NUC970/N9H30 U-Boot v2016.11 User Manual

1 U-Boot user manual

The U-Boot utility is a multi-platform, open-source, universal boot-loader with comprehensive support for loading and managing boot images, such as the Linux kernel. It supports the following features:

- Network download: TFTP, BOOTP, DHCP
- Serial download: s-record, binary (via Kermit)
- Flash management: erase, read, update, yaffs2
- Flash types: SPI flash, NAND flash
- Memory utilities: dump, compare, copy, write
- Interactive shell: commands with scripting features

NUC970/N9H30 U-Boot version is v2016.11. It is downloaded from http://www.denx.de/wiki/U-Boot/SourceCode
To know more detailed description of U-Boot can visit U-Boot official website http://www.denx.de/wiki/view/DULG/UBoot

1.1 Configuration

U-Boot supports two ways for configuration. One is modifying the definitions in configuration file, and another is by menu configuration.

1.1.1 Editing configuration file

U-Boot is configurable by modifying the definitions in configuration file.
NUC970 and N9H30 configuration files are located under include/configs/ directory.
Below are the definitions in nuc970_evb.h.

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

#define CONFIG_SYS_TEXT_BASE            0xE00000

#define CONFIG_SYS_LOAD_ADDR            0x8000

#define CONFIG_SYS_HZ                   1000
#define CONFIG_SYS_MEMTEST_START         0xA00000
#define CONFIG_SYS_MEMTEST_END           0xB00000

#define CONFIG_ARCH_CPU_INIT
#undef  CONFIG_USE_IRQ

#define CONFIG_CMDLINE_TAG      1       /* enable passing of ATAGs      */
#define CONFIG_SETUP_MEMORY_TAGS 1
#define CONFIG_INITRD_TAG       1

#define CONFIG_CMD_TIMER
/* #define CONFIG_NUC970_HW_CHECKSUM */
```

- EXT_CLK: External crystal clock frequency

- CONFIG_SYS_TEXT_BASE: U-Boot text base address
- CONFIG_SYS_LOAD_ADDR: the load address for downloading image
- CONFIG_SYS_HZ: timer frequency
- CONFIG_SYS_MEMTEST_START: start address of memory test
- CONFIG_SYS_MEMTEST_END: end address of memory test
- CONFIG_CMD_TIMER: Use timer relative command
- CONFIG_NUC970_HW_CHECKSUM: Use SHA-1 to calculate the checksum of Linux kernel (otherwise, use crc32 to calculate checksum), It should cooperate with mkimage tool, Please reference chapter 1.8.2 .

```
#define CONFIG_SYS_USE_SPIFLASH
#define CONFIG_SYS_USE_NANDFLASH
#define CONFIG_SYS_NO_FLASH    // that is, no *NOR* flash
#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_DISPLAY_CPUINFO */

#define CONFIG_BOOTDELAY        3

#define CONFIG_SYS_SDRAM_BASE   0
#define CONFIG_NR_DRAM_BANKS    2     /* there are 2 sdram banks for nuc970
*/
#define CONFIG_SYS_INIT_SP_ADDR 0xBC008000

#define CONFIG_BAUDRATE         115200
#define CONFIG_SYS_BAUDRATE_TABLE       {115200, 57600, 38400}

/*#define CONFIG_NUC970_EMAC1*/
/*#define CONFIG_CMD_NET */
#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_KPI_NUC970*/
/*#undef CONFIG_SYS_ICACHE_OFF */
/*#undef CONFIG_SYS_DCACHE_OFF */
```

```
/*#define CONFIG_SYS_ICACHE_OFF*/
#define CONFIG_SYS_DCACHE_OFF
```

- CONFIG_SYS_USE_SPIFLASH: Use SPI flash
- CONFIG_SYS_USE_NANDFLASH: Use NAND flash
- CONFIG_ENV_IS_IN_NAND: Environment variables are stored in NAND flash
- CONFIG_ENV_IS_IN_SPI_FLASH: Environment variables are stored in SPI flash
- CONFIG_ENV_IS_IN_MMC: Environment variables are stored in eMMC
- CONFIG_HW_WATCHDOG: Enable hardware watchdog timer function (Enable CONFIG_NUC970_WATCHDOG at the same time)
- CONFIG_SYS_BOOTM_LEN: Maximum kernel size
- CONFIG_DISPLAY_CPUINFO: Display CPU relative information
- CONFIG_BOOTDELAY: default boot delay time
- CONFIG_SYS_INIT_SP_ADDR: the stack pointer during system initialization
- CONFIG_BAUDRATE: UART baud rate
- CONFIG_NUC970_EMAC0: Use NUC970/N9H30 EMAC0
- CONFIG_NUC970_EMAC1: Use NUC970/N9H30 EMAC1
- CONFIG_NUC970_ETH: Support NUC970/N9H30 Ethernet
- CONFIG_NUC970_PHY_ADDR: PHY address
- CONFIG_CMD_NET: support network relative commands
- CONFIG_ETHADDR: MAC address
- CONFIG_SYS_RX_ETH_BUFFER: the number of Rx Frame Descriptors
- CONFIG_KPI_NUC970: Support NUC970 KPI
- CONFIG_SYS_ICACHE_OFF: Disable I-Cache
- CONFIG_SYS_DCACHE_OFF: Disable D-Cache

```
/*
 * BOOTP options
 */
#if 1
#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
*/
#endif


/*
 * Command line configuration.
 */
#if 0
#include <config_cmd_default.h>

#undef CONFIG_CMD_LOADS
#undef CONFIG_CMD_SOURCE
#endif
```

```
#define CONFIG_CMD_PING         1
#define CONFIG_CMD_DHCP         1
#define CONFIG_CMD_JFFS2        1


#ifdef CONFIG_SYS_USE_SPIFLASH
#undef CONFIG_CMD_IMLS  /*====================> SPI only */
#undef CONFIG_CMD_JFFS2
#endif
```

- CONFIG_BOOTP_SERVERIP: TFTP server IP not overruled by DHCP server.
- CONFIG_CMD_PING: Use ping command
- CONFIG_CMD_DHCP: Use DHCP command
- CONFIG_CMD_JFFS2: Support JFFS2 command

```
#ifdef CONFIG_SYS_USE_NANDFLASH
#define CONFIG_NAND_NUC970
#define CONFIG_CMD_NAND            1
#define CONFIG_CMD_UBI             1
#define CONFIG_CMD_UBIFS           1
#define CONFIG_CMD_MTDPARTS        1
#define CONFIG_MTD_DEVICE               1
#define CONFIG_MTD_PARTITIONS      1
#define CONFIG_RBTREE              1
#define CONFIG_LZO                 1
#define MTDIDS_DEFAULT “nand0=nand0”
#define MTDPARTS_DEFAULT “mtdparts=nand0:0x200000@0x0(u-
boot),0x1400000@0x200000(kernel),-(user)”
#define MTD_ACTIVE_PART “nand0,2”
#define CONFIG_CMD_NAND_YAFFS2 1
#define CONFIG_YAFFS2         1
#define CONFIG_SYS_MAX_NAND_DEVICE 1
#define CONFIG_SYS_NAND_BASE       0xB000D000
#ifdef CONFIG_ENV_IS_IN_NAND
#define CONFIG_ENV_OFFSET               0x80000
#define CONFIG_ENV_SIZE            0x10000
#define CONFIG_ENV_SECT_SIZE       0x20000
#define CONFIG_ENV_RANGE         (4 * CONFIG_ENV_SECT_SIZE)  /* Env range :
0x80000 ~ 0x100000 */
#define CONFIG_ENV_OVERWRITE
#endif
#endif#define CONFIG_SYS_NAND_U_BOOT_OFFS    (0x100000)    /* Offset to RAM
U-Boot image */
```

```
#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: Enable NUC970/N9H30 NAND function
- CONFIG_CMD_NAND: Use nand command
- CONFIG_MTD_DEVICE: Enable MTD device
- CONFIG_MTD_PARTITIONS: Enable MTD partition
- CONFIG_CMD_UBI: Enable UBI
- CONFIG_CMD_UBIFS: Enable UBIFS file system
- CONFIG_CMD_MTDPARTS: Use MTD partition command.
- CONFIG_RBTREE: Enable the configuration UBI need
- CONFIG_LZO: Enable the configuration UBI need
- MTDIDS_DEFAULT: Set MTD ID name, it needs to be the same as Linux kernel.
- MTDPARTS_DEFAULT: MTD partition configuration
- CONFIG_SYS_MAX_NAND_DEVICE: Maximum NAND device
- CONFIG_SYS_NAND_BASE: NAND controller base address
- CONFIG_ENV_OFFSET: flash offset address that environment variables are stored.
- CONFIG_ENV_SIZE: The space reserved for environment variables
- CONFIG_ENV_SECT_SIZE: The sector size of flash that environment variables are stored.
- CONFIG_ENV_RANGE: The range of environment variables, from CONFIG_ENV_OFFSET to CONFIG_ENV_OFFSET + CONFIG_ENV_RANGE. (When the block is a bad block, U-Boot will store environment variables to next block.)
- CONFIG_SYS_NAND_U_BOOT_OFFS: The NAND flash offset address that U-Boot is stored.
- CONFIG_SPL_TEXT_BASE: SPL U-Boot text base address
- CONFIG_SPL_STACK: SPL U-Boot stack address

- CONFIG_SYS_UBOOT_SIZE: U-Boot total space (code + data + heap)
- CONFIG_SYS_PHY_UBOOT_BASE: U-Boot execution address
- CONFIG_SYS_NAND_U_BOOT_SIZE: U-Boot image size
- CONFIG_SYS_NAND_PAGE_SIZE: NAND flash page size
- CONFIG_SYS_NAND_BLOCK_SIZE: NAND flash block size
- CONFIG_SYS_NAND_PAGE_COUNT: The page count per NAND flash block

```
/* SPI flash test code */
#ifdef CONFIG_SYS_USE_SPIFLASH
#define CONFIG_NUC970_SPI       1
#define CONFIG_CMD_SPI          1
#define CONFIG_CMD_SF           1
#define CONFIG_SPI              1
#define CONFIG_SPI_FLASH        1
/*#define CONFIG_SPI_FLASH_MACRONIX   1 */
#define CONFIG_SPI_FLASH_WINBOND   1
/*#define CONFIG_SPI_FLASH_EON 1 */
/*#define CONFIG_SPI_FLASH_SPANSION 1 */
/*#define CONFIG_SPI_FLASH_USE_4K_SECTORS*/
#define CONFIG_SPI_FLASH_BAR
#ifdef CONFIG_ENV_IS_IN_SPI_FLASH
#define CONFIG_ENV_OFFSET       0x80000
#define CONFIG_ENV_SIZE         0x10000
#define CONFIG_ENV_SECT_SIZE    0x10000
#define CONFIG_ENV_OVERWRITE
#endif
#endif
```

- CONFIG_CMD_SF: Use SPI flash sf command.
- CONFIG_SPI_FLASH_MACRONIX: Use MACRONIX SPI flash
- CONFIG_SPI_FLASH_WINBOND: Use Winbond SPI flash
- CONFIG_SPI_FLASH_EON: Use EON SPI flash
- CONFIG_SPI_FLASH_SPANSION: Use Spansion SPI flash
  According to your platform, enable one of them (MACRONIX/WINBOND/EON/SPANSION)
- CONFIG_ENV_OFFSET: The offset of flash that environment variables are stored
- CONFIG_ENV_SIZE: The space reserved for environment variables

```
#define CONFIG_SYS_PROMPT          "U-Boot> "
#define CONFIG_SYS_CBSIZE          256
#define CONFIG_SYS_MAXARGS         16
#define CONFIG_SYS_PBSIZE          (CONFIG_SYS_CBSIZE +
sizeof(CONFIG_SYS_PROMPT) + 16)
#define CONFIG_SYS_LONGHELP        1
#define CONFIG_CMDLINE_EDITING          1
#define CONFIG_AUTO_COMPLETE
#define CONFIG_SYS_HUSH_PARSER
```

```
#define CONFIG_SYS_PROMPT_HUSH_PS2 "> "
```

- CONFIG_SYS_PROMPT: Show prompt message
- CONFIG_SYS_LONGHELP: Display detailed help message.
- CONFIG_CMDLINE_EDITING: Permit command line editing.

```
/* Following block is for LCD support */
#define CONFIG_LCD
#define CONFIG_NUC970_LCD
#define LCD_BPP                         LCD_COLOR16
#define CONFIG_LCD_LOGO
#define CONFIG_LCD_INFO
#define CONFIG_LCD_INFO_BELOW_LOGO
#define CONFIG_SYS_CONSOLE_IS_IN_ENV
#define CONFIG_SYS_CONSOLE_OVERWRITE_ROUTINE
```

- CONFIG_LCD: Enable LCD
- CONFIG_NUC970_LCD: Compile NUC970/N9H30 driver
- LCD_BPP: The number of bits per pixel output to LCD.
- CONFIG_LCD_LOGO: Show the LOGO to LCD
- CONFIG_LCD_INFO: Show U-Boot version and NUC970/N9H30 relative information to LCD.
- CONFIG_LCD_INFO_BELOW_LOGO: Show NUC970/N9H30 relative information below the LOGO.
- CONFIG_SYS_CONSOLE_IS_IN_ENV: stdin/stdout/stderr use the setting of environment variables
- CONFIG_SYS_CONSOLE_OVERWRITE_ROUTINE: stdin/stdout/stderr switch to serial port

```
/* Following block is for MMC support */
#define CONFIG_NUC970_MMC
#define CONFIG_CMD_MMC
#define CONFIG_CMD_FAT
#define CONFIG_MMC
#define CONFIG_GENERIC_MMC
#define CONFIG_DOS_PARTITION
#define CONFIG_NUC970_SD_PORT0
//#define CONFIG_NUC970_SD_PORT1
//#define CONFIG_NUC970_EMMC    /* Don't enable eMMC(CONFIG_NUC970_EMMC)
and NAND(CONFIG_NAND_NUC970) at the same time! */
#ifdef CONFIG_ENV_IS_IN_MMC
#define CONFIG_SYS_MMC_ENV_DEV  2
#define CONFIG_ENV_OFFSET       0x80000
#define CONFIG_ENV_SIZE         0x10000
#define CONFIG_ENV_SECT_SIZE    512
#define CONFIG_ENV_OVERWRITE
```

```
#endif
```

- CONFIG_NUC970_MMC: Compile NUC970/N9H30 driver
- CONFIG_CMD_MMC: Support MMC command
- CONFIG_CMD_FAT: Support FAT command
- CONFIG_MMC: Support MMC
- CONFIG_GENERIC_MMC: Support generic MMC
- CONFIG_DOS_PARTITION: Support DOS partition
- CONFIG_NUC970_SD_PORT0: Support SD port 0
- CONFIG_NUC970_SD_PORT1: Support SD port 1
- CONFIG_NUC970_EMMC: Support eMMC
- CONFIG_SYS_MMC_ENV_DEV: The MMC device number that environment variables stored
- CONFIG_ENV_OFFSET: Environment variables offset
- CONFIG_ENV_SIZE: Environment variables size
- CONFIG_ENV_SECT_SIZE: Environment variables sector size

```
/* Following block is for EHCI support*/
#if 1
#define CONFIG_CMD_USB
#define CONFIG_CMD_FAT
#define CONFIG_USB_STORAGE
#define CONFIG_USB_EHCI
#define CONFIG_USB_EHCI_NUC970
#define CONFIG_EHCI_HCD_INIT_AFTER_RESET
#define CONFIG_DOS_PARTITION
#endif
```

- CONFIG_CMD_USB: Support USB command
- CONFIG_CMD_FAT: Support FAT command
- CONFIG_USB_STORAGE: Support USB storage
- CONFIG_USB_EHCI: Support USB 2.0
- CONFIG_USB_EHCI_NUC970: Support NUC970/N9H30 USB 2.0
- CONFIG_DOS_PARTITION: Support DOS partition

```
#define CONFIG_NUC970_GPIO

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

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

#endif
```

- CONFIG_NUC970_GPIO: Enable GPIO function
- CONFIG_SYS_MALLOC_LEN: The space reserved for malloc
- CONFIG_STACKSIZE: Stack size.

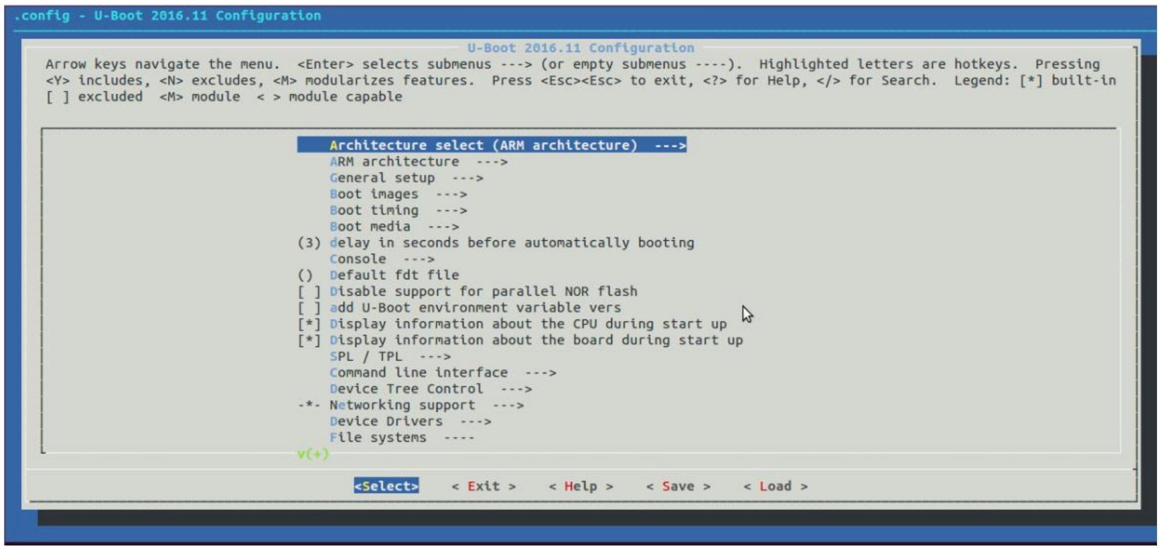
```
#define CONFIG_KPI_NUC970

#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: Enable KPI function
- CONFIG_KPI_PA_PORT: Select KPI key from port A
- CONFIG_KPI_PH_PORT: Select KPI key from port H
  (user only can select one of CONFIG_KPI_PA_PORT or CONFIG_KPI_PH_PORT)
- CONFIG_KPI_ROW_NUM: Set KPI matrix row number
- CONFIG_KPI_COL_NUM: Set KPI matrix column number
- CONFIG_KPI_DEBOUNCE: Set de-bounce sampling cycle

1.1.2 Menu configuration

U-Boot v2016.11 supports menu configuration by command “make menuconfig”.

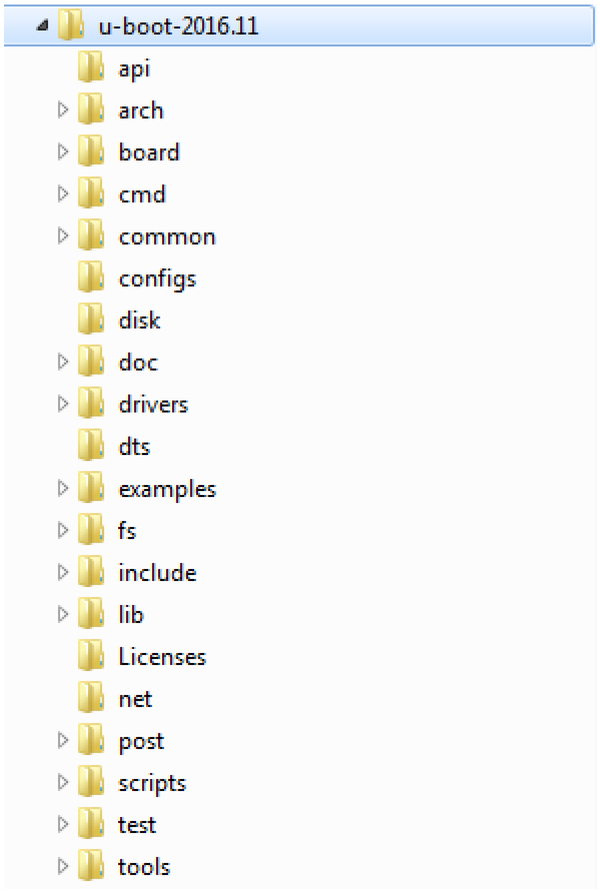
```
# make menuconfig
```



```
.config - U-Boot 2016.11 Configuration
                              U-Boot 2016.11 Configuration
  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).  Highlighted letters are hotkeys.  Pressing
  <Y> includes, <N> excludes, <M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </> for Search.  Legend: [*] built-in
  [ ] excluded  <M> module  < > module capable

                    Architecture select (ARM architecture)  --->
                    ARM architecture  --->
                    General setup  --->
                    Boot images  --->
                    Boot timing  --->
                    Boot media  --->
                (3) delay in seconds before automatically booting
                    Console  --->
                ()  Default fdt file
                [ ] Disable support for parallel NOR flash
                [ ] add U-Boot environment variable vers
                [*] Display information about the CPU during start up
                [*] Display information about the board during start up
                    SPL / TPL  --->
                    Command line interface  --->
                    Device Tree Control  --->
                -*- Networking support  --->
                    Device Drivers  --->
                    File systems  ----
                v(+)

                  <Select>    < Exit >    < Help >    < Save >    < Load >
```

1.2 Directory structure

The directory structure of U-Boot source code is as below.



- api: U-Boot machine/arch independent API for external apps
- arch: This directory contains CPU relative source code.
- The CPU relative source code of NUC970/N9H30 is under arch/arm/cpu/arm926ejs/nuc970.
- board: This directory contains board relative source code.
  The board relative source code of NUC970/N9H30 is under board/nuvoton/nuc970_evb.
- cmd: This directory contains miscellaneous U-Boot command
- common: This directory contains U-Boot command and other common source code.
- configs: This directory contains default configuration of vendors.
- disk: This directory contains disk partition related source code.
- doc: This directory contains miscellaneous README document.
- drivers: This directory contains miscellaneous driver source code.
  The driver relative source code of NUC970/N9H30 is under directory drivers. For instance the Ethernet driver is under drivers/net/nuc970_eth.c
- dts: This directory contains two files, Makefile and Kconfig for device tree.
  If device tree configuration, CONFIG_OF_CONTROL, is enabled, the Makefile builds the internal U-Boot fdt. See doc/README.fdt-control for more details.
- examples: This directory contains some examples. For instance, mips.lds is the linker script file for MIPS.
- fs: This directory contains miscellaneous file systems. For instance, FAT, yaffs2.
- include: This directory contains header file and configuration file. NUC970/N9H30 configuration file is under include/configs/nuc970_evb.h
- lib: This directory contains miscellaneous library.
- Licenses: This directory contains GPL relative document.
- net: This directory contains network relative source code. For instance, tftp.c, ping.c,
- post: Supply a default implementation for post_hotkeys_pressed() for boards without hotkey support.

- scripts: contains sources for various helper programs used throughout
  the kernel for the build process.
- test: This directory contains some test programs. See test/README for more details.
- tools: This directory contains some tools. For instance, mkimage is the tool to make a image.

1.3 Compile U-Boot

1.3.1 Compile command

Clean all object code.

```
# make distclean
```

Generate default configuration (Choose one of NUC970 or N9H30)

```
# make nuc970_defconfig (for NUC970)
# make n9h30_defconfig (for N9H30)
```

Start compilation.

```
# make
```

1.3.2 Output file after compilation

If you compile successfully, you can get Main U-Boot and SPL U-Boot:
Main U-Boot : Full function U-Boot
SPL U-Boot : Move Main U-Boot from NAND flash to DDR and boot Main U-Boot
SPL U-Boot : It's only for NAND boot；SPI boot and eMMC boot need Main U-Boot only.

Main U-Boot and SPL U-Boot are generated in root directory and sub-directory spl:
Main U-Boot files are generated in root directory.

- u-boot - Elf executable file (for download with GDB or IDE)
- u-boot.bin - binary file (You can use Nu-Writer to burn it to NAND/SPI flash、eMMC)
- u-boot.map –Linker memory map file

SPL U-Boot files are generated in sub-directory spl

- u-boot-spl - Elf executable file (for download with GDB or IDE)
- u-boot-spl.bin - binary file (You can use Nu-Writer to burn it to NAND/SPI flash、eMMC)
- u-boot-spl.map –Linker memory map file

1.3.3 Main U-Boot link address

Main U-Boot link address is defined in include/configs/nuc970_evb.h.

```
#define CONFIG_SYS_TEXT_BASE            0xE00000
```

The default setting of U-Boot link address is 0xE00000

If boot mode is NAND Boot, please also modify the definition in include/configs/nuc970_evb.h

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

1.3.4 SPL U-Boot link address

SPL U-Boot link address is defined in include/configs/nuc970_evb.h
Default address is 0x200, if you want to modify it to other address, please find the following code segment, and replace 0x200 with new address.

```
#define CONFIG_SPL_TEXT_BASE    0x200
```

1.4 Add SPI configuration

In default configuration, SPI is disabled. Update include/configs/nuc970_evb.h and use "make menuconfig " to update configurations for supporting SPI.

1.4.1 Enable SPI configuration in nuc970_evb.h

Edit include/configs/nuc970_evb.h. Enable definition "CONFIG_SYS_USE_SPIFLASH".
Set environment variables in SPI flash by enabling definition "CONFIG_ENV_IS_IN_SPI_FLASH" and disabling definition "CONFIG_ENV_IS_IN_NAND".

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

1.4.2 Enable NUC970/N9H30 SPI driver

By "make menuconfig", enable NUC970/N9H30 SPI driver and select SPI in Quad mode or Normal mode.

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

1.4.3 Enable sf/spi command

By "make menuconfig", enable sf/spi command.

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

1.4.4 Enable Legacy SPI Flash Interface support

By "make menuconfig", enable SPI Flash interface support and SPI flash Bank/Extend address support. Besides, select SPI flash per your DEV board. Below is an example that Winbond SPI flash support.

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

```
      [*]   Winbond SPI flash support
```

1.5 NAND AES secure boot example (NUC970 only)

NAND AES secure boot need Main U-Boot and SPL U-Boot, and use Nu-Writer to encrypt SPL U-Boot by AES and burn to NAND flash.

1.5.1 Compile Main U-Boot and SPL U-Boot

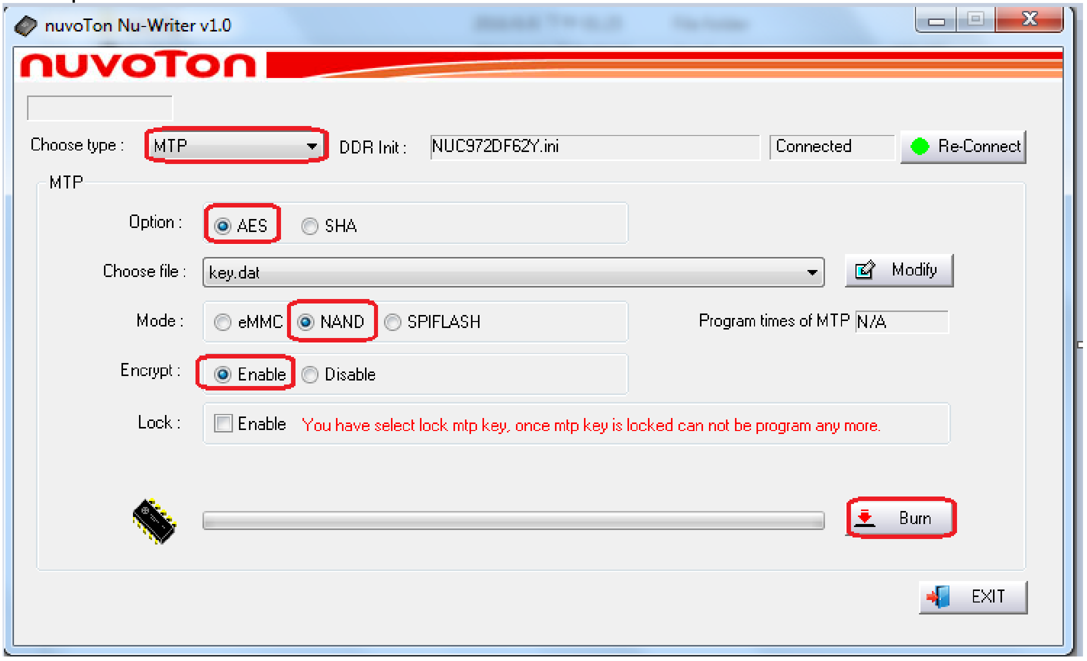
```
# make distclean
# make nuc970_defconfig (for NUC970)
# make
```

After compilation success, find out the binary file of Main U-Boot and SPL U-Boot.

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

1.5.2 Burn key to MTP

Choose type : select "MTP".
Option: AES
Mode: NAND
Encrypt: Enable
Then press "Burn" button.


nuvoTon Nu-Writer v1.0

Choose type : MTP
DDR Init : NUC972DF62Y.ini
Connected
Re-Connect

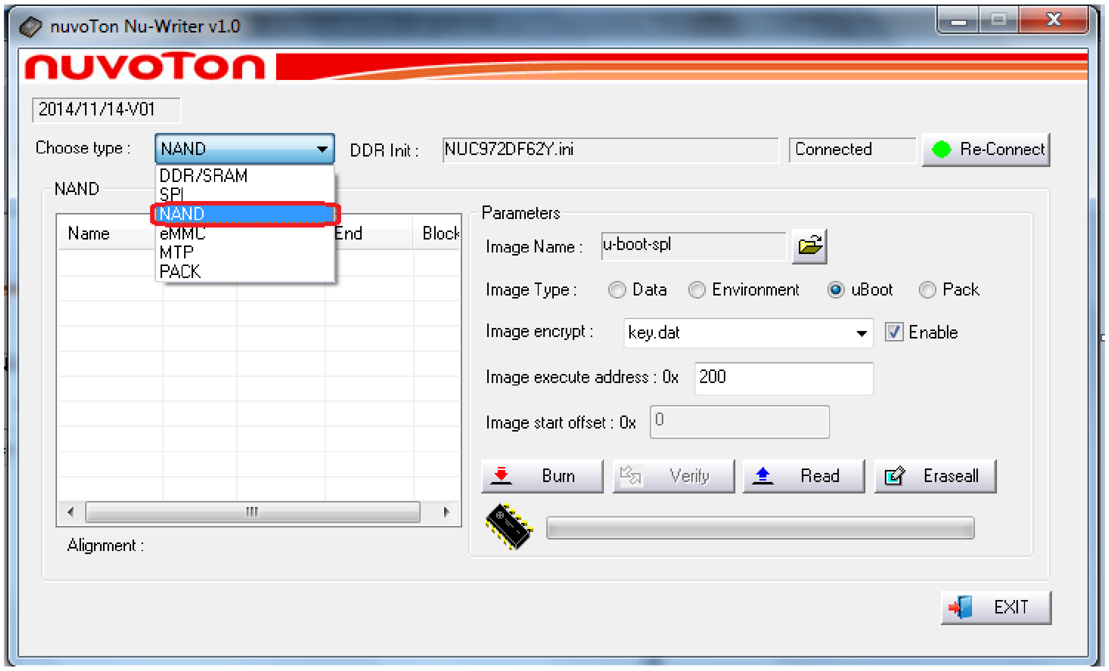
MTP

Option : [✓ AES] SHA
Choose file : key.dat
Modify
Mode : eMMC [✓ NAND] SPIFLASH
Program times of MTP N/A
Encrypt : [✓ Enable] Disable
Lock : ☐ Enable You have select lock mtp key, once mtp key is locked can not be program any more.

Burn
EXIT

1.5.3 Burn SPL U-Boot

Choose type : select "NAND".


nuvoTon Nu-Writer v1.0

2014/11/14-V01

Choose type : NAND

DDR Init : NUC972DF62Y.ini

Connected

NAND

Parameters

Image Name : u-boot-spl

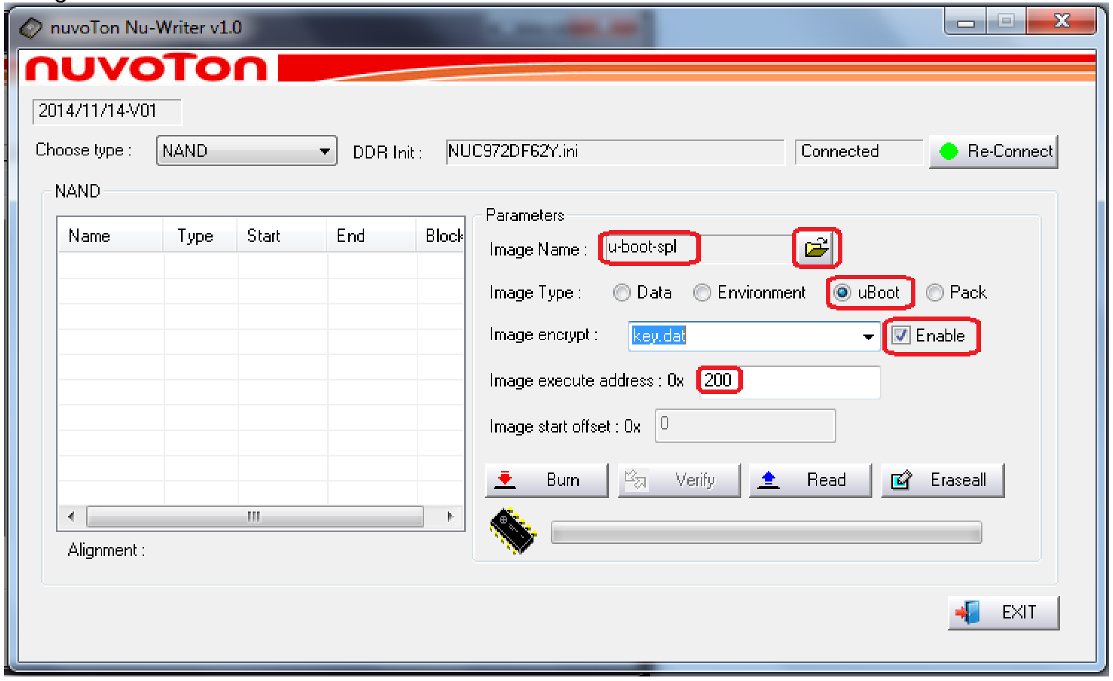
Image Type : ( ) Data ( ) Environment (●) uBoot ( ) Pack

Image encrypt : key.dat ☑ Enable

Image execute address : 0x 200

Image start offset : 0x 0

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


nuvoTon Nu-Writer v1.0

2014/11/14-V01

Choose type : NAND

DDR Init : NUC972DF62Y.ini

Connected

NAND

Parameters

Image Name : u-boot-spl

Image Type : ( ) Data ( ) Environment (●) uBoot ( ) Pack

Image encrypt : key.dat ☑ Enable

Image execute address : 0x 200

Image start offset : 0x 0

Then press “Burn” button.

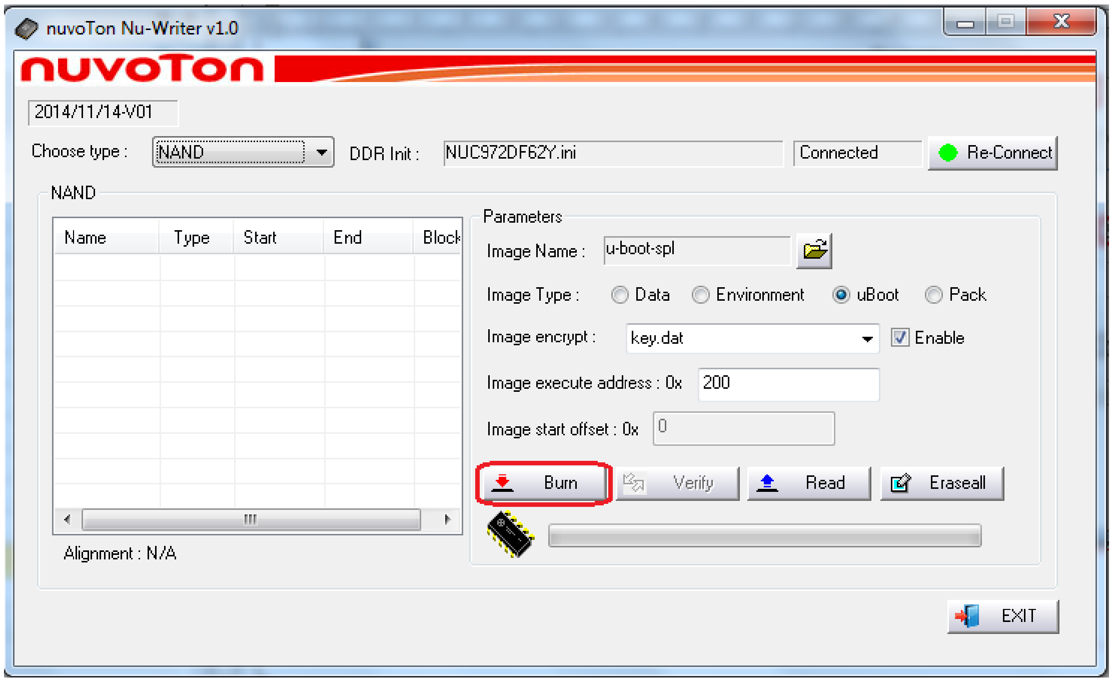


Image Name : u-boot-spl

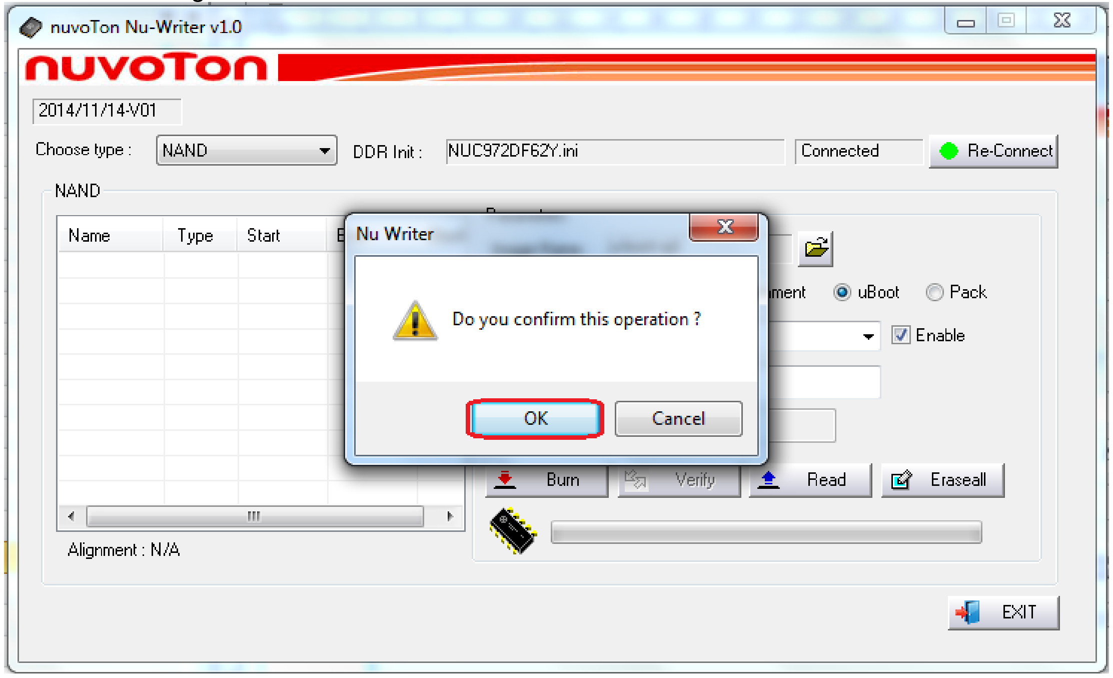
Image Type : ( ) Data ( ) Environment (●) uBoot ( ) Pack

Image encrypt : key.dat [✓] Enable

Image execute address : 0x 200

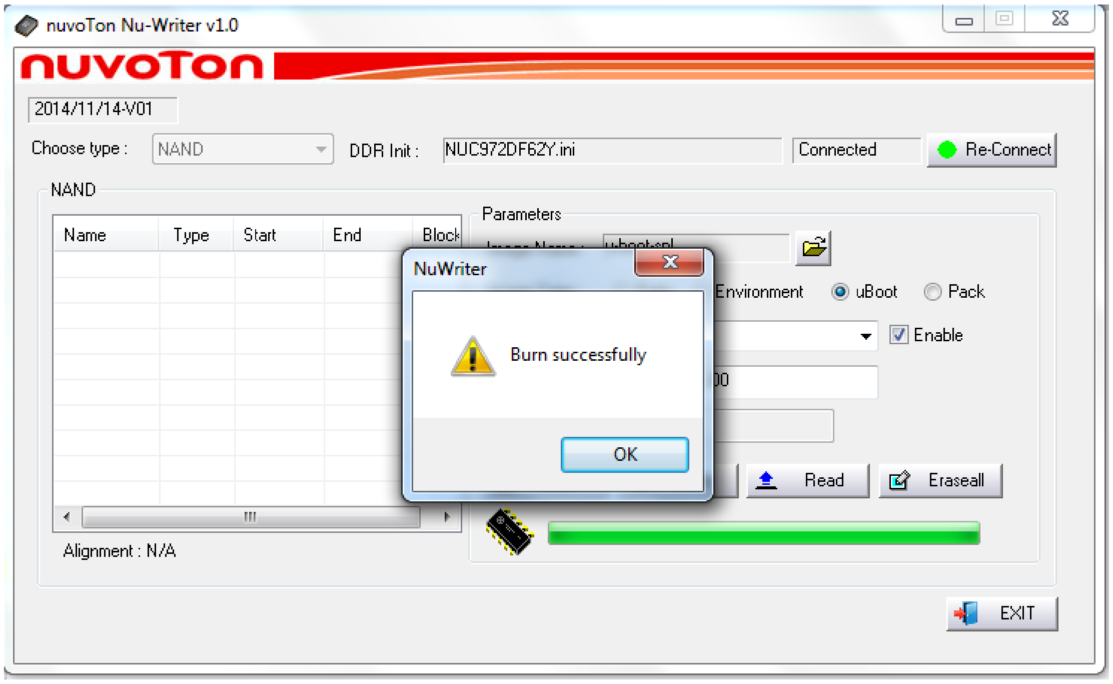
Image start offset : 0x 0

There is a dialog box, choose “OK”.



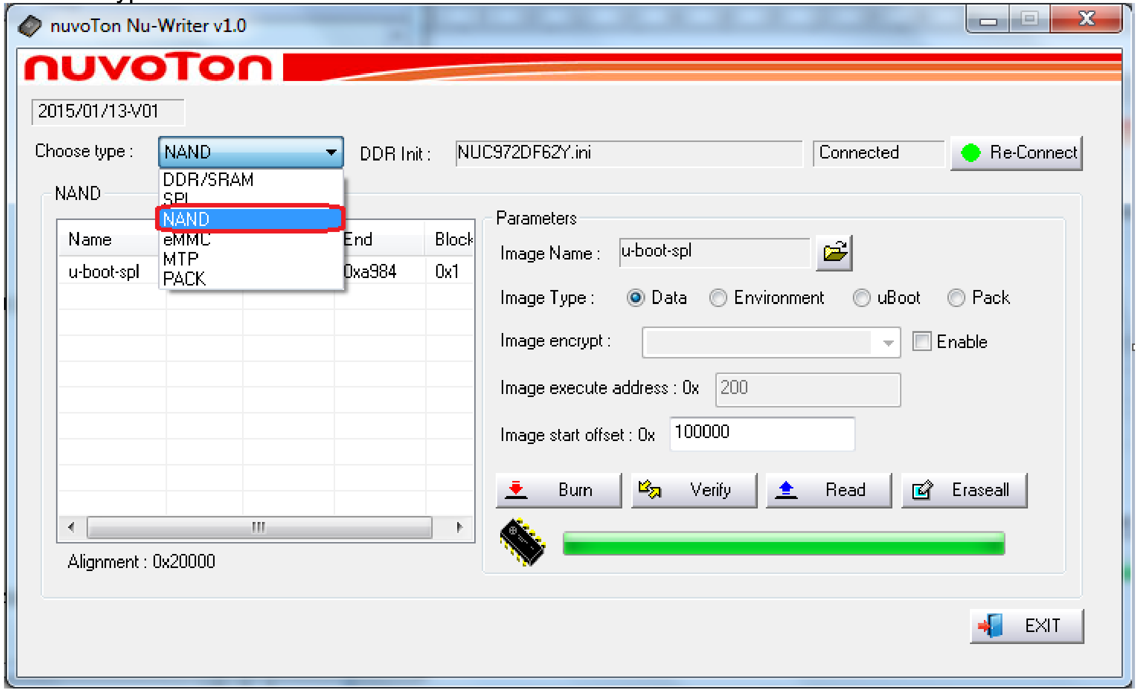
Do you confirm this operation ?

A dialog box shows burn successfully message.

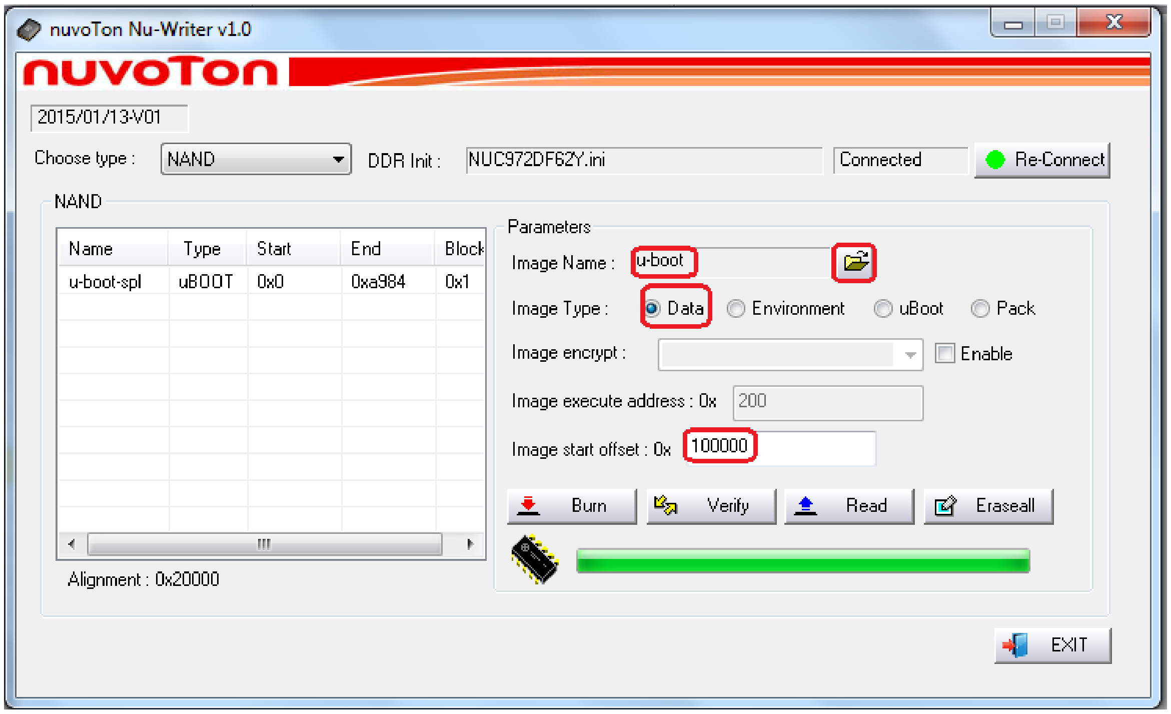


1.5.4 Burn Main U-Boot

Choose type: select "NAND".

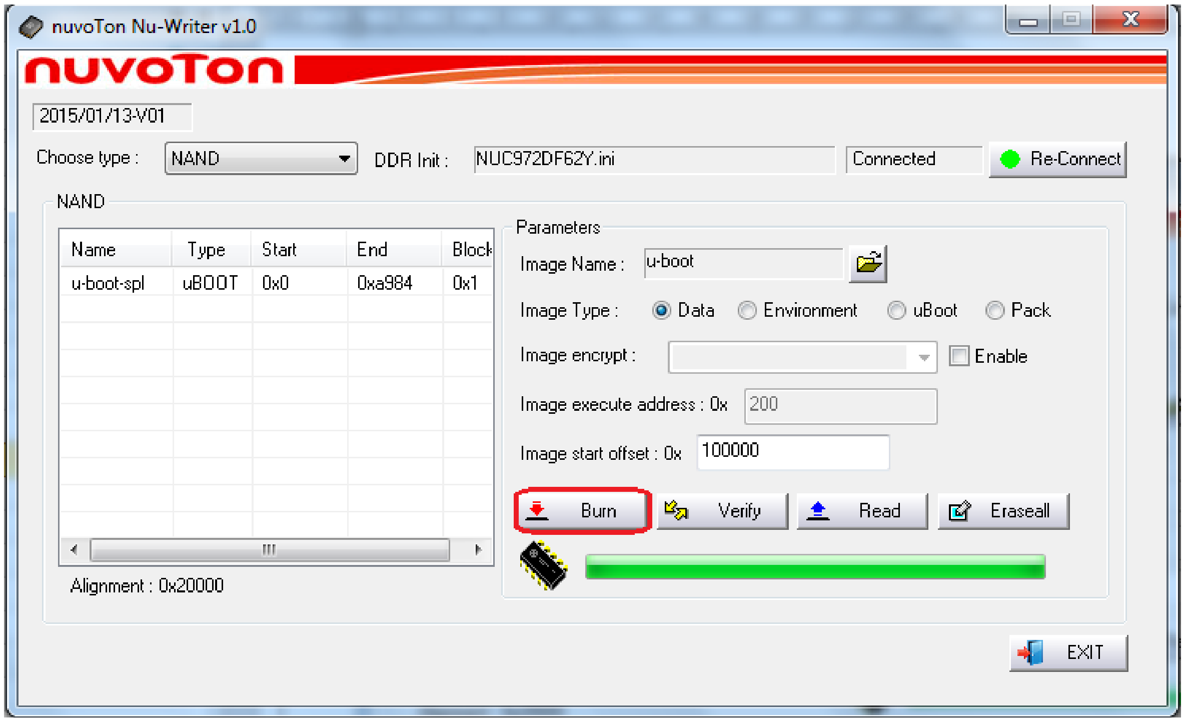


Set burning parameters.
Image Name: select u-boot.bin,
Image Type: select Data,
Image execute address:0x Fill in 100000



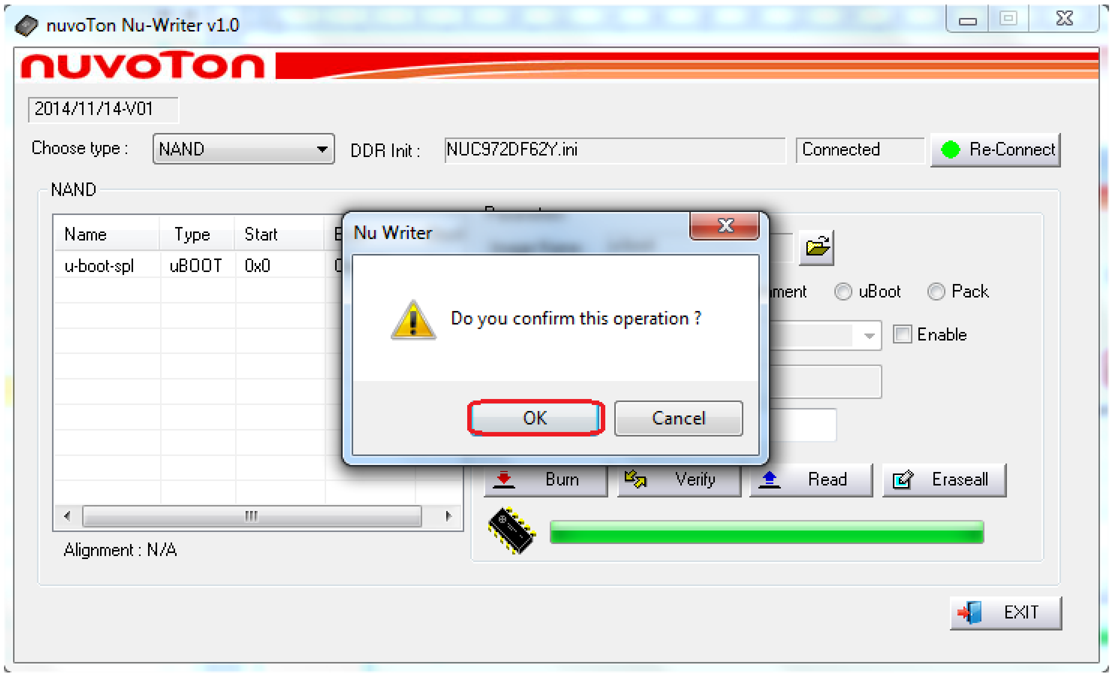
Name	Type	Start	End	Block
u-boot-spl	uBOOT	0x0	0xa984	0x1

Press "Burn" button.



Name	Type	Start	End	Block
u-boot-spl	uBOOT	0x0	0xa984	0x1

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



Name	Type	Start
u-boot-spl	uBOOT	0x0

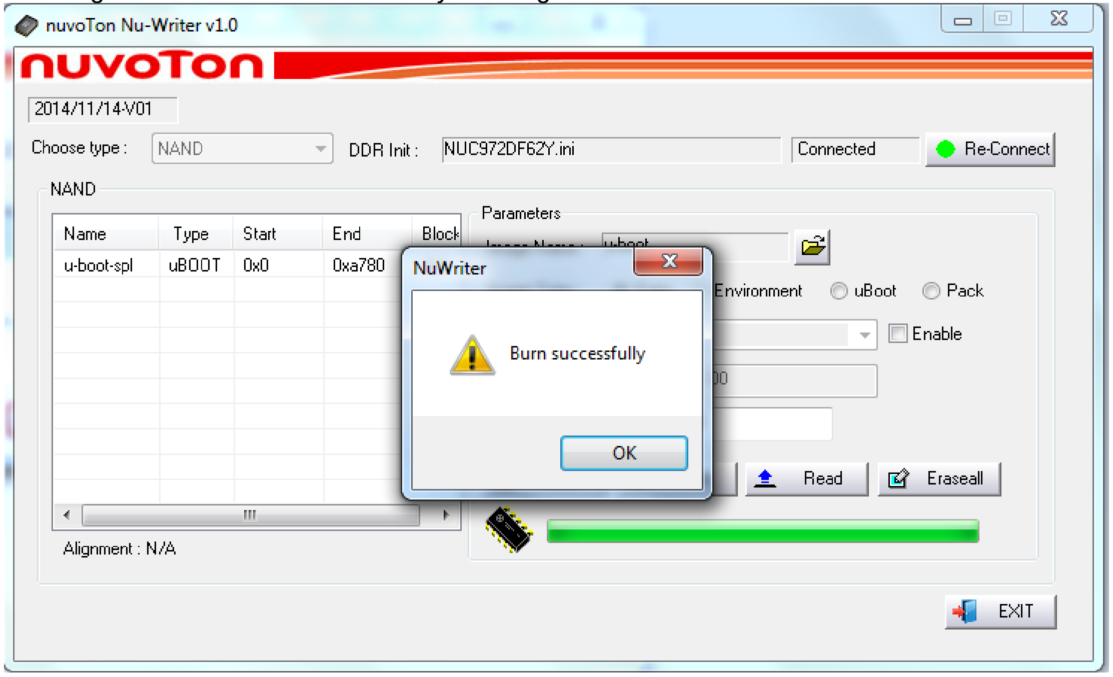
Alignment : N/A

Nu Writer

Do you confirm this operation ?

OK Cancel

A dialog box shows burn successfully message.



Name	Type	Start	End	Block
u-boot-spl	uBOOT	0x0	0xa780	

Alignment : N/A

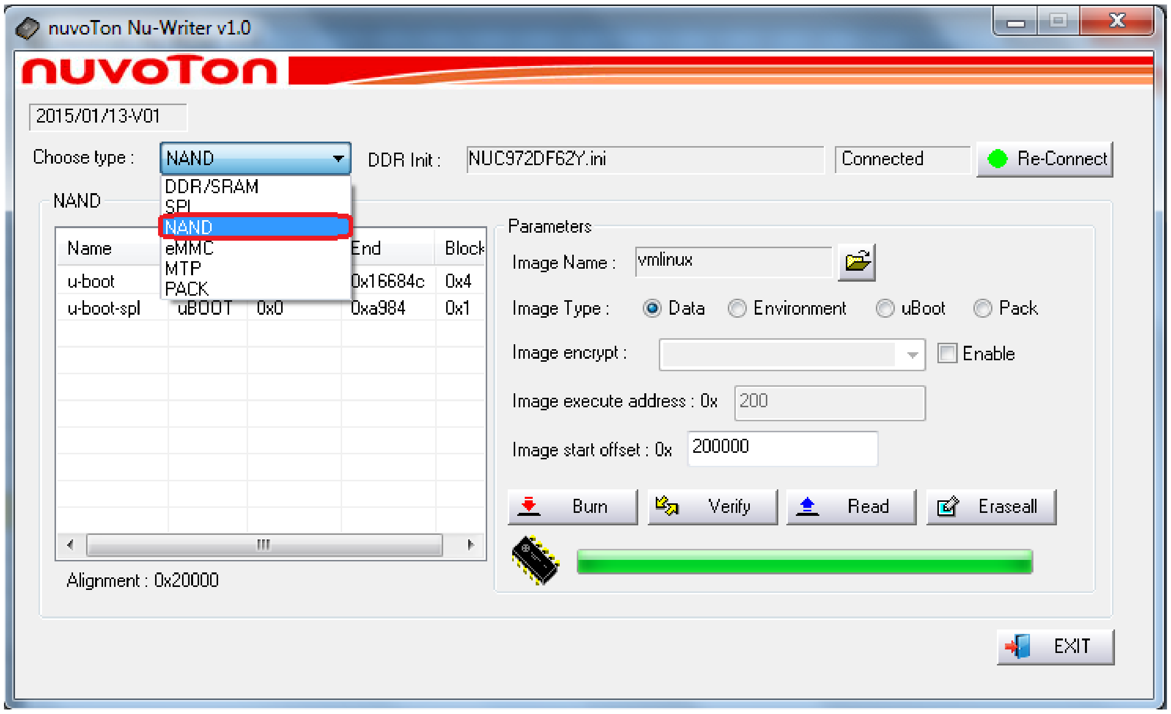
NuWriter

Burn successfully

OK

1.5.5 Burn Linux kernel

Choose type : select “NAND”.



nuvoTon Nu-Writer v1.0

2015/01/13-V01

Choose type : NAND

DDR Init : NUC972DF62Y.ini

Connected

Re-Connect

NAND

Name	Type	Start	End	Block
u-boot			0x16684c	0x4
u-boot-spl	uBOOT	0x0	0xa984	0x1

Alignment : 0x20000

Parameters

Image Name : vmlinux

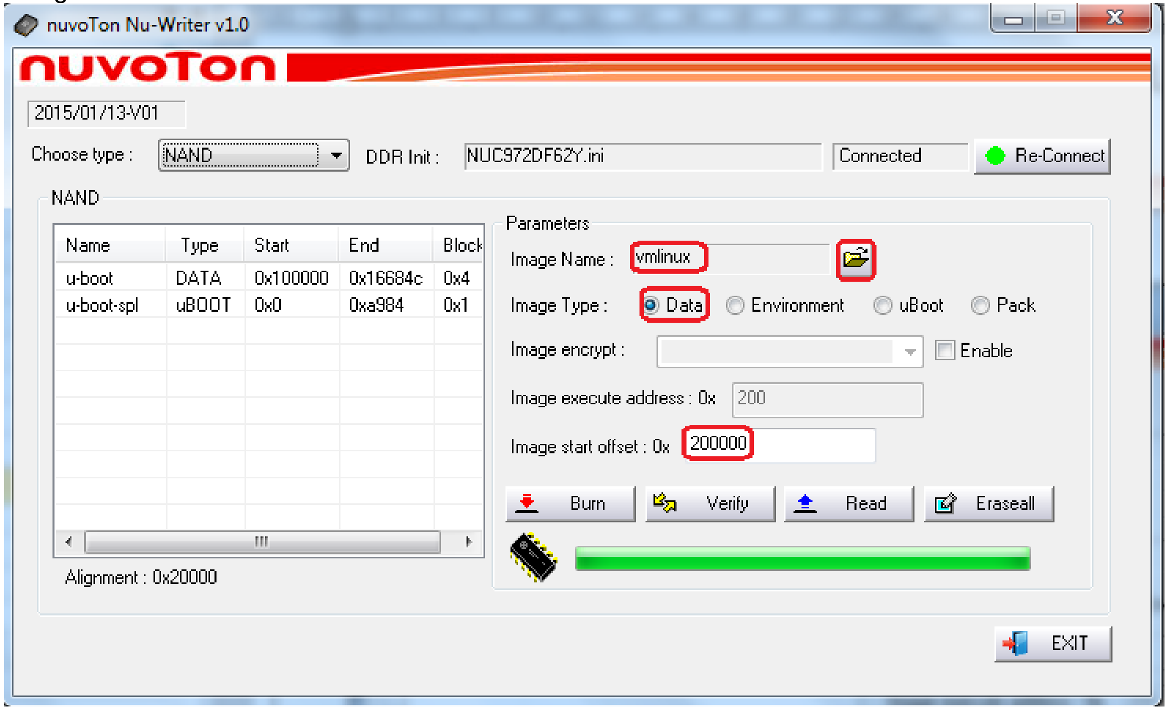
Image Type : [✓ Data] Environment uBoot Pack

Image encrypt : Enable

Image execute address : 0x 200

Image start offset : 0x 200000

Set burning parameters.
Image Name: select vmlinux.ub (Please reference 1.8 to know how to generate vmlinux.ub).
Image Type: select Data,
Image execute address:0x Fill in 200000



nuvoTon Nu-Writer v1.0

2015/01/13-V01

Choose type : NAND

DDR Init : NUC972DF62Y.ini

Connected

Re-Connect

NAND

Name	Type	Start	End	Block
u-boot	DATA	0x100000	0x16684c	0x4
u-boot-spl	uBOOT	0x0	0xa984	0x1

Alignment : 0x20000

Parameters

Image Name : vmlinux

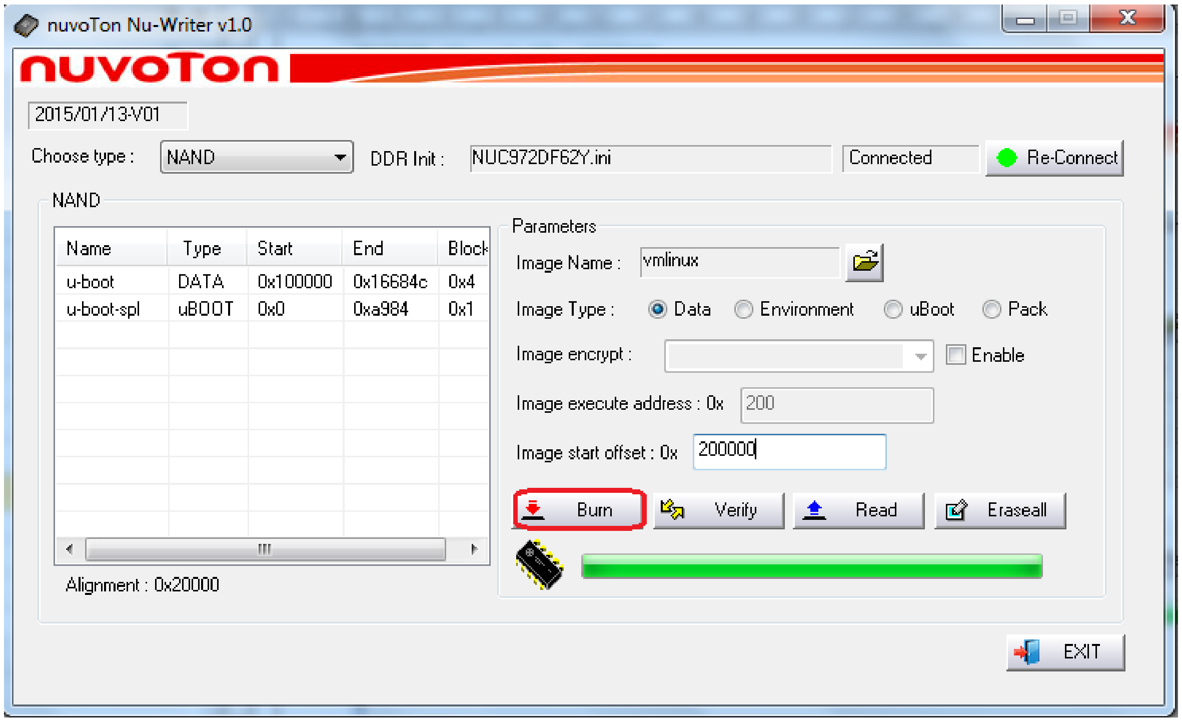
Image Type : [✓ Data] Environment uBoot Pack

Image encrypt : Enable

Image execute address : 0x 200

Image start offset : 0x 200000

Press “Burn” button.



Name	Type	Start	End	Block
u-boot	DATA	0x100000	0x16684c	0x4
u-boot-spl	uBOOT	0x0	0xa984	0x1

Alignment : 0x20000

Image Name : vmlinux

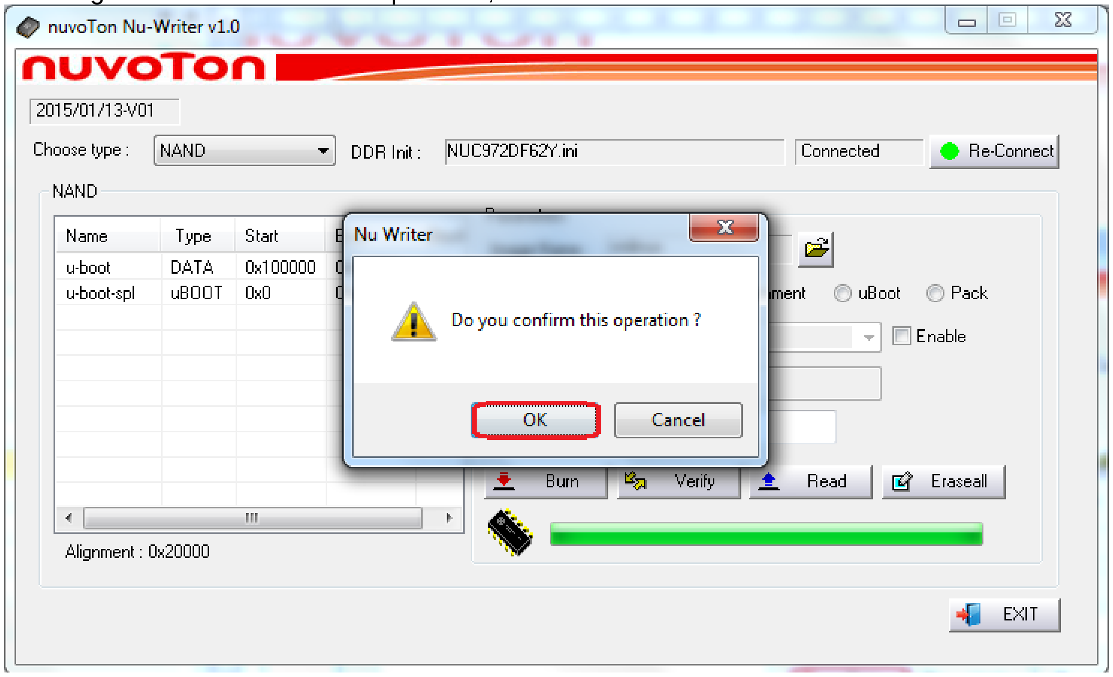
Image Type : [✓ Data] Environment uBoot Pack

Image encrypt : Enable

Image execute address : 0x 200

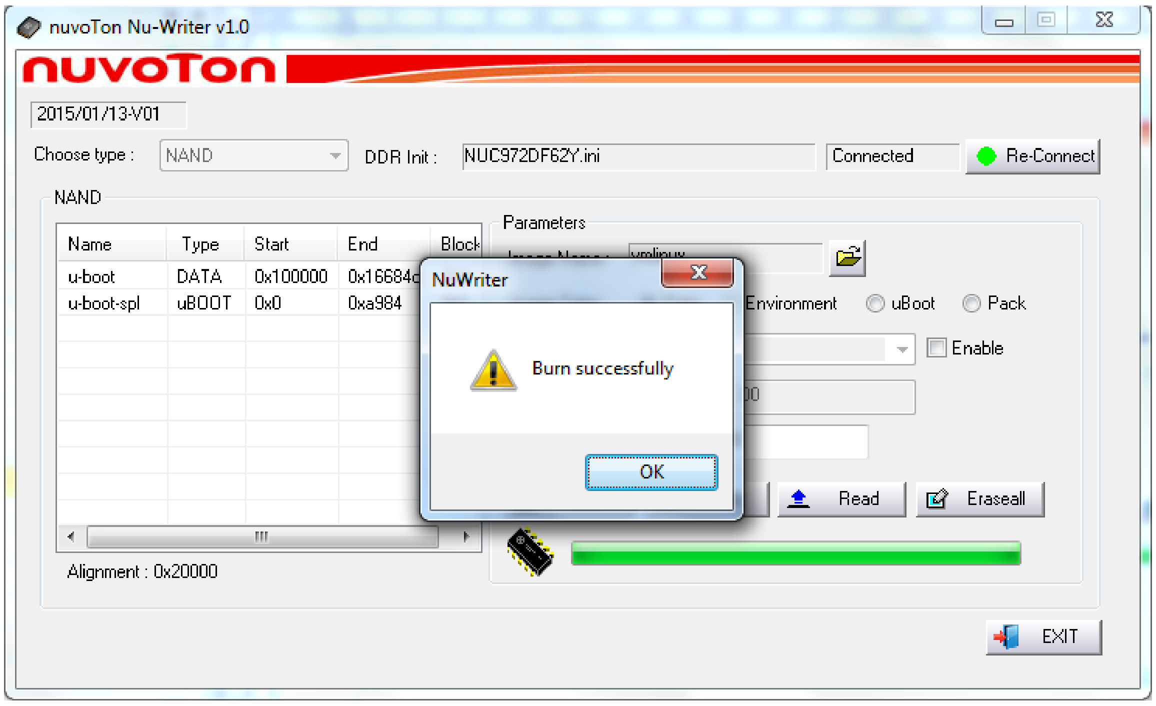
Image start offset : 0x 200000

A dialog box ask if confirm this operation, choose “OK”



Do you confirm this operation ?

A dialog box shows burn successfully message.



Name	Type	Start	End	Block
u-boot	DATA	0x100000	0x16684c	
u-boot-spl	uBOOT	0x0	0xa984	

Alignment : 0x20000

Burn successfully

1.5.6 nboot command to boot Linux kernel in NAND flash

Following example demonstrate use nboot command to read Linux kernel image stored in NAND flash offset 0x200000 to DDR address 0x7fc0. Then use bootm command to boot Linux kernel.

```
U-Boot> nboot 0x7fc0 0 0x200000

Loading from nand0, offset 0x200000
   Image Name:
   Image Type:   ARM Linux Kernel Image (uncompressed)
   Data Size:    1639744 Bytes = 1.6 MiB
   Load Address: 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 ...
```

1.6 U-Boot Command

U-boot provides a powerful command line interface which may be accessed through a terminal emulator connected to the target board's serial port. For example type "help" at the command prompt will print a list of all the available commands:

```
U-Boot> help
0       - do nothing, unsuccessfully
1       - do nothing, successfully
?       - alias for 'help'
base    - print or set address offset
bdinfo  - print Board Info structure
boot    - boot default, i.e., run 'bootcmd'
bootd   - boot default, i.e., run 'bootcmd'
…
```

For most commands, you do not need to type in the full command name; instead it is sufficient to type a few characters. For instance, help can be abbreviated as h. Almost all U-Boot commands expect numbers to be entered in hexadecimal input format. (Exception: for historical reasons, the sleep command takes its argument in decimal input format.)

1.6.1 Bootm command

Since Linux kernel image is stored in network、NAND、SPI、USB、MMC, we can download Linux kernel to DDR by those storage relative command, then boot Linux kernel by bootm command.
Hence, bootm command is used to boot Linux kernel or other application program.
bootm command format is as below:

```
U-Boot> help bootm
bootm - boot application image from memory

Usage:
bootm [addr [arg ...]]
    - boot application image stored in memory
        passing arguments 'arg ...'; when booting a Linux kernel,
        'arg' can be the address of an initrd image
```

Suppose we have downloaded Linux kernel to DDR address 0x7fc0, then we can boot Linux kernel by bootm command.

```
U-Boot> bootm 0x7fc0
## Booting kernel from Legacy Image at 00007fc0 ...
   Image Name:
   Image Type:   ARM Linux Kernel Image (uncompressed)
   Data Size:    1639744 Bytes = 1.6 MiB
   Load Address: 00008000
   Entry Point:  00008000
   Verifying Checksum ... OK
   XIP Kernel Image ... OK
OK

Starting kernel ...
```

1.6.2 Go command

- go: start application

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

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

Below example is to start an application program that has been downloaded to DDR 0x100000

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



 Hello World!
```

1.6.3 Network relative command

- ping

Transmit ICMP ECHO_REQUEST packet to network host

```
U-Boot> help ping
ping - send ICMP ECHO_REQUEST to network host

Usage:
ping pingAddress
U-Boot>
```

Before using this command, you have to set variables ipaddr that is the IP address of your platform.
Below is an example that set IP address of NUC970/N9H30 to 192.168.0.101 and ping a remote PC whose IP address is 192.168.0.100

```
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

Download image via network using TFTP protocol.

```
U-Boot> help tftp
tftpboot - boot image via network using TFTP protocol

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

Before using this command, you have to set variables ipaddr and serverip.
Below is an example to download a Linux kernel image by TFTP protocol. First, set IP address of NUC970/N9H30 and TFTP server to 192.168.0.101 and 192.168.0.100 respectively. Second, download Linux kernel image to address 0x200000 by TFTP protocol. Third, boot Linux kernel by command bootm.

```
U-Boot> 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

Download image via network using DHCP/TFTP protocol

```
U-Boot> help dhcp
dhcp - boot image via network using DHCP/TFTP protocol

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

Below is an example to download Linux kernel image to address 0x7fc0 by DHCP/TFTP protocol. You don't have to set ipaddr for your platform, since DHCP server will assign an IP for you.

```
U-Boot> dhcp 0x7fc0 vmlinux.ub
BOOTP broadcast 1
*** Unhandled DHCP Option in OFFER/ACK: 7
*** Unhandled DHCP Option in OFFER/ACK: 7
DHCP client bound to address 192.168.0.102
Using emac device
TFTP from server 192.168.0.100; our IP address is 192.168.0.102; sending
through gateway 192.168.0.100
Filename 'vmlinux.ub'.
Load address: 0x7fc0
Loading: *#################################################################
      ###############################################
      1 MiB/s
done
Bytes transferred = 1639808 (190580 hex)
U-Boot> bootm 0x7fc0
## Booting kernel from Legacy Image at 00007fc0 ...
   Image Name:
   Image Type:   ARM Linux Kernel Image (uncompressed)
   Data Size:    1639744 Bytes = 1.6 MiB
   Load Address: 00008000
   Entry Point:  00008000
   Verifying Checksum ... OK
   XIP Kernel Image ... OK
OK

Starting kernel ...
```

- bootp

Download image via network using BOOTP/TFTP protocol

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

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

Below is an example to download Linux kernel image to address 0x7fc0 by BOOTP/TFTP protocol. You don't have to set ipaddr for your platform, since DHCP server will assign an IP for you.

```
U-Boot> bootp 0x7fc0 vmlinux.ub
BOOTP broadcast 1
*** Unhandled DHCP Option in OFFER/ACK: 7
*** Unhandled DHCP Option in OFFER/ACK: 7
DHCP client bound to address 192.168.0.102
Using emac device
TFTP from server 192.168.0.100; our IP address is 192.168.0.102; sending
through gateway 192.168.0.100
Filename 'vmlinux.ub'.
Load address: 0x7fc0
Loading: *#################################################################
      ###############################################
      1 MiB/s
done
Bytes transferred = 1639808 (190580 hex)
U-Boot> bootm 0x7fc0
## Booting kernel from Legacy Image at 00007fc0 ...
   Image Name:
   Image Type:   ARM Linux Kernel Image (uncompressed)
   Data Size:    1639744 Bytes = 1.6 MiB
   Load Address: 00008000
   Entry Point:  00008000
   Verifying Checksum ... OK
   XIP Kernel Image ... OK
OK
```

```
Starting kernel ...
```

1.6.4 Nand flash commands

- nand: NAND Sub-system

U-Boot supports NAND flash relative commands, including nand info/device/erase/read/write. Command format is as below:

```
U-Boot> help nand
nand - NAND sub-system

Usage:
nand info - show available NAND devices
nand device [dev] - show or set current device
nand read - addr off|partition size
nand write - addr off|partition size
    read/write 'size' bytes starting at offset 'off'
    to/from memory address 'addr', skipping bad blocks.
nand read.raw - addr off|partition [count]
nand write.raw - addr off|partition [count]
    Use read.raw/write.raw to avoid ECC and access the flash as-is.
nand erase[.spread] [clean] off size - erase 'size' bytes from offset 'off'
    With '.spread', erase enough for given file size, otherwise,
    'size' includes skipped bad blocks.
nand erase.part [clean] partition - erase entire mtd partition'
nand erase.chip [clean] - erase entire chip'
nand bad - show bad blocks
nand dump[.oob] off - dump page
nand scrub [-y] off size | scrub.part partition | scrub.chip
    really clean NAND erasing bad blocks (UNSAFE)
nand markbad off [...] - mark bad block(s) at offset (UNSAFE)
nand biterr off - make a bit error at offset (UNSAFE)
U-Boot>
```

Below example show NAND Page size/OOB size/Erase size information by nand info/device command.

```
U-Boot> nand info

Device 0: nand0, sector size 128 KiB
  Page size      2048 b
  OOB size         64 b
  Erase size   131072 b
U-Boot> nand device

Device 0: nand0, sector size 128 KiB
  Page size      2048 b
  OOB size         64 b
  Erase size   131072 b
U-Boot>
```

nand erase.chip erase whole chip.

```
U-Boot> nand erase.chip

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

Below is an example to write a Linux kernel image to NAND flash. The Linux kernel image is allocated at DDR 0x500000 and its size is 0x190580 bytes. We will write it to NAND flash offset 0x200000. Then, read the Linux kernel image back to DDR 0x7fc0. At last, use command, bootm, to boot Linux kernel image.

```
U-Boot> nand write 0x500000 0x200000 0x190580

NAND write: device 0 offset 0x200000, size 0x190580
 1639808 bytes written: OK
U-Boot> nand read 0x7FC0 0x200000 0x190580

NAND read: device 0 offset 0x200000, size 0x190580
 1639808 bytes read: OK
U-Boot> bootm 0x7FC0
## Booting kernel from Legacy Image at 007FC0 ...
   Image Name:
   Image Type:   ARM Linux Kernel Image (uncompressed)
   Data Size:    1639744 Bytes = 1.6 MiB
   Load Address: 00008000
   Entry Point:  00008000
   Verifying Checksum ... OK
   Loading Kernel Image ... OK
OK

Starting kernel ...
```

- nboot: boot from NAND device

Command format is as below:

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

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

Below example use nboot to read Linux kernel image from NAND flash offset 0x200000 to DDR address 0x7fc0. Then boot Linux kernel by command bootm.

```
U-Boot> nboot 0x7fc0 0 0x200000

Loading from nand0, offset 0x200000
   Image Name:
   Image Type:   ARM Linux Kernel Image (uncompressed)
   Data Size:    1639744 Bytes = 1.6 MiB
   Load Address: 00007FC0
   Entry Point:  00008000
U-Boot> bootm 0x7fc0
## Booting kernel from Legacy Image at 00007fc0 ...
   Image Name:
   Image Type:   ARM Linux Kernel Image (uncompressed)
   Data Size:    1639744 Bytes = 1.6 MiB
   Load Address: 00008000
   Entry Point:  00008000
   Verifying Checksum ... OK
   XIP Kernel Image ... OK
OK

Starting kernel ...
```

1.6.5 SPI flash commands

U-Boot supports SPI flash relative commands including sf probe/read/write/erase/update. The command format is as below.

```
U-Boot> help sf
sf - SPI flash sub-system
```

```
Usage:
sf probe [[bus:]cs] [hz] [mode]    - init flash device on given SPI bus and
chip select
sf read addr offset len  - read `len' bytes starting at `offset' to memory
at `addr'
sf write addr offset len - write `len' bytes from memory at `addr' to flash
at `offset'
sf erase offset [+]len   - erase `len' bytes from `offset' `+len' round up
`len' to block size
sf update addr offset len     - erase and write `len' bytes from memory at
`addr' to flash at `offset'
U-Boot>
```

Note that you have to run command, sf probe, first before using sf read/write/erase/update. You can designate SPI speed in argument of sf probe. Below is an example to set SPI clock to 18 MHz.

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

Below is an example to read a Linux kernel image from SPI flash. First, use “sf probe” command to set SPI clock to 18 MHz. Then, “sf read” command read Linux kernel image stored at SPI flash offset 0x200000 to DDR 0x7fc0. Finally, use command, bootm, to boot Linux kernel image.

```
U-Boot> sf probe 0 18000000
SF: Detected EN25QH16-104HIP with page size 64 KiB, total 16 MiB
U-Boot> sf read 0x7FC0 0x200000 0x190580
U-Boot> bootm 0x7FC0
## Booting kernel from Legacy Image at 007FC0 ...
   Image Name:
   Image Type:   ARM Linux Kernel Image (uncompressed)
   Data Size:    1639744 Bytes = 1.6 MiB
   Load Address: 00008000
   Entry Point:  00008000
   Verifying Checksum ... OK
   Loading Kernel Image ... OK
OK

Starting kernel ...
```

1.6.6 Memory commands

- md: Memory display.

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

```

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

Below is an example to display the memory content from address 0x10000 to 0x100ff.

```
U-Boot> md 0x1000
00001000: bffbcf5c 5ffb56ff fcff5f41 ff67760b    \....V._A_...vg.
00001010: fcd227e3 dffefeeb 70cf7cb3 dbefc7cb    .'.......|.p....
00001020: fbda3e3b eb3e9ebb aa3abc95 e5fbbb2f    ;>....>...:./...
00001030: ffbbb319 effe9d7d bfbeeb09 ff7b4f31    ....}.......1O{.
00001040: f7bf3973 eaff296c e6fce35e 6fffcd7f    s9..l)..^......o
00001050: cfd28a65 8cd69f2b efeece87 677f3b8f    e...+........;.g
00001060: def67b1d deff7ece 3ffd4003 ffbf32c2    .{...~...@.?.2..
00001070: feef5b67 ffdfa2e6 b7ffe1d3 efffb707    g[..............
00001080: ed2fee4b 6fd852b9 cbf765dd 796dc3de    K./..R.o.e....my
00001090: ff9fcff9 ef7bae38 efb0aff3 f8fdf324    ....8.{.....$...
000010a0: fda577b7 cfbbebcc d5936aa0 088f362f    .w.......j../6..
000010b0: ff6bae5a beff9df1 eadded74 3de9fd3d    Z.k.....t...=..=
000010c0: dbff79bf 6f32ccf1 89bfa6b1 fbafeebf    .y....2o........
000010d0: 77f5b6cd bd7fe7fc 6e2366f2 dff7a5fc    ...w.....f#n....
000010e0: f9ff160b edba6d61 fbf88f79 ffef7b76    ....am..y...v{..
000010f0: 3efabd8c fbfaebe2 6f7d807a ffae9ace    ...>....z.}o....
U-Boot>
```

- mw: Memory write

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

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

Below is an example to write 4 words 0s to address 0x10000.

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

Display the memory content of address 0x10000. The first 4 words of address 0x10000 are 0s.

```
U-Boot> md 0x10000
00010000: 00000000 00000000 00000000 00000000    ................
```

```
00010010: e58c3004 e59c3008 e0843003 e58c3008    .0...0...0...0..
00010020: e1a01105 e1a03305 e0613003 e0833005    .....3...0a..0..
00010030: e1a02103 e0632002 e1a02102 e0862002    .!... c..!... ..
00010040: e58282d0 e58242d4 e59f3220 e0831001    .....B.. 2......
00010050: e5913110 e58232d8 e58262c8 e3a0300c    .1...2...b...0..
00010060: e58232b4 e59f321c e5823014 e254a000    .2...2...0....T.
00010070: 0a00006e e1a02305 e0422105 e0822005    n....#...!B.. ..
00010080: e1a03102 e0623003 e1a03103 e0863003    .1...0b..1...0..
00010090: e59342d8 e51b0038 eb015a3f e1a03000    .B..8...?Z...0..
000100a0: e59f01e4 e1a01004 e1a0200a eb007cc5    ......... ...|..
000100b0: ea00005e e2813040 e1a03183 e083300e    ^...@0...1...0..
000100c0: e0863003 e2832004 e5822000 e5832008    .0... ... ... ..
000100d0: e08c3001 e283308e e1a03103 e0863003    .0...0...1...0..
000100e0: e2833004 e5837000 e2811001 e2800001    .0...p..........
000100f0: e3500005 1affffee e1a03305 e0433105    ..P......3...1C.
U-Boot>
```

- cmp: Memory compare.

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

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

Below is an example to compare 64 words of address 0x8000 with address 0x9000.

```
U-Boot> cmp 0x8000 0x9000 64
word at 0x00008000 (0xe321f0d3) != word at 0x00009000 (0xe59f00d4)
Total of 0 word(s) were the same
U-Boot>
```

- mtest: simple RAM read/write test

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

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

Below is an example to test RAM read/write from address 0xa00000 to address 0xb00000 0x20 (32) iterations.

```
U-Boot> mtest 0xa00000 0xb00000 5a5a5a5a 20
Testing 00a00000 ... 00b00000:
Iteration:     32Pattern A5A5A5A5  Writing...            Reading...Tested
32 iteration(s) with 0 errors.
U-Boot>
```

1.6.7 USB commands

- usb: USB sub-system

```
usb: USB sub-system

U-Boot> help usb
usb - USB sub-system

Usage:
usb start - start (scan) USB controller
usb reset - reset (rescan) USB controller
usb stop [f] - stop USB [f]=force stop
usb tree - show USB device tree
usb info [dev] - show available USB devices
usb storage - show details of USB storage devices
usb dev [dev] - show or set current USB storage device
usb part [dev] - print partition table of one or all USB storage devices
usb read addr blk# cnt - read `cnt' blocks starting at block `blk#'
    to memory address `addr'
usb write addr blk# cnt - write `cnt' blocks starting at block `blk#'
    from memory address `addr'
U-Boot>
```

- usb reset

```
U-Boot> usb reset
(Re)start USB...
USB0:   USB EHCI 0.95
scanning bus 0 for devices... 2 USB Device(s) found
       scanning usb for storage devices... 1 Storage Device(s) found
U-Boot>
```

- usb start

```
U-Boot> usb start
(Re)start USB...
USB0:   USB EHCI 0.95
scanning bus 0 for devices... 2 USB Device(s) found
       scanning usb for storage devices... 1 Storage Device(s) found
U-Boot>
```

- usb tree

```
U-Boot> usb tree
USB device tree:
  1  Hub (480 Mb/s, 0mA)
  |  u-boot EHCI Host Controller
  |
  |+-2  Mass Storage (480 Mb/s, 200mA)
       Kingston DT 101 II 0019E000B4955B8C0E0B0158

U-Boot>
```

- usb info

```
U-Boot> usb info
1: Hub,  USB Revision 2.0
 - u-boot EHCI Host Controller
 - Class: Hub
 - PacketSize: 64  Configurations: 1
 - Vendor: 0x0000  Product 0x0000 Version 1.0
   Configuration: 1
   - Interfaces: 1 Self Powered 0mA
     Interface: 0
     - Alternate Setting 0, Endpoints: 1
     - Class Hub
     - Endpoint 1 In Interrupt MaxPacket 8 Interval 255ms

2: Mass Storage,  USB Revision 2.0
 - Kingston DT 101 II 0019E000B4955B8C0E0B0158
 - Class: (from Interface) Mass Storage
 - PacketSize: 64  Configurations: 1
 - Vendor: 0x0951  Product 0x1613 Version 1.0
   Configuration: 1
   - Interfaces: 1 Bus Powered 200mA
     Interface: 0
     - Alternate Setting 0, Endpoints: 2
     - Class Mass Storage, Transp. SCSI, Bulk only
     - Endpoint 1 In Bulk MaxPacket 512
     - Endpoint 2 Out Bulk MaxPacket 512

U-Boot>
```

- usb storage

```
U-Boot> usb storage
  Device 0: Vendor: Kingston Rev: PMAP Prod: DT 101 II
            Type: Removable Hard Disk
            Capacity: 3875.0 MB = 3.7 GB (7936000 x 512)
U-Boot>
```

- usb dev

```
U-Boot> usb dev

USB device 0: Vendor: Kingston Rev: PMAP Prod: DT 101 II
            Type: Removable Hard Disk
            Capacity: 3875.0 MB = 3.7 GB (7936000 x 512)
U-Boot>
```

- usb part

```
U-Boot> usb part

Partition Map for USB device 0  --   Partition Type: DOS

Part Start Sector    Num Sectors    UUID      Type
  1  8064            7927936        1dfc1dfb-01    0b Boot
U-Boot>
```

- usb read: read `cnt' blocks starting at block `blk#' to memory address `addr'.
- usb write: write `cnt' blocks starting at block `blk#' from memory address `addr'.

Below is an example that write device 0 block #2, 1 block from 0x10000, and read back device 0 block #2, 1 block to 0x20000. Then compare the memory content of 0x10000 and 0x20000 with 1 block (512 bytes).

```
U-Boot> usb write 0x10000 2 1

USB write: device 0 block # 2, count 1 ... 1 blocks write: OK
U-Boot> usb read 0x20000 2 1

USB read: device 0 block # 2, count 1 ... 1 blocks read: OK
U-Boot>
```

```
U-Boot> cmp 0x10000 0x20000 80
Total of 128 word(s) were the same
U-Boot>
```

- usbboot: boot from USB device

```
U-Boot> help usb  boot
usbboot - boot from USB device

Usage:
usbboot loadAddr dev:part
U-Boot>
```

Before using usbboot, you have to write Linux kernel image into USB device.
It can be achieved by command, usb write. However, we have to know the start block(sector) number where Linux kernel image put to. Below we use command, usb part, to show the partition map of USB device 0.

```
U-Boot> usb part

Partition Map for USB device 0  --   Partition Type: DOS

Part Start Sector   Num Sectors    UUID      Type
  1  8064           7927936        1dfc1dfb-01    0b Boot
U-Boot>
```

The start sector (block) number is 369 (0x171). Therefore, we use command, usb write, to write Linux kernel image to device 0 block # 369(0x171). The block count can be computed as below. The Linux kernel image is downloaded at 0x200000. It can be downloaded by ICE or TFTP or other tools. And the Linux kernel size is 1639808 bytes. 1639808/512 = 3202.75. So, it needs 3203 (0xc83) blocks to store the Linux kernel.

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

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

Now, the Linux kernel is stored in device 0 block # 369(0x171). So, We can load Linux kernel from USB device 0 partition 1 by command, usbboot.

```
U-Boot> usbboot 0x7fc0 0:1

Loading from usb device 0, partition 1: Name: usbda1  Type: U-Boot
   Image Name:
   Image Type:   ARM Linux Kernel Image (uncompressed)
   Data Size:    1639744 Bytes = 1.6 MiB
   Load Address: 00007FC0
   Entry Point:  00008000
U-Boot> bootm 0x7fc0
## Booting kernel from Legacy Image at 00007fc0 ...
   Image Name:
   Image Type:   ARM Linux Kernel Image (uncompressed)
   Data Size:    1639744 Bytes = 1.6 MiB
   Load Address: 00008000
   Entry Point:  00008000
   Verifying Checksum ... OK
   XIP Kernel Image ... OK
OK

Starting kernel ...
```

Besides, U-Boot supports command fatls and fatload that can access USB device files from file system. Below is an example that lists USB device file by command fatls and loads USB device file by command fatload.

```
U-Boot> fatls usb 0:1
  1639808   vmlinux.ub

1 file(s), 0 dir(s)

U-Boot>
U-Boot> fatload usb 0:1 0x7fc0 vmlinux.ub
reading vmlinux.ub
1639808 bytes read in 90 ms (17.4 MiB/s)
U-Boot>
U-Boot> bootm 0x7fc0
## Booting kernel from Legacy Image at 00007fc0 ...
   Image Name:
   Image Type:   ARM Linux Kernel Image (uncompressed)
   Data Size:    1639744 Bytes = 1.6 MiB
   Load Address: 00008000
   Entry Point:  00008000
   Verifying Checksum ... OK
   XIP Kernel Image ... OK
OK

Starting kernel ...
```

1.6.8 Environment variable commands

- setenv: set environment variables

```
U-Boot> help setenv
setenv - set environment variables

Usage:
setenv [-f] name value ...
    - [forcibly] set environment variable 'name' to 'value ...'
setenv [-f] name
    - [forcibly] delete environment variable 'name'
U-Boot>
```

Below is an example to set environment variable, ipaddr, to 192.168.0.101
And use command, echo, to show the value of ipaddr.

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

- saveenv: save environment variables to persistent storage.

```
U-Boot> help saveenv
saveenv - save environment variables to persistent storage

Usage:
saveenv
U-Boot>
```

- env: environment handling commands

```
U-Boot> help env
env - environment handling commands

Usage:
env default [-f] -a - [forcibly] reset default environment
env default [-f] var [...] - [forcibly] reset variable(s) to their default
values
env delete [-f] var [...] - [forcibly] delete variable(s)
env edit name - edit environment variable
env export [-t | -b | -c] [-s size] addr [var ...] - export environment
env import [-d] [-t | -b | -c] addr [size] - import environment
env print [-a | name ...] - print environment
env run var [...] - run commands in an environment variable
env save - save environment
env set [-f] name [arg ...]

U-Boot>
```

1.6.9 Decrypt commands (NUC970 only)

In addition to the original commands U-Boot support. NUC970 U-Boot added a command for decryption. It supports AES decryption only. The command format is as below.

```
U-Boot> help decrypt
decrypt - Decrypt image(kernel)

Usage:
decrypt decrypt aes SrcAddr DstAddr Length - Decrypt the image from SrcAddr
to DstAddr with lenth [Length].
Example : decrypt aes 0x8000 0x10000 0x200- decrypt the image from 0x8000
to 0x10000 and lenth is 0x200

decrypt program aes EnSecure - program AES key to MTP and [Enable/Disable]
secure boot.
Example :  decrypt program aes 1 - program AES key to MTP and Enable secure
boot.
Example :  decrypt program aes 0 - program AES key to MTP but Disable
secure boot.

Note that before enabling secure boot, you have to burn U-Boot with the
same AES key!
```

Otherwise, your system will be locked!!!
For instance, decrypt a Linux kernel image from 0x200000 to 0x400000 with length 0x190580. The command is as below.
U-Boot> decrypt aes 0x200000 0x400000 0x190580

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

The command “decrypt program” can burn AES key to MTP, and set if enable secure boot mode or not.
For instance, if burn AES key to MTP, but”NOT” enable secure boot, command is:

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

If burn AES key to MTP, and also enable secure boot, command is:

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

Note that before enable secure boot,you have to confirm using Nu-Writer to burn U-Boot encrypted with the same AES key ,otherwise,your system will be locked and boot failed when you reset or power-on again,therefore,it must be very carefully to use this command.

1.6.10 MMC commands

- mmc : MMC sub-system

U-Boot support MMC relative command, include read/write/erase/list/dev.
The command format is as below.

```
U-Boot> help mmc
mmc - MMC sub system

Usage:
mmc read addr blk# cnt
mmc write addr blk# cnt
mmc erase blk# cnt
mmc rescan
mmc part - lists available partition on current mmc device
mmc dev [dev] [part] - show or set current mmc device [partition]
mmc list - lists available devices
U-Boot>
```

mmc list : list all mmc device

```
U-Boot> mmc list
mmc: 0
mmc: 1
mmc: 2
U-Boot>
```

NUC970/N9H30 supports mmc device including SD port 0, SD port 1 and eMMC.
User can modify following three definitions in nuc970_evb.h according to your platform.

```
#define CONFIG_NUC970_SD_PORT0
#define CONFIG_NUC970_SD_PORT1
#define CONFIG_NUC970_EMMC
```

The default setting enables SD port 0 and SD port 1, eMMC and NAND can not be used at the same time, NAND is disabled by default setting.
If SD port 0, SD port 1 and eMMC are all enabled, mmc device numbers are as below:
Device number 0 is SD port 0
Device number 1 is SD port 1
Device number 2 is eMMC

If your platform supports SD port 0 and eMMC (doesn’t support SD port 1), you have to disable the definition CONFIG_NUC970_SD_PORT1 in nuc970_evb.h
The command, mmc list, can see the following result:

```
U-Boot> mmc list
```

```
mmc: 0
mmc: 1
U-Boot>
```

Device number 0 is SD port 0
Device number 1 is eMMC

If your platform supports eMMC (doesn’t support SD port 0 and SD port 1), you have to disable the definition CONFIG_NUC970_SD_PORT0 and CONFIG_NUC970_SD_PORT1 in nuc970_evb.h
The command, mmc list, can see the following result:

```
U-Boot> mmc list
mmc: 0
U-Boot>
```

Device number 0 is eMMC

Following example is user enable SD port 0, SD port 1 and eMMC.
User set current device to SD port 1 by “mmc dev” command, then use mmclist command to display SD relative information.

```
U-Boot> mmc dev 1
mmc1 is current device
U-Boot> mmcinfo
Device: mmc
Manufacturer ID: 3
OEM: 5344
Name: SD02G
Tran Speed: 25000000
Rd Block Len: 512
SD version 2.0
High Capacity: No
Capacity: 1.8 GiB
Bus Width: 4-bit
U-Boot>
```

Following example is user sets current device to device 0 (SD port 0) by “mmc dev” command.
Use “mmc erase” command to erase SD card block 0x30 and 0x31, and copy data from DDR address 0x8000 to SD card block 0x30 and 0x31, then read SD card block 0x30 and 0x31 to DDR 0x500000. Finally compare the data in DDR address 0x8000 with address 0x500000 to validate SD access.

```
U-Boot> mmc dev 0
mmc0 is current device
U-Boot> mmc erase 0x30 2


MMC erase: dev # 0, block # 48, count 2 ... 2 blocks erase: OK
U-Boot> mmc write 0x8000 0x30 2
```

```
MMC write: dev # 0, block # 48, count 2 ... 2 blocks write: OK
U-Boot> mmc read 0x500000 0x30 2

MMC read: dev # 0, block # 48, count 2 ... 2 blocks read: OK
U-Boot> cmp.b 0x8000 0x500000 0x400
Total of 1024 byte(s) were the same
U-Boot>
```

Following example is user sets current device to devide 2 (eMMC) by “mmc dev” command.
Use “mmc erase” command to erase SD card block 1024 to 2047, and copy data from DDR address 0x8000 to SD card block 1024 ~ 2047, then read SD card block 1024 ~ 2047 to DDR 0x500000. Finally compare the data in DDR address 0x8000 with address 0x500000 to validate SD access.

```
U-Boot> mmc dev 2
mmc2(part 0) is current device
U-Boot> mmc erase 0x400 0x400

MMC erase: dev # 2, block # 1024, count 1024 ... 1024 blocks erase: OK
U-Boot> mmc write 0x8000 0x400 0x400

MMC write: dev # 2, block # 1024, count 1024 ... 1024 blocks write: OK
U-Boot> mmc read 0x500000 0x400 0x400

MMC read: dev # 2, block # 1024, count 1024 ... 1024 blocks read: OK
U-Boot> cmp.b 0x8000 0x500000 0x4000
Total of 16384 byte(s) were the same
U-Boot>
```

We can access SD/eMMC card by “mmc” command. Besides, we can access the files in SD/eMMC card by “fatls” and “fatload” command.
Following example use “fatls” command to list the file in SD port 0, then “fatload” command to read Linux kernel image (vmlinux.ub) to DDR address 0x7fc0,finally boot Linux kernel by “bootm” command.

```
U-Boot> fatls mmc 0
  1639808   vmlinux.ub
        0   4gsd.txt

2 file(s), 0 dir(s)

U-Boot> fatload mmc 0 0x7fc0 vmlinux.ub
reading vmlinux.ub
1639808 bytes read in 301 ms (5.2 MiB/s)
U-Boot> bootm 0x7fc0
## Booting kernel from Legacy Image at 00007fc0 ...
```

```
   Image Name:
   Image Type:   ARM Linux Kernel Image (uncompressed)
   Data Size:    1639744 Bytes = 1.6 MiB
   Load Address: 00008000
   Entry Point:  00008000
   Verifying Checksum ... OK
   XIP Kernel Image ... OK
OK

Starting kernel ...
```

1.6.11 MTD commands

- mtdparts : define flash/nand partitions

U-Boot supports MTD partition relative command,including add/del/list.

```
U-Boot> help mtd
mtdparts - define flash/nand partitions

Usage:
mtdparts
    - list partition table
mtdparts delall
    - delete all partitions
mtdparts del part-id
    - delete partition (e.g. part-id = nand0,1)
mtdparts add <mtd-dev> <size>[@<offset>] [<name>] [ro]
    - add partition
mtdparts default
    - reset partition table to defaults
-----
this command uses three environment variables:
'partition' - keeps current partition identifier
partition  := <part-id>
<part-id>  := <dev-id>,part_num
'mtdids' - linux kernel mtd device id <-> u-boot device id mapping
mtdids=<idmap>[,<idmap>,...]
<idmap>    := <dev-id>=<mtd-id>
<dev-id>   := 'nand'|'nor'|'onenand'<dev-num>
<dev-num>  := mtd device number, 0...
<mtd-id>   := unique device tag used by linux kernel to find mtd device
(mtd->name)
'mtdparts' - partition list
mtdparts=mtdparts=<mtd-def>[;<mtd-def>...]
```

```
<mtd-def>  := <mtd-id>:<part-def>[,<part-def>...]
<mtd-id>   := unique device tag used by linux kernel to find mtd device
(mtd->name)
<part-def> := <size>[@<offset>][<name>][<ro-flag>]
<size>     := standard linux memsize OR '-' to denote all remaining space
<offset>   := partition start offset within the device
<name>     := '(' NAME ')'
<ro-flag>  := when set to 'ro' makes partition read-only (not used, passed
to kernel)
U-Boot>
```

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

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

mtdparts list all the mtd partitions

```
U-Boot> mtdparts
device nand0 <nand0>, # parts = 3
 #: name                size            offset          mask_flags
 0: u-boot              0x00100000      0x00000000      0
 1: kernel              0x01400000      0x00100000      0
 2: user                0x06b00000      0x01500000      0
active partition: nand0,0 - (u-boot) 0x00100000 @ 0x00000000
defaults:
mtdids  : nand0=nand0
mtdparts: mtdparts=nand0:0x100000@0x0(u-boot),0x1400000@0x100000(kernel),-
(user)
U-Boot>
```

1.6.12 UBI commands

- ubi : ubi commands

U-Boot supports UBI relative command, including info/create/read/write.

```
U-Boot> help ubi
ubi - ubi commands
Usage:
ubi part [part] [offset]
```

```
 - Show or set current partition (with optional VID header offset)
ubi info [l[ayout]] - Display volume and ubi layout information
ubi create[vol] volume [size] [type] - create volume name with size
ubi write[vol] address volume size - Write volume from address with size
ubi read[vol] address volume [size] - Read volume to address with size
ubi remove[vol] volume - Remove volume
[Legends]
 volume: character name
 size: specified in bytes
 type: s[tatic] or d[ynamic] (default=dynamic)
U-Boot>
```

ubi part : Display or set current partition

```
U-Boot> ubi part user
Creating 1 MTD partitions on "nand0":
0x000001500000-0x000008000000 : "mtd=2"
UBI: attaching mtd1 to ubi0
UBI: physical eraseblock size:   131072 bytes (128 KiB)
UBI: logical eraseblock size:    126976 bytes
UBI: smallest flash I/O unit:    2048
UBI: VID header offset:          2048 (aligned 2048)
UBI: data offset:                4096
UBI: attached mtd1 to ubi0
UBI: MTD device name:            "mtd=2"
UBI: MTD device size:            107 MiB
UBI: number of good PEBs:        855
UBI: number of bad PEBs:         1
UBI: max. allowed volumes:       128
UBI: wear-leveling threshold:    4096
UBI: number of internal volumes: 1
UBI: number of user volumes:     1
UBI: available PEBs:             17
UBI: total number of reserved PEBs: 838
UBI: number of PEBs reserved for bad PEB handling: 8
UBI: max/mean erase counter: 6/4
U-Boot>
```

ubi info : display capacity and ubi information

```
U-Boot> ubi info l
UBI: volume information dump:
UBI: vol_id          0
```

```
UBI: reserved_pebs   826
UBI: alignment       1
UBI: data_pad        0
UBI: vol_type        3
UBI: name_len        9
UBI: usable_leb_size 126976
UBI: used_ebs        826
UBI: used_bytes      104882176
UBI: last_eb_bytes   126976
UBI: corrupted       0
UBI: upd_marker      0
UBI: name            nandflash

UBI: volume information dump:
UBI: vol_id          2147479551
UBI: reserved_pebs   2
UBI: alignment       1
UBI: data_pad        0
UBI: vol_type        3
UBI: name_len        13
UBI: usable_leb_size 126976
UBI: used_ebs        2
UBI: used_bytes      253952
UBI: last_eb_bytes   2
UBI: corrupted       0
UBI: upd_marker      0
UBI: name            layout volume
U-Boot>

ubifsmount : mount ubifs volume
U-Boot> help ubifsmount
ubifsmount - mount UBIFS volume
Usage:
ubifsmount <volume-name>
    - mount 'volume-name' volume
U-Boot> ubifsmount ubi0:nandflash
UBIFS: mounted UBI device 0, volume 0, name "nandflash"
UBIFS: mounted read-only
UBIFS: file system size:   103485440 bytes (101060 KiB, 98 MiB, 815 LEBs)
UBIFS: journal size:       5206016 bytes (5084 KiB, 4 MiB, 41 LEBs)
UBIFS: media format:       w4/r0 (latest is w4/r0)
```

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

ubifsls : list files in a directory

```
U-Boot> help ubifsls
ubifsls - list files in a directory
Usage:
ubifsls [directory]
    - list files in a 'directory' (default '/')
U-Boot> ubifsls
<DIR>         160  Thu Jan 01 00:08:09 1970  tt
U-Boot>
```

ubifsumount : unmount UBIFS volume

```
U-Boot> help ubifsumount
ubifsumount - unmount UBIFS volume
Usage:
ubifsumount     - unmount current volume
U-Boot> ubifsumount
Unmounting UBIFS volume nandflash!
U-Boot>
```

1.6.13 YAFFS2 commands

- yaffs : yaffs commands

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

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

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

ydevconfig configure YAFFS mount point

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

ydevls list YAFFS mount points

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

ymount mount YAFFS

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

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

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

ymkdir make a directory

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

yrmdir delete a directory

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

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

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

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

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

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

yrm delete a file

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

yumount unmounts YAFFS

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

1.7 Environment variables

1.7.1 Environment variables configuration

Environment variables can store in NAND flash、SPI flash or eMMC, user can modify below three definitions in nuc970_evb.h:

- CONFIG_ENV_IS_IN_NAND: environment variables are stored in NAND flash
- CONFIG_ENV_IS_IN_SPI_FLASH: environment variables are stored in SPI flash
- CONFIG_ENV_IS_IN_MMC: environment variables are stored in eMMC

Note that only one of them can be defined.

User can configure the flash offset address environment variables stored and the space reserved for environment variables in nuc970_evb.h:

- CONFIG_ENV_OFFSET: the flash offset address environment variables stored
- CONFIG_ENV_SIZE: the space reserved for environment variables

1.7.2 Default environment variables

U-Boot has some default environment variables. If the variables are not stored in flash, U-Boot will assign default value to the variables.
Below are the default environment variables.

- baudrate

Console baudrate, the value is from CONFIG_BAUDRATE in nuc970_evb.h

- bootdelay

It's the boot delay time when U-Boot run the command script in bootcmd. Its unit is second. Before it is countdown to 0, hit any key to stop running script in bootcmd.

- ethact

It sets which Ethernet interface state is active, since nuc970 ethernet driver set device name to emac, ethact can be set to emac only.

- ethaddr

Ethernet MAC address. ethaddr value is from CONFIG_ETHADDR in nuc970_evb.h

- stderr

Set stderr, default value is serial

- stdin

Set stdin, default value is serial

- stdout

Set stdout, default value is serial

1.7.3 Command Script

Below are script relative commands

- bootcmd

Whenever U-Boot boots up, U-Boot executes the script in bootcmd.
Following example set bootcmd as: read Linux kernel from SPI flash offset 0x200000 to DDR address 0x7fc0, and boot Linux kernel.
Remember to save the environment variables to flash.

```
U-Boot> 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

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

```
U-Boot> 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>
```

NOTE: If you have set the U-Boot environment variables, that user should select the "Command line partition table parsing" in the kernel device setting. And then, the parameters will be passed into kernel.

```
Device Drivers  --->
     -*- Memory Technology Device (MTD) support  --->
          <*>   Command line partition table parsing
```

1.7.4 New added environment variable

NUC970/N9H30 U-Boot adds below new environment variable.

- watchdog: Enable or disable watchdog timer function.

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

The parameter "mode" can be on or off.
on: watchdog timer function enabled
off: watchdog timer function disabled
For instance, disable watchdog function

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

Enable watchdog function

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

Remember save the environment variables to flash.

```
U-Boot> saveenv
```

1.8 mkimage tool

U-Boot supports a number of different image formats that can be downloaded, saved to flash and executed. The types of such image files supported by U-Boot, include:

- Linux Kernel
- Script files
- Standalone binaries
- RAM disk images

These images are often referred to as an ".ub" files, as that is the file extension name that is often used to name them.

1.8.1 Use mkimage to generate Linux kernel image

The mkimage tool is located in tools/mkimage. Below is an example to encapsulate an ARM Linux kernel binary file (970image). Linux kernel load address is 0x8000 and entry point is 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: set CPU architecture to arm
-O: set operating system to linux
-T: set image type to kernel
-a: set load address to 0x8000
-e: set entry point to 0x8000
-d: set image data from 970image

1.8.2 Checksum calculation (SHA-1 or crc32, NUC970 only)

mkimage tool calculates the checksum of Linux kernel image and puts it in the header.
The command “bootm” also calculates the checksum of Linux kernel image loaded to DDR and compares the checksum with the value in header of Linux kernel image. If they are different, it shows the error message “Verifying Checksum ... Bad Data CRC” and stop booting Linux kernel. If they are the same, it shows the message “Verifying Checksum ... OK” and continues to boot Linux kernel. The default option of mkimage tool is using software crc32 to calculate the checksum of Linux kernel image. It’s time-consuming. If you want to speed up booting Linux kernel, you can use hardware SHA-1 to calculate the checksum of Linux kernel.
NUC970/N9H30 adds a new parameter “-S” to calculate Linux kernel checksum.
The original checksum calculation method of mkimage tool is crc32, NUC970/N9H30 provides another option, SHA-1, below is an example uses SHA-1 to calculate Linux kernel checksum. Add option “-S sha1”. Remember to enable CONFIG_NUC970_HW_CHECKSUM in nuc970_evb.h

```
u-boot/tools# ./mkimage -A arm -O linux  -T kernel -S sha1 -a 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: set CPU architecture to arm
-O: set operating system to linux
-T: set image type to kernel
-a: set load address to 0x8000
-e: set entry point to 0x8000
-S: set checksum calculation to sha1
-d: set image data from 970image

If you select crc32 to calculate Linux kernel checksum,below is an example : Added an option “-S crc32”. Remember to disable CONFIG_NUC970_HW_CHECKSUM in nuc970_evb.h

```
u-boot/tools# ./mkimage -A arm -O linux  -T kernel -S crc32 -a 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: set CPU architecture to arm
-O: set operating system to linux
-T: set image type to kernel
-a: set load address to 0x8000
-e: set entry point to 0x8000
-S: set checksum calculation to crc32
-d: set image data from 970image

1.8.3 AES encrypt (NUC970 only)

Besides, NUC970/N9H30 mkimage tool adds AES encrypt function added two new parameters, -E AES, Encrypt image. –K, designates the key file that AES encryption uses. Following example encapsulate and encrypt an ARM Linux kernel image (970image), AES key is in key.dat. Linux kernel load address is 0x7fc0, execution address is 0x8000.

```
u-boot/tools# ./mkimage -A arm -O linux  -T kernel -a 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: set CPU architecture to arm
-O: set operating system to linux
-T: set image type to kernel
-a: set load address to 0x8000
-e: set entry point to 0x8000
-d: set image data from 970image
-G: set encryption to AES
-H: Designate the key file that AES encryption uses

There is a file named key.dat in directory tools/, the file content is AES key. User can edit key.dat to modify key

The content of key.dat is as below:

```
0x34125243
0xc41a9087
0xa14cae80
0xc4c38790
0x672ca187
0xd4f785a1
0x562fa1c5
0x78561198
```

Each line has 4 bytes, 8 lines total.
Each line starts with 0x, please do not modify it. If you want to modify key, please directly modify the value after 0x.
Please modify this file in Linux environment,if you edit it in Windows and copy to Linux, you can use dos2unix tool to transform key.dat,otherwise the AES key for encryption is wrong.

```
u-boot/tools$ dos2unix key.dat
dos2unix: converting file key.dat to Unix format.
```

1.9 Security issue (NUC970 only)

1.9.1 Encrypt

To protect the security of image file and avoid the hacker, we encrypt image by mkimage tool. Below is an example for AES encryption.

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

1.9.2 Decrypt

We can also decrypt the image in U-Boot command line,below is an example to decrypt an encrypted Linux kernel located in address 0x200000 to 0x400000,Linux kernel size is 0x190580.

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

1.9.3 Risk

After the image is decrypted, hacker can use the command “md” to display the memory and know the image content.

There is a solution for this security issue：

- Disable md command.

Modify include/config_cmd_default.h
Remark the line #define CONFIG_CMD_MEMORY

1.10 Watchdog timer

1.10.1 Watchdog timer configuration

Modify below definition in nuc970_evb.h to enable NUC970/N9H30 watchdog timer function.

```
#define CONFIG_HW_WATCHDOG
```

Remark the definition will disable NUC970/N9H30 watchdog timer function.

1.10.2 Watchdog timer environment variables

When NUC970/N9H30 watchdog timer configuration is enabled, user can set environment variable “watchdog” to “off” or “0”, the watchdog timer function will be disabled without modifying configuration file or recomplation.

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

Set environment variable “watchdog” to “on” will enable watchdog timer function again.

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

After modifying the environment variable “watchdog”, remember to use command “saveenv” to save variable “watchdog” to flash.
If the configuration of watchdog timer in nuc970_evb.h is disabled, it is meaningless to modify environment variable “watchdog”.

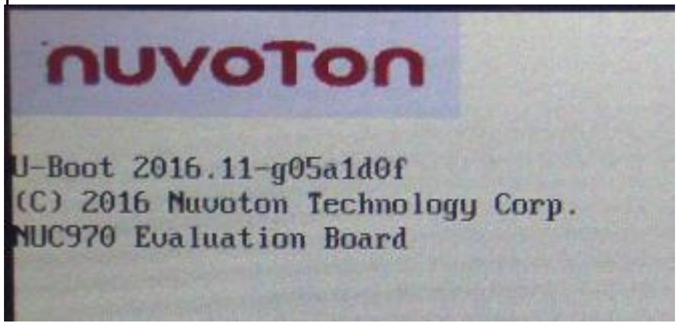
1.10.3 Watchdog timer period

When Watchdog timer function is enabled and system is idle more than 14 seconds, Watchdog timer will reset system；Whenever use enter a command (input Enter key), the idle time will be reset to 0.

1.11 U-Boot LCD

1.11.1 NUC970/N9H30 LCD display content

During U-Boot boot up, Nuvoton LOGO and U_Boot relative information is displayed in LCD panel.



```
U-Boot 2016.11-g05a1d0f
(C) 2016 Nuvoton Technology Corp.
NUC970 Evaluation Board
```

1.11.2 How to replace LOGO

To change Nuvoton LOGO to another one, if the company name is abc, LOGO file name is abc.bmp, the steps are as below:

- Put abc.bmp in tools/logos
- Modify tools/Makefile

Search the below seven lines,

```
# 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
```

Add below definition behind above seven lines.

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

1.12 GPIO

U-Boot GPIO can be used for LED.
NUC970/N9H30 U-Boot provides the function to set GPIO. User can access GPIO by NUC970/N9H30 GPIO driver interface.

1.12.1 NUC970/N9H30 GPIO

NUC970/N9H30 GPIO port includes port A ~ port J, each port has 16 pins.
Note that GPIO port C pin 15 and GPIO port J pin 5~15 are reserved, please do not use them.
NUC970/N9H30 U-Boot assign each pin a GPIO number, for instance, GPIO port A pin 0 number is GPIO_PA0, GPIO port B pin 2 number is GPIO_PB2
User has to pass GPIO number when calling NUC970/N9H30 GPIO driver function.

1.12.2 GPIO driver interface

NUC970/N9H30 provides following GPIO APIs

```
int gpio_request(unsigned gpio, const char *label);
int gpio_direction_input(unsigned gpio);
int gpio_direction_output(unsigned gpio, int value);
int gpio_get_value(unsigned gpio);
int gpio_set_value(unsigned gpio, int value);
```

The first parameter of each API is GPIO number.

- gpio_request

Confirm GPIO is in used or not, the second parameter is not used, you can fill in 0.
If the designated GPIO pin is switched to other function (not GPIO), there will be error message.
For instance, when we use GPIO port D0, and calling gpio_request():
gpio_request(GPIO_PD0,NULL);
If port D0 is switched to other function, you will get below error message.
[gpio_request] Please Check GPIO pin [96], multi-function pins = 0x6

- gpio_direction_input

Set GPIO pin to input mode.

- gpio_direction_output

Set GPIO pin to output mode and output value.

- gpio_get_value

Read GPIO pin value

- gpio_set_value

Set GPIO pin output value.

1.12.3 Example

Below example set GPIO port G0 ~ port G5 output value to 0x101010.

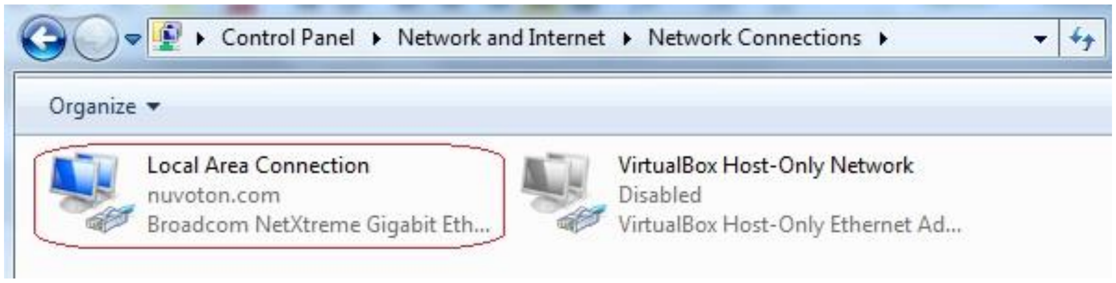
```
gpio_request(GPIO_PG0,NULL);
gpio_direction_output(GPIO_PG0, 0);
gpio_request(GPIO_PG1,NULL);
gpio_direction_output(GPIO_PG1, 1);
gpio_request(GPIO_PG2,NULL);
gpio_direction_output(GPIO_PG2, 0);
gpio_request(GPIO_PG3,NULL);
gpio_direction_output(GPIO_PG3, 1);
gpio_request(GPIO_PG4,NULL);
gpio_direction_output(GPIO_PG4,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

1.13 Network test environment

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

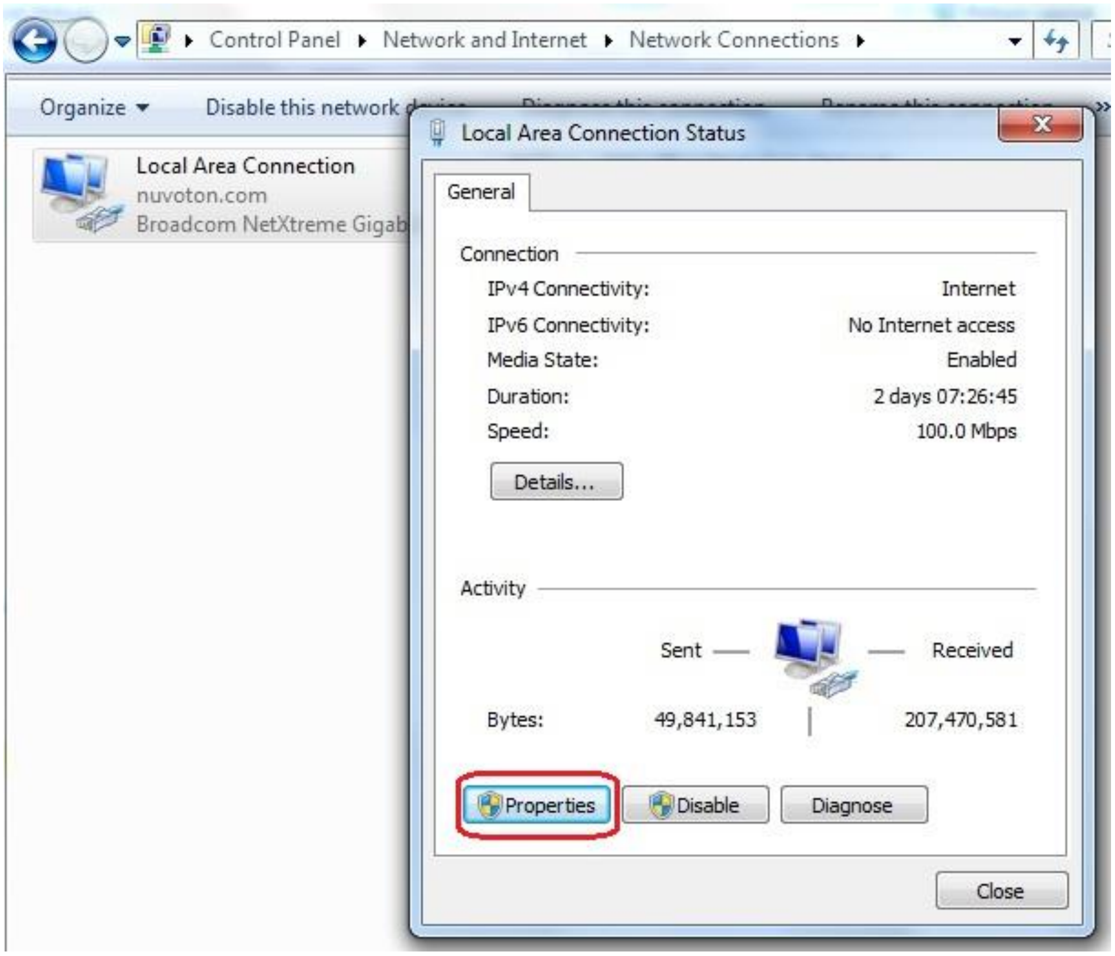
1.13.1 Set static IP address

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



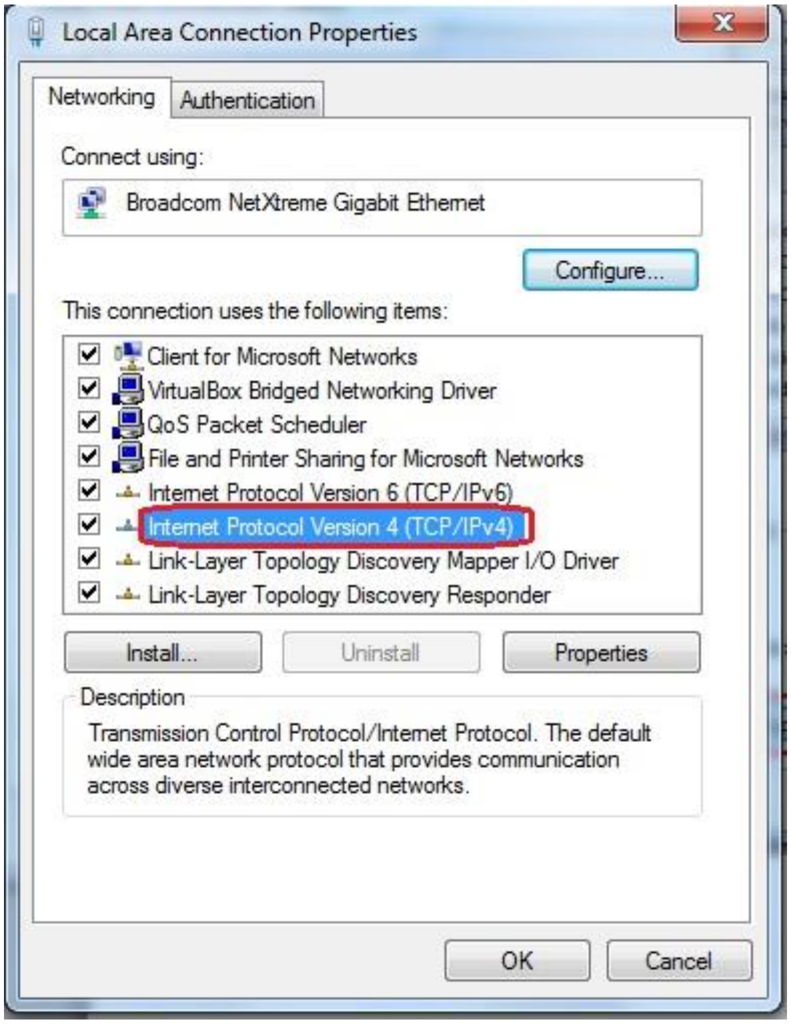
Control Panel ▸ Network and Internet ▸ Network Connections ▸
Organize ▾
Local Area Connection
nuvoton.com
Broadcom NetXtreme Gigabit Eth...
VirtualBox Host-Only Network
Disabled
VirtualBox Host-Only Ethernet Ad...

- Choose Properties



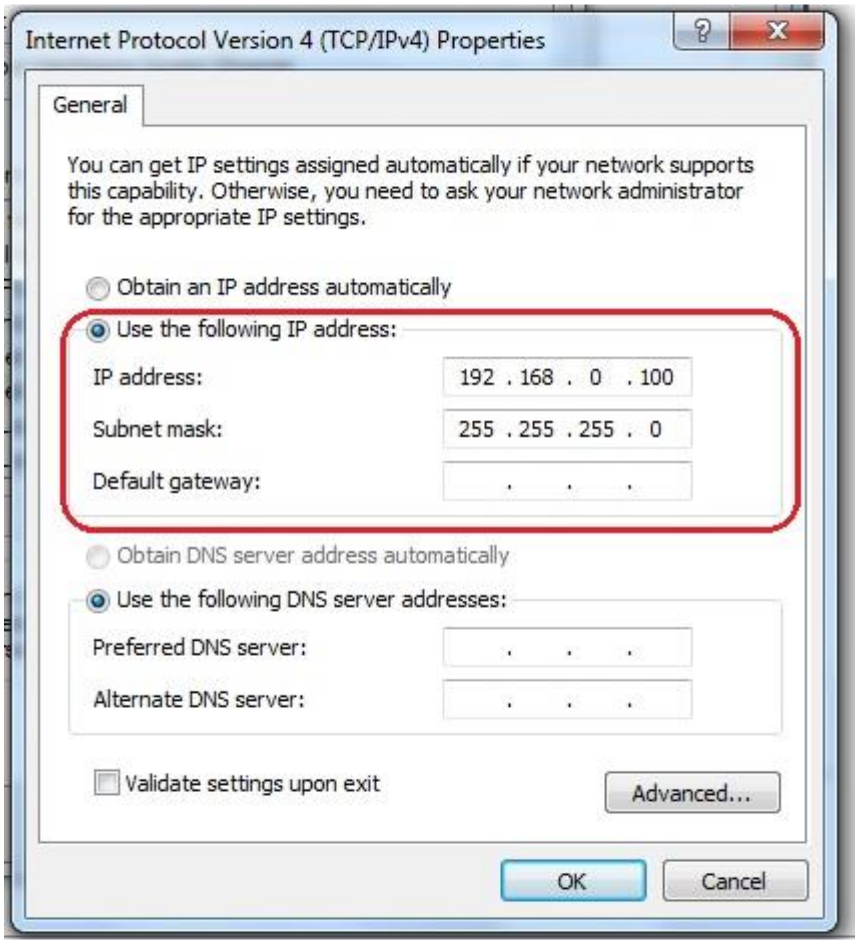
IPv4 Connectivity: Internet
IPv6 Connectivity: No Internet access
Media State: Enabled
Duration: 2 days 07:26:45
Speed: 100.0 Mbps
Bytes: 49,841,153 | 207,470,581

- Choose Internet Protocol Version 4 (TCP/IPv4)


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

Transmission Control Protocol/Internet Protocol. The default wide area network protocol that provides communication across diverse interconnected networks.

- Set static IP address


Internet Protocol Version 4 (TCP/IPv4) Properties

General

You can get IP settings assigned automatically if your network supports this capability. Otherwise, you need to ask your network administrator for the appropriate IP settings.

Obtain an IP address automatically [✓ Use the following IP address:]

IP address: 192 . 168 . 0 . 100

Subnet mask: 255 . 255 . 255 . 0

Default gateway: . . .

Obtain DNS server address automatically [✓ Use the following DNS server addresses:]

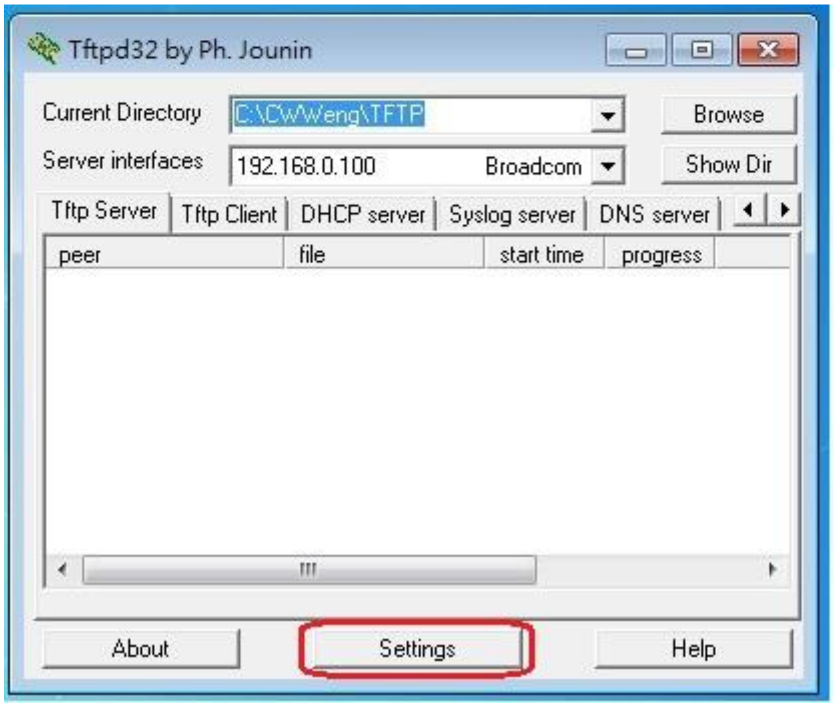
Preferred DNS server: . . .

Alternate DNS server: . . .

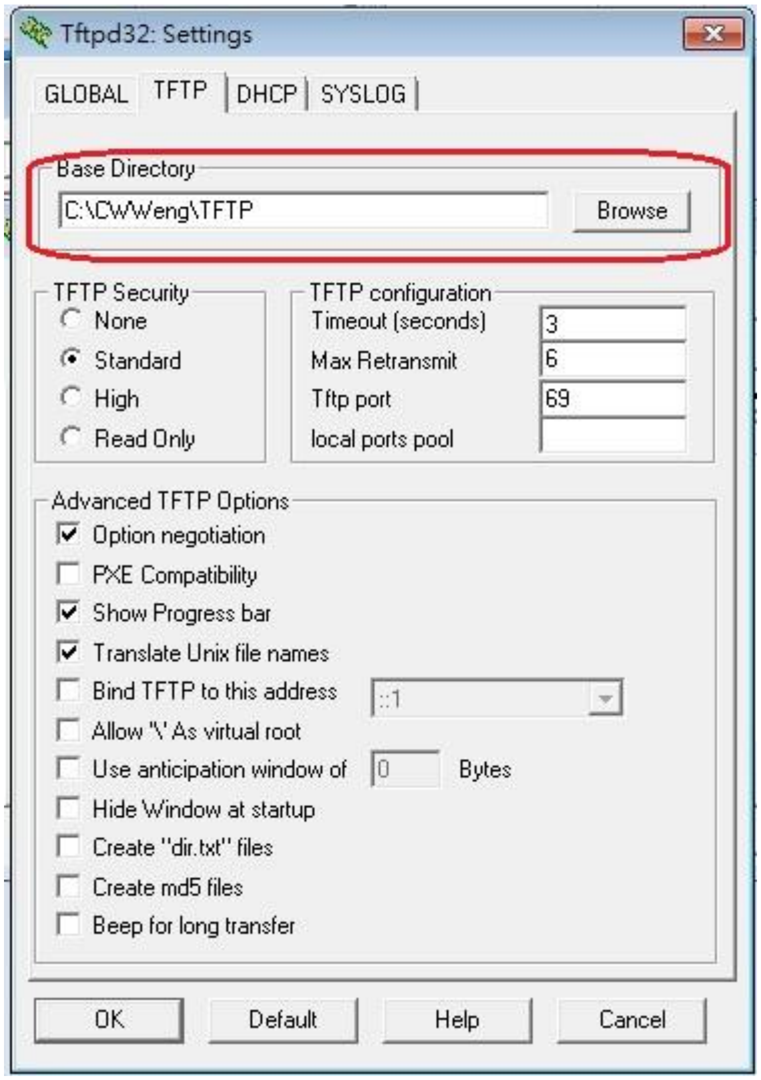
Validate settings upon exit

1.13.1 TFTP and DHCP server

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



Choose Settings to setup TFTP server and DHCP server.
Set base directory for TFTP server.



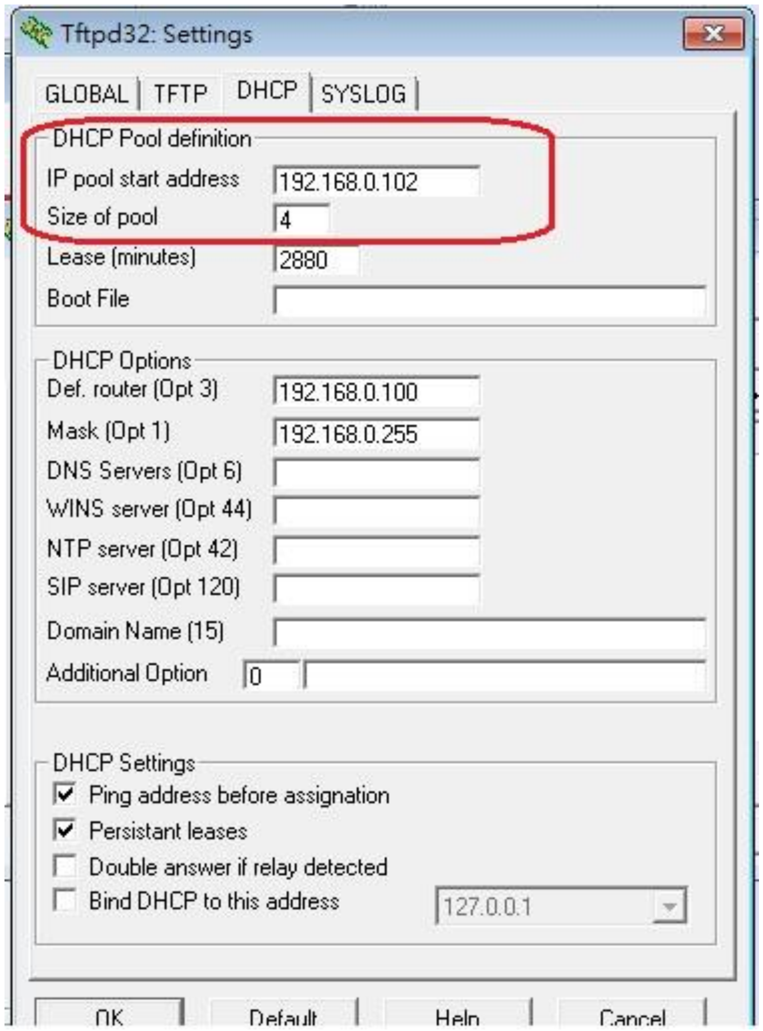
Base Directory: C:\CWWeng\TFTP

TFTP Security: ( ) None [✓ Standard] ( ) High ( ) Read Only

TFTP configuration:
Timeout (seconds): 3
Max Retransmit: 6
Tftp port: 69
local ports pool:

Advanced TFTP Options:
- [✓] Option negotiation
- [ ] PXE Compatibility
- [✓] Show Progress bar
- [✓] Translate Unix file names
- [ ] Bind TFTP to this address ::1
- [ ] Allow '\' As virtual root
- [ ] Use anticipation window of 0 Bytes
- [ ] Hide Window at startup
- [ ] Create "dir.txt" files
- [ ] Create md5 files
- [ ] Beep for long transfer

Set DHCP pool definitions. Below is an example to set IP pool start from 192.168.0.102


Tftpd32: Settings

GLOBAL | TFTP | DHCP | SYSLOG

DHCP Pool definition

IP pool start address: 192.168.0.102

Size of pool: 4

Lease (minutes): 2880

Boot File:

DHCP Options

Def. router (Opt 3): 192.168.0.100

Mask (Opt 1): 192.168.0.255

DNS Servers (Opt 6):

WINS server (Opt 44):

NTP server (Opt 42):

SIP server (Opt 120):

Domain Name (15):

Additional Option: 0

DHCP Settings

- [✓] Ping address before assignation
- [✓] Persistant leases
- [ ] Double answer if relay detected
- [ ] Bind DHCP to this address: 127.0.0.1

1.14 Speed up SPI flash boot

In SPI flash boot, U-Boot uses the command “sf read” to read Linux kernel from SPI flash to DDR，and uses the command “bootm” to boot Linux kernel. There are two methods to shorten the SPI flash boot up time including speed up SPI and calculate checksum of Linux kernel image by SHA-1.

1.14.1 Speed up SPI

Set SPI clock to higher rate can speed up SPI.

Below is an example that uses “sf probe” command to set SPI clock to 18 MHz that is the highest SPI clock.

```
U-Boot> sf probe 0 18000000
SF: Detected W25Q128 with page size 4 KiB, total 16 MiB
U-Boot>
```

1.14.2 Calculate checksum of Linux kernel image by SHA-1 (NUC970 only)

The command “bootm” calculates the checksum of Linux kernel by hardware can also shorten boot time. Please reference 1.8.2 that introduce how-to use SHA-1 to calculate checksum to speed up booting Linux kernel.

1.15 Use Power-on setting to overwrite NAND flash page size and ECC type

U-Boot gets NAND flash page size and ECC type by scanning NAND ID table in NAND initialization. If you use a NAND flash that is not supported in the ID table, or the page size and ECC type mismatch with ID table, you can use Power-on setting switch on EVB to overwrite them.

1.15.1 Config 6/7 for page size

Config 7 = 0, Config 6 = 0 set NAND flash page size to 2K Byte.

Config 7 = 0, Config 6 = 1 set NAND flash page size to 4K Byte.

Config 7 = 1, Config 6 = 0 set NAND flash page size to 8K Byte.

1.15.2 Config 8/9 for ECC type

Config 9 = 0, Config 8 = 0 set NAND flash ECC type to BCH T12.

Config 9 = 0, Config 8 = 1 set NAND flash ECC type to BCH T15.

Config 9 = 1, Config 8 = 0 set NAND flash ECC type to BCH T24.

1.16 U-Boot for Tomato

For Tomato, there is a Tomato configuration that U-Boot boot from SPI flash and automatically boot Linux kernel named “970image.sha” from SD card.

1.16.1 Build Tomato U-Boot

Clean all object code.

```
# make distclean
```

Compile U-Boot

```
# make nuc970_tomato_defconfig
# make
```

1.16.2 Tomato U-Boot link address

The link address of Tomato U-Boot is defined in include/configs/nuc970_tomato.h.

```
#define CONFIG_SYS_TEXT_BASE        0x2000000
```

CONFIG_SYS_TEXT_BASE defines U-Boot link address.

Tomato U-Boot link address is 0x2000000.

1.16.3 Use mkimage tool to pack Linux kernel

Tomato U-Boot boot SPI flash and automatically boot Linux kernel from SD card. The Linux kernel should be named 970image.sha and located in the root directory of SD card. In order to speed up boot time, Tomato U-Boot uses hardware checksum, hence, Please use mkimage to pack Linux kernel as below. Note that the Linux kernel should be named 970image.sha。

Tomato U-Boot boot SPI flash and automatically boot Linux kernel from SD card. The Linux kernel should be named 970image.sha and located in the root directory of SD card. In order to speed up boot time, Tomato U-Boot uses hardware checksum, hence, Please use mkimage to pack Linux kernel as below. Note that the Linux kernel should be named 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
```

1.17 U-Boot for Eth2UART

For Eth2UART, there is a Eth2UART configuration that U-Boot boot from SPI flash and automatically boot Linux kernel named “eth2uart.ub” from SD card.

1.17.1 Build Eth2UART U-Boot

Clean all object code.

```
# make distclean
```

Compile U-Boot

```
# make nuc970_eth2uart_defconfig
# make
```

1.17.2 Eth2UART U-Boot link address

The link address of Eth2UART U-Boot is defined in include/configs/nuc970_eth2uart.h.

```
#define CONFIG_SYS_TEXT_BASE        0x2000000
```

CONFIG_SYS_TEXT_BASE defines U-Boot link address.

Eth2UART U-Boot link address is 0x2000000.

1.17.3 Use mkimage tool to pack Linux kernel

Eth2UART U-Boot boot SPI flash and automatically boot Linux kernel from SD card. The Linux kernel should be named eth2uart.ub and located in the root directory of SD card.

```
u-boot/tools# ./mkimage -A arm -O linux  -T kernel -a 0x8000 -e  0x8000
-d 970image eth2uart.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
```

1.18 Notice

U-Boot SPI sub-system uses NUC970/N9H30 SPI0. Its multi-function pins are B6, B7, B8, B9, B10, and B11. Hence, you have to check whether these pins are connected.