```

```
UBI: number of internal volumes: 1
```

```
UBI: number of user volumes: 1
```

```
UBI: available PEBs: 17
```

```
UBI: total number of reserved PEBs: 838
```

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

```
UBI: max/mean erase counter: 6/4
```

```
U-Boot>
```

ubi info 顯示容積及 ubi 信息

```
U-Boot> ubi info 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: LZO
UBIFS: reserved for root: 5114338 bytes (4994 KiB)
U-Boot>
```

ubifsls 列出 ubifs 文件系統中的目錄或文件

```
U-Boot> help ubifsls
ubifsls - list files in a directory
Usage:
ubifsls [directory]
    - list files in a 'directory' (default '/')
U-Boot>
```

```
U-Boot> ubifs1s
<DIR>          160  Thu Jan 01 00:08:09 1980  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>
```

### 10.13 YAFFS2 命令

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

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

```
ytrace - show/set yaffs trace mask
yumount - unmount yaffs
ywr      - write file to yaffs
ywrn     - write file from memory to yaffs
```

ydevconfig 配置 YAFFS 掛載點

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

ydevls 顯示 YAFFS 掛載點

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

ymount 掛載 YAFFS

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

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

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

ymkdir 建立一個目錄

```
U-Boot> ymkdir nand/test
U-Boot> yls -l nand
```



test	2032	257	directory
lost+found	2032	2	directory

yrmdir 刪除一個目錄

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

ywr / 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/wrn.bin 0x200000			
Copy nand/wrn.bin to 0x00200000... [DONE]			

yrn 刪除一個檔案

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

## 11 環境變數

### 11.1 環境變數的配置

環境變數可以存放在 NAND flash、SPI flash 或 eMMC,可以透過修改配置檔 (nuc980\_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 (nuc980\_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.

### 11.2 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 nuc980\_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 nuc980\_evb.h 中的 CONFIG\_BOOTDELAY
- **ethact**  
控制目前哪一個 interface 的狀態為 active, 因為 nuc980 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 nuc980\_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

### 11.3 命令腳本

下列是和命令腳本相關的環境變數

- **bootcmd**

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

```
U-Boot> setenv bootcmd sf probe 0 50000000\; 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 10.11 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
```

#### 11.4 新增的環境變數

除了 U-Boot 支援的環境變數之外,NUC980 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
```

## 12 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” .

### 12.1 使用 mkimage 產生影像檔

mkimage 工具放在 tools/mkimage, 下面的範例是將 ARM Linux 內核 (980image) 透過 mkimage 打包成一個影像檔 (980image.ub). Linux 內核下載位址是 0x8000, 程式進入點是 0x8000.

```
u-boot/tools# ./mkimage -A arm -O linux -T kernel -a 0x8000 -e 0x8000 -d 980image 980image.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 980image

### 12.2 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；NUC980 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.NUC980 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, NUC980 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 nuc980\_evb.h 中的 CONFIG\_NUC980\_HW\_CHECKSUM 選項打開.

```
u-boot/tools# ./mkimage -A arm -O linux -T kernel -S sha1 -a 0x8000 -e 0x8000 -d 980image 980image.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: 設置程式進入點位址為 0x8000
- d: 設置影像檔的來源為 980image

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 nuc980\_evb.h 中的 CONFIG\_NUC980\_HW\_CHECKSUM 選項.

```
u-boot/tools# ./mkimage -A arm -O linux -T kernel -S crc32 -a 0x8000 -e 0x8000 -d 980image 980image.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: 設置程式進入點位址為 0x8000
- d: 設置影像檔的來源為 980image

## 13 Watchdog timer

### 13.1 Watchdog timer 配置

若要將 NUC980 的 watchdog timer 功能打開,請將 nuc980\_evb.h 當中以下兩個定義打開.

```
#define CONFIG_NUC980_WATCHDOG  
  
#define CONFIG_HW_WATCHDOG
```

反之,若要將 NUC980 的 watchdog timer 功能關掉,請將上面兩個定義注釋掉.

### 13.2 Watchdog timer 環境變數

當 NUC980 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 NUC980 的配置 watchdog timer 功能是關掉時,修改環境變數 “watchdog” 是沒有意義的.

### 13.3 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.



## 14 GPIO

U-Boot GPIO 最常見的應用是拿來點亮 LED. NUC980 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.

### 14.1 NUC980 GPIO

NUC980 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. NUC980 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 NUC980 GPIO 驅動程式時, 都必須傳入 GPIO 編號.

### 14.2 GPIO 驅動程式介面

NUC980 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.

### 14.3 使用範例

下面這個範例是將 GPIO port G0 ~ port G5 的數值設為 0x101010.

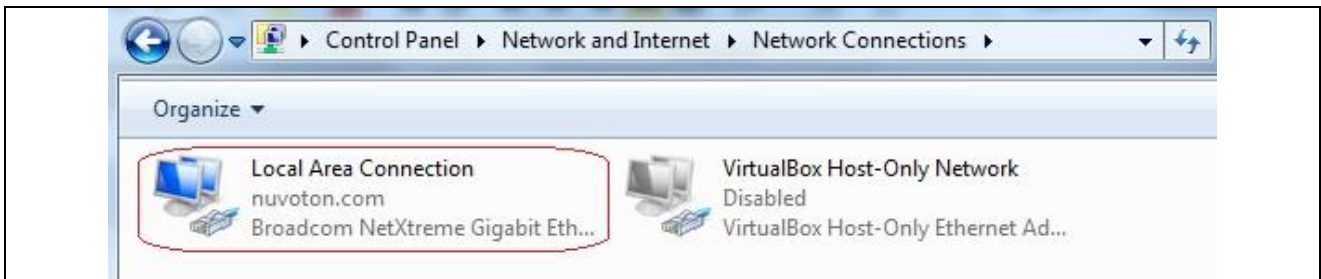
```
gpio_request(GPIO_PG0, NULL);  
gpio_direction_output(GPIO_PG0, 0);  
gpio_request(GPIO_PG1, NULL);  
gpio_direction_output(GPIO_PG1, 1);  
gpio_request(GPIO_PG2, NULL);  
gpio_direction_output(GPIO_PG2, 0);  
gpio_request(GPIO_PG3, NULL);  
gpio_direction_output(GPIO_PG3, 1);  
gpio_request(GPIO_PG4,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

## 15 網路測試環境

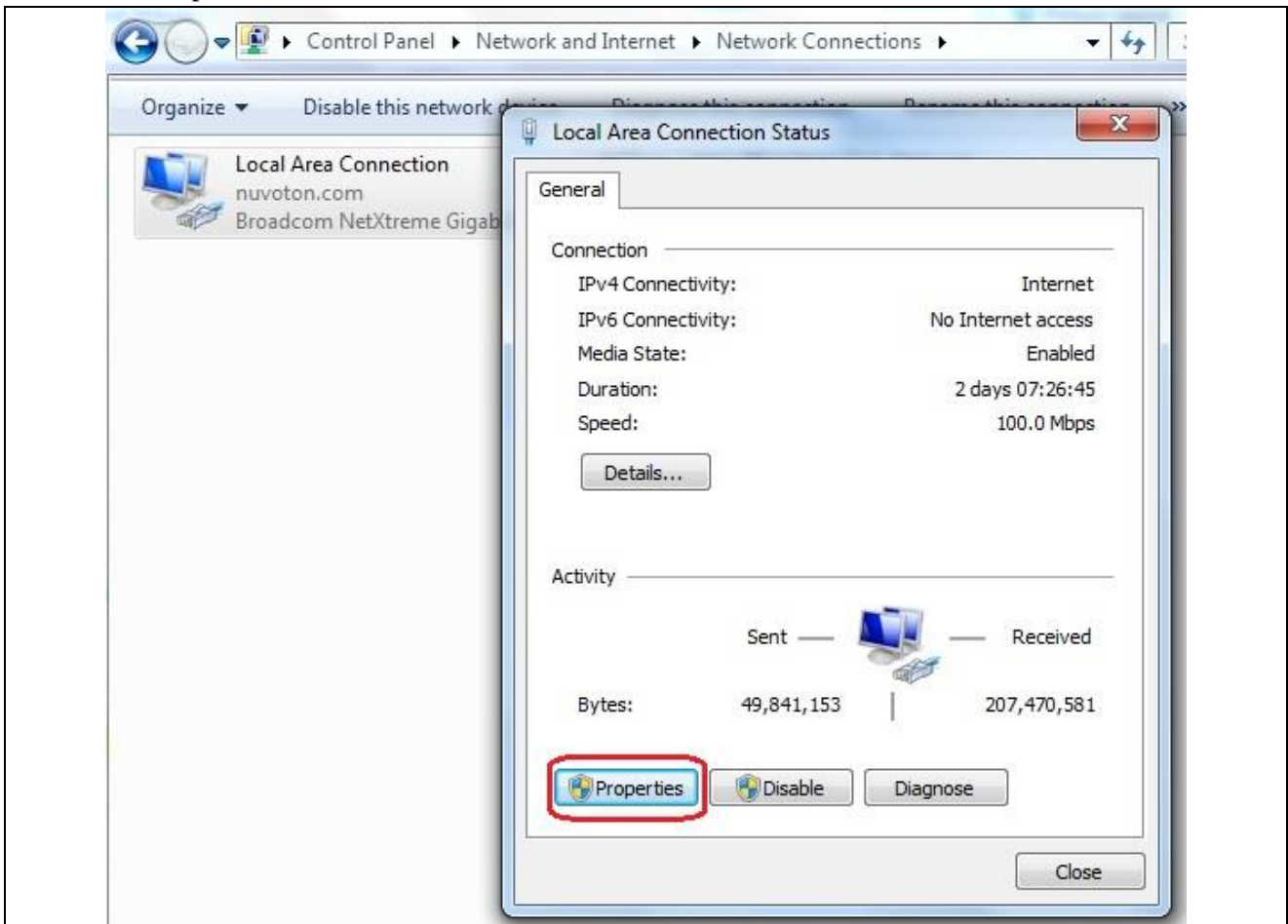
我們需要一部 PC, 該 PC 有固定 IP 位址並且安裝 TFTP/DHCP 伺服器應用程式。

### 15.1 設置固定 IP 位址

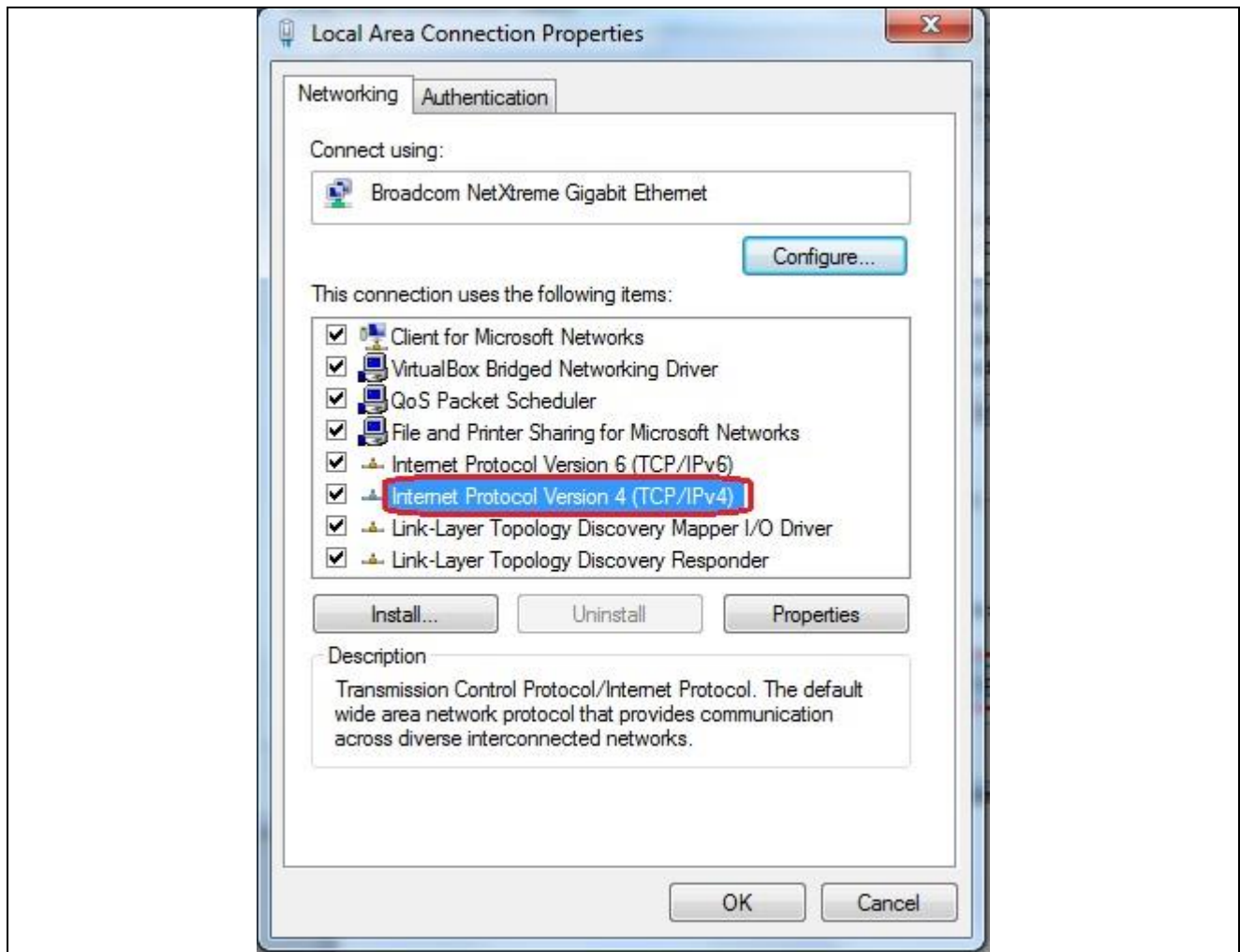
點選 Local Area Connection (Control Panel -> Network and Internet -> Network Connections)



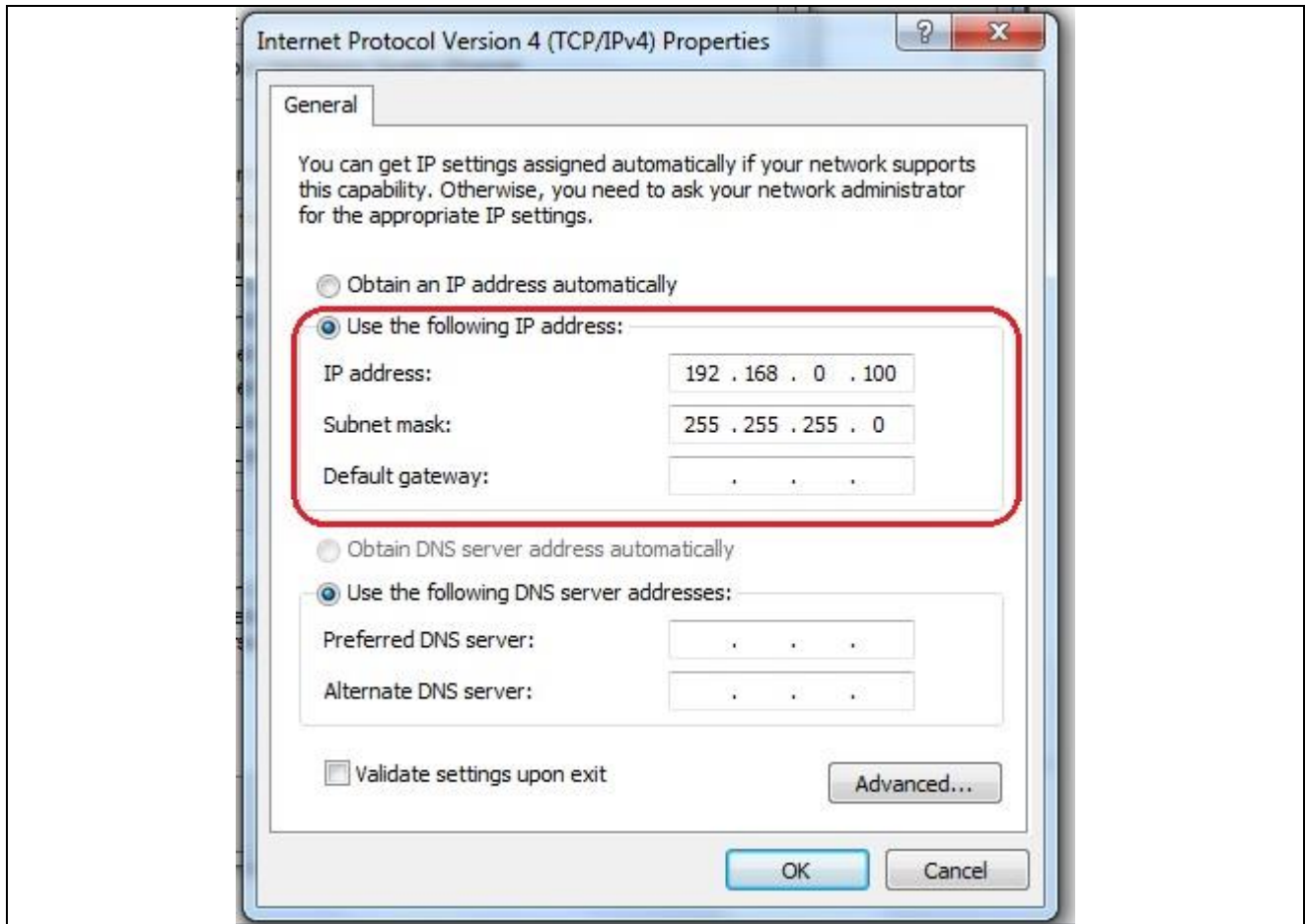
點選 Properties



點選 Internet Protocol Version 4 (TCP/IPv4)

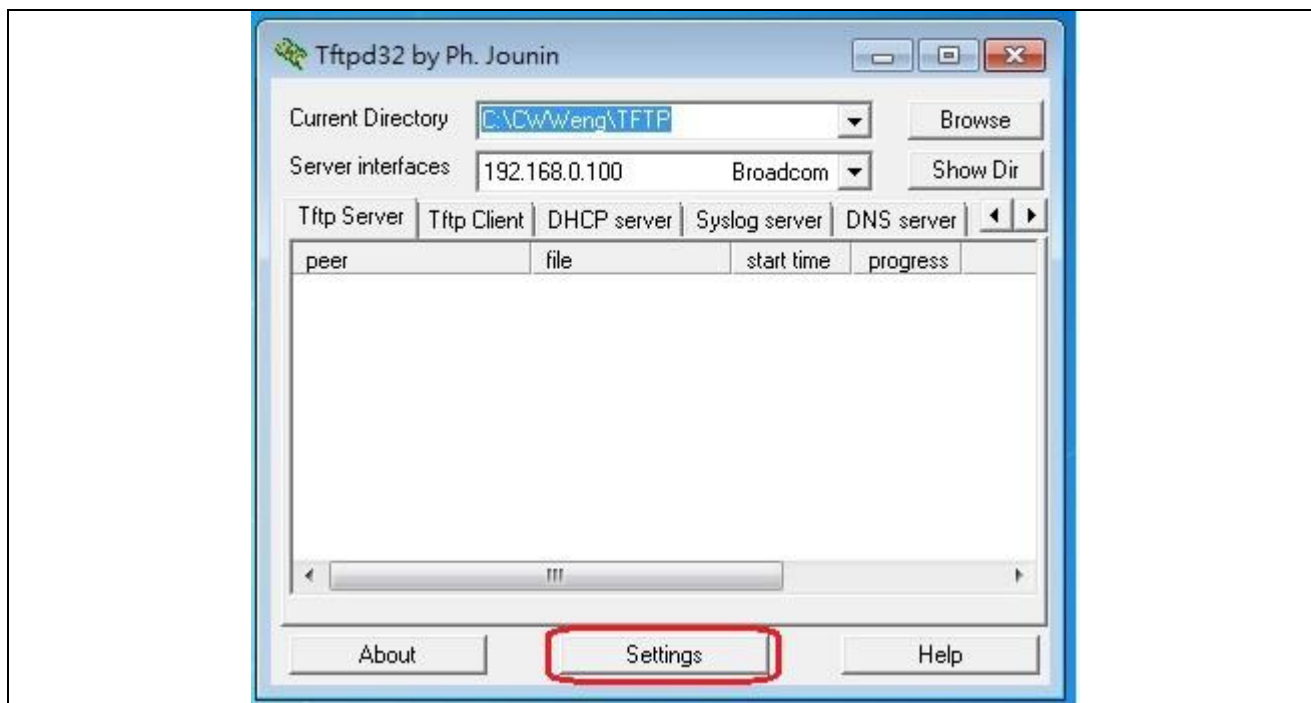


設置固定 IP 位址

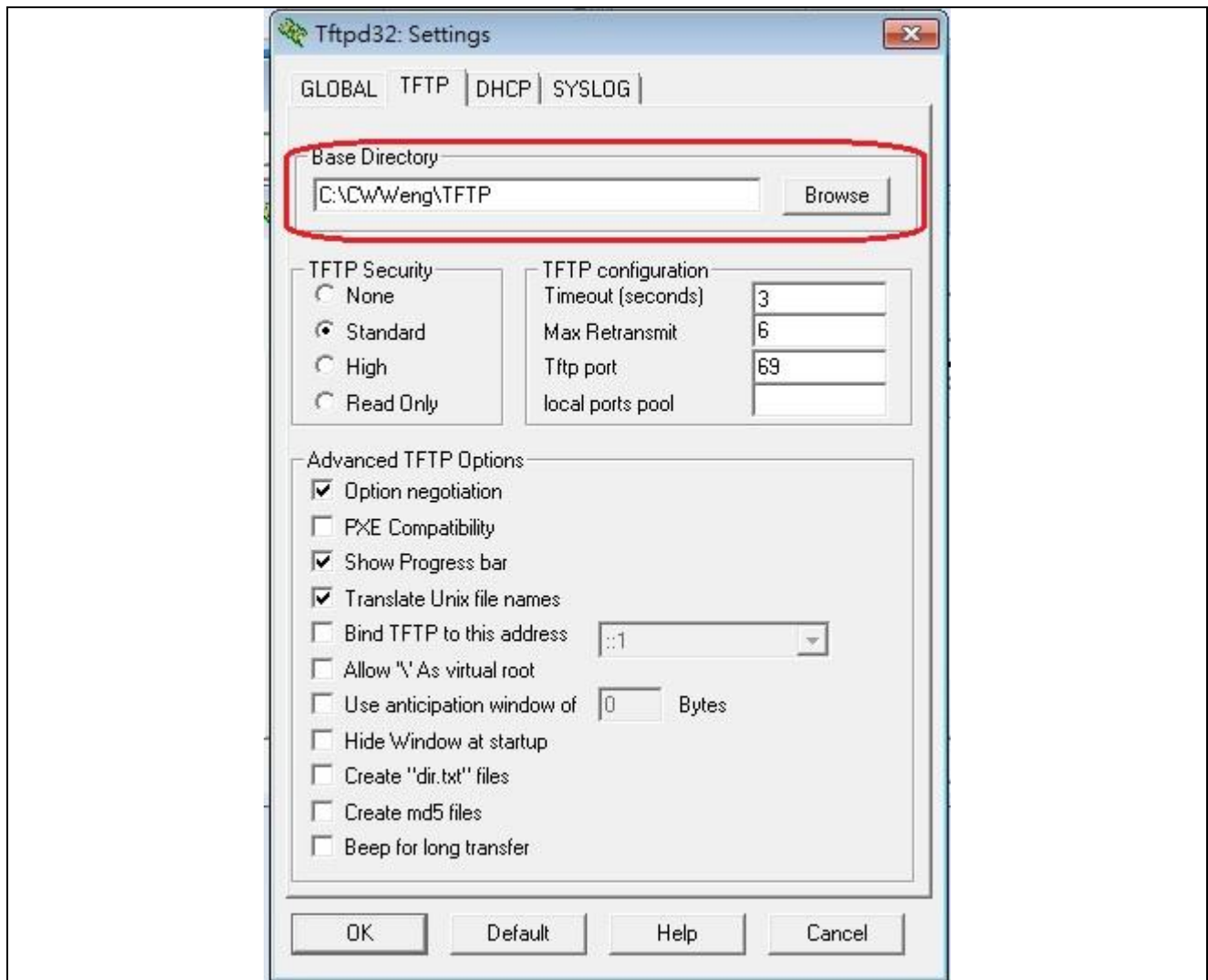


## 15.2 TFTP 和 DHCP 伺服器

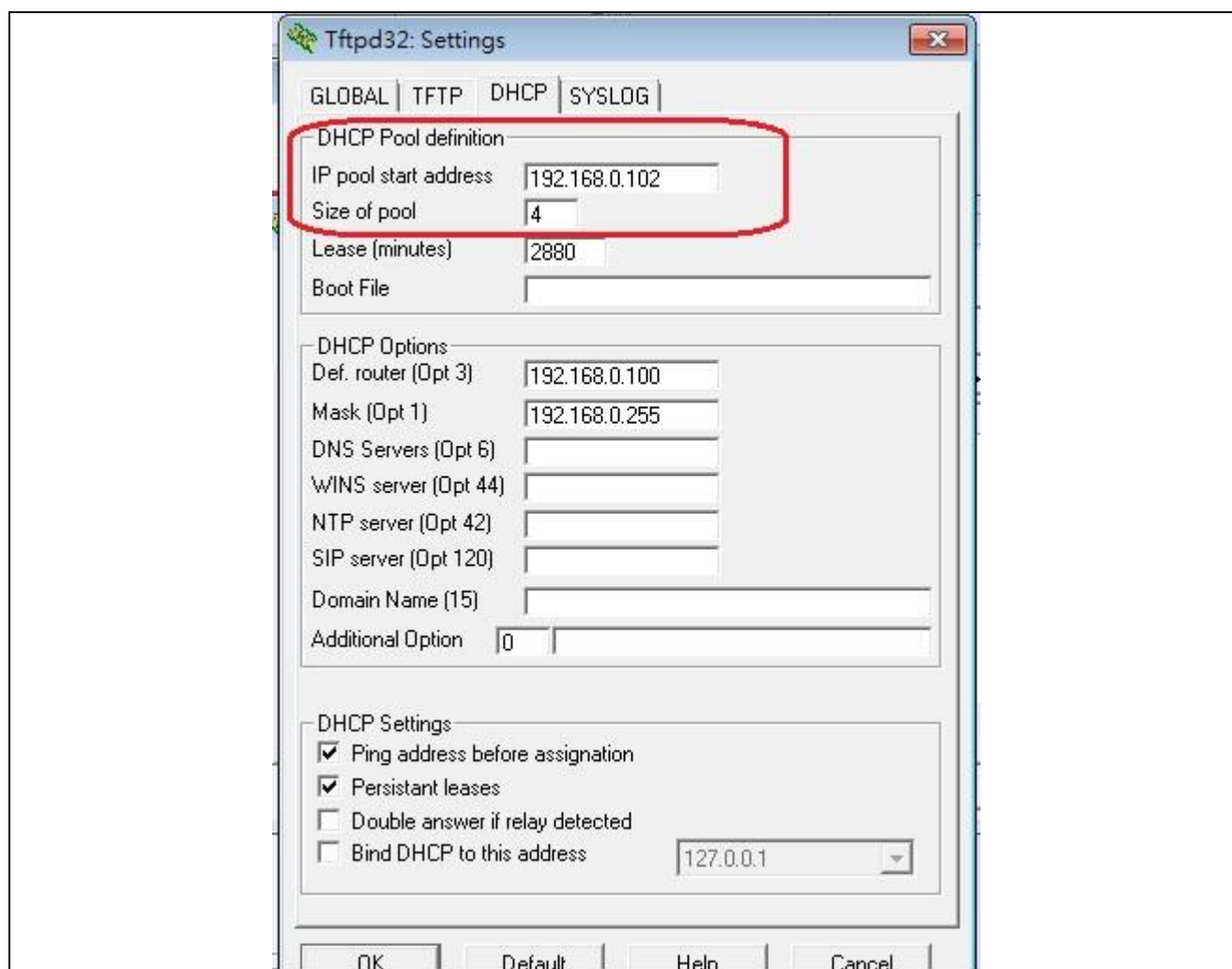
TFTPD32 是一個免費、源代碼公開的應用程式, 它包含 TFTP 和 DHCP 伺服器等功能. 可以從下面網址進行下載<http://www.jounin.net/tftp-server-for-windows.html>



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





## 16 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.

### 16.1 提高 SPI 運作速度

提高 SPI 運作速度可以從下列方法著手.

- 讓 SPI 運作在 Quad 模式
- 調高 SPI 時鐘

以下的範例是透過 sf probe 命令將 SPI 時鐘設在 50 MHz.

```
U-Boot> sf probe 0 50000000
SF: Detected w25q128 with page size 4 KiB, total 16 MiB
U-Boot>
```

### 16.2 採用 SHA-1 硬體計算 checksum

bootm 開機過程中採用硬體計算 checksum,也可以縮短開機時間.請參考 12.2 章節採用 SHA-1 硬體計算 checksum,來縮短 bootm 開機時間.

## 17 NAND flash 相關 Power-on Setting

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.

### 17.1 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

### 17.2 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 設定成不使用 ECC.

Config 9 = 0, Config 8 = 1 將 NAND flash ECC type 設定成 BCH T12.

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

## 18 修改歷史

Date	Revision	Description
2019.06.24	1.00	初版
2020.09.08	1.10	錯誤修正

### 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.