Create Root Filesystem on Flash

Application Note for NUC970/NUC980 Series

Document Information

Abstract	This application note introduces how to create a root filesystem on Flash.
Apply to	NUC970/NUC980 series.

For additional information or questions, please contact: Nuvoton Technology Corporation.

www.nuvoton.com

Table of Contents

1 Overview

For the NUC970/980 series microprocessor, Linux default configuration root filesystem is operated on RAM so that the system can read/write data in a short time. However, RAM filesystem cannot store any data when powered off. In order to keep data in root filesystem, mounting file system on Flash is an easier way to use.

There are two types of Flash on evaluation boards, NAND Flash and NOR Flash.

NAND Flash has some bad block problems. Bad block is factory-generated inevitable defection, and it will cause some Flash block unable to be read. NAND Flash normally uses YAFFS2 format because YAFFS2 has bad block management function. NOR Flash does not have bad block problem and it usually uses jffs2 format.

2 Create YAFFS2 Format Root Filesystem on NAND Flash

2.1 Modify Buildroot Configuration

Follow the steps below to modify Buildroot configuration.

```
~/Buildroot$ make menuconfig
```

```
> Filesystem images
[ ] cpio the root filesystem (for use as an initial RAM filesystem)
[ ] initial RAM filesystem linked into linux kernel
.
.
.
[*] tar the root filesystem
[*] yaffs2 root filesystem
```

1. Cancel the default configuration which create root file system on RAM.
   - Cancel cpio the root filesystem (for use as an initial RAM filesystem)
   - Cancel initial RAM filesystem linked into linux kernel
2. Separate Linux kernel and root filesystem from uImage and tar the root filesystem with yaffs2 format.
   - Tick tar the root filesystem
   - Tick yaffs2 root filesystem

2.2 Modify Linux Kernel Configuration

Follow the steps below to modify Linux kernel configuration.

```
~/Buildroot$ make linux-menuconfig
```

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

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

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

1. Cancel RAM filesystem and RAM disk support.
   - Cancel Initial RAM filesystem and RAM disk (initramfs/initrd) support
2. Use MTD partition command in u-boot.
   - Tick Command line partition table parsing
3. Ensure that Linux kernel supports YAFFS2 format:
   - Tick yaffs2 file system support
   - Tick Autoselect yaffs2 format
   - Tick Enable yaffs2 xattr support

2.3 Modify mtdblock Auto Mount

Some particular evaluation boards have "*mdev*" file determining which mtdblock will be auto mounted. You can check if the path */NUC970_Buildroot/board/nuvoton/rootfs-xxx/etc* has *mdev* file or not. The default setting is auto mount mtdblock1 to mtdblock9. In other words, it will mount u-boot and Linux kernel and will cause error. To avoid this problem, modify *mdev* with the step below. This demo is operated on NuMaker-RTU-NUC980 (Chili Board).

```
user@ubuntu:~/NUC970_Buildroot/board/nuvoton/rootfs-chili/etc$ gedit mdev
```

Modify the source code below:

```
mtdblock([3-9]+)        0:0 660 */sbin/automount.sh $MDEV X${ACTION}
```

2.4 Modify u-boot env.txt

env.txt is a u-boot environment variable file. u-boot can read data to RAM from Flash, boot kernel from RAM.

The variable **bootcmd** will be executed when u-boot starts and the variable **bootdelay** has been defined.

The following command will read the kernel image from Flash to RAM and boot from RAM address 7fc0.

```
loadkernel=nand read 7fc0 200000 1400000
bootcmd=run loadkernel;bootm 7fc0;
```

u-boot can use MTD to partition Flash into three mtdblocks and create YAFFS2 format root filesystem on mtdblock2. The Flash address and size below can be changed to anywhere you want.

mtdblock0 stores u-boot image and its Flash offset is from 0x0 to 0x200000.

mtdblock1 stores kernel image and its Flash offset is from 0x200000 to 0x1600000.

mtdblock2 stores root filesystem image and its Flash offset is from 0x1600000 to the end.

```
bootargs=noinitrd root=/dev/mtdblock2 rootfstype=yaffs2 rootflags=inband-tags
console=ttyS0 rdinit=/sbin/init mem=64M mtdparts=nand0:0x200000@0x0(u-
boot),0x1400000@0x200000(kernel),-(user) ignore_loglevel
```

The whole *env.txt* content is shown below:

```
baudrate=115200
bootdelay=1
stderr=serial
stdin=serial
stdout=serial
loadkernel=nand read 7fc0 200000 1400000
```

```
bootcmd=run loadkernel;bootm 7fc0;
bootargs=noinitrd root=/dev/mtdblock2 rootfstype=yaffs2 rootflags=inband-tags
console=ttyS0 rdinit=/sbin/init mem=64M mtdparts=nand0:0x200000@0x0(u-
boot),0x1400000@0x200000(kernel),-(user) ignore_loglevel
```

2.5 Program Image to Evaluation Board

After compiling, program the following files to the evaluation board. Each image has particular address and these addresses depend on *env.txt.*

Image Name	Image Type	Address
u-boot-spl.bin	Uboot	0x200
u-boot.bin	Data	0x100000
env.txt	Env	0x80000
uImage	Data	0x200000
rootfs_yaffs2	Data	0x1600000

Table 1-1 NAND Flash Program Image

2.6 Check Root Filesystem

Type the following command on device terminal to check if the root filesystem has been mounted on or not.

Compare the MTD partition you defined in env.txt with the message after you type the command below.

```
#cat /proc/mtd
dev:    size   erasesize  name
mtd0: 00200000 00020000 "u-boot"
mtd1: 01200000 00020000 "kernel1"
mtd2: 07200000 00020000 "user"
```

3 Create JFFS2 Format Root Filesystem on NOR Flash

3.1 Modify Buildroot Configuration

Follow the steps below to modify Buildroot configuration.

```
~/Buildroot$ make menuconfig
```

```
    > Filesystem images
    [ ] cpio the root filesystem (for use as an initial RAM filesystem)
    [ ] initial RAM filesystem linked into linux kernel
    [*] tar the root filesystem
    [*] jffs2 root filesystem
          Flash Type(Parallel flash with 128 kb erase size ->
            (X) Parallel flash with 128 kb erase size
```

1. Cancel the default configuration that creates root file system on RAM.
   - Cancel cpio the root filesystem (for use as an initial RAM filesystem)
   - Cancel initial RAM filesystem linked into linux kernel
2. Separate Linux kernel and root filesystem from uImage and tar the root filesystem with JFFS2 format.
   - Choose Flash Type to Parallel Flash with 128 KB erase size.
3. Use JFFS2 root filesystem.

3.2 Modify Linux Kernel Configuration

Follow the steps below to modify Linux kernel configuration.

```
~/Buildroot$ make linux-menuconfig
```

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

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

    > File systems > Miscellaneous filesystems
    [*] JFFS2 summary support
      [*] JFFS2 XATTR support
      [*] JFFS2 POSIX Access Control Lists
      [*] JFFS2 Security Labels
```

1. Cancel RAM filesystem and RAM disk support.
   - Cancel Initial RAM filesystem and RAM disk (initramfs/initrd) support.
2. Use MTD partition command in u-boot.
   - Tick Command line partition table parsing.
3. Ensure that Linux kernel supports JFFS2 format:
   - Tick JFFS2 summary support
   - Tick JFFS2 XATTR support
   - Tick JFFS2 POSIX Access Control Lists
   - Tick JFFS2 Security Labels

3.3 Modify mtdblock Auto Mount

Some particular evaluation boards have "*mdev*" file determining which mtdblock will be auto mounted. You can check if the path */NUC970_Buildroot/board/nuvoton/rootfs-xxx/etc* has "*mdev*" file or not. The default setting is auto mount mtdblock1 to mtdblock9. In other words, it will mount u-boot and Linux kernel and will cause error. To avoid this problem, modify "*mdev*" file with the following step. This demo is operated on NuMaker-RTU-NUC980(Chili Board).

```
user@ubuntu:~/NUC970_Buildroot/board/nuvoton/rootfs-chili/etc$ gedit mdev
```

Modify the source code below:

```
mtdblock([3-9]+)        0:0 660 */sbin/automount.sh $MDEV X${ACTION}
```

3.4 Modify JFFS2 Erase Size

The rootfs_jffs2 image cannot be directly programed into an evaluation board. Thus, you can use mkfs tool to generate jffs2.img. Because JFFS2 format needs the minimum erase size of 8K, you should modify setting in Linux driver.

```
user@ubuntu:~ /NUC970_Buildroot/output/build/linux-master/drivers/mtd/spi-nor$ gedit spi-nor.c
```

Find the Flash driver and model number on your Flash and replace SECT_4K with 0. For example, Chili Board's Flash driver is spi-nor and model number is w25q256.

```
    { "w25x05", INFO(0xef3010, 0, 64 * 1024,  1,  SECT_4K) },
    { "w25x10", INFO(0xef3011, 0, 64 * 1024,  2,  SECT_4K) },
    { "w25x20", INFO(0xef3012, 0, 64 * 1024,  4,  SECT_4K) },
    { "w25x40", INFO(0xef3013, 0, 64 * 1024,  8,  SECT_4K) },
    { "w25x80", INFO(0xef3014, 0, 64 * 1024,  16, SECT_4K) },
    { "w25x16", INFO(0xef3015, 0, 64 * 1024,  32, SECT_4K) },
    { "w25x32", INFO(0xef3016, 0, 64 * 1024,  64, SECT_4K) },
    { "w25q32", INFO(0xef4016, 0, 64 * 1024,  64, SECT_4K) },
    { "w25q32dw", INFO(0xef6016, 0, 64 * 1024,  64, SECT_4K | SPI_NOR_DUAL_READ |
SPI_NOR_QUAD_READ) },
    { "w25x64", INFO(0xef3017, 0, 64 * 1024, 128, SECT_4K) },
    { "w25q64", INFO(0xef4017, 0, 64 * 1024, 128, SECT_4K) },
    { "w25q64dw", INFO(0xef6017, 0, 64 * 1024, 128, SECT_4K | SPI_NOR_DUAL_READ |
SPI_NOR_QUAD_READ) },
    { "w25q128fw", INFO(0xef6018, 0, 64 * 1024, 256, SECT_4K | SPI_NOR_DUAL_READ |
SPI_NOR_QUAD_READ) },
    { "w25q80", INFO(0xef5014, 0, 64 * 1024,  16, SECT_4K) },
    { "w25q80bl", INFO(0xef4014, 0, 64 * 1024,  16, SECT_4K) },
    { "w25q128", INFO(0xef4018, 0, 64 * 1024, 256, SECT_4K) },
    { "w25q256", INFO(0xef4019, 0, 64 * 1024, 512,0) },
```

Build Linux kernel.

```
user@ubuntu:~NUC970_Buildroot$ make
```

3.5 Use mkfs Tool to Generate Root Filesystem Image

Install mkfs tool first.

```
$apt-get install mtd-utils
```

After installing successfully and compiling Linux kernel, go to Image folder.

Create rootfs folder to unzip *rootfs.tar*.

```
user@ubuntu:~NUC970_Buildroot/output/images$  mkdir rootfs
user@ubuntu:~NUC970_Buildroot/output/images$  sudo tar -C ./rootfs -xf rootfs.tar
```

Use mkfs to generate *jffs2.img*.

```
user@ubuntu:~/NUC970_Buildroot/output/images$ sudo mkfs.jffs2 -s 0x1000 -e 0x10000 -p
0x800000 -d /home/user/Buildroot/NUC980_Chili/NUC970_Buildroot/output/images/rootfs -o
jffs2.img
```

The mkfs usage is shown below:

```
mkfs [options] [-t <type>] [fs-options] <device> [<size>]
Options:
 -t, --type=<type>  filesystem type; when unspecified, ext2 is used
     fs-options     parameters for the real filesystem builder
     <device>       path to the device to be used
     <size>         number of blocks to be used on the device
 -V, --verbose      explain what is being done;
                      specifying -V more than once will cause a dry-run
 -h, --help         display this help
 -V, --version      display version
```

3.6 Modify u-boot env.txt

env.txt is a u-boot environment variable file. u-boot can read data to RAM from Flash, boot kernel from RAM.

The variable **bootcmd** will be executed when u-boot starts and the variable **bootdelay** has been defined.

The following command will read the kernel image from Flash to RAM and boot from RAM address 7fc0. The difference of command between NAND Flash and NOR Flash is NAND Flash uses **nand read** and the NOR Flash uses **sf read**. **Sf probe** should have been executed first because NOR Flash should be initiated first.

```
setspi=sf probe 0 30000000
loadkernel=sf read 7fc0 200000 600000
bootcmd=run setspi;loadkernel;bootm 7fc0;
```

u-boot can use MTD to partition Flash into three mtdblocks and create JFFS2 format root filesystem on mtdblock2. The Flash address and size listed below can be changed to anywhere you want.

- mtdblock0 stores u-boot image and its Flash offset is from 0x0 to 0x200000.
- mtdblock1 stores kernel image and its Flash offset is from 0x200000 to 0x800000.
- mtdblock2 stores root filesystem image and its Flash offset is from 0x800000 to the end.

```
bootargs=noinitrd root=/dev/mtdblock2 rw rootfstype=jffs2 console=ttyS0 rdinit=/sbin/init
mem=32M mtdparts=m25p80:0x200000@0x0(u-boot),0x600000@0x200000(kernel),-(user)
ignore_loglevel
```

The whole *env.txt* content is shown below:

```
baudrate=115200
bootdelay=1
stderr=serial
stdin=serial
stdout=serial
setspi=sf probe 0 30000000;
loadkernel=sf read 7fc0 200000 600000
bootcmd=run setspi;run loadkernel;bootm 7fc0;
bootargs=noinitrd root=/dev/mtdblock2 rw rootfstype=jffs2 console=ttyS0 rdinit=/sbin/init
mem=32M mtdparts=m25p80:0x200000@0x0(u-boot),0x600000@0x200000(kernel),-(user)
ignore_loglevel
```

3.7 Program Image to Evaluation Board

Program the following Image files to the target board. Each image has particular address and these addresses depend on *env.txt*.

Image Name	Image Type	Address
u-boot.bin	Loader	0xe00000
env.txt	Env	0x80000
uImage	Data	0x200000
jffs2.img	Data	0x800000

Table 1-1 NAND Flash Program Image

3.8 Check Root Filesystem

Type the following command on device terminal to check if the root filesystem has been mounted on or not.

Compare the MTD partition you defined in *env.txt* with the message after you type the command below.

```
#cat /proc/mtd
dev:    size   erasesize  name
mtd0: 00200000 00020000 "u-boot"
mtd1: 00600000 00020000 "kernel"
mtd2: 01200000 00020000 "user"
```

4 Conclusion

Creating root filesystem on Flash can not only store data but also separate u-boot, Linux kernel, and root filesystem. It is easy to control the whole system because you can only update Linux kernel but not the whole image.

YAFFS2 has bad block management and higher read/write speed than JFFS2. However, YAFFS2 is not suitable for NOR Flash because NOR Flash does not have bad block problem. JFFS2 is more suitable than YAFFS2 on NOR Flash.

When you create root file system on Flash, you must partition Flash into several mtdblocks. Partitioning Flash can clearly define every area's function, for example, mtdblock 1 is used to store u-boot, and mtdblock 2 is used to store Linux kernel. Mounting on necessary block on Linux will prevent others from reading or writing important data.

Revision History

Date	Revision	Description
2021.07.19	1.00	1. Initially issued.

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.