ARM926EJ-S™
32-bit Microprocessor

NUC980 Linux BSP User Manual

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

Table of Contents

1 NUC980 Linux BSP Introduction

This BSP supports Nuvoton NUC980 series. The NUC980 series targeted for general purpose 32-bit microprocessor embeds an outstanding CPU core ARM926EJ-S, a RISC processor designed by Advanced RISC Machines Ltd., runs up to 300 MHz, with 16 KB I-cache, 16 KB D-cache and MMU, 16KB embedded SRAM and 16.5 KB IBR (Internal Boot ROM) for booting from USB, NAND , SD/eMMC and SPI Flash.

The NUC980 series is equipped with a large number of high speed digital peripherals, such as two 10/100 Mbps Ethernet MAC supporting RMII, a USB 2.0 high speed host/device and a USB 2.0 high speed host controller, up to six USB 1.1 host lite interfaces, two CMOS sensor interfaces supporting CCIR601 and CCIR656 type sensor, two SD interfaces supporting SD/SDHC/SDIO card, a NAND Flash interface supporting SLC and MLC type NAND Flash, an I^2S interface supporting I^2S and PCM protocol. Also, the NUC980 series offers a built-in hardware cryptography accelerator supporting RSA, ECC, AES, SHA, HMAC and a random number generator (RNG).

The NUC980 series provides up to ten UART interfaces, two ISO-7816-3 interfaces, a Quad-SPI interface, two SPI interfaces, up to four I^2C interfaces, four CAN 2.0B interfaces, eight channels PWM output, 8-channel 12-bit SAR ADC, six 32-bit timers, WDT (Watchdog Timer), WWDT (Window Watchdog Timer), 32.768 kHz XTL and RTC (Real Time Clock). The NUC980 series also supports two 10-channel peripheral DMA (PDMA) for automatic data transfer between memories and peripherals.

The NUC980 Linux BSP includes following contents:

- Linux 4.4 kernel source code and NUC980 device drivers.
- GCC 4.8.4 cross compiler with EABI support.
- uClibc-0.9.33
- Binutils-2.24
- Demo program for device drivers, busybox, mtd-util, and other open source applications.
- U-Boot 2016.11 source code including NUC980 device drivers.
- Flash programming tool Nu-Writer, and its Windows driver.
- User manuals.

1.1 Development Environment

This BSP only provides cross development tool chain in Linux environment. Therefore, Linux platform is a must to build Linux kernel, U-Boot, and applications using the cross compiling tool chain in BSP. This platform could be a dedicate Linux server or running on virtual machine. PC can communicate with NUC980 Dev Board via different communication interfaces, such as UART, USB or Ethernet, as well as debug port, JTAG. Above interfaces could be used to load binary file to EV board for execution. JTAG interface could be used for chip level debug. USB interface is the interface used by NuWriter to program NAND, SPI, and eMMC. Following figure is an example of development environment.

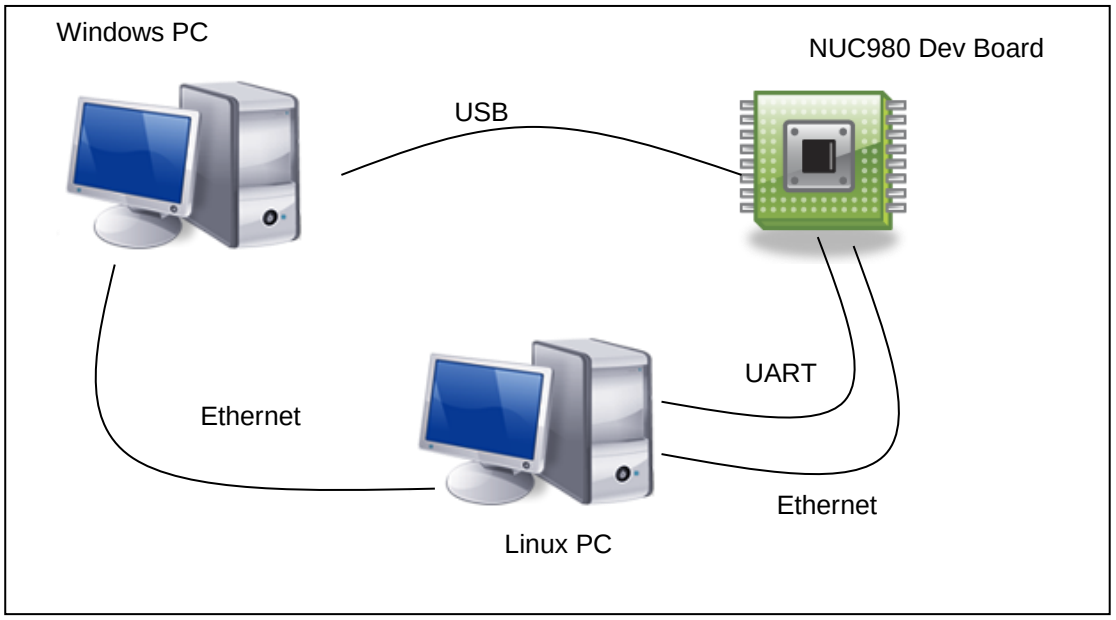

Figure 1-1 Development Environment

1.2 Dev Board Setting

The NUC980 series supports different boot modes, it can boot from SPI, NAND, eMMC/SD, or enter USB ISP mode. The booting mode is selected by PG[1:0] jumper. Because most I/O pins support multiple functions, the jumpers on DEV board must be set according to the enabled peripherals. Please refer to “NUC980 Development Board User Manual” for the usage of DEV board.

2 BSP Installation

2.1 System Requirements

The NUC980 Linux BSP provides cross compilation tools based on Linux operating system. We have tested this BSP in different x86 Linux distributions, including Ubuntu, CentOS, and Debian…etc. Because there are so many distributions out there with different system configuration, sometimes it is necessary to change system setting or manually install some missing component in order to cross compile.

The Linux development environment could either be native, or install in a virtual machine execute on top of other operating system. This chapter introduces how to install CentOS Linux to VMware virtual machine, and the steps to install NUC980Linux BSP.

2.2 Download and Installation VMware Virtual Machine

VMware provides free virtual machine VMware player 5.0.2 for users to download from VMware official website http://www.vmware.com/. Select “Products” → “Free Products” → “Player”, click “Download” button, select “5.0 (latest)” as “Major Version” and “5.0.2 (latest)” as “Minor Version” and then download “VMware Player for Windows 32-bit and 64-bit”. Please refer to the figure below.

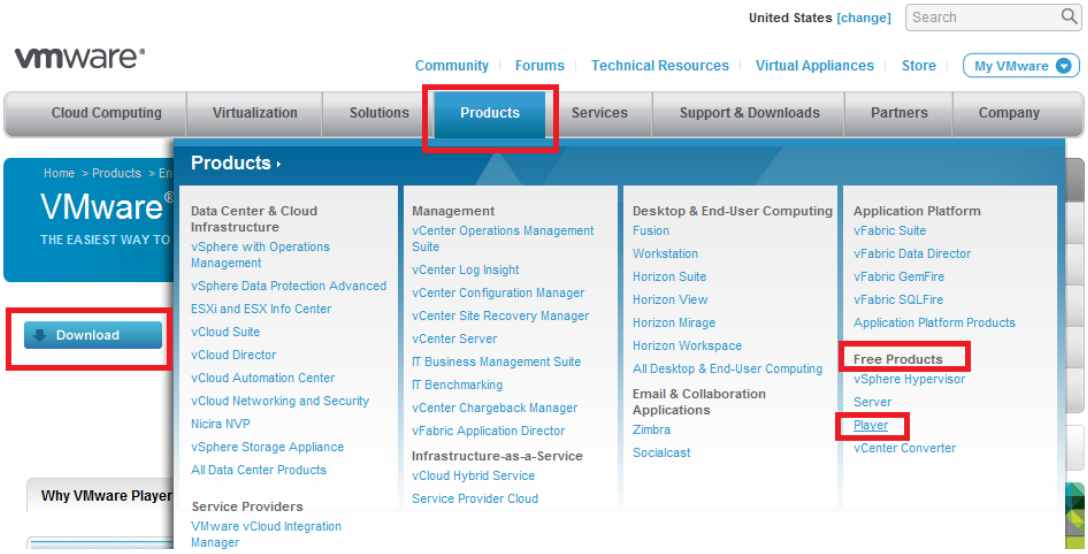

Figure 2-1 Download VMWare Player

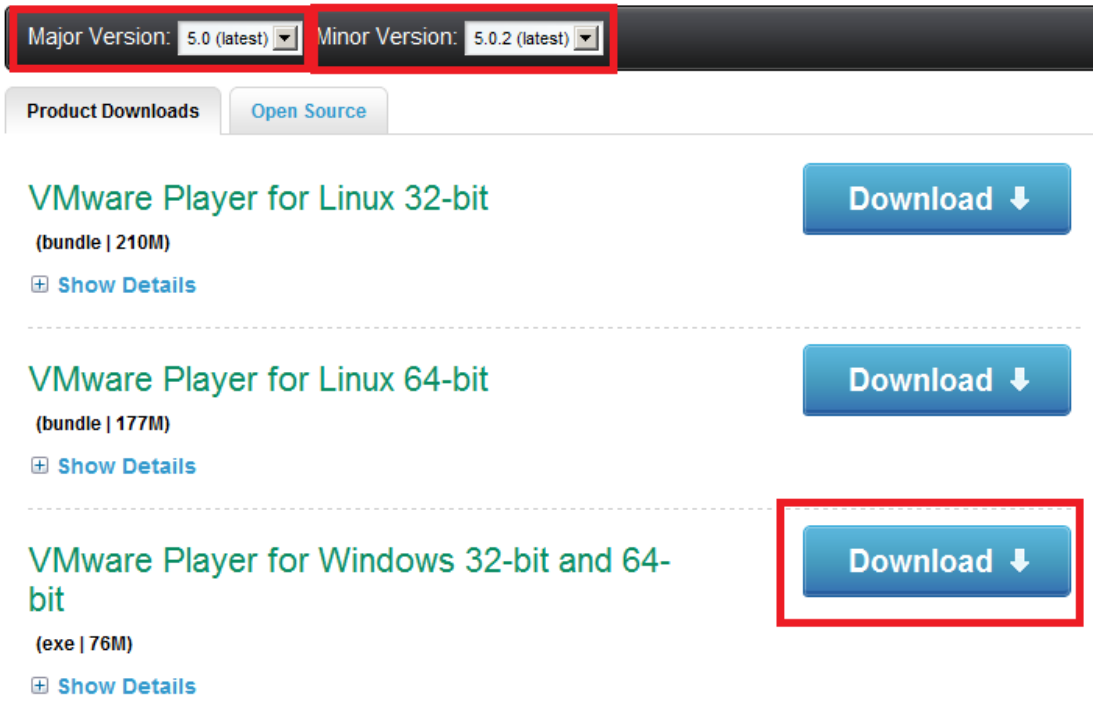

Figure 2-2 Download VMWare Player for Windows

After download is complete, double click the downloaded file.

Figure 2-3 Install VMWare Player (1)

Then click “Next” to continue installation steps.

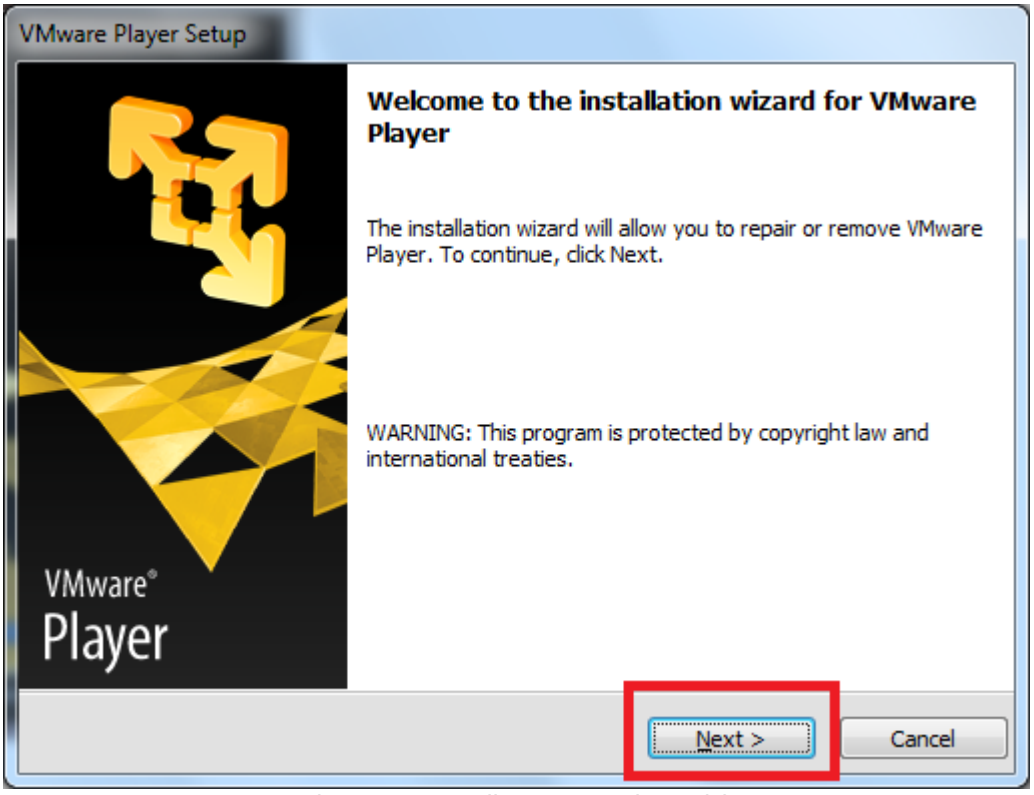

Figure 2-4 Install VMWare Player (2)

At last, double click the installed file to create a virtual machine.

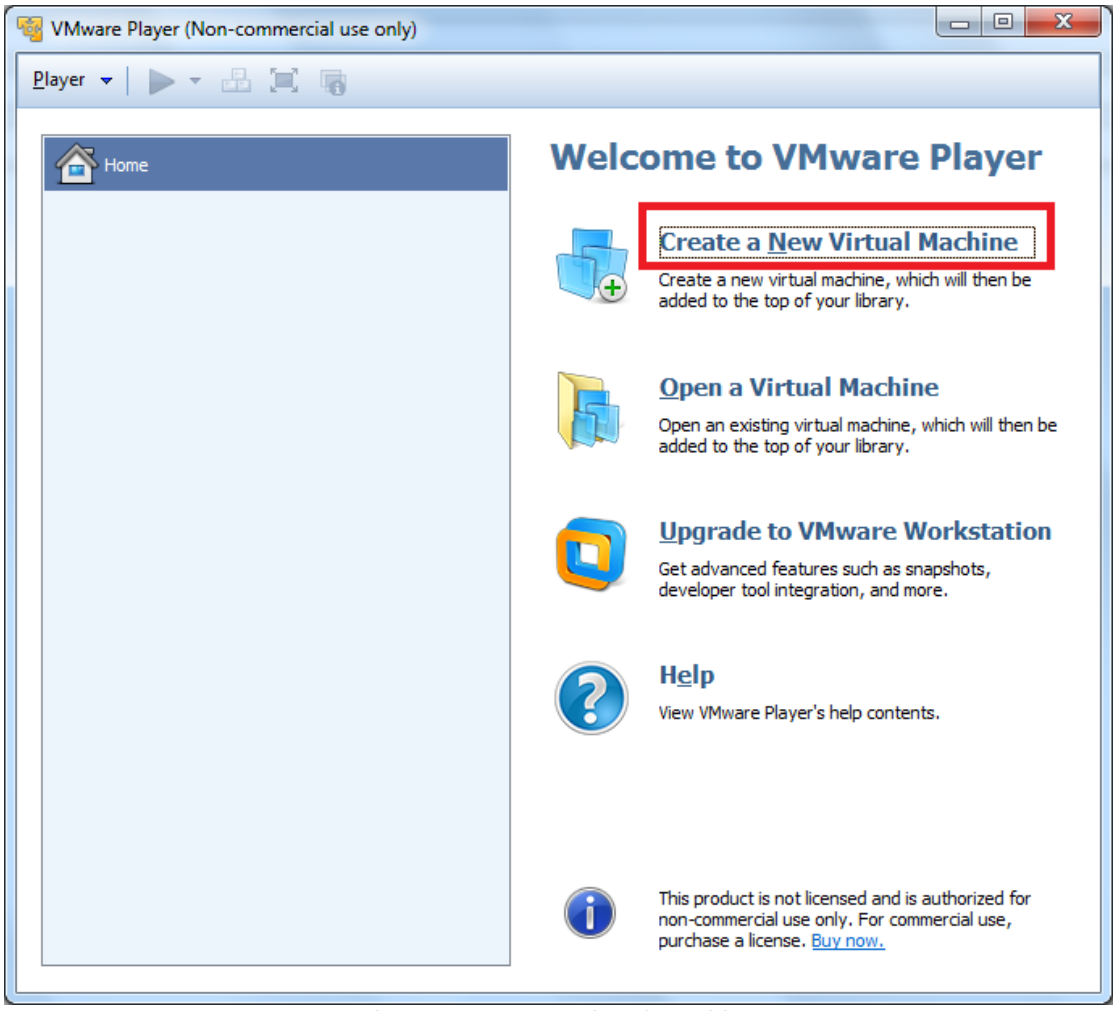

Figure 2-5 Create Virtual Machine

2.3 Download and installation CentOS Linux

The following introduces the procedure to install CentOS 6.4 under VMWare. It is pretty much the same with installing as native operating system. First connect to http://www.centos.org/, and enter the download page by selecting “CentOS 6 Releases” → “i386”. Select “CentOS-6.4-i386-bin-DVD 1.iso” to download CentOS 6.4.

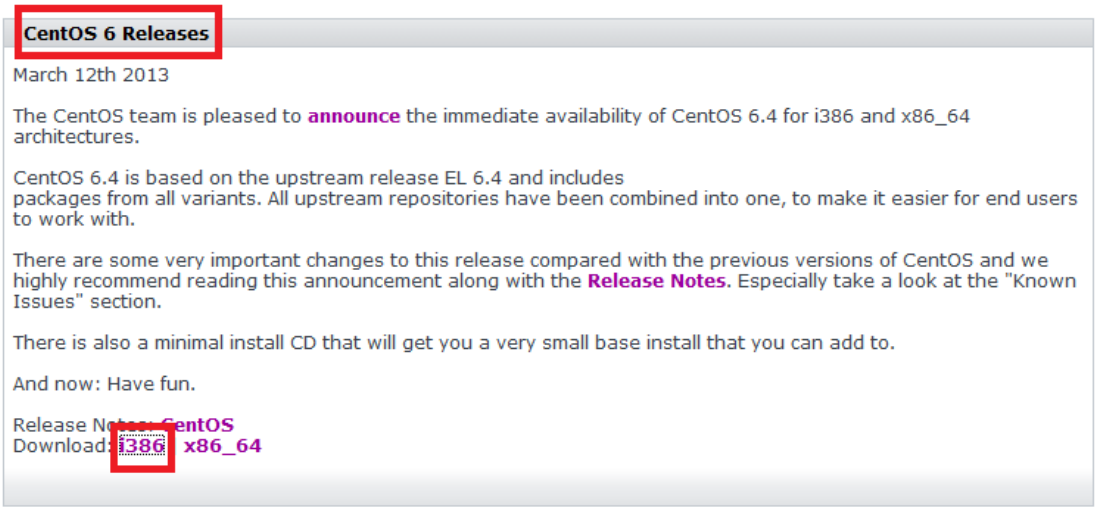

Figure 2-6 Download CentOS ISO Image (1)

Figure 2-7 Download CentOS ISO image (2)

If VMware virtual machine is already installed, click “Create a New Virtual Machine” to install Linux virtual machine, otherwise follow the steps in previous section to complete VMware installation.

First, click “Installer disc image file (iso):” and “Browse…” and select the downloaded CentOS 6.4 iso as image source file.

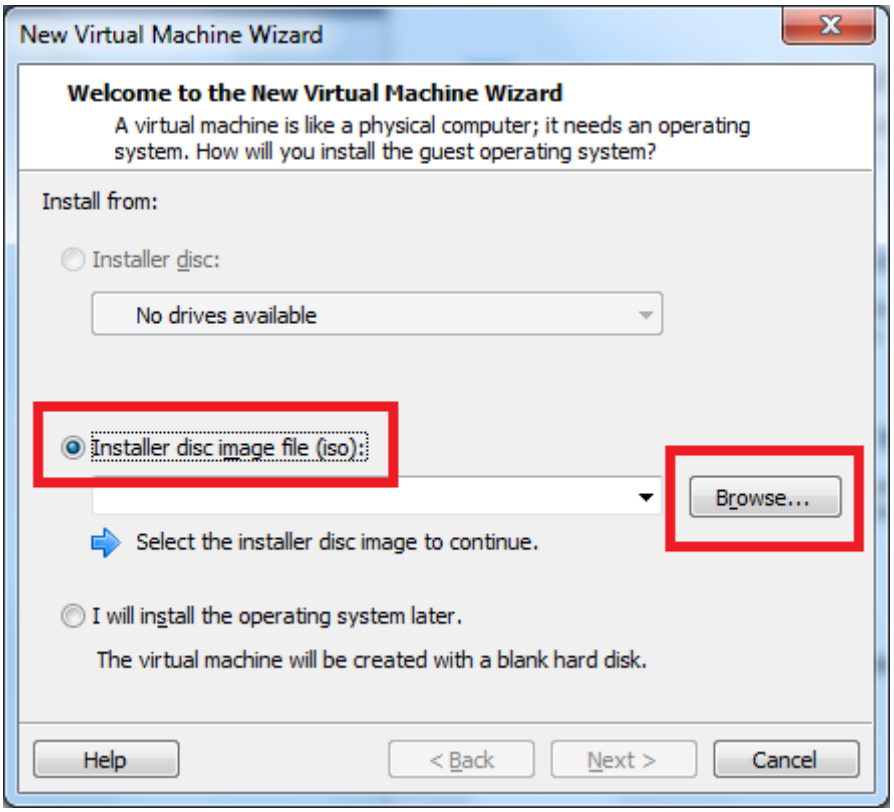

Figure 2-8 Select Linux Image File (1)

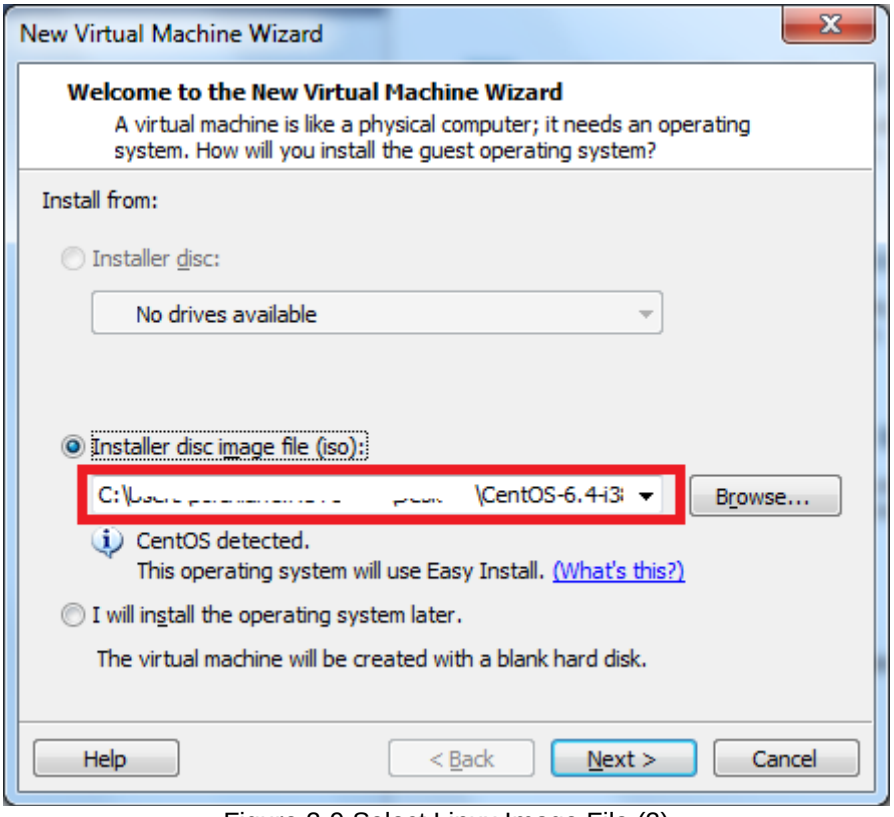

Figure 2-9 Select Linux Image File (2)

Input “Full name:”, “User name:”, “Password:”, and “Confirm:” click Confirm button. Please

remember the username and password, as they will be used to login CentOS later.

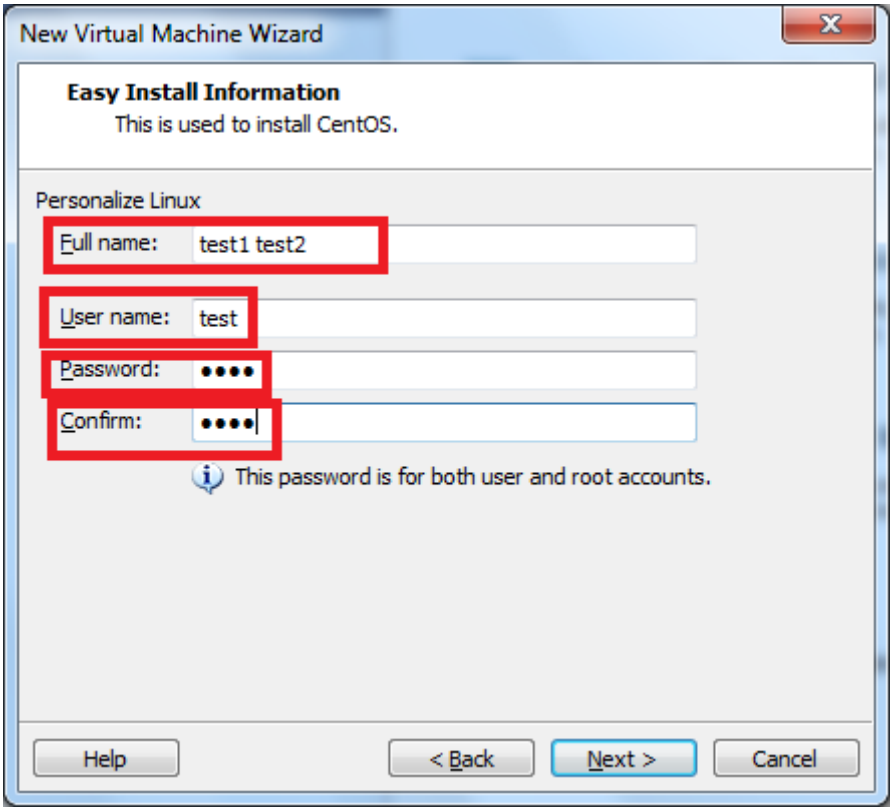

Figure 2-10 Configure Username and Password

Next, input “Virtual machine name:” and “Location:” if necessary, otherwise keep the default value.

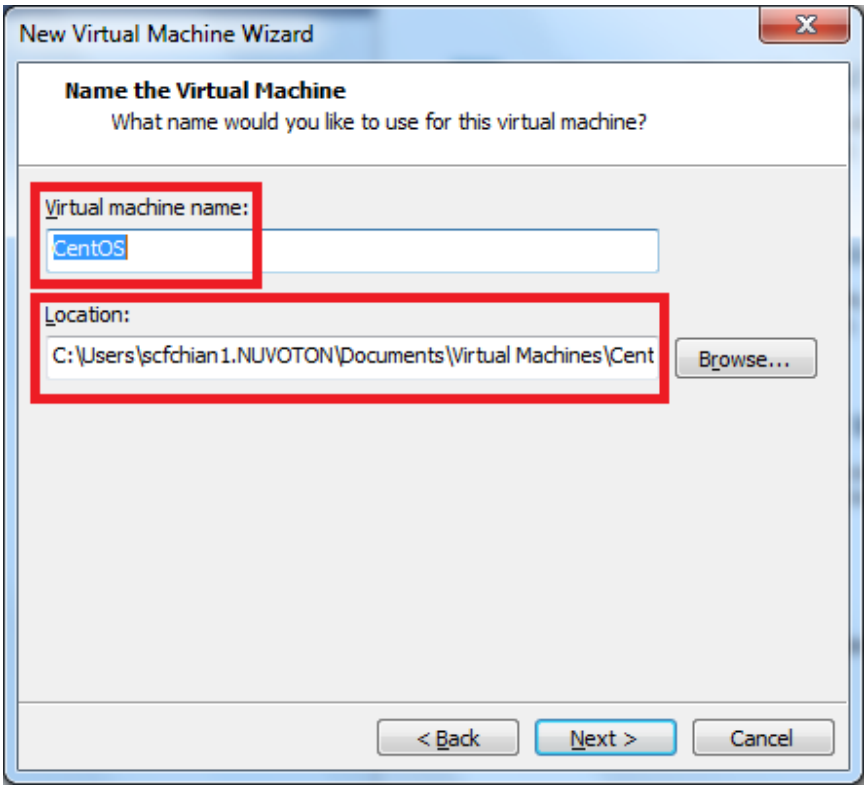

Figure 2-11 Assign Virtual Machine Name and Location

In the next step, input “Maximum disk size (GB):” value, it is recommend to reserve at least 60GB disk space, and select “Store virtual disk as a single file”.

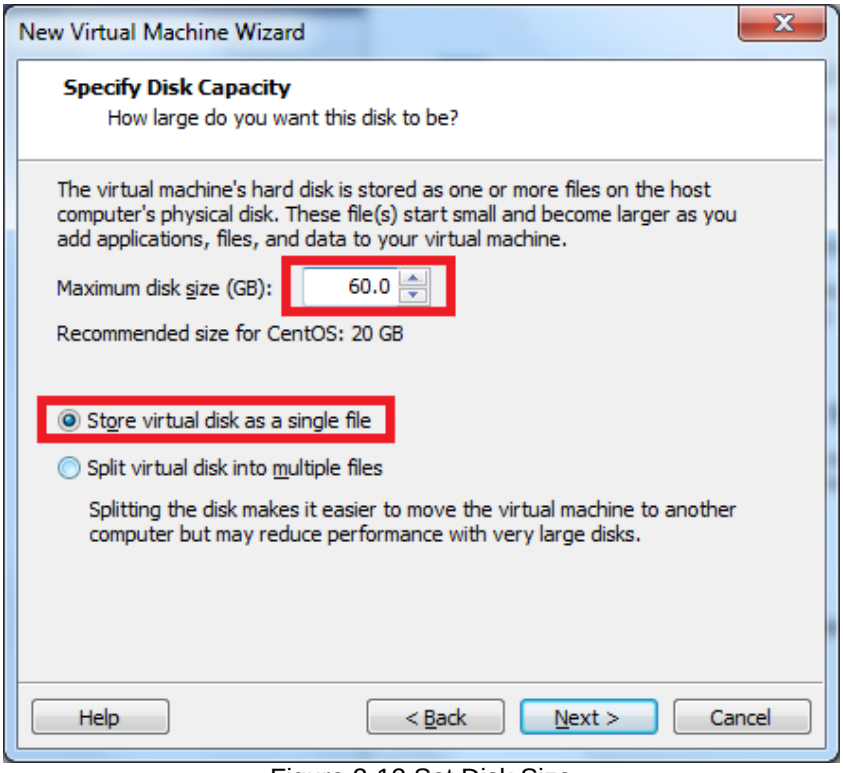

Figure 2-12 Set Disk Size

The last step is to click “Finish” to complete the CentOS configuration.

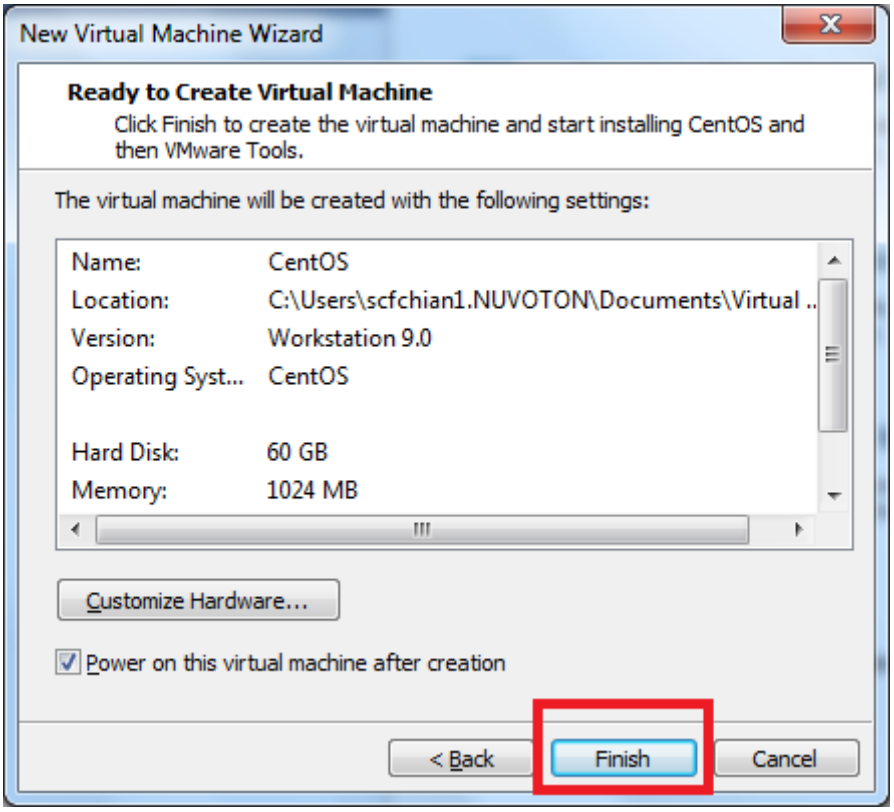

Figure 2-13 Complete Virtual Machine Installation

VMware will automatically complete reset of the CentOS installation procedures.

Figure 2-14 Launch Virtual Machine

A CentOS login window will shows up after installation complete. You can login using the username and password set in previous step.

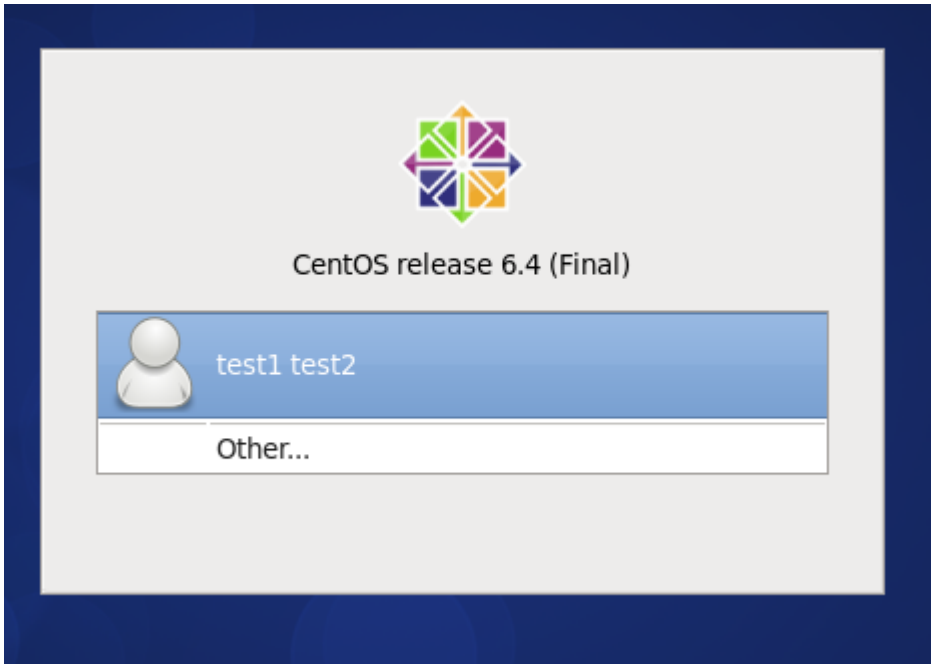

Figure 2-15 Login Linux Virtual Machine

2.4 Install Missing Packages

Each Linux distribution selects different packages to install. Most distributions do not install all packages mandatory for NUC980 Linux BSP. Also lack of some optional packages which might be useful during development. The following lists some packages that may be missing after installed a Linux distribution but are mandatory or recommend, and could be installed later manually.

Package name	Function	Mandatory/ Optional
Patch	Application for apply patch file	Mandatory
libc6-dev	Contains required libraries to link with cross compiling tool chain. (i386 version)	Mandatory
libncurses5-dev	Contains required header files to build menuconfig interface	Mandatory
git-all	Software version control tool	Mandatory
Minicom or cutecom	Serial terminal which could display the bootloader message or Linux console output.	Optional

Each Linux distribution has its own package management system. Ubuntu users could use apt-get command or Synaptic Package Manager to install packages. Fedora users could use rpm command or Package Manager to install packages. Please refer to the manual of the Linux distribution you use to install missing components.

2.5 BSP Installation Procedures

Linux BSP contains three directories. Content of each directory listed in following table:

Directory name	Content
BSP	A tar ball contains cross compiler, root file system, and pre-build tool chain.
Documents	BSP related documents.
Tools	NuWriter tool and its driver for Windows and SD writer tool.

Source code could be download using repo tool. Below list the steps of doing so.
Make sure you have a bin/ directory in your home directory and that is included in your path.

```
$ mkdir ~/bin
$ export PATH=~/bin:$PATH
```

Download the repo tool and ensure it is executable.

```
$ curl https://storage.googleapis.com/git-repo-downloads/repo > ~/bin/repo
$ chmod a+x ~/bin/repo
```

Create an empty directory to hold working directory.

```
$ mkdir WORKING_DIR
$ cd WORKING_DIR
```

Configure git with your real name and email address.

```
$ git config --global user.name "Your Name"
$ git config --global user.email "you@example.com"
```

Use one of following commands to download manifest file for NUC980 BSP. The first command download from Github, and the second command download from Gitee. Users can use the command to select the site with faster download speed.

```
$ repo init -u git://github.com/OpenNuvoton/manifest.git -b nuc980-2020.09
-m github.xml
```

Or

```
$ repo init -u https://gitee.com/OpenNuvoton/manifest.git -b nuc980-2020.09
-m gitee.xml
```

Then download source code.

```
$ repo sync
```

After downloading the source code, please copy the tar ball under BSP directory to Linux machine and use following command to extract the file.

```
$ tar zxvf nuc980bsp.tar.gz
```

After entering nuc980bsp directory, execute the installation script install.sh to install tool chain, rootfs, and pre-build image. This script requires the administrator privilege to execute. You can use “su” command to switch to root and execute the installation script.

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

Or execute this script as root by using sudo command. (This method works for those distributions do not open the root account privilege, such as Ubuntu)

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

Below is the console output during installation, the path input should be the same as the WORKING_DIR set previously.

```
Now installing arm_linux_4.8 tool chain to /usr/local/
Setting tool chain environment
Installing arm_linux_4.8 tool chain successfully
Install rootfs, applications, u-boot and Linux kernel
Please enter absolute path for installing(eg:/home/<user name>) :
BSP will be installed in /<path you input>/nuc980bsp
/home/someone
Extract rootfs and pre-build images
...
...
NUC980 BSP installion complete
```

If your Linux server has already installed the arm_linux_4.8 tools, the installation script will ask whether to overwrite existing tool chain. Otherwise, the script will install the tool chain into /usr/local/arm_linux_4.8 without asking. For the first case, if you want to update the tool chain, you can select Y(or yes、y、YES), then hit Enter key.

After install the tool chain, the installation script will ask for the absolute path for install kernel and applications. The table below listed the item will be installed in the specified location

Directory name	Content
applications	Demo applications and other open source applications, such as busybox, wireless tool…
buildroot	A set of Makefiles and patches that simplifies and automates the process of building a complete and bootable Linux environment for an embedded system
image/kernel	Pre build kernel using default configuration
image/U-Boot	Pre build U-Boot images support either NAND or SPI Flash and env.txt file which stores U-Boot's environment variable. The default execution address of U-Boot images is 0xE00000.
linux-4.4.y	Linux kernel source code
rootfs	Root file system
u-boot-2016.11	U-Boot V2016.11 source code
nuwriter	NuWriter command line tool for Linux

The installation script will try to configured the installed directory with correct owner and group, and add the path of compiler into $PATH. However, this does not work correctly in every Linux distribution. User might need to set the owner/group of installed directory with correct user's name, and add /usr/local/arm_linux_4.8/bin to $PATH manually.

Note: Please logout and re-login after installation complete to make the changes take effect.

2.6 Clone from Git Server

All source code could be clone from public git repository. After clone from remote repository, users can sync with latest source code from remote repository with git pull command. Following table list the link to the repositpories of each software package. The Git command usage is beyond the scope of this document, pleae refer to the manuals in https://git-scm.com/ for the Git command usage.

Software Package	Link to repository
Applications	https://github.com/OpenNuvoton/NUC980_Linux_Applications.git

	https://gitlab.com/OpenNuvoton/NuMicro-ARM7-ARM9-Family/NUC980_Linux_Applications.git https://gitee.com/OpenNuvoton/NUC980_Linux_Applications.git
buildroot	https://github.com/OpenNuvoton/NUC980_Buildroot.git https://gitlab.com/OpenNuvoton/NuMicro-ARM7-ARM9-Family/NUC980_Buildroot.git https://gitee.com/OpenNuvoton/NUC980_Buildroot.git
linux-4.4.y	https://github.com/OpenNuvoton/NUC980-linux-4.4.y.git https://gitlab.com/OpenNuvoton/NuMicro-ARM7-ARM9-Family/NUC980-linux-4.4.y.git https://gitee.com/OpenNuvoton/NUC980-linux-4.4.y.git
uboot.v2016.11	https://github.com/OpenNuvoton/NUC970_U-Boot_v2016.11.git https://gitlab.com/OpenNuvoton/NuMicro-ARM7-ARM9-Family/NUC970_U-Boot_v2016.11.git https://gitee.com/OpenNuvoton/NUC970_U-Boot_v2016.11.git
NuWriter	https://github.com/OpenNuvoton/NUC980_NuWriter.git https://gitlab.com/OpenNuvoton/NuMicro-ARM7-ARM9-Family/NUC980_NuWriter.git https://gitee.com/OpenNuvoton/NUC980_NuWriter.git
Linux Command Line NuWriter	https://github.com/OpenNuvoton/NUC980_NuWriter_CMD.git https://gitlab.com/OpenNuvoton/NuMicro-ARM7-ARM9-Family/NUC980_NuWriter_CMD.git https://gitee.com/OpenNuvoton/NUC980_NuWriter_CMD.git
SD Writer	https://github.com/OpenNuvoton/NUC980_SDWriter.git https://gitlab.com/OpenNuvoton/NuMicro-ARM7-ARM9-Family/NUC980_SDWriter.git https://gitee.com/OpenNuvoton/NUC980_SDWriter.git

3 Linux Kernel

3.1 Configuration Interface for the Kernel

Linux supports different kinds of configuration. Users can disable some unnecessary functions to save resource of kernel system.
To enter the page of Linux configuration, please type “make menuconfig” command in shell.
It’s multi-layer menu in configuration system. In the current page, user can press “up”, “down”, ”left”, ”right” four keys to control the layer of configuration system. Select kernel function by pressing “up” or “down” key and select menu function in the bottom of page by pressing “left” or “right” key. To enter the next layer of configuration page, user can press “enter” key.
There are five functions at the bottom of menu page. User can disable or enable kernel function by pressing space key when cursor stays at “Select”. The symbol in front of the selection function “[ ]” stands for this function is disabled, “[*]” stands for this function is enabled and “[M]” stands for this function is built as module and can be loaded dynamically.
Menu page can return to upper layer by pressing space key when cursor stays at “Exit” at the bottom of menu page. If it’s at the top layer of configuration system, system will inform user if wants to save the configuration and exit.
The help screen will show when cursor is at “Help” by pressing space key. To save current configuration or load old configuration, use can press space key when cursor is at “Save” or “Load” at the bottom of menu page.
The kernel configuration file will be named “.config” and saved in the linux-3.10.x directory.

3.2 Default Configuration

There is a default configuration for the NUC980 series chips provided by Nuvoton. Before modifying any configuration of kernel, we recommend to load the default configuration of kernel first. User can type “make nuc980_defconfig” command to do that. Sometimes if system cannot boot up, user can load the default configuration to recovery kernel to safe status.

3.3 Linux Kernel Configuration

This section introduces the configuration to enable kernel function according to different NUC980 driver or functions.

3.3.1 Basic Configuration of System

- Mount the module

Some drivers only support dynamic load, for example “USB Wi-Fi driver” or “USB device driver” … and so on. Please enable the following function to support that. When system is booted up at shell, user can use “insmod <module name>” to load module.

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

- Remove module

If some module drivers need to be removed by system, please enable the following function to support module removing. To remove module, user can use “rmmod <module name>” command to do that.

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

- Boot Options – root file system is based on RAM

Boot option can configure system, including the type of root file system, the size of memory, baud-rate of uart console… and so on. The following example is a simple configuration and there are many commands can be supported by kernel. User can refer to the document which is at Documentation/kernel-parameters.txt.

```
Boot options  --->
```

```
     (root=/dev/ram0 console=ttyS0, 115200n8 rdinit=/sbin/init mem=64M)
Default kernel command string
          Kernel command line type (Use bootloader arguments if available)
--->
```

- Boot Options – root file system is based on YAFFS2 (NAND Flash)

If root file system is at NAND Flash and use YAFFS2 file system, user needs to enable YAFFS2 file system (please refer to 3.3.4) and disable RAM file system function.

```
General setup  --->
     [ ] Initial RAM filesystem and RAM disk (initramfs/initrd) support
```

The following is an example to boot up YAFFS2 root file system. User must set the U-Boot environment argument first. Then a YAFFS2 root file system image needs to be done first and write it to the mtdblock2 in Linux system.

```
Boot options  --->
     (noinitrd root=/dev/mtdblock2 rootfstype=yaffs2 rootflags=inband-tags
console=ttyS0, 115200n8 rdinit=/sbin/init mem=64M) Default kernel command
string
          Kernel command line type (Use bootloader arguments if available)
--->
```

If boot up YAFFS2 root file system directly without uboot emvironment arguments. The kernel setting is as follow.

```
Boot options  --->
     (noinitrd root=/dev/mtdblock2 rootfstype=yaffs2 rootflags=inband-tags
console=ttyS0, 115200n8 rdinit=/sbin/init mem=64M
mtdparts=nand0:0x200000@0x0(u-boot),0x1400000@0x200000(kernel),-(user)
ignore_loglevel) Default kernel command string
          Kernel command line type (Use bootloader arguments if available)
--->
```

- Boot Options – root file system is based on JFFS2 (SPI Flash)

If root file system is at SPI Flash and use JFFS2 file system, user needs to enable JFFS2 file system (please refer to 3.3.4) and disable RAM file system function.

```
General setup  --->
     [ ] Initial RAM filesystem and RAM disk (initramfs/initrd) support
```

The following is an example to boot up JFFS2 root file system. A JFFS2 root file system image needs to be done first by mkfs.jffs2 utility and write it to the mtd1 in Linux system

```
Boot options  --->
     (root=/dev/mtdblock1 rw rootfstype=jffs2 console=ttyS0,115200n8
rdinit=/sbin/init mem=64M) Default kernel command string
          Kernel command line type (Use bootloader arguments if available)
--->
```

- Boot Options – root file system is based on UBIFS (NAND Flash)

If root file system is at NAND Flash and use UBIFS file system, user needs to enable UBIFS file system (please refer to 3.3.4) and disable RAM file system function.

```
General setup  --->
     [ ] Initial RAM filesystem and RAM disk (initramfs/initrd) support
```

The following is an example to boot up UBIFS root file system. A UBIFS root file system image needs to be done first and write it to the mtd2 in Linux system

```
Boot options  --->
     (noinitrd ubi.mtd=2 root=ubi0:system rw rootfstype=ubifs
console=ttyS0, 115200n8 rdinit=/sbin/init mem=64M) Default kernel command
string
          Kernel command line type (Use bootloader arguments if available)
--->
```

- Boot Options – root file system is based NFS (Network File System)

At the development stage of Linux application, user oftern wants to modify testing application. This method can reduce some development time by mounting NFS rootfs.

```
Boot options  --->
     (noinitrd root=/dev/nfs nfsroot=x.x.x.x:/path_to_nfs_rootfs
ip=y.y.y.y:z.z.z.z:g.g.g.g:m.m.m.m console=ttyS0,115200n8 rdinit=/sbin/init
mem=64M) Default kernel command string
```

x.x.x.x and z.z.z.z is the server ip, y.y.y.y is the client ip, g.g.g.g is the gateway ip and m.m.m.m is the net mask.

And user needs to enable network function (please refer to 3.3.2) and the following item additionally.

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

Of course, NFS function must be enabled.

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

- Boot Options – Support device tree

To enable device tree support, please configure kernel as below.

```
[*] Flattened Device Tree support
[*]   Support for the traditional ATAGS boot data passing
```

3.3.2 Network

- TCP/IP

To enable basic network functions, please enable the following configurations.

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

- Wi-Fi Wireless

If wireless device is used, user needs to enable the following functions additionally.

```
[*] Networking support  --->
     [*]   Wireless  --->
          <*>  Nuvoton external WiFi driver support
          <*>   cfg80211 - wireless configuration API
          [*]     cfg80211 wireless extensions compatibility
Device Drivers  --->
     [*] Network device support  --->
          [*]   Network core driver support
          [*]   Wireless LAN  --->
```

3.3.3 Drivers

- Audio Interface

The following is I²S interface configuration:

```
Device Drivers  --->
     <*> Sound card support  --->
          <*>   Advanced Linux Sound Architecture  --->
               <*>   ALSA for SoC audio support  --->
                    <*>   SoC Audio for NUC980 series
                    <*>     NUC980 I2S support for demo board
                              I2S Mode Selection (Master Mode)  --->
```

The I²S supports master and slave mode. User can decide which mode is used by selecting in the configuration menu.
If the I²S function is enabled, NAU8822 codec driver is also enabled automatically. To use I²S with audio codec function well, user needs to enable I²C function at the same time.
If audio application is wrote by old OSS architecture, user can enable the following two functions to do that. User can refer to the example in the BSP which source code is at BSP/applications/demos/alsa_audio.

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

- Cryptographic Accelerator

To support Cryptographic Accelerator function, user needs to enable PF_KEY sockets function support in Networking support menu page.

```
[*]  Networking support  --->
     Networking options --->
          <*> PF_KEY sockets
```

Then enable Cryptographic API related functions.

```
Cryptographic API  --->
     <*>  Userspace cryptographic algorithm configuration
```

```
    [*]  Disable run-time self ests
    <*>  Software async crypto daemon
    <*>  User-space interface for hash algorithm
    <*>  User-space interface for symmetric key cipher algorithms
    [*]  Hardware crypto devices --->
         <*> Support for NUC980 Cryptographic Accelerator
```

The NUC980 Cryptographic Accelerator function supports AES, DES and 3-DES crypto algorithm. It also supports SHA and HMAC hash algorithm. User can refer to the example named crypto (source is at BSP/applications/demos/crypto directory)

The NUC980 Cryptographic Accelerator function supports a Pseudo Random Number Generator (PRNG). To enable NUC980 H/W PRNG, above NUC980 crypto device driver must be enabled first. Then the following H/W RNG device option will present for selection.

```
Device Drivers  --->
    Character Devices --->
         <*> Hardware Random Number Generator Core support
              <*> Nuvoton HW Random Number Generator support
```

- DMA

The DMA function is supported by the NUC980 series. To support it in kernel, user needs to enable “NUC980 DMA support” in “DMA Engine support” menu page.
User can learn DMA functions in kernel by referring to source which is at linux-3.10.x/drivers/dma/dmatest.c. A test client will be also enabled by enabling “DMA Test Client” function, it’s a shortcut to understand the procedure of DMA in kernel.

```
Device Drivers  --->
    [*]DMA Engine support  --->
         <*>   NUC980 DMA support
         <*>   DMA Test client
```

- User space memory management

If user wants to get a physical and virtual address of memory by enabling this user space memory management function.

```
Device Drivers  --->
 Character devices  --->
      [*] Support for /dev/nuc980-mem
```

- Ethernet

The NUC980 series supports two Ethernet ports. They can be enabled simulataniously. To support network port, PHY driver also needs to be enabled additionally.
The PHY chip on the development board is provided by ICPlus, the configuration will need to be modified if different PHY is used.
The NUC980 Ethernet ports support Wake on Lan (WoL) feature using Magic Packet by configure with ethtool tool. The Magic Packet complies with the format defined in AMD’s Magic Packet Technology white paper. For the usage of ethtool, please refer to section 4.1.

```
Device Drivers  --->
    [*] Network device support  --->
         <*>Dummy net driver support
         [*]   Ethernet driver support  --->
```

```
                <*>     Nuvoton NUC980 Ethernet MAC 0
                <*>     Nuvoton NUC980 Ethernet MAC 1
          -*-   PHY Device support and infrastructure  --->
                <*>   Drivers for ICPlus PHYs
```

- EBI

The EBI function is supported by the NUC980 series. To support it in kernel, user needs to enable “NUC980 EBI driver” in “Misc devices” menu page..

```
Device Drivers  --->
     Misc devices  --->
          <*> NUC980 EBI driver
```

- Etimer

When Linux kernel runs, it uses basic timer function of NUC980 to be timer. The NUC980 also supports four enhanced timers which can output 50% duty cycle or capture function. Four channel of Etimer can be controlled individually.
The following is an example which Etimer channel 0 and channel 1 are as output by general purpose pin PC.6 and PC.8. And channel 2 and channel 3 are as capture pins by PC.11 and PC.13.
If the channel is unused, select the function to “No output” or “No input”.
Enable “NUC980 Enhance Timer (ETIMER) wake-up support” to support Etimer wake-up function.

```
Device Drivers  --->
     Misc devices  --->
          <*> NUC980 Enhance Timer (ETIMER) support
          <*> NUC980 Enhance Timer (ETIMER) wake-up support
               NUC980 ETIMER channel 0 toggle output pin (Output to PC6)  -
--->
               NUC980 ETIMER channel 0 capture input pin (No input)  --->
               NUC980 ETIMER channel 1 toggle output pin (Output to PC8)  -
--->
               NUC980 ETIMER channel 1 capture input pin (No input)  --->
               NUC980 ETIMER channel 2 toggle output pin (No output)  --->
               NUC980 ETIMER channel 2 capture input pin (Input from PC11)
--->
               NUC980 ETIMER channel 3 toggle output pin (No output)  --->
               NUC980 ETIMER channel 3 capture input pin (Input from PC13)
--->
```

Application can control etimer function by ioctl() function. The driver supports etimer wake-up function, periodic, toggle out, tiger counting mode and free counting mode functions now.
The wake-up function use etimer to wake up system from Power-down mode periodically. The value captured by ETIMER at capture mode (trigger counting mode) can be read back by using read() function. The unit of value is us, it stands for time interval between two triggers. Free counting mode can measure the external pin input frequency, the unit of value is Hz. User can refer to example code in the BSP (source code path is at BSP/applications/demos/etimer) to develop the related application.

- Smartcard

The NUC980 series has two smartcard interfaces that comply with ISO-7816 and EMV 2000

specification. If the system needs to access smartcard, please refer to the kernel configuration below to enable smartcard driver. This driver supports both T = 0 and T = 1 protocols. The card detection level and power-on level, which is depend on the on board circuit and card slot design can be configured individually. Ether Port A or Port C can be selected for smartcard interface 0, and Port C or Port F can be selected for smartcard interface 1. When enable Perform EMV check checkbox, the driver will perform a more strict protocol check comply with EMV 2000, so some smartcards will be reported as faliuare cards.

```
Device Drivers  --->
     Misc devices  --->
          <*> NUC980 Smartcard Interface support
          [ ]   Perform EMV check
          [*]   NUC980 SC0 support
                  NUC980 SC0 pin selection (Use port A)  --->
                  NUC980 SC0 CD pin config (CD high as card insert)  --->
          [ ]     Inverse SC0 power pin level
          [*]   NUC980 SC1 support
                  NUC980 SC1 pin selection (Use port C)  --->
                  NUC980 SC1 CD pin config (CD high as card insert)  --->
          [ ]     Inverse SC1 power pin level
```

User applications can control smartcard using ioctl() function call. Below listed the commands support by smartcard driver and their purpose. User can refer to example code in the BSP (source code path is at BSP/applications/demos/sc) for the usage of these commands.
SC_IOC_GETSTATUS: Check slot staus, for example card inserted or removed..
SC_IOC_ACTIVATE: Activate smartcard, report ATR length if success.
SC_IOC_READATR: Read the ATR (Answer to reset) of activated smartcard.
SC_IOC_DEACTIVATE: Deactivate smartcard.
SC_IOC_TRANSACT: Send ADPU command and read response through sc_transact structure.
Please note that before enter power-down mode, all inserted cards will be deactivate. User application needs to activate the cards in order to do transaction.

- GPIO

To support GPIO function controlled by kernel, please enable “NUC980 GPIO support” and “/sys/class/gpio/…”function.

```
Device Drivers  --->
     [*] GPIO Support  --->
          [*]   /sys/class/gpio/... (sysfs interface)
               <*>   NUC980 GPIO support
               <*>     NUC980 external I/O wake-up support
```

The number of each GPIO pin will be described below.
Driver will keep 32 numbers for each group of GPIO from port A to port J. So the number for the GPIOA will be 0x000~0x01F, GPIOB will be 0x020~0x03F, GPIOC will be 0x040~0x05F, GPIOD will be 0x060~0x07F, GPIOE will be 0x080~0x09F, GPIOF will be 0x0A0~0x0BF, GPIOG will be 0x0C0~0x0DF, GPIOH will be 0x0E0~0x0FF, GPIOI will be 0x100~0x11F and GPIOJ will be 0x120~0x13F.
Application can control each GPIO port by using sysfs. The following is the description of GPIO action based on sysfs interface.

- /sys/class/gpio/export: which GPIO pin will be exported

- /sys/class/gpio/unexport: which GPIO pin will be un-exported
- /sys/class/gpio/gpio0/direction : set GPIOA0 direction to in or output
- /sys/class/gpio/gpio0/value : set or read the value to/from GPIOA0

The following is an example to let GPIOA0 output high:

```
$ echo 0 > /sys/class/gpio/export
$ echo out >/sys/class/gpio/gpio0/direction
$ echo 1 >/sys/class/gpio/gpio0/value
```

User also can refer to the example which source code is at BSP/applications/demos/gpio.
The driver can also control GPIO pin by the following steps.

- Add #inlcude <linux/gpio.h> in the target driver.
- Decide which GPIO pin will be use according to the definition in the arch\arm\mach-nuc980\inlcude\mach\gpio.h.

Take NUC980_PC7 GPIO pin as example.

Set to input mode	gpio_direction_input(NUC980_PC7);
Set to output mode and value	gpio_direction_output(NUC980_PC7,1);
Set to output high	gpio_set_value(NUC980_PC7, 1);
Set to output low	gpio_set_value(NUC980_PC7, 0);
Read the value	gpio_get_value(NUC980_PC7);
Check if GPIO is in use	gpio_request(NUC980_PC7, "NUC980_PC7");
Get the GPIO interrupt number:	gpio_to_irq(NUC980_PC7);

Example:

```
static irqreturn_t PC7IntHandler(int irq, void *dev_id)
{
printk(KERN_INFO "PC7IntHandler:irq=%d \n",irq);
    return IRQ_HANDLED;
}
int xxx_init(void)
{
int ret,irqno;
ret = gpio_request(NUC980_PC7, "NUC980_PC7");
if (ret) printk("NUC980_PC7 failed ret=%d\n",ret);
irqno=gpio_to_irq(NUC980_PC7);
request_irq(irqno,  PC7IntHandler,
                 IRQF_TRIGGER_RISING | IRQF_TRIGGER_FALLING,
                "NUC980_PC7",
                 NULL);
}
```

- Use GPIO to wakeup system

User can use GPIO to wakeup system. Please enable "NUC980 external I/O wake-up support" to do that. User can modify EINT0, EINT1, EINT2, EINT3, EINT4, EINT5, EINT6 and EINT7 on "linux-3.10.x/drivers/gpio/gpio-nuc980.c". Find discription of EINT0 as below

```
/*
 * @brief       External Interrupt 0 Handler
 * @details     This function will be used by EINT0,
```

```
 *              when enable IRQ_EXT0_H0 or IRQ_EXT0_F11 in eint0
 */
/*
static irqreturn_t nuc980_eint0_interrupt(int irq, void *dev_id){
        printk("@0\n");
        return IRQ_HANDLED;
}
*/

/* If enable IRQ_EXT0_H0 or IRQ_EXT0_F11 , linux will enable EINT0
 * User can modify trigger tiypes as below :
 * IRQF_TRIGGER_FALLING / IRQF_TRIGGER_RISING / IRQF_TRIGGER_HIGH /
IRQF_TRIGGER_LOW
 */
struct nuc980_eint_pins eint0[]={
//{IRQ_EXT0_H0, nuc980_eint0_interrupt,IRQF_TRIGGER_FALLING |
IRQF_TRIGGER_RISING,"eint0"},
//{IRQ_EXT0_F11,nuc980_eint0_interrupt,IRQF_TRIGGER_FALLING |
IRQF_TRIGGER_RISING,"eint0"},
{0,0,0,0}
};
```

User can enable nuc980_eintX_interrupt, X=0~7, and set structure of strucnuc980_eint_pins of eintX, X=0~7.
Example, if you want to use PH0 of EINT0 to wakeup and set trigger of PH0 to rising and falling trigger. You can set as follows.

```
static irqreturn_t nuc980_eint0_interrupt(int irq, void *dev_id){
        printk("@0\n");
        return IRQ_HANDLED;
}
struct nuc980_eint_pins eint0[]={
{IRQ_EXT0_H0, nuc980_eint0_interrupt,IRQF_TRIGGER_FALLING |
IRQF_TRIGGER_RISING,"eint0"},
{0,0,0,0}
};
```

- Use GPIO to simulate I²C interface

User can use GPIO to simulate I²C function. Please enable the following function to do that.

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

User can select I²C pin by modifying i2c_gpio_adapter_data structure in arch/arm/mach-nuc980/dev.c. For example, .sda_pin = NUC980_PG1, .scl_pin = NUC980_PG0 will use PG0 as SCL pin and PG1 will be SDA pin.

- I²C

The configuration of I²C is listed as follows:

```
Device Drivers  --->
     <*> I2C support  --->
          I2C Hardware Bus support  --->
               <*> NUC980 I2C Driver for Port 0
               NUC980 I2C0 pin selection (SDA:PA_0  SCL:PA_1)  --->
               <*> NUC980 I2C Driver for Port 1
               NUC980 I2C1 pin selection (SDA:PA_13  SCL:PA_14)  --->
               <*> NUC980 I2C Driver for Port 2
               NUC980 I2C2 pin selection (SDA:PB_7  SCL:PB_5)  --->
               <*> NUC980 I2C Driver for Port 3
               NUC980 I2C3 pin selection (SDA:PB_1  SCL:PB_3)  --->
```

There are many groups can be select for I²C port 1, like GPIO port-A, port-B…
If the I²C function support is selected in kernel configuration, kernel will use build in I²C interface of NUC980 to communicate with other device.
The BSP builds in five I²C port 0 client devices. There i2 NAU8822 by default. User can modify those devices to I²C port1/2/3 by modifying nuc980_i2c_client0 structure in arch/arm/mach-nuc980/dev.c.

User can enable slave eeprom mode. Please enable the following function to do that.

```
Device Drivers  --->
        <*> I2C support  --->
            [*] I2C slave support
             <*> I2C eeprom slave driver
```

- MTD NAND Fash

To enable NAND Flash function, user needs to enable the following function in kernel configuration.

```
Device Drivers  --->
     Generic Driver Options  --->
          <*> Nuvoton NUC980 FMI function selection
               Select FMI device to support (Support MTD NAND Flash)  --->
     -*- Memory Technology Device (MTD) support  --->
          <*>   Command line partition table parsing
          <*>   Caching block device access to MTD devices
          -*- NAND Device Support  --->
               -*-   Nuvoton NUC980 MTD NAND
```

It's necessary to enable "Command line partition table parsing" function when the basic configuration of Flash driver is passed from U-Boot.
The default configuration of Flash driver will partition MTD into three blocks, there are /dev/mtdblock0, /dev/mtdblock1, /dev/mtdblock2 when system boots up.
The first block is the space for the U-Boot, second one is for the kernel and the last one is for mounting YAFFS2 or UBIFS.

If user wants to modify the block size or increase or decrease number of partition, please modify uboot/include/nuc980_evb.h or drivers/mtd/nand/nuc980_nand.c.

- PWM

To enable PWM function, user needs to enable the following functions. The PWM pin maybe needs to change according to hardware connection.
"No output" is stand for those unused PWM channels.

```
Device Drivers  --->
     [*] Pulse-Width Modulation (PWM) Support  --->
          <*>   NUC980 PWM support
               NUC980 PWM channel 0 output pin (Output from PF5)  --->
               NUC980 PWM channel 1 output pin (Output from PF6)  --->
               NUC980 PWM channel 2 output pin (No output)  --->
               NUC980 PWM channel 3 output pin (No output)  --->
```

This section will describe PWM control method by using sysfs. After system boots up, there are four PWM (pwmchip0~3) in /sys/class/pwm directory. Each group stands for one PWM channel. Before using it, enter target PWM directory and execute "echo 0 > export" command to enable this PWM channel. If enable success, there is a pwm0 directory will be created and user can control this PWM channel.
There are some files in the new created directory, their meaning is listed in the following table.

File Name	Purpose
period	Control cycle, which the unit is ns. The shortest time supported by the driver is us. Example (control cycle is 20us) $ echo 20000 > period
duty_cycle	Set duty cycle of PWM, which the unit is ns. The shortest time supported by the driver is us. Example (duty cycle is 15us) $ echo 15000 > duty_cycle
polarity	Set polarity, it can be normal or inverse output. Example: Normal output: $ echo normal > polarity Inverse output:$ echo inversed > polarity
enable	Enable or disable function. Example: Enable function: $echo 1 > enable Disable function: $echo 0 > enable

The following is a PWM0 example which control cycle is 300us and duty cycle is 33%.

```
$ cd sys/class/pwm
$ ls
pwmchip0  pwmchip1  pwmchip2  pwmchip3
$ cd pwmchip0
$ ls
device     export     npwm       power      subsystem  uevent     unexport
$ echo 0 > export
$ ls
device     npwm       pwm0       uevent
export     power      subsystem  unexport
```

```
$ cd pwm0/
$ ls
duty_cycle  enable      period      polarity    power       uevent
$ echo 1 > enable
$ echo 300000 > period
$ echo 100000 > duty_cycle
```

- Realtek RTL8188 802.11 Wi-Fi

To support RTL8188 USB Wi-Fi module, user needs to enable wireless network function, USB host, loadable module support and the usage of this driver is described as follows,

1. Load driver module by using insmod command.

```
$ insmod rtl8188eu.ko
```

2. Enable wireless interface

```
$ ifconfig lo up
$ ifconfig wlan0 up
```

3. Use wpa_supplicant utility to connect with wireless AP

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

Wpa_supplicant needs configuration file, the following are examples of configuration file for it.

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

Note: Nuvoton cannot provide RTL8188 driver source code.

- Realtek RTL8192 802.11 Wi-Fi

To support RTL8192 SDIO Wi-Fi module, user needs to enable wireless network function, SDIO host, loadable module support and the following function.

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

The usage of this driver is described as follows.

4. Load driver module by using insmod command.

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

5. Enable wireless interface

```
$ ifconfig wlan0 up
```

6. Use wpa_supplicant utility to connect with wireless AP

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

Wpa_supplicant needs configuration file, the following are examples of configuration file for it.

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

Note: Nuvoton cannot provide RTL8192 driver source code.

- RS232, RS485, IrDA

The NUC980 series supports 11 serial ports which can be configured individually. Please follow the instruction below to enable serial port function.
User can enable or disable each port on configuration page. Most of the ports have various GPIO pins can be selected except UART0, UART3 and UART5.
User can enable or disable wake-up function on configuration page except UART0, UART3 and UART5, UART7, UART9.
The UART0 is kept for console and user doesn't need to configure it.

```
Device Drivers  --->
     Character devices  --->
          Serial drivers  --->
               [*]  NUC980 serial support
               [*]    NUC980 UART1 support
               [*]      Enable UART1 CTS wake-up function
                        NUC980 UART1 pin selection (Tx:PE2, Rx:PE3)  --->
               [*]    NUC980 UART2 support
               [*]      Enable UART2 CTS wake-up function
                        NUC980 UART2 pin selection (Tx:PF11, Rx:PF12)  --->
               [*]    NUC980 UART3 support
               [*]    NUC980 UART4 support
               [*]      Enable UART4 CTS wake-up function
                        NUC980 UART4 pin selection (Tx:PC10, Rx:PC11)  --->
               [*]    NUC980 UART5 support
               [*]    NUC980 UART6 support
               [*]      Enable UART6 CTS wake-up function
                        NUC980 UART6 pin selection (Tx:PB2, Rx:PB3)  --->
               [*]    NUC980 UART7 support
                        NUC980 UART7 pin selection (Tx:PG4, Rx:PG5)  --->
               [*]    NUC980 UART8 support
               [*]      Enable UART8 CTS wake-up function
                        NUC980 UART8 pin selection (Tx:PE10, Rx:PE11)  --->
               [*]    NUC980 UART9 suppor
                        NUC980 UART9 pin selection (Tx:PH2, Rx:PH3)  --->
               [*]    NUC980 UART10 support
               [*]      Enable UART10 CTS wake-up function
```

```
                        NUC980 UART10 pin selection (Tx:PB10, Rx:PB11)  ---
>
               [*]     Console on NUC980 serial port
```

If serial port is configured as IrDA function, user needs to enable serial port function and IrDA function additionally like the following items.

```
[*] Networking support  --->
          <*>   IrDA (infrared) subsystem support  --->
               Infrared-port device drivers  --->
                    <*> NUC980 SIR on UART
```

- SD Card

To enable SD interface, user needs to enable the following functions. The NUC980 series supports one SD card interface.

```
Device Drivers  --->
     <*>MMC/SD/SDIO card support  --->
          <*>   MMC block device driver
          <*>     Use bounce buffer for simple hosts
          <*>   Nuvoton NUC980 SD Card support
```

After system booting, if any card is detected the device mmcblk0 will be create in /dev directory. If more than one partition is created in the card, the device will be created sequentially, like mmcblk0, mmcblk1 and so on

- SPI

The NUC980 series supports three SPI interfaces. They can be enabled individually or not. The following describes configuring two SPI interfaces.

```
Device Drivers  --->
     [*] SPI support  --->
          <*>  Nuvoton NUC980 Series QSPI Port 0
               QSPI0 pin selection (Normal mode)  --->
               QSPI0 TX/RX by PDMA or not (Use PDMA)  --->
          < >     QSPI0 enable pin for the second chip select
          <*>  Nuvoton NUC980 Series SPI Port 0
               SPI0 IO port selection (Port D)  --->
               SPI0 TX/RX by PDMA or not (Use PDMA)  --->
          < >  SPI0 enable pin for the second chip select
```

The QSPI0 supports normal (4-pin) or quad (6-pin) mode, enable or disable PDMA, additional second chip select pin (SS1) can be selected for the QSPI0.
In SPI0 and SPI1, normal mode, , enable or disable PDMA, additional second chip select pin (SS1) can be selected.

If SPI Flash device is also used, user needs to enable MTD function like the following items.

```
Device Drivers  --->
     <*> Memory Technology Device (MTD) support  --->
          <*>   Caching block device access to MTD devices
          Self-contained MTD device drivers  --->
```

```
<*> Support most SPI Flash chips (AT26DF, M25P, W25X, ...)
```

User also needs to enable JFFS2 file system functions in order to use SPI Flash device correctly. The configuration of JFFS2 is described in the file system section. User can refer to it for more detail.
If user wants to use new SPI Flash device which is not included in BSP, it's necessary to modify id table in driver and it can be identified be kernel correctly.
Please modify the m25p_ids structure in drivers/mtd/devices/m25p80.c file.

```
static const struct spi_device_id m25p_ids[] = {
/* Atmel -- some are (confusingly) marketed as "DataFlash" */
{ "at25fs010",  INFO(0x1f6601, 0, 32 * 1024,   4, SECT_4K) },
{ "at25fs040",  INFO(0x1f6604, 0, 64 * 1024,   8, SECT_4K) },
...
{ "en25qh16", INFO(0x1c7015, 0, 64 * 1024, 32, 0) },
...
{ "cat25128", CAT25_INFO(2048, 8, 64, 2) },
     { },
};
```

The nuc980_spi_flash_data structure also needs to be modified for the same purpose.
The string (name) at type argument must be the same with the first argument of m25p_ids structure otherwise system can't recognize it correctly.

```
static struct flash_platform_data nuc980_spi_flash_data = {
     .name = "m25p80",
     .parts =  nuc980_spi_flash_partitions,
     .nr_parts = ARRAY_SIZE(nuc980_spi_flash_partitions),
     .type = "en25qh16",
};
```

If user wants to modify partition number of SPI flash, nuc980_spi_flash_partitions structure in arch/arm/mach-nuc980/dev.c file also needs to be modified.

```
static struct mtd_partition nuc980_spi_flash_partitions[] = {
     {
          .name = "SPI flash",
          .size = 0x0200000,
          offset = 0,
     },
};
```

- SPI slave

The NUC980 series supports SPI slave. The following is an example for configuring QSPI0 to SPI slave.

```
Device Drivers  --->
     -> Misc devices
          <*> NUC980 QSPI0 slave mode support
               QSPI0 pin selection by transfer mode (Normal mode)  --->
```

```
          < > NUC980 SPI0 slave mode support
          < > NUC980 SPI1 slave mode support
```

The SPI slave driver are located in directory drivers/misc. The file naming are nuc980-qspi0-slave.c, nuc980-spi0-slave.c, and nuc980-spi1-slave.c.
Below is sample that slave receives command “0x9f”, then prepare data and reply to master.

```
static int QSPI0_Slave_Thread_TXRX(struct nuc980_spi *hw)
{
        unsigned char rx;
        unsigned long flags;
        int i;

        while(1) {

                wait_event_interruptible(slave_done, (slave_done_state !=
0));
                rx = __raw_readl(hw->regs + REG_RX);
                //printk("Receive [0x%x] \n", rx);

                switch (rx) {
                        case 0x9f:
                                for (i = 0; i < QSPI0_SlaveDataLen; i++)
                                        QSPI0_SlaveData[i] = i;

                                nuc980_enable_txth_int(hw);
                                break;
                        default:
                                break;
                }

                InTransmitted = 0;
                slave_done_state = 0;
                nuc980_enable_rxth_int(hw);
        }

        return 0;
}
```

Users can modify the slave code according to their application.

- SPI NAND

The NUC980 series supports SPI NAND. The following describes configuring SPI NAND. First, enable SPI.

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

```
     [*] SPI support  --->
          <*>  Nuvoton NUC980 Series QSPI Port 0
               QSPI0 pin selection (Normal mode)  --->
          <*>  QSPI0 enable pin for the second chip select
               Pin selection (Use SS1 (PD0))  --->
          <*>  Nuvoton NUC980 Series SPI Port 0
               SPI0 transfer mode selection (Normal mode)  --->
               SPI0 IO port selection (Port D)  --->
          < >  SPI0 enable pin for the second chip select
```

Then, user needs to enable MTD function like the following items.

```
Device Drivers  --->
     <*> Memory Technology Device (MTD) support  --->
          <*>   Caching block device access to MTD devices
          Self-contained MTD device drivers  --->
               <*> Support most SPI Flash chips (AT26DF, M25P, W25X, ...)
          <*>   NAND Device Support --->
          <*>   SPI-NOR Device Support --->
```

Besides, user needs to enable SPI NAND driver.

```
Device Drivers  --->
     <*> Staging drivers  --->
          <*>   SPINAND Device Support
```

If user wants to use new SPI NAND Flash device which is not included in BSP, it's necessary to modify id table in driver and it can be identified be kernel correctly.
Please modify the nand_flash_ids structure in drivers/mtd/nand/nand_ids.c file.

```
struct nand_flash_dev nand_flash_ids[] = {
        /*
         * Some incompatible NAND chips share device ID's and so must be
         * listed by full ID. We list them first so that we can identify
         * the most specific match.
         */
        {"W25N01GV 1G 3.3V",
                { .id = {0xef, 0xaa} }, SZ_2K, 128, SZ_128K, 0, 2, 64,
NAND_ECC_INFO(1, SZ_512) },
        {"TC58NVG0S3E 1G 3.3V 8-bit",
                { .id = {0x98, 0xd1, 0x90, 0x15, 0x76, 0x14, 0x01, 0x00} },
                  SZ_2K, SZ_128, SZ_128K, 0, 8, 64, NAND_ECC_INFO(1,
SZ_512),
                  2 },
...
{NULL}
};
```

The nuc980_qspi0_flash_data structure in arch/arm/mach-nuc980/dev.c also needs to be modified for the same purpose.
The string (name) at name argument must be the same with SPI NAND driver name otherwise kernel can't run the driver.
For instance, the driver name of Micron SPI NAND is mt29f.

```
static struct spi_driver spinand_driver = {
        .driver = {
                .name           = "mt29f",
                .of_match_table = spinand_dt,
        },
        .probe          = spinand_probe,
        .remove         = spinand_remove,
};
```

The name field of nuc980_qspi0_flash_data must also named mt29f

```
static struct flash_platform_data nuc980_qspi0_flash_data = {
     .name = "mt29f",
     ……
};
```

And also update .modalias field of table nuc980_qspi0_board_info in dev.c

```
static struct spi_board_info nuc980_qspi0_board_info[] __initdata = {
#ifdef CONFIG_MTD_M25P80
          {
                .modalias = "mt29f",
                …….
          },
#endif
};
```

- USB Host

To enable USB Host function, please check "USB support" in "Device Drivers" menu. The NUC980 USB Host is equipped with EHCI (USB 2.0) and OHCI (USB1.1) Host controllers. All of the following items must be checked to enable both Host controllers.

```
Device Drivers  --->
     [*] USB support  --->
          <*>   Support for Host-side USB
          <*>     EHCI HCD (USB 2.0) support
          <*>     NUC980 EHCI (USB 2.0) support
          NUC980 USB Host port power pin select (No USBH_PPWRx and
USBH_OVRCUR) --->
          [*]     NUC980 turn off Usb Hots VBUS power while power down
          <*>     OHCI HCD support
          [*]     NUC980 OHCI (USB 1.1) support
```

According to the pin configuration of the target NUC980 series, select the corresponding multi-

function pin setting.

```
[ ] PE.14 and PE.15 for USBH_PPWR0/1, PH.1 for USBH_OVRCUR
[ ] PF.10, PH.1 for USBH_OVRCUR
[ ] No USBH_PPWRx, PH.1 for USBH_OVRCUR
[X] No USBH_PPWRx and USBH_OVRCUR
```

If target board's USB port power is controlled by a Power Switch Controller, the NUC980 must have USBH_PPWRx and USBH_OVRCUR pins to communicate with it. Depend on the target NUC980 chip's pin configuration, USB Host port0 and port1 power can be controlled by PE.14 and PE.15 respectively; or both be controlled by PF.10. PH.1 is dedicated assigned to USBH_OVRCUR for over-current detection.
If VBUS of USB Host ports are connected to +5V directly, USBH_PPWRx pins are not required any more. In this condition, USBH_PPWRx can be configured as GPIO pins and user must select "No USBH_PPWRx" items from this menu.
User can also select the last item "No USBH_PPWRx and USBH_OVRCUR" to release USBH_PPWRx and USBH_OVRCUR pins. In this condition, user can obtain 2 or 3 free GPIO pins, but take the risk of being unaware of over-current dangerous.

The NUC980 USB Host driver power down function supports option to turn off VBUS while power down.

```
 [*]     NUC980 turn off Usb Hots VBUS power while power down
```

If this option is not enabled, the NUC980 USB Host will turn off USB PHY power while power down. USB bus is supsended. The VBUS power will be kept. USB devices can still consume power from NUC980. NUC980 USB Host can resume USB devices when kernel is woken up.
If this option is selected, the NUC980 USB Host will turn off both USB PHY power and VBUS power. In this case, all USB devices are disconnected while power down. On kernel wake up, the NUC980 USB Host turns on the VBUS power again and USB devices will be re-enumerated.

- USB mass storage class device support

Besides selecting NUC980 USB Host controller driver, user may have to select supporting device classes. For example, if user want to support mass storage device, it's necessary to enable "SCSI device support" first. After enabled SCSI device support, "USB Mass Storage Support" option will be present in "USB support" menu. Select it to enable Mass Storage Device supporting.

```
Device Drivers  --->
     SCSI device support  --->
          <*>   SCSI device support
               <*> legacy / proc/scsi/ support
               <*> SCSI disk support
               <*> SCSI media changer support
               [*] Asynchronous SCSI scanning
               [*] SCSI low-level drivers
     [*] USB support  --->
          <*>   USB Mass Storage Support
```

- USB video class device support

To support USB video class (UVC), the following "Media Support" options must be enabled.

```
Device Drivers  --->
Multimedia support  --->
```

```
  <*>  Cameras/video grabbers support
  <*>  Media Controller API
  <*>  V4L2 sub-device userspace API
          <*>  V4L2 int device
  <*>  Media USB Apapters  --->
       <*>  USB Video Class (UVC)
       [*]    UVC input events device support
  <*>  V4L platform device
```

- USB host and HID device

To support HID class devices, such as USB mouse and USB keyboard, except to enable USB Host function, user must also enable “HID bus support” and “Input device support”. As the following:

```
Device Drivers --->
HID support  --->
 HID bus support  --->
      <*>   User-space I/O driver support for HID subsystem
      <*>   Generic HID driver
 USB HID support  --->
      <*> USB HID transport layer
Input device support  --->
 <*>   Mouse interface
 [*]     Provide legacy /dev/psaux device
 <*>   Event interface
 [*]   Keyboards  --->
      <*>   AT keyboard
 [*]   Mice  --->
      <*>   PS/2 mouse
```

- USB Host and USB modem

To use USB modem device, except to enable USB Host function, user must also enable “USB Serial Converter support” and “USB driver for GSM and CDMA modems”. As the following:

```
Device Drivers →
  [*] USB Support →
       [*] USB Serial Converter support --->
            [*] USB driver for GSM and CDMA modems
```

- USB Device

```
Device Drivers --->
[*] USB support --->
 <*> USB Gadget Support --->
      USB Peripheral Controller --->
           <*> NUC980 USB Device Controller
      <M> USB Gadget Driver
```

```
      <M> Mass Storage Gadget
```

After compiling kernel, three driver module files will be outputted. (fs/configfs/configfs.ko, drivers/usb/gadget/libcomposite.ko and drivers/usb/gadget/g_mass_storage.ko)
User needs to copy those file to rootfs or somewhere they can be accessed by system. (Like USB mass storage device)
The following is an example by using USB mass storage gadget function.

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

- Video Capture

To support video capture function, user needs to enable “Cameras/video grabbers support” item first and enable “NUC980 Video-in support” in “Encoders, decoders, sensors and other helper chips” item. Finally, user can select model of video capture device. BSP supports OV7725, OV5640, NT99050 and NT99141 drivers now.

```
Device Drivers  --->
  <*> I2C support  --->
       I2C Hardware Bus support  --->
            <*> NUC980 I2C Driver for Port 1
                  NUC980 I2C1 pin selection (SDA:PB_6  SCL:PB_4) --->
            <*> NUC980 I2C Driver for Port 2
                  NUC980 I2C2 pin selection (SDA:PB_7  SCL:PB_5) --->
 [*] Multimedia support  --->
      [*]   Camera/video grabbers support
      [*]   Media Controller API
      [*]   V4L2 sub-device userspace API
      [*]   V4L platform devices  --->
           Encoders, decoders, sensors and other helper chips  --->
                <*> Nuvoton NUC980 Video-In0 Support
                     (3)   Max frame buffer
                     (24000000) Video frequency
                     Nuvoton NUC980 Image Sensor Selection(NT99141)--->
                  <*> Nuvoton NUC980 Video-In1 Support
                     (3)   Max frame buffer
                     (24000000) Video frequency
                     Nuvoton NUC980 Image Sensor Selection(NT99141)--->
```

If the I²C interface is used by video capture device to configure arguments, the I²C function also needs to be enabled first. User can refer to I²C section to do that.
The V4L2 API is supported by video capture driver in BSP and user can refer to the example in BSP/applications/demos/cap directory. Introdution API in cap sample code as below:

xioctl(fd, VIDIOC_S_FMT, &fmt) : Set Image Height, width and format
xioctl(fd, VIDIOC_DQBUF, &buf) : Get Image
xioctl(fd, VIDIOC_QBUF, &buf) : Relase Image
xioctl(fd, VIDIOC_STREAMON, &type) : Start CAP

xioctl(fd, VIDIOC_STREAMOFF, &type) : Stop CAP

- Watchdog Timer

To support watchdog timer function, please enable the following items.
Timeout period in default is 2.03 seconds and this time can be modified via ioctl() command function (WDIOC_SETTIMEOUT) by application program.
There are three different time cycles can be supported by watchdog driver. If command argument is smaller than 2 and the timeout period will be 0.53 second. If command argument is between 2 to 8 and timeout period will be 2.03 seconds. And if argument is larger than 8 and timeout period will be 8 seconds. There is an example in BSP/applications/demos/wdt directory for user to reference.
To support watchdog timer wake-up function, please enable “NUC980 WDT wake-up support” item.

```
Device Drivers  --->
     [*] Watchdog Timer Support  --->
          <*>   Nuvoton NUC980 Watchdog Timer
          <*>     NUC980 WDT wake-up support
```

- Window Watchdog Timer

Please enable the following items to support window watchdog timer function.

```
Device Drivers  --->
     [*] Watchdog Timer Support  --->
          <*>   Nuvoton NUC980 Window Watchdog Timer
```

There are three major differences between window watchdog timer and watchdog timer. First, the configuration of window watchdog timer cannot be modified after enabling its function. Second, window watchdog timer only can be reset in specific time slot, but watchdog timer can be reset at any time if timeout doesn’t occur. In application, user needs to use WDIOC_GETTIMELEFT ioctl() argument to get the available time to reset. If return value is 0 and application can use WDIOC_KEEPALIVE argument to let system reset otherwise system will be reset right now. And the third, window watchdog timer does not count while CPU is in idle and power-down mode. Linux kernel automatically put CPU into idle mode between each timer tick if the system is no busy. So the timeout period of window watchdog timer timeout period varies depending on the system loading. An example code is in BSP/applications/demos/wwdt for reference.

- RTC

User can enable or disable wake-up function on configuration page.

```
Device Drivers --->
     [*] Real Time Clock --->
          <*> NUC980 RTC driver
          [*]   Enable RTC wake-up function
```

- CAN

NUC980 series support 2 CAN ports which can be configured individually. Please follow the instruction below to enable CAN port function.
User can enable or disable each port on configuration page. CAN0 port has various GPIO pins can be selected.
User can enable or disable wake-up function on configuration page

```
-*- Networking support  --->
     <*>   CAN bus subsystem support  --->
          --- CAN bus subsystem support
```

```
          <*>  CAN Gateway/Router (with netlink configuration)
               CAN Device Drivers  --->
                    <*> Platform CAN drivers with Netlink support
                    [*] CAN bit-timing calculation
                    <*> NUC980 CAN0/CAN1 devices  --->
                         --- NUC980 CAN0/CAN1 devices
                         [*]  NUC980 CAN0 support
                         [*]    Enable CAN0 wake-up function
                                NUC980 CAN0 pin selection (Tx:PB11,
Rx:PB10)  --->
                                   (X) Tx:PB11, Rx:PB10
                                   ( ) Tx:PH3, Rx:PH2
                                   ( ) Tx:PI4, Rx:PI32
                         [*]  NUC980 CAN1 support
                         [*]    Enable CAN0 wake-up function
                                NUC980 CAN1 pin selection (Tx:PH15,
Rx:PH14)  --->
                                   (X) Tx:PH15, Rx:PH14
```

An example code is in BSP/applications/demos/CAN for reference

- IIO ADC
  User can use normal mode ADC via IIO archtechture, please refer the following configurations to enable it.

```
Device Drivers  --->
     <*> Industrial I/O support  --->
  [*]   Enable buffer support within IIO
  [*]   Enable triggered sampling support
  Analog to digital converters  --->
    <*> Nuvoton NUC980 Normal ADC driver
        Reference voltage selection (Internal AVDD, 3.3V)  --->
  [ ]   Enable internal bandgap
  <*>   Select external channel 0
  <*>   Select external channel 1
  <*>   Select external channel 2
  < >   Select external channel 3
  < >   Select external channel 4
  < >   Select external channel 5
  < >   Select external channel 6
  < >   Select external channel 7
```

There are two reference voltage selections can be choosen, they are internal AVDD 3.3V and VREF input.
Besides, select the external channels that you want to use. For above configuration example, external channel 0, 1 and 2 are enabled to use.

Application can use the following command to get the result converted by ADC function.

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

X stands for the channel of ADC (X=0~8).

- SCUART

There are two smart card interfaces built in NUC980 series. They have additional UART function to simulate as basic UART port when there is not enough UARTs to use in system.
In this mode, the SC_CLK pin will be used to as transmit function and SC_DATA will be receive function.

```
Device Drivers --->
     Character devices --->
          Serial drivers --->
                [*] NUC980 Smartcard UART mode support
                [*] NUC980 SCUART0 support
                    NUC980 SCUART0 pin selection (Tx:PA5, Rx:PA4) --->
                [*] NUC980 SCUART1 support
                    NUC980 SCUART1 pin selection (Tx:PC7, Rx:PC8) --->
```

The device node for the SCUART is /dev/ttySCU0 or /dev/ttySCU1. The basic operation of SCUART is the same with normal UART but have a lot of limitations, for example, there are only four levels of FIFO and can't support flow control function, can't support RS485 and IrDA transmission mode. It is better to use normal UART only if they are all occupied by system.

- Loopback device

A loop device is a pseudo-device that makes a file accessible as a block device. Before use, a loop device must be connected to an existing file in the file system.
Please enable the following items to support loopback device.

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

The usage of loopback device lists as the following steps.
1. Create image file for mounting on loopback device.

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

2. Format image (take FATFS for example)

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

3. Mount image

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

3.3.4 File System

- FAT

FAT is common file system and can be seen usually on SD card or USB mass storage device. User can enable the following items to support it.

```
File systems  --->
     DOS/FAT/NT Filesystems  --->
          <*> MSDOS fs support
```

```
          <*> VFAT (Windows-95) fs support
          (437) Default codepage for FAT
          (iso8859-1) Default iocharset for FAT
```

Command for mounting the first partition on SD card is listed as follows.

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

- JFFS2

JFFS2 is one of the file system used on NAND Flash. Please enable the following items to support it.

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

- ROMFS

ROMFS is one of the file system used on root file system. Please enable the following items to support it.

```
File systems  --->
     [*] Miscellaneous filesystems  --->
          <*>   ROM file system support
               RomFS backing stores (Block device-backed ROM file system
support)  --->
```

- YAFFS2

YAFFS2 is one of the file system used on NAND Flash. Before enabling the following items, user needs to enable MTD item “Caching block device access to MTD devices Device drivers” first.

```
File systems  --->
     [*] Miscellaneous filesystems  --->
          <*>   yaffs2 file system support
          <*>   Autoselect yaffs2 format
          <*>   Enable yaffs2 xattr support
```

Command for mounting YAFFS2 is listed as follows.

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

- exFAT

exFAT is a new generation file system created by Microsoft. It is more flexible about size of single file and total capacity of device.

```
File systems  --->
     DOS/FAT/NT Filesystems  --->
          <*> exFAT fs support
```

Command for mounting the first partition on SD card is listed as follows.

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

- FUSE and NTFS

FUSE (Filesystem in Userspace) is a kind of file system that is implemented for user space. User can implement much kind of file systems by FUSE. The famous file system that is implement by FUSE are NTFS-3G or SSHFS and so on. The following is an example that implements Microsoft

NTFS (NTFS-3G) by FUSE.
Please enable the following item to support FUSE function.

```
File systems  --->
     <*> FUSE (Filesystem in Userspace) support
```

NTFS-3G is an open source project developed and implemented by Tuxera. It is a driver which can read and write NTFS on Linux and source code can be downloaded from http://www.tuxera.com/community/ntfs-3g-download page. Please refer to user's manual of ntfs-3g to compile it. And mount it by the following command.

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

- UBIFS

Please enable the following items to support it.

```
Device Drivers  --->
-*- Memory Technology Device (MTD) support  --->
<*>   Enable UBI - Unsorted block images  --->
File systems  --->
     [*] Miscellaneous filesystems  --->
          <*>   UBIFS file system support
          [*]     Advanced compression options
          [*]       LZO compression support
```

3.3.5 Speed Up SPI Boot with JFFS2 File System in SPI Flash

The first step is to set SPI to Quad mode.

```
Device Drivers  --->
[*] SPI support  --->
<*>   Nuvoton NUC980 Series SPI Port 0
    SPI0 pin selection by transfer mode (Quad mode)  --->
```

The second step, set page size to 0x1000 while use mkfs.jffs2 to build a JFFS2 root file system

```
$ mkfs.jffs2  -s 0x1000 -e 0x10000 -p 0x800000 -d  rootfs_jffs2/ -
o jffs2.img
```

Some SPI Flash's sector size is 4K that is defined in spi_nor_ids [] of drivers/mtd/spi-nor/spi-nor.c For instance, Winbond W25Q128 is 4K sectoer size.

```
{ "w25q128", INFO(0xef4018, 0, 64 * 1024, 256, SECT_4K) },
```

Howerever, the minimum sector size of mkfs.jffs2 tool is 8K. Hence, the attribute "SECT_4K" should be removed. The modification is as below.

```
{ "w25q128", INFO(0xef4018, 0, 64 * 1024, 256, 0) },
```

The third step, use sumtool and enable JFFS2_SUMMARY in kernel configuration
Below is an example of sumtool usage

```
$ sumtool -i jffs2.img -o jffs2_sumtool.img -e 0x10000
```

JFFS2_SUMMARY configuration is located as below.

```
-> File systems
     -> Miscellaneous filesystems (MISC_FILESYSTEMS)
     -> Journalling Flash File System v2 (JFFS2) support (JFFS2_FS)
```

```
    [*]     JFFS2 summary support
```

3.3.6 FIQ

To make sure the real time of interrupt, user can use FIQ instead of IRQ. This section includes an example which describes how to use timer2 FIQ.
Please enable the following item to support FIQ in system.

```
Kernel Configuration
     System Type  --->
          [*] Nuvoton NUC980 FIQ support
```

Example for timer0:
User needs to inster *init_FIQ(0)* code in the initialization function of driver.

```
static int __init xxx_init(void) {
    ...
    init_FIQ(0);
    ...
}
```

Then add the following code and insert use_fiq() function in the suitable position of drvier. (Should replace general irq operation code.)

```
/*IRQ handler for the timer*/
void nuc980_timer0_interrupt(void) {
    // ... add some code here
    __raw_writel(0x1, REG_TMR_ISR(TIMER0)); /* clear timer0 flag */
}

static uint8_t fiqStack[1024];
extern unsigned char fiq_handler, fiq_handler_end;
static struct fiq_handler timer0_fiq = {
    .name = "timer0_fiq_handler"
};

void use_fiq(void) {
    int ret;
    struct pt_regs regs;

    ret = claim_fiq(&timer0_fiq);
    if (ret)
        return;
    set_fiq_handler(&fiq_handler, &fiq_handler_end - &fiq_handler);
    // set some registers use in FIQ handler
    regs.ARM_r8 = (long)nuc980_timer0_interrupt;
    regs.ARM_r10 = (long)REG_AIC_FIQNUM;
```

```
    regs.ARM_sp = (long)fiqStack + sizeof(fiqStack) - 4;
    set_fiq_regs(&regs);
    /* Enable the FIQ */
    __raw_writel(__raw_readl(REG_AIC_SRCCTL4) & ~0x00000007,
                 REG_AIC_SRCCTL4);
    enable_fiq(IRQ_TIMER0);
}
```

Note that, the regs.ARM_r8 must be the address of fiq handler function and regs.ARM_r10 must be the address of REG_AIC_FIQNUM register.

3.3.7 Power Management

Linux kernel also supports power management function. The system can enter power-down mode to save power consumption and wake up later using enabled wake up source(s). To enable power management support, please enable following kernel features befor compilation.

```
Power management options  --->
     [*] Suspend to RAM and standby
```

With the kernel with power management function enabled, issue following under shell can put the system enter power-down mode. In power-down mode, all unnessery clocks will be turned off, and DDR put into self-refresh mode. And only enabled wake up source can bring the system back to normal operation mode.

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

Note that to minimize the power consumption, GPIO pin needs extra pull up/down setting before enter power-down mode. This is pretty much depending on the board design, so please add the your control code in at the begining of nuc980_suspend_enter() function in arch/arm/mach-nuc980/pm.c

3.4 Linux Kernel Compilation

After the kernel configuration is finished, type “make” command to compile kernel in linux-3.10.x directory. If no error happens, the kernel image file and kernel zip file will be output to upper image directory. You can use “make uImage” command to build an image file that has a U-Boot wrapper if mkimage tool is installed or use “make dtbs” to generate dtb (device tree blob) file if dtc tool is installed.

```
$ make
......
  Kernel: arch/arm/boot/Image is ready
cp arch/arm/boot/Image   ../image/980image
updating: ../image/980image (deflated 31%)
  GZIP    arch/arm/boot/compressed/piggy.gzip
  CC      arch/arm/boot/compressed/misc.o
  AS      arch/arm/boot/compressed/piggy.gzip.o
  LD      arch/arm/boot/compressed/vmlinux
  OBJCOPY arch/arm/boot/zImage
  Kernel: arch/arm/boot/zImage is ready
$ ls ../image/
```

```
980image
```

4 Linux User Applications

4.1 Sample Applications

There are some sample applications in the applications/ directory. Content of each directory listed in the following table

Directory	Description

alsa-utils-1.0.23	ALSA command line utilities.*
	
	Cross compilation command as below:
	
	$ ./configure -host=arm-linux -disable-nls --disable-xmlto PKG_CONFIG_LIBDIR=/usr/local/arm_linux_4.8/usr/lib -disable-alsamixer
	
	$ make
	
	Sample mixer setting for playback:
	
	● $./amixer set PCM 85%
	● $./amixer set Headphone 90%
	
	Sample mixer setting for recording:
	
	When source is Mic:
	
	● $./amixer set “Mic Bias” on
	● $./amixer set “Input PGA” 100%
	● $./axmier set ADC 90%
	
	When source is Line In:
	
	● $./axmier set "Right Input Mixer R2" on
	● $./axmier set "Left Input Mixer L2" on
	● $./axmier set "L2/R2 Boost" 100%
	● $./axmier set ADC 90%
	
	Playback command:
	
	● $./aplay <file name>
	
	To playback the sample sound file in BSP, please use following command:
	
	$ cd usr
	
	$./aplay –c 2 –f S16_LE alsa/8k2ch.pcm
	
	Recording command:
	
	● $./arecord –d 10 –f S16_LE –c2 –r8000 –t wav –D plughw:0,0 <file name>

	Command to record and play simultaneously: ● $./arecord -f S16_LE -r 8000 -c 2 -D plughw:0,0 | ./aplay
benchmark/netperf-2.6.0	Network performance benchmarking tool. Cross compilation command below: $./configure –host=arm-linux
busybox-1.22.1/	Busybox source code. Cross compilation command below: 1. $ make menuconfig 2. Select applets to be build 3. $ make
demos/alsa_audio	Audio sample application. *
demos/cap	Video capture sample application. *
demos/can	CAN bus sample application
demos/crypto	Encryption/decryption sample application. *
demos/etimer	Enhanced timer sample application. *
demos/ebi	External Bus Interface sample application. *
demos/gpio	GPIO sample application. *
demos/irda	IrDA sample application. *
demos/lcm/	LCD sample application. *
demos/rtc	RTC sample application. *
demos/uart	UART sample application. *
demos/wdt	Watchdog timer sample application. *
demos/wwdt	Window watchdog timer sample application. *
demos/dma	DMA sample application. *
i2c-tools	I2c-tools utility. Simply type make to compile $ make
yaffs2utils	yaffs2 command tool. Simply type make to compile $ make

lzo-2.09	Compress/decompress utility.
	Cross compilation command below:
	$ cd lizo-2.09
	$ ./configure --host=arm-linux --prefix=$PWD/../install
	$ make
	
	$ make install

libuuid-1.0.3	Utility to create UUID. Cross compilation command below:
	$ cd libuuid-1.0.3
	$ ./configure --host=arm-linux --prefix=$PWD/../install
	$ make
	
	$ make install

mtd-utils.tar.gz	mtd-utils source code. Required to use lzo-2.09.tar.gz

	and libuuid-1.0.3.tar.gz
	Cross compilation command below:
	$ cd mtd-utils
	$ export CROSS=arm-linux-
	$ export WITHOUT_XATTR=1
	$ export DESTDR=$PWD/../install
	$ export LZOCPPFLAGS=-I/home/install/include
	$ export LZOLDFLAGS=-L/home/install/lib
	$ make
	
	$ make install

ethtool-4.6	ethtool is the standard Linux utility for controlling network drivers and hardware, particularly for wired Ethernet devices. To cross compile this tool, use "./configure CC=arm-linux-gcc --host=arm-linux;make" command to build ethtool. The command to enable Wake on Lan function is "./ethtool –s eth0 wol g", and the commad to diable Wake on Lan is "./ethtool –s eth0 wol d".

*. The execution result will be incorrect if the driver is not enabled in kernel configuration and/or jumper/ switch setting on EV board setting is inconsistent with kernel configuration.

4.2 Cross Compilation

Sometimes a project requires porting an application to ARM platform. Many open source projects already supports cross compiling. Simply follow these projects' document to configure for cross compiling to build executable files or libraries for ARM platform.
If the application's Makefile doesn't support cross compilation options, the modification of Makefile is necessary. The Makefile used for cross compiling could be alike with the original one, only part of it needs to be modified

- The prefix of tool chain must be set. For example, the original Makefile use gcc for compiling, the new Makefile use arm-linux-gcc for cross compiling. Other tools for example, as and ld need to change to arm-linux-as and arm-linux-ld respectively.
- The path of library and include files need to be set. The cross compiler doesn't use the glibc or other library using in x86 system. Rather it links with uClibc which consumes less system resource.

Here is a simple Makefile for your reference.

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

ROOT = /usr/local/arm_linux_4.8/usr
LIB =$(ROOT)/lib
INC :=$(ROOT)/include

CC=arm-linux-gcc -O2 -I$(INC)
WEC_LDFLAGS=-L$(LIB)
STRIP=arm-linux-strip

TARGET = hello
```

```
SRCS := hello.c

LIBS=  -lc -lgcc -lc
all:
	$(CC) $(WEC_LDFLAGS) $(SRCS) -o $(TARGET) $(LIBS)
	$(STRIP) $(TARGET)

clean:
	rm -f *.o
	rm -f $(TARGET)
	rm -f *.gdb
```

5 Revision Hisotry

Date	Revision	Description
2018.07.07	1.00	Initially issued.
2019.01.24	1.01	Updated content.
2019.03.21	1.02	Added kernel configuration for USB modem.
2020.09.08	1.10	Fix ADC, PWM, and ethtool description error.

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.