NuMicro® Family

ARM926EJ-S™-based Microprocessor

NUC980 Linux 5.10 BSP User Manual

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 5.10 kernel source code and NUC980 device drivers.
- Buildroot (A tool that simplifies and automates the process of building a complete Linux system for an embedded system, using cross-compilation).
- U-Boot 2020.07 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 Evaluation 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. Figure 1-1 is an example of development environment.

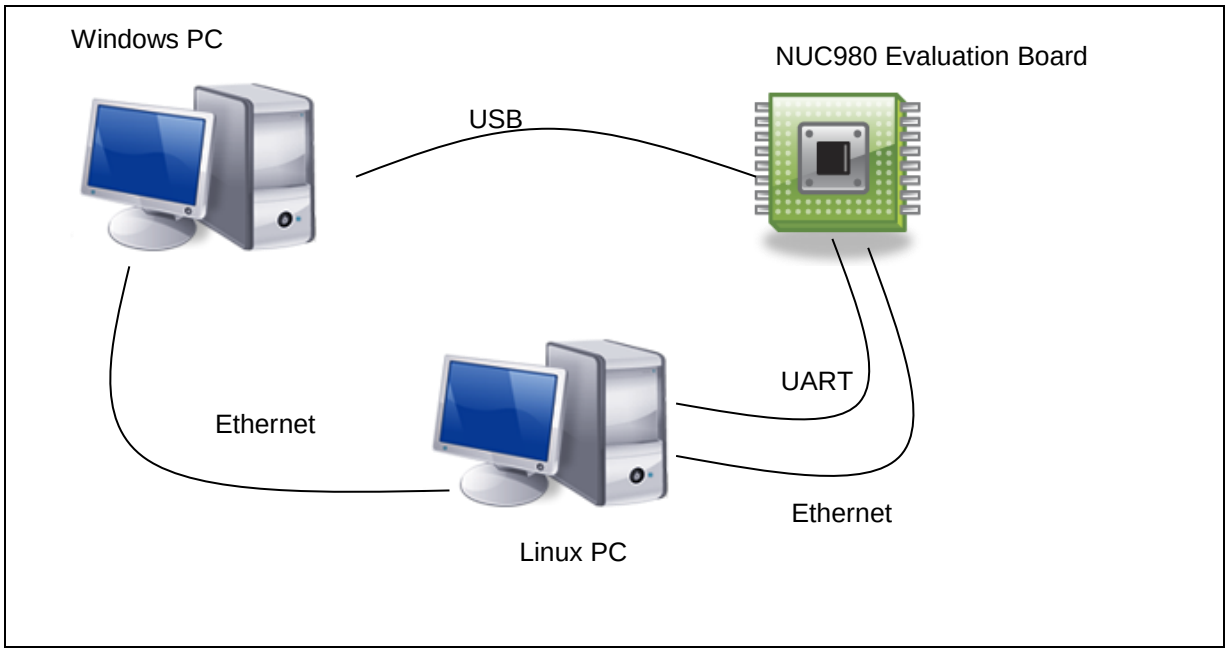

Figure 1-1 Development Environment

1.2 Evaluation Board Setting

The NUC980 series supports different boot modes, which 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 the evaluation board must be set according to the enabled peripherals. Please refer to "*NUC980 Evaluation Board User Manual*" for the board usage.

2 DEVELOPMENT ENVIRONMENT SETUP

The following is required to develop projects in the Buildroot Project environment. A host system with a minimum of 20 Gbytes of free disk space that is running a supported Linux distribution (i.e. recent releases of Fedora, CentOS, Debian, or Ubuntu), and appropriate packages installed on the system you are using for building.

Nuvoton provides two environments for building images, Docker and Linux. Docker is a virtual machine based on host Linux OS, so the setting in the Docker will not affect the host OS and the Docker can create an environment only for building images. Linux distribution will be updated and may result in building image error, so Docker provided by Nuvoton is a better way than Linux

2.1 Docker

Docker is an open-source project based on Linux contains, which is similar to virtual machines, but containers are more portable, more resource-friendly, and more dependent on the host operating system. Docker provides a quick and easy way to get up and running with buildroot. Install docker, Example for Ubuntu 20.04:

First, update your existing list of packages:

```
$ sudo apt-get update
```

Next, install a few prerequisite packages which lets apt use packages over HTTPS:

```
$sudo apt install apt-transport-https ca-certificates curl software-properties-common
```

Then add Docker's official GPG key for the official Docker repository to your system:

```
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
```

Use the following command to set up the **stable** repository; add the Docker repository to APT sources:

```
$sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu focal stable"
```

Next, update the package database with the Docker packages from the newly added repo:

```
$ sudo apt-get update
```

Finally, install Docker:

```
$ sudo apt-get install docker-ce docker-ce-cli containerd.io
```

Then use the Dockerfile to generate the Docker image environment. after completion, use the repo utility to download the Buildroot project after entering the Docker image.

Pleaes use git tool to clone these Dockefiles, *build.sh*, *Dockerfile*, *join.sh*, and *README.md* from the URL https://github.com/OpenNuvoton/MA35D1_Docker_Script.git

Set up docker image.

```
$./build.sh
```

Docker share folder will mount /home/$USER/shared

Enter docker image, and you will see "[user name]@[container id]:~$"

```
$./join.sh
Nuc980_test
test@575f:~$
```

Create a shared/buildroot folder and enter

```
test@575f:~$ mkdir shared/buildroot
test@575f:~$ cd shared/buildroot
```

The first time you use repo, you need to set up the GIT environment.

```
test@575f:~$ git config --global user.email "test@test.test.test"
test@575f:~$ git config --global user.name "test"
```

Use git to download Buildroot project

```
test@575f:~$ git clone https://github.com/OpenNuvoton/MA35D1_Buildroot.git
```

You can check Docker documentation:

https://docker-curriculum.com/

https://docs.docker.com/get-started/

https://github.com/OpenNuvoton/docker

2.2 Linux

The necessary packages must be installed before using Buildroot project. In Linux system, use the command below to install essential Buildroot Project host packages.

- Mandatory packages
  - which
  - sed
  - make (version 3.81 or any later)
  - binutils – build-essential (only for Debian based systems)
  - gcc (version 4.8 or any later)
  - g++ (version 4.8 or any later)
  - bash
  - patch
  - gzip
  - bzip2
  - perl (version 5.8.7 or any later)
  - tar
  - cpio
  - unzip
  - rsync
  - file (must be in /usr/bin/file)
  - bc
- Optional packages
  - python (version 2.7 or any later)
  - ncurses5

3 BUILD IMAGE

This section provides the detailed information along with the process for building an image.

3.1 Build Configurations

Please visit the OpenNuvoton website to download the Buildroot project. The Buildroot project provides default configuration for the NUC980 series. Before modifying any Buildroot configuration, it is recommended to load the default configure of Buildroot first. User can type “make nuvoton_nuc980_iot_defconfig” command. Sometimes if system cannot boot up, user can load the default configuration to recovery Buildroot to safe status. The following are NUC980 configuration files that can be selected.

- nuvoton_nuc980_defconfig
- nuvoton_nuc980_iot_defconfig
- nuvoton_nuc980_iot_g2_defconfig
- nuvoton_nuc980_chili_defconfig
- nuvoton_nuc980_lorag_defconfig
- nuvoton_nuc980_eth2uart_defconfig
- nuvoton_nuc980_chili_matter_defconfig

The command is shown below:

```
$ make nuvoton_nuc980_iot_defconfig (make <DEFCONFIG>)
```

DEFCONFIG=<default configuration name> is the configuration name which points to the configuration file in buildroot/configs/.

Sometimes fine-tune Buildroot configuration, for example to enable some features that are not enabled by default. The Buildroot provides an interface to enter configuration menu by typing “make menuconfig” command.

```
$ make menuconfig
```

This is a multi-layer menu in configuration system. In the current page, user can press arrow 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 can be loaded dynamically.

Menu page can be returned 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 up 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 buildroot configuration file will be named “.config” and be saved in the buildroot source tree directory.

3.2 Build Image

Use the “make” command to build an image. The following command is an example of how to build an image:

```
$ make
```

Update the NUC980 source code by deleting the following files in the dl folder.

```
$ cd dl
$ rm –rf uboot linux
```

You need to change through “make menuconfig” to get the corresponding uImage.

- uImage provides the full system to boot with U-Boot, Linux Kernel and file system; do the following:

```
$ make menuconfig
(Modify 'Bootloaders ---> Board defconfig' to 'nuc980' or 'nuc980_iot' or 'nuc980_chili' or 'nuc980_lorag' or 'nuc980_eth2uart' and Save)
$ make uboot-dirclean linux-dirclean
$ make
(Wait while it compiles and
find nuc980-xxx.dtb (xxx maybe 'chili' or 'dev-v1.0' or 'eth2uart' or 'iot-v1.0' or 'lorag')  and uImage
```

3.3 Deploy Image

After building an image, you can find the image at <build directory>/output/images. Each image can create a u-boot, Linux kernel, file system. After the computer is connected to the NUC980 demo board, use “NuWriter” to write the uImage and devicetree (*.dtb* file) to the NUC980 demo board. The NuWriter is a programing tool for the NUC980 series. For the use of NuWriter, users can refer to the *NUC980 NuWriter User Manual*.

4 BUILDROOT PROJECT CUSTOMIZATION

4.1 Toolchain

Add toolchain to environment variables:

```
/buildroot$ source output/host/environment-setup
```

Create the source code file for this example: helloworld.c:

```
#include <stdio.h>
int main() {
    // printf() displays the string inside console
    printf("Hello, World!\n");
    return 0;
}
```

And compile it.

```
$ $CC helloworld.c -o helloworld
$ ls
Helloword.c  helloworld
```

4.2 U-Boot

If you just want to configure u-boot, you can use make command to configure u-boot.

Configure u-boot:

```
$ make uboot-menuconfig
(Modify and Save)
$ make uboot-rebuild
```

4.3 Linux Kernel

If you just want to configure Linux, you can use make command to configure Linux.

Configure Linux:

```
$ make linux-menuconfig
(Modify and Save)
$ make linux-rebuild
```

4.4 Others

If you modify the source code and rebuild the source code, you can find the source code under *buildroot/output/build*. The U-boot modification example is as follows:

```
(Modify buildroot/output/build/uboot-custom/FILES and Save)
$ make uboot-rebuild
```

5 LINUX KERNEL

5.1 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 (If it is under buildroot, enter “make linux-menuconfig”).

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 up 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-5.10.y directory. (If it is under buildroot, .config saved in buildroot/output/build/uboot-custom/ directory).

5.1.1 Drivers

Audio Interface

The following lists the options that needs to enable to support NUC980 I²S driver.

```
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
                              NUC980 I2S pin selection (Port A)  --->
```

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.

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
          [*]   PF_KEY MIGRATE
```

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
     <*>   User-space interface for random number generator algorithms
     <*>   User-space interface for AEAD cipher algorithms
     [*]   Enable obsolete cryptographic algorithms for userspace
     [*]  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
```

Here introduces the device tree node describes the attribute of NUC980 Cryptographic Accelerator.

```
          crypto@b001c000{
```

“compatible” must set to “nuvoton,nuc980-crypto”.

```
               compatible = "nuvoton,nuc980-crypto";
```

Register base address is 0xb001c000.

```
               reg = <0xb001c000 0x1000>;
```

The interrupt number is 35.

```
               interrupts = <35 4 1>;
               status = "okay";
          };
```

The crypto raw driver supports ioctl interface to performace NUC980 Cryptographic Accelerator ECC and RSA.

```
          crypto_raw {                                   /* Important note!
Must also enable crypto@b001c000 */
               compatible = "nuvoton,nuc980-crypto-raw";
               status = "okay";
          };
```

The crypto prng driver supports NUC980 Cryptographic Accelerator PRNG.

```
          crypto_prng {                                  /* Important note!
Must also enable crypto@b001c000 */
```

```
            compatible = "nuvoton,nuvoton-rng";
            status = "okay";
        };
```

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-5.10.y/drivers/dma/dmatest.c. A test client will also be enabled by enabling "DMA Test Client" function, which is a shortcut to understand the procedure of DMA in kernel.

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

Each PDMA controller has its own device node in device tree, below is the example for PDMA.

```
    dma@b0008000 {
```

"compatible" must set to "nuvoton,nuc980-dma".

```
        compatible = "nuvoton,nuc980-dma";
```

Base address of PDMA is 0xb0080000.

```
        reg = <0xb0008000 0x2000>;
```

Interrupt number of PDMA is 25 and 26.

```
        interrupts = <25 4 1>, <26 4 1>;
```

Set "status" to "enable" to enable PDMA, otherwise set to "disable".

```
        status = "okay";
    };
```

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 evaluation 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 6.1.

```
Device Drivers  --->
    [*] Network device support  --->
        <*> Dummy net driver support
        [*] Ethernet driver support  --->
            [*]   Nuvoton devices
            <*>     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. The NUC980 consists an EBI interface that supports up to 3 memory banks. The EBI mapping address is located at 0x6000_0000 ~ 0x602F_FFFF and the total memory space is 3MB. When system request address hits EBI's memory space, the corresponding EBI chip select signal is assert and EBI state machine operates.To support it in kernel, user needs to enable "NUC980 EBI driver" in "Misc devices" menu page..

```
Device Drivers  --->
     Misc devices  --->
          <*> NUC980 EBI driver
              NUC980 EBI Timing Setting (Timing Slowest)  --->
```

Each EBI bank has its own node in device tree. Here use EBI bank 0 as an example.

```
     ebi@b0010000 {
```

"compatible" should set to "nuvoton,nuc980-ebi" for EBI.

```
     compatible = "nuvoton,nuc980-ebi";
```

The register defines the base address and size of EBI bank, which should set to 0xb0010000, 0xb0010010, and 0xb0010020 for EBI bank 0, 1, and 2.

```
     reg = <0xb0010000 0x1000>;
```

The EBI controller provides a flexible timing control for different external device. "EBI_TIMING_FASTEST","EBI_TIMING_VERYFAST","EBI_TIMING_FAST","EBI_TIMING_NORMAL" , "EBI_TIMING_SLOW","EBI_TIMING_VERYSLOW" and EBI_TIMING_SLOWEST are the timing attribute of the EBI bank. There is an ioctl command defined in uapi/misc/nuc980ebi.h that user application can issue an ioctl commands to control timing in different external devices.

EBI_IOC_SET

Set timing for different external devices. Demonstrate how to map an external SRAM to 0x60000000 memory spaces, as follows.

```
     int fd;
     unsigned char *pEbiBuffer;
     unsigned long uEbiSize;
     struct nuc980_set_ebi ebi;

     fd = open("/dev/ebi",O_RDWR);
     if(fd < 0)
          printf("open ebi error\n");

     ebi.bank = 0;
     ebi.base = 0x60000000;

     ebi.busmode = EBI_OPMODE_NORMAL;
     ebi.CSActiveLevel = EBI_CS_ACTIVE_LOW;
     ebi.width = EBI_BUSWIDTH_16BIT;
     ebi.timing = EBI_TIMING_SLOWEST;
     ioctl(fd, EBI_IOC_SET, &ebi);
```

```
    uEbiSize = 2 * 1024 ; //2K
    pEbiBuffer = mmap(NULL, uEbiSize, PROT_READ|PROT_WRITE, MAP_SHARED,
fd, 0);
    if (pEbiBuffer == MAP_FAILED) {
        printf("mmap() failed\n");
        exit(0);
    }
```

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. This driver can be enabled with following kernel configuration.

```
Device Drivers  --->
    Misc devices  --->
        <*> NUC980 Enhance Timer (ETIMER) support
```

Here describes the etimer0 related node in NUC980 device tree.

```
    etimer0: etimer0@b0050000 {
```

The base address of timer is 0xb0050000 + 0x1000 * x / 2 where x = 0, 2, and 0xb0050000 + 0x1000 * (x – 1) / 2 where x = 1, 3.

```
        reg = <0x0 0xb0050000 0x0 0x100>;
```

“compatible” should set to “nuvoton,nuc980-timer”, and register base address set to the address just mentioned.

```
    compatible = "nuvoton,nuc980-timer";
```

Interrupt number for timer 0~3 are 16, 17, 30, and 31 respectively.

```
    interrupts = <16 4 1>;
```

The “port-number” should be equal to the timer number.

```
    port-number = <0>;
```

status configure the timer status after system boot up, could be ether “okay” or “disabled”.

```
    status = "disabled";
```

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.

There are several ioctl commands defined in uapi/misc/nuc980_timer.h that user application can issue these ioctl commands to control timer to operate in different modes.

- TMR_IOC_CLKLXT

  Switch timer peripheral clock source to LXT.

- TMR_IOC_CLKHXT

Switch timer peripheral clock source to HXT.

- TMR_IOC_STOP

  Stop timer.

- TMR_IOC_PERIODIC

  Start timer to operate in periodic mode with specified frequency.

- TMR_IOC_PERIODIC_FOR_WKUP

  Start timer to operate in periodic mode and enable timer timeout event to wake up system.

- TMR_IOC_TOGGLE

  Start timer to operate in toggle output mode with specified frequency.

- TMR_IOC_EVENT_COUNTING

  Start timer to operate in event counting mode. User application can read the event counter with read() function.

- TMR_IOC_FREE_COUNTING

  Start timer to operate in free counting mode. User can read the calculated duration with the read() function.

- TMR_IOC_TRIGGER_COUNTING

  Start timer to operate in trigger counting mode. User can read the calculated duration with the read() function.

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
```

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 entering Power-down mode, all inserted cards will be deactivate. User application needs to activate the cards in order to do transaction.

SDH

The NUC980 contains one SDH interfaces that can connect with SD memory card, SDIO device, or eMMC. The SDH interface supports SD clock rate up to 52 MH.

The kernel options that enable SDH controller support are list below.

```
Device Drivers  --->
  <*> MMC/SD/SDIO card support  --->
    <*>   MMC block device driver
    <*>   Nuvoton NUC980 SD Card support
```

Here introduces the device tree node describes the attribute of NUC980 SDH controller.

```
     sdh@b0018000 {
```

“compatible” must set to “nuvoton,nuc980-sdh”.

```
          compatible = "nuvoton,nuc980-sdh";
```

“reg” defines the base address and size of SDH control register. Should set to 0xb0180000

```
          reg = <0xb0018000 0x1000>;
```

“interrupts” defines the interrupt number, should set to 27

```
          interrupts = <27 4 1>;
```

Set “okay” to enable NUC980 SDH controller, otherwise set to “disabled”.

```
          status = "okay";
     };
```

CCAP

The NUC980 contains duo CMOS sensor interfaces. The driver that supports these interfaces is:

drivers/media/platform/nuc980-ccap.c

The kernel options that enable NUC980 CMOS sensor interface support are listed below.

```
Device Drivers  --->
  <*> I2C support
  I2C Hardware Bus support  --->
     <*> GPIO-based bitbanging I2C
  <*> Multimedia support  --->
    [*]   Cameras/video grabbers support
    [*]   Media Controller API
    [*]   V4L2 sub-device userspace API
    [*]   V4L platform devices  --->
        <*>   NUC980 camera controller support
    [*]   Memory-to-memory multimedia devices  --->
```

The CMOS sensor driver also needs to be enabled. Below is an example enables HiMax HM1055 series sensor.

```
Device Drivers  --->
  <*> Multimedia support  --->
```

```
    I2C Encoders, decoders, sensors and other helper chips  --->
      <*> HiMax HM1055 sensor support
```

Here is an example of CMOS sensor interface device node.

```
     Ccap0@b0024000 {
```

The “compatible” should set to “nuvoton,nuc980-ccap”. And base address is 0xb0024000 for interface 0 and 0x40140000 for interface 1. The interrupt of interface 0 is 14 and interface 1 is 33.

```
          compatible = "nuvoton,nuc980-ccap";
          reg = <0xb0024000 0x1000>;
          interrupts = <14 4 1>;
          status = "disabled";
```

Below are the description of the CMOS sensor and can be different from sensor to senor. Please refer to the sensor’s spec for the sensor.

```
          port {
                    /* Parallel bus endpoint */
                    ccap0_1: endpoint {
                         remote-endpoint = <&hm1055_0>;
                         hsync-active = <0>;     /* Active low */
                         vsync-active = <0>;     /* Active low */
                         pclk-sample = <1>;      /* Rising */
                    };
               };
          };

          i2c_gpio0: i2c-gpio-0 {
               compatible = "i2c-gpio";
               sda-gpios=<&gpio 0x27 (GPIO_ACTIVE_HIGH | GPIO_OPEN_DRAIN)>;
               scl-gpios=<&gpio 0x25 (GPIO_ACTIVE_HIGH | GPIO_OPEN_DRAIN)>;
               i2c-gpio,delay-us = <20>;        /* ~100 kHz */
               #address-cells = <1>;
               #size-cells = <0>;
               status = "disabled";
               hm1055@24 {
                    compatible = "himax,hm1055";
                    reg = <0x24>;
                    pinctrl-names = "default";
                    pinctrl-0 = <&pinctrl_ccap0>;
                    reset-gpios = <&gpio 0x47 GPIO_ACTIVE_LOW>; /* PC7 */
                    powerdown-gpios = <&gpio 0x21 GPIO_ACTIVE_HIGH>; /* PB1
*/
                    port-number = <0>;
                    frequency = <24000000>;
```

```
                    port {
                         hm1055_0: endpoint {
                              remote-endpoint = <&ccap0_1>;
                         };
                    };
                        };
          };
```

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)
          [*]   Character device (/dev/gpiochipN) support
              Memory mapped GPIO drivers  --->
               <*>   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 can also 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);
}
```

Below is a reference of the pinctrl and GPIO node defined in device tree for NUC980. The “compatible” should set to “nuvoton,nuc980-gpio” provides the base address of system control registers to pinctrl driver.

```
     gpio: gpio@b004000 {
          compatible = "nuvoton,nuc980-gpio";
```

The register base address of GPIO ports are adjacent to each other.

```
          gpio: gpio@b0004000 {
               compatible = "nuvoton,nuc980-gpio";
               reg = <0xb0004000 0x1000>;
               interrupts = <57 4 1>;
               port-number = <0>;
               map-addr = <0xf0004000>;
               pinctrl-names = "default";
               pinctrl-0 = <&pinctrl_gpio>;
               #gpio-cells = <2>;
               gpio-controller;
               interrupt-controller;
```

```
                #interrupt-cells = <2>;
                ...
```

Set “eint0-config” to <0 0 0> for enable PA1 interrupt and set PA1 to both edge trigger

```
                eint0-config = <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 NUC980 contains four I²C interfaces. The following lists the options that needs to enable to support NUC980 I²C function:

```
Device Drivers  --->
     <*> I2C support  --->
          I2C Hardware Bus support  --->
               <*> NUC980 I2C Driver for Port 0
               <*> NUC980 I2C Driver for Port 1
               <*> NUC980 I2C Driver for Port 2
               <*> NUC980 I2C Driver for Port 3
```

If the I²C function support is selected in kernel configuration, kernel will use build in I²C interface of NUC980 to communicate with other device.

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
```

Here is a device node example for NUC980.

```
     i2c0: i2c0@b0080000 {
```

“compatible” must set to “nuvoton, nuc980-i2c0”.

```
          compatible = "nuvoton,nuc980-i2c0";
```

Register base address of I2C0~3 are 0xb0080000 + 0x1000 * X, where X is the I2C port number.

```
          reg = <0xb0080000 0x1000>;
```

Interrupt numbers for port 0 ~ 3 are 53, 54, , 47, and 48.

```
          interrupts = <53 4 1>;
```

“bus_num” defines the I²C port number.

```
            bus_num = <0>;
```

"bus_freq" defines the I²C bus clock frequency. Could be 100 kHz, 400 kHz, and 1000 kHz.

```
            bus_freq = <100000>;
            status = "disable";
        };
```

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  --->
        <*>   Caching block device access to MTD devices
        NAND  --->
            <*> Raw/Parallel NAND Device Support  --->
                -*-   Nuvoton NUC980 MTD NAND
```

Here is the device node sample that describes the attribute of NAND interface.

```
    fmi@b0019000 {
```

"compatible" should set to "nuvoton,nuc980-fmi", "nand". Set "okay" to enable NUC980 NAND controller, otherwise set to "disabled".

```
            compatible = "nuvoton,nuc980-fmi";
            status = "okay";
```

The remaining attributes are: "nand-ecc-mode" set the ECC mode and should set to "hw_oob_first". The "nand-ecc-algo" is "bch", "nand-bus-width" defines the bus width and is always 8 for NUC980. The "nand-ecc-strength" can be 8, 12, or 24. The value should set according to the NAND Flash's requirement. "nand-ecc-step-size" is 512 for T8, T12 and is 1024 for T24. "nand-on-flash-bbt" attribute tells Linux kernel to use the on chip bad block table.

```
            nand-ecc-mode = "hw_oob_first";
            nand-ecc-algo = "bch";
            nand-bus-width = <8>;
            nand-ecc-strength = <8>;
            nand-ecc-step-size = <512>;
            nand-on-flash-bbt;
```

The partition table should be a sub-node of the Flash node and should be named "partitions". "compatible" must be "fixed-partitions".

```
            partitions {
                compatible = "fixed-partitions";
```

"#address-cells" and "#size-cells" must both be present in the partitions sub-node of the Flash device. There are two valid values for both:

<1>: for partitions that require a single 32-bit cell to represent their size/address (aka the value is below 4 GiB)

<2>: for partitions that require two 32-bit cells to represent their size/address (aka the value is 4 GiB or greater).

```
                #address-cells = <1>;
                #size-cells = <1>;
```

“reg” is the partition's offset and size within the Flash. “label” is the name for this partition. If omitted, the label is taken from the node name (excluding the unit address). “read-only” is a hint to Linux that this partition should only be mounted read-only.

```
                uboot@0 {
                     label = "uboot";
                     reg = <0x0000000 0x200000>;
                     read-only;
                };
                kernel@200000 {
                     label = "kernel";
                     reg = <0x200000 0x1400000>;
                };
                user@1600000 {
                     label = "user";
                     reg = <0x1600000 0x6480000>;
                };
          };
```

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” stands for those unused PWM channels.

File Name	Purpose
period	Control cycle, where 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, where 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, which 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

```
Device Drivers  --->
     [*] Pulse-Width Modulation (PWM) Support  --->
          <*>   NUC980 PWM0 support
          <*>   NUC980 PWM1 support
```

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, and their meaning is listed in the following table.

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 --->
```

The usage of this driver is described as follows.

1. Load driver module by using insmod command.

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

2. Enable wireless interface.

```
$ ifconfig wlan0 up
```

3. Use wpa_supplicant utility to connect with wireless AP.

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

4. 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

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
            [*]     Console on NUC980 serial port
            [*]    NUC980 UART1 support
            [*]      Enable UART1 CTS wake-up function
            [*]    NUC980 UART2 support
            [*]      Enable UART2 CTS wake-up function
            [*]    NUC980 UART3 support
            [*]    NUC980 UART4 support
            [*]      Enable UART4 CTS wake-up function
            [*]    NUC980 UART5 support
            [*]    NUC980 UART6 support
            [*]      Enable UART6 CTS wake-up function
            [*]    NUC980 UART7 support
            [*]    NUC980 UART8 support
            [*]      Enable UART8 CTS wake-up function
            [*]    NUC980 UART9 suppor
            [*]    NUC980 UART10 support
            [*]      Enable UART10 CTS wake-up function
```

Here is a device node example for NUC980 UART.

```
    uart0:serial@b0070000 {
```

“compatible” should set to “nuvoton,nuc980-uart”.

```
        compatible = "nuvoton,nuc980-uart";
```

The register base address for UART ports are 0xb0070000 + 0x10000 * X, where X is the port number for UART port 0~9.

```
        reg = <0x0 0xb0070000 0x0 0x10000>;
```

Interrupt numbers for port 0 ~ 9 are 36, 37, 38, 43, 39, 44, 40, 45, 41 and 46.

The “port-number” for UART port 0~9 are 0~9.

User can set “pdma-enable” to 1 to enable UART PDMA mode or set it to 0 to disable UART PDMA mode.

```
        interrupts = <GIC_SPI 59 IRQ_TYPE_LEVEL_HIGH>;
        port-number = <0>;
        pdma-enable = <0>;
        status = "okay";
    };
```

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 by transfer mode (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 spi_nor_dev_ids structure in drivers/mtd/spi-nor/core.c file.

```
static const struct spi_device_id spi_nor_dev_ids[] = {
        /*
         * Allow non-DT platform devices to bind to the "spi-nor" modalias,
and
         * hack around the fact that the SPI core does not provide uevent
         * matching for .of_match_table
         */
        {"spi-nor"},

        /*
         * Entries not used in DTs that should be safe to drop after
replacing
         * them with "spi-nor" in platform data.
         */
        {"s25sl064a"},  {"w25x16"},     {"m25p10"},     {"m25px64"},

        /*
```

```
         * Entries that were used in DTs without "jedec,spi-nor" fallback
and
         * should be kept for backward compatibility.
         */
        {"at25df321a"}, {"at25df641"},  {"at26df081a"},
        {"mx25l4005a"}, {"mx25l1606e"}, {"mx25l6405d"}, {"mx25l12805d"},
        {"mx25l25635e"},{"mx66l51235l"},
        {"n25q064"},    {"n25q128a11"}, {"n25q128a13"}, {"n25q512a"},
        {"s25fl256s1"}, {"s25fl512s"},  {"s25sl12801"}, {"s25fl008k"},
        {"s25fl064k"},
        {"sst25vf040b"},{"sst25vf016b"},{"sst25vf032b"},{"sst25wf040"},
        {"m25p40"},     {"m25p80"},     {"m25p16"},     {"m25p32"},
        {"m25p64"},     {"m25p128"},
        {"w25x80"},     {"w25x32"},     {"w25q32"},     {"w25q32dw"},
        {"w25q80bl"},   {"w25q128"},    {"w25q256"},

        /* Flashes that can't be detected using JEDEC */
        {"m25p05-nonjedec"},         {"m25p10-nonjedec"},         {"m25p20-
nonjedec"},
        {"m25p40-nonjedec"},         {"m25p80-nonjedec"},         {"m25p16-
nonjedec"},
        {"m25p32-nonjedec"},         {"m25p64-nonjedec"},         {"m25p128-
nonjedec"},

        /* Everspin MRAMs (non-JEDEC) */
        { "mr25h128" }, /* 128 Kib, 40 MHz */
        { "mr25h256" }, /* 256 Kib, 40 MHz */
        { "mr25h10" },  /*   1 Mib, 40 MHz */
        { "mr25h40" },  /*   4 Mib, 40 MHz */

        { },
};
```

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
          < > 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 by transfer mode (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  --->
          NAND  --->
               <*> SPI NAND device Support  ----
```

If user wants to use new SPI NAND Flash device which is not included in BSP, it's necessary to modify SPI NAND table in MTD driver and it can be identified correctly.

Please modify the chips field of spinand_manufacturer structure. For instance, you'd like to use Winbond SPI NAND. Please modify drivers/mtd/nand/spi/winbond.c and fill in the SPI NAND information in spinand_info data structure.

```
static const struct spinand_info winbond_spinand_table[] = {
        SPINAND_INFO("W25M02GV",
                     SPINAND_ID(SPINAND_READID_METHOD_OPCODE_DUMMY, 0xab),
                     NAND_MEMORG(1, 2048, 64, 64, 1024, 20, 1, 1, 2),
                     NAND_ECCREQ(1, 512),
                     SPINAND_INFO_OP_VARIANTS(&read_cache_variants,
                                              &write_cache_variants,
                                              &update_cache_variants),
```

```
                    0,
                    SPINAND_ECCINFO(&w25m02gv_ooblayout, NULL),
                    SPINAND_SELECT_TARGET(w25m02gv_select_target)),
        SPINAND_INFO("W25N02JWZEIF",
                    SPINAND_ID(SPINAND_READID_METHOD_OPCODE_DUMMY,   0xbf,
0x22),
                    NAND_MEMORG(1, 2048, 64, 64, 2048, 20, 1, 1, 1),
                    NAND_ECCREQ(1, 512),
                    SPINAND_INFO_OP_VARIANTS(&read_cache_variants,
                                             &write_cache_variants,
                                             &update_cache_variants),
                    0,
                    SPINAND_ECCINFO(&w25m02gv_ooblayout, NULL))
...
        SPINAND_INFO("W25N512GVEIR",
                    SPINAND_ID(SPINAND_READID_METHOD_OPCODE_DUMMY,   0xaa,
0x20),
                    NAND_MEMORG(1, 2048, 64, 64, 512, 20, 1, 1, 1),
                    NAND_ECCREQ(1, 512),
                    SPINAND_INFO_OP_VARIANTS(&read_cache_variants,
                                             &write_cache_variants,
                                             &update_cache_variants),
                    0,
                    SPINAND_ECCINFO(&w25m02gv_ooblayout, NULL)),
};
```

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
          <*>       Support for NUC980 EHCI (USB 2.0)
          <*>       Generic EHCI driver for a platform device
          <*>     OHCI HCD support
          <*>       Support for NUC980 OHCI (USB 1.1)
          <*>       Generic OHCI driver for a platform device
```

Here are the device nodes that describe the USB EHCI (USB 2.0) in device tree.

```
                usbh_ehci@b0015000 {
```

“compatible” must set to “nuvoton,nuc980-ehci”.

```
                compatible = "nuvoton,nuc980-ehci";
```

EHCI register base address is 0xb0015000.

```
                reg = <0xb0015000 0x1000>;
```

EHCI interrupt number is 23.

```
                interrupts = <23 4 1>;
```

USB Host pinctrl provides the following selections:

```
                usbh {
                     pinctrl_usbh_ppwr_ovc: usbh-ppwr-ovc {
                          nuvoton,pins =
                               <4 0xa 1 0
                                4 0xc 1 0>;
                     };
                     pinctrl_usbh_ppwr: usbh-ppwr {
                          nuvoton,pins =
                               <4 0xc 1 0>;
                     };
                     pinctrl_usbh_ovc: usbh-ovc{
                          nuvoton,pins =
                               <4 0xa 1 0>;
                     };
                };              pinctrl-names = "default";
```

The default selection is “pinctrl_usbh_ppwr_ovc”, which means use PE.10 for USBH_OVC and PE.12 for USBH_PWREN. If removed the “pinctrl-0”, USB Host driver will not use PE.10 and PE.12 and user can freely use these two pins.

```
                pinctrl-0 = <&pinctrl_usbh_ppwr_ovc>;
```

“ov_active” is used to specify the active level of USBH_OVC input. 0 is for “active low” and 1 is for “active high”.

For example, if “active low” is specified, a low level presented on USBH_OVC pin will result in over-current and subsequently disabled EHCI by hardware.

```
                ov_active = <0>;   /* over-current active level, 1: active
high; 0: active low */
```

“pm_vbus_off” is used to specify turn off VBUS or not when chip entering Power-down mode.

```
           pm_vbus_off = <0>;   /* 1: turn-off VBUS when suspend; 0: keep
VBUS power */
                status = "okay";
           };
```

Here are the device nodes that describe the USB OHCI (USB 1.1) in device tree.

```
           usbh_ohci@b0017000{
```

“compatible” must set to “nuvoton,nuc980-ohci”.

```
            compatible = "nuvoton,nuc980-ohci";
```

OHCI register base address is 0xb0017000.

```
            reg = <0xb0017000 0x1000>;
```

OHCI interrupt number is 24.

```
            interrupts = <24 4 1>;
```

The followings are for OHCI Lite 0 ~ 5, which are all disabled by default. Modify status from “disabled” to “okay” can enable it.

```
        usbh_lite0@b0017000{
            compatible = "nuvoton,nuc980-usbh-lite0", "usbh-ohci";
            reg = <0xb0017000 0x1000>;
            interrupts = <24 4 1>;
            pinctrl-names = "default";
            pinctrl-0 = <&pinctrl_lite0_PB10_PB9>;
            status = "disabled";
        };

        usbh_lite1@b0017000{
            compatible = "nuvoton,nuc980-usbh-lite1", "usbh-ohci";
            reg = <0xb0017000 0x1000>;
            interrupts = <24 4 1>;
            pinctrl-names = "default";
            pinctrl-0 = <&pinctrl_lite1_PE1_PE0>;
            status = "disabled";
        };

        usbh_lite2@b0017000{
            compatible = "nuvoton,nuc980-usbh-lite2", "usbh-ohci";
            reg = <0xb0017000 0x1000>;
            interrupts = <24 4 1>;
            pinctrl-names = "default";
            pinctrl-0 = <&pinctrl_lite2_PE3_PE2>;
            status = "disabled";
        };

        usbh_lite3@b0017000{
            compatible = "nuvoton,nuc980-usbh-lite3", "usbh-ohci";
            reg = <0xb0017000 0x1000>;
            interrupts = <24 4 1>;
            pinctrl-names = "default";
            pinctrl-0 = <&pinctrl_lite3_PE5_PE4>;
```

```
            status = "disabled";
        };

        usbh_lite4@b0017000{
            compatible = "nuvoton,nuc980-usbh-lite4", "usbh-ohci";
            reg = <0xb0017000 0x1000>;
            interrupts = <24 4 1>;
            pinctrl-names = "default";
            pinctrl-0 = <&pinctrl_lite4_PE7_PE6>;
            status = "disabled";
        };

        usbh_lite5@b0017000{
            compatible = "nuvoton,nuc980-usbh-lite5", "usbh-ohci";
            reg = <0xb0017000 0x1000>;
            interrupts = <24 4 1>;
            pinctrl-names = "default";
            pinctrl-0 = <&pinctrl_lite5_PE9_PE8>;
            status = "disabled";
        };   };
```

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
               USB Gadget precomposed configurations --->
               <M> Mass Storage Gadget
```

Here are the device nodes that describe the USB device in device tree.

```
usbdev@b0016000{
```

“compatible” must set to “nuvoton,nuc980-usbdev”.

```
compatible = "nuvoton,nuc980-usbdev";
```

Register base address is 0xb0016000.

```
reg = <0xb0016000 0x1000>;
```

Interrupt number is 29.

```
interrupts = <29 4 1>;
```

After compiling kernel, three driver module files will be outputted. (fs/configfs/configfs.ko, drivers/usb/gadget/libcomposite.ko, drivers/usb/gadget/function/usb_f_mass_storage.ko and drivers/usb/gadget/legacy/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 usb_f_mass_storage.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

Timeout period is 2.03 seconds by default 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 function, please enable the following items. To support watchdog timer wake-up function, please enable “NUC980 WDT wake-up support” item.

```
Device Drivers  --->
     [*] Watchdog Timer Support  --->
          [*]   Update boot-enabled watchdog until userspace takes over
          (1)   Timeout value for opening watchdog device
          <*>   Nuvoton NUC980 Watchdog Timer
          <*>     NUC980 WDT wake-up support
```

Below is the device node for WDT.

```
     wdt: wdt@b0040000 {
```

The “compatible” attribute should set to “nuvoton,nuc980-wdt”.

```
     compatible = "nuvoton,nuc980-wdt";
```

Register base address is 0xb0040000.

```
     reg = <0xb0040000 0x100>;
```

Interrupt number of WDT is 1.

```
     interrupts = <1 4 1>;
```

Set “status” to “okay” to enable WDT, and “disabled” to disable WDT.

```
     status = "okay";
```

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.

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

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

Below is the device node for WWDT.

```
     Wwdt: wwdt@b0040100 {
```

The "compatible" attribute should set to "nuvoton,nuc980-wwdt".

```
     compatible = "nuvoton,nuc980-wwdt";
```

Register base address is 0xb0040100.

```
     reg = <0xb0040100 0x100>;
```

Interrupt number of WWDT is 2.

```
     interrupts = <2 4 1>;
```

Set "status" to "okay" to enable WWDT, and "disabled" to disable WWDT.

```
     status = "okay";
```

RTC

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

```
Device Drivers --->
     [*] Real Time Clock --->
          <*> NUC980 RTC driver
```

Here is the device node describes RTC in device tree.

```
     rtc: rtc@b0041000 {
```

"compatible" must set to "nuvoton,nuc980-rtc".

```
          compatible = "nuvoton,nuc980-rtc";
```

"reg" defines the base address and size of RTC register map. The base address is 0x b0041000 for RTC. And the RTC interrupt number is 15.

```
          reg = <0xb0041000 0x1000>;
          interrupts = <15 4 1>;
          status = "okay";
```

```
    };
```

CAN

NUC980 series support 2 CAN ports which can be configured individually. Please follow the instruction below to enable CAN port function.

```
-*- 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
                         [*]  NUC980 CAN1 support
```

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

Below is the node describes CAN attribute in device tree.

```
     can0: can@b00a0000 {
```

“compatible” must set to “nuvoton,nuc980-can0”.

```
          compatible = "nuvoton,nuc980-can0";
```

“reg” defines the base address and size of CAN register map. They The base address is 0xb00a1000 + 0x1000 * X, where X is the port number.

```
          reg = <0xb00a0000 0x1000>;
```

Interrupt numbers for port 0 ~ 3 are 58, 59, 62 and 42.

```
          interrupts = <58 4 1>;
          status = "disabled";
     };
```

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
```

5.2 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_iot_defconfig” command to do that (If it is under buildroot, enter “make nuvoton_nuc980_iot_defconfig”). Sometimes if system cannot boot up, user can load the default configuration to recovery kernel to safe status.

5.3 Linux Kernel Configuration

This section introduces the configuration to enable kernel function according to different NUC980 driver or functions. In Buildroot environment, enter “make linux-menuconfig” to edit Linux kernel configuration.

5.3.1 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 5.3.3) 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 booting 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 5.3.3) 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 5.3.3) 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.

User needs to enable network function (please refer to 5.3.2) and the following item additionally.

```
 [*] Networking support  --->
     Networking options  --->
           <*>  Packet socket
           [*]  TCP/IP networking
           [*]  IP: multicasting
           [*]  IP: kernel level autoconfiguration
           [*]     IP: DHCP support
           [*]     IP: BOOTP support
           [*]     IP: RARP support
```

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
```

5.3.2 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
```

To enable EMAC support in Linux, please enable following options in configuration interface.

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

```
      [*] Nuvoton devices
      <*>   Nuvoton NUC980 Ethernet MAC 0
      <*>   Nuvoton NUC980 Ethernet MAC 1
```

Each EMAC controller has its own device node in device tree. The base address of EAMC controller, set to b0012000 for EMAC0 and b0022000 for EMAC1.

```
          emac0@b0012000 {
```

“compatible” must set to “nuvoton,nuc980-emac0” for EMAC0 and “nuvoton,nuc980-emac1” for EMAC1.

```
          compatible = "nuvoton,nuc980-emac0";
```

“reg” defines the base address and size of EMAC control registers. The base address is 0xb0012000 for EAMC0 and 0xb0022000 for EMAC1..

```
          reg = <0xb0012000 0x1000>;
```

Interrupt numbers are 21 and 19 for EMAC0, 22 and 20 for EMAC1.

```
          interrupts = <21 4 1>, <19 4 1>;
```

Set “status” to “enable” to enable EMAC, otherwise set to “disable”.

```
          status = "okay";
     };
```

5.3.3 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)  --->
```

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
```

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
```

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.

5.3.4 **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, issuing following under shell can put the system into 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 entering 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

5.4 Linux Kernel Compilation

After the kernel configuration is finished, type “make” command (type “make linux-rebuild” if under buildroot) to compile kernel in linux-5.10.y 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
```

6 LINUX USER APPLICATIONS

6.1 Sample Applications

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

Directory	Description
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. *

*. 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.

7 REVISION HISOTRY

Date	Revision	Description
2023.09.01	1.01	Correcting spelling, grammar, and formatting of the document
2023.07.26	1.00	Initial version.

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.