

实验一交叉工具链的安装

【实验目的】

了解交叉工具链的编译过程及其使用。

说明：在实验中命令提示符为“\$”表示在主机上运行，“#”表示在目标板上运行

【实验环境】

- 1、ubuntu 12.04 发行版
- 2、FS4412 平台

【实验步骤】

- 1、如果要自己编译工具链，从以下链接下载源码

crosstools-ng 下载地址

<http://ymorin.is-a-geek.org/download/crosstool-ng/>

同时对每一个版本都有相应的补丁我们尽量把这些补丁打上，这些补丁的下载地址是

<http://ymorin.is-a-geek.org/download/crosstool-ng/01-fixes/>

- 2、解压工具链压缩包

```
$ cd ~  
$ mkdir toolchain  
$ cd toolchain
```

将第一天/工具/gcc-4.6.4.tar.xz 拷贝到 toolchain 目录下并解压

```
$ tar xvf gcc-4.6.4.tar.xz
```

- 3、环境变量的添加

修改文件/etc/bash.bashrc 添加如下内容

```
export PATH=$PATH:/home/linux/toolchain/gcc-4.6.4/bin
```

重启配置文件

```
$ source /etc/bash.bashrc
```

4、工具链的测试

```
$ arm-none-linux-gnueabi-gcc -v

Using built-in specs.

COLLECT_GCC=arm-none-linux-gnueabi-gcc

COLLECT_LTO_WRAPPER=/home/david/Exynos4412/toolchain/gcc-4.6.4/bin/./libexec/gcc/arm-arm1176jzfssf-linux-gnueabi/4.6.4/lto-wrapper

Target: arm-arm1176jzfssf-linux-gnueabi

Configured with: /work/builddir/src/gcc-4.6.4/configure --build=i686-build_pc-linux-gnu
--host=i686-build_pc-linux-gnu --target=arm-arm1176jzfssf-linux-gnueabi
--prefix=/opt/TuxamitoSoftToolchains/arm-arm1176jzfssf-linux-gnueabi/gcc-4.6.4
--with-sysroot=/opt/TuxamitoSoftToolchains/arm-arm1176jzfssf-linux-gnueabi/gcc-4.6.4/arm-arm1176jzfssf-linux-gnueabi/sysroot --enable-languages=c,c++ --with-arch=armv6zk
--with-cpu=arm1176jzf-s --with-tune=arm1176jzf-s --with-fpu=vfp --with-float=softfp
--with-pkgversion='crosstool-NG hg+default-2685dfa9de14 - tc0002' --disable-sjlj-exceptions
--enable-__cxa_atexit --disable-libmudflap --disable-libgomp --disable-libssp
--disable-libquadmath --disable-libquadmath-support
--with-gmp=/work/builddir/arm-arm1176jzfssf-linux-gnueabi/buildtools
--with-mpfr=/work/builddir/arm-arm1176jzfssf-linux-gnueabi/buildtools
--with-mpc=/work/builddir/arm-arm1176jzfssf-linux-gnueabi/buildtools
--with-ppl=/work/builddir/arm-arm1176jzfssf-linux-gnueabi/buildtools
--with-cloog=/work/builddir/arm-arm1176jzfssf-linux-gnueabi/buildtools
--with-libelf=/work/builddir/arm-arm1176jzfssf-linux-gnueabi/buildtools
--with-host-libstdcxx='-static-libgcc -Wl,-Bstatic,-lstdc++, -Bdynamic -lm' --enable-threads=posix
--enable-target-optspace --without-long-double-128 --disable-nls --disable-multilib
--with-local-prefix=/opt/TuxamitoSoftToolchains/arm-arm1176jzfssf-linux-gnueabi/gcc-4.6.4/arm-arm1176jzfssf-linux-gnueabi/sysroot --enable-c99 --enable-long-long

Thread model: posix

gcc version 4.6.4 (crosstool-NG hg+default-2685dfa9de14 - tc0002)
```

这样我们的交叉工具链就安装好了

实验二 u-boot 的烧写及使用

【实验目的】

了解 u-boot 的常用命令和 linux 内核的引导。

【实验环境】

- 1、ubuntu 12.04 发行版
- 2、u-boot-2013.03
- 3、FS4412 平台
- 4、交叉编译器 arm-none-linux-gnueabi-gcc

【实验步骤】

1、SD 启动盘制作

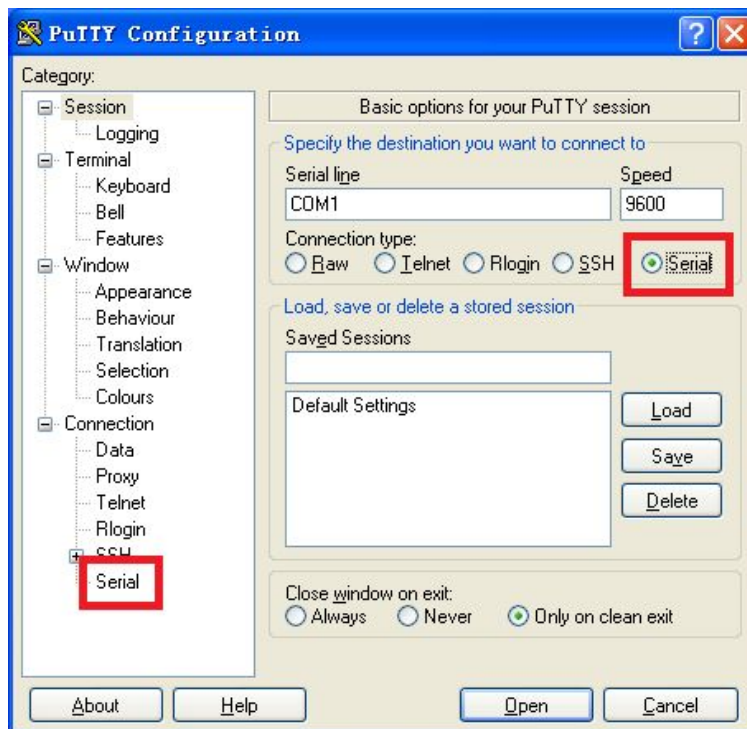
- a) 将第一天/工具/sdfuse_q 拷贝到 Linux 下
- b) 将 SD 卡插入电脑并识别
- c) 进入 sdfuse_q 执行如下操作

```
$ sudo ./mkuboot.sh /dev/sdb
```

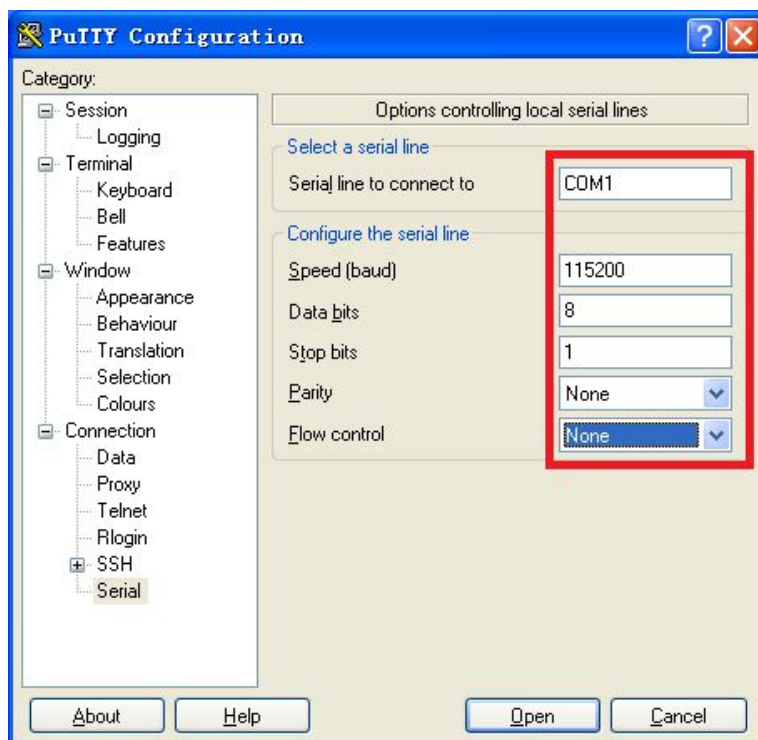
- d) 在 SD 卡中创建目录 sdupdate 并把第一天/镜像中的 u-boot-fs4412.bin 拷贝到这个目录下

2、u-boot 的烧写

- a) 连接串口和板子，运行串口通信程序(putty 第一天工具中)



选择右上角的”Serial”，然后点击左下角的”Serial”



按照自己的主机的情况选择 COM 口其他必须一直，然后点击 open 打开串口



- b) 关闭开发板电源，将拨码开关 SW1 调至(1000)(SD 启动模式)后打开电源
- c) 将刚才做好的 SD 启动盘插入 SD 卡插槽
- d) 重新打开开发板能够看到如下界面

```
COM6 - PuTTY
Input CLK [ 50 MHz] is higher than limit [40 MHz]
Set CLK to 40000 KHz
EMMC clock output: 40000 KHz
>>>get part_type fail : 0
MMC0: 3728 MB
SD sclk_mmc is 400K HZ
SD sclk_mmc is 50000K HZ
SD sclk_mmc is 50000K HZ
MMC1: 7460 MB
env_valid = 1
*** Warning - using default environment

In: serial
Out: serial
Err: serial

Checking Boot Mode ... SDMMC
Net: dm9000
dm9000 i/o: 0x5000000, id: 0x90000a46
DM9000: running in 16 bit mode
MAC: 11:22:33:44:55:66
Hit any key to stop autoboot: 0
FS4412 #
```

在倒计时时按任意键

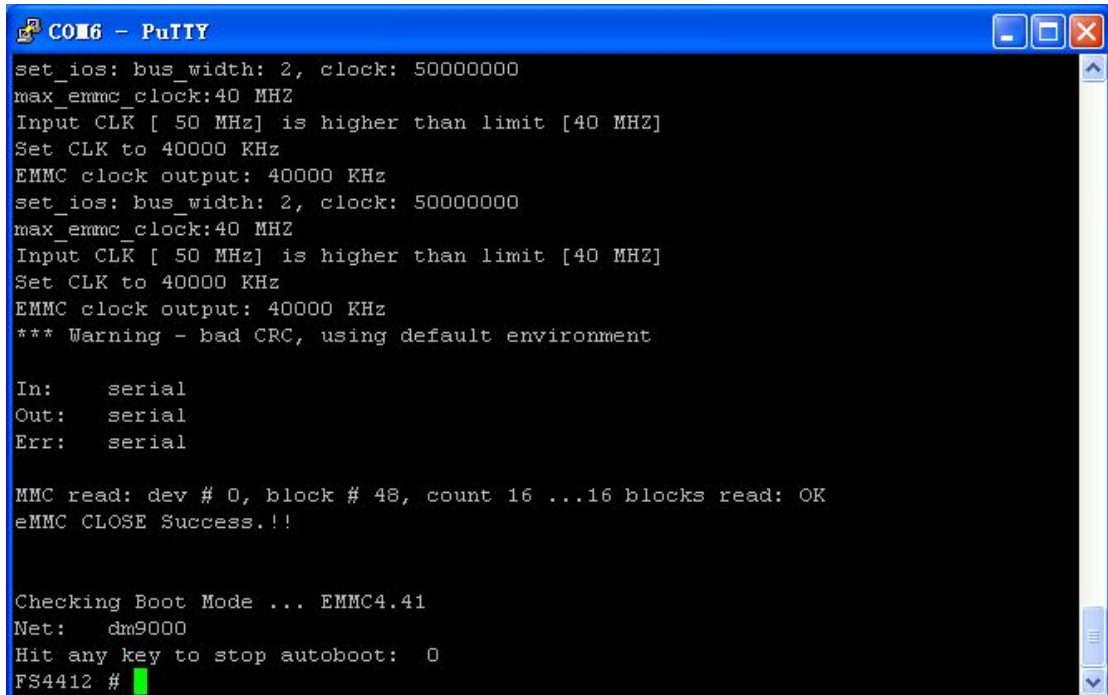
- e) 烧写

在终端上执行

```
sdfuse flashall
```

等待终端无输出是表示烧写结束

- f) 关闭开发板电源，将拨码开关 SW1 调至 0110(EMMC 启动模式)后打开电源可以看到如下界面表示烧写成功



```
COM6 - PuTTY
set_ios: bus_width: 2, clock: 50000000
max_emmc_clock:40 MHz
Input CLK [ 50 MHz] is higher than limit [40 MHz]
Set CLK to 40000 KHz
EMMC clock output: 40000 KHz
set_ios: bus_width: 2, clock: 50000000
max_emmc_clock:40 MHz
Input CLK [ 50 MHz] is higher than limit [40 MHz]
Set CLK to 40000 KHz
EMMC clock output: 40000 KHz
*** Warning - bad CRC, using default environment

In:  serial
Out: serial
Err: serial

MMC read: dev # 0, block # 48, count 16 ...16 blocks read: OK
eMMC CLOSE Success.!!

Checking Boot Mode ... EMMC4.41
Net:  dm9000
Hit any key to stop autoboot:  0
FS4412 #
```

在倒计时时按任意键

3、通过网络加载内核和文件系统

- 将第一天/镜像文件/uImage 拷贝到 ubuntu 的 /tftpboot 下
- 将第一天/镜像文件/rootfs.tar.xz 拷贝到 ubuntu 的 /source 下并解压
- 将第一天/镜像文件/exynos4412-fs4412.dtb 拷贝到 ubuntu 的 /tftpboot 下
- 修改虚拟机 nfs 配置文件/etc/exports，添加如下内容并重启 nfs 服务

```
/source/rootfs *(rw,sync,no_subtree_check,no_root_squash)
```

- 重新驱动 nfs 服务

```
$ sudo /etc/init.d/nfs-kernel-server restart
```

- 设置启动参数

```
#setenv serverip 192.168.9.120

#setenv ipaddr 192.168.9.233

#setenv bootcmd tftp 41000000 uImage;tftp 42000000 exynos4412-fs4412.dtb;bootm 41000000
- 42000000

#setenv          bootargs          root=/dev/nfs          nfsroot=192.168.9.120:/source/rootfs          rw
```

```
console=ttySAC2,115200    init=/linuxrc  ip=192.168.9.233
```

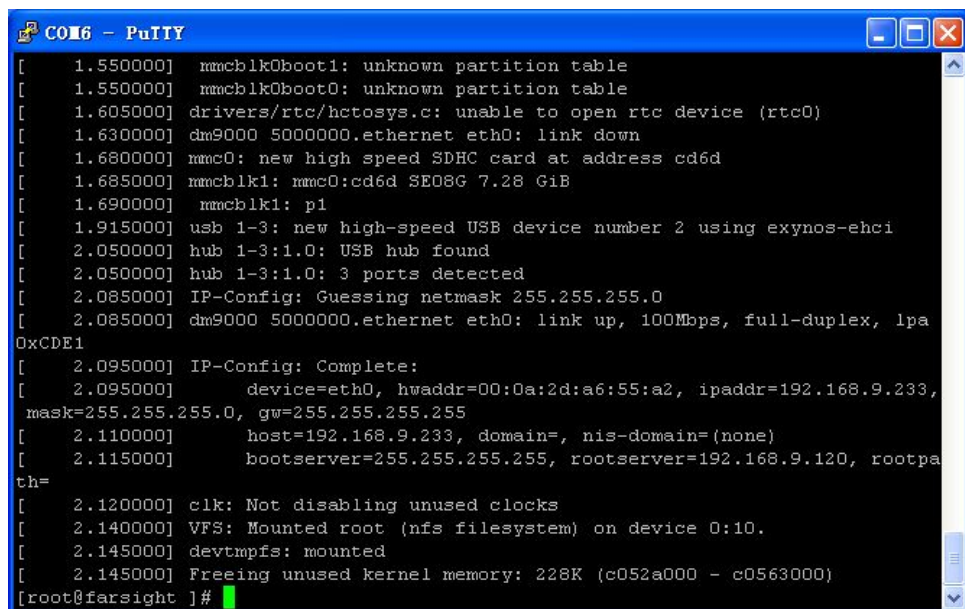
```
# saveenv
```

注意：192.168.9.120 对应 Ubuntu 的 ip

192.168.9.233 对应板子的 ip

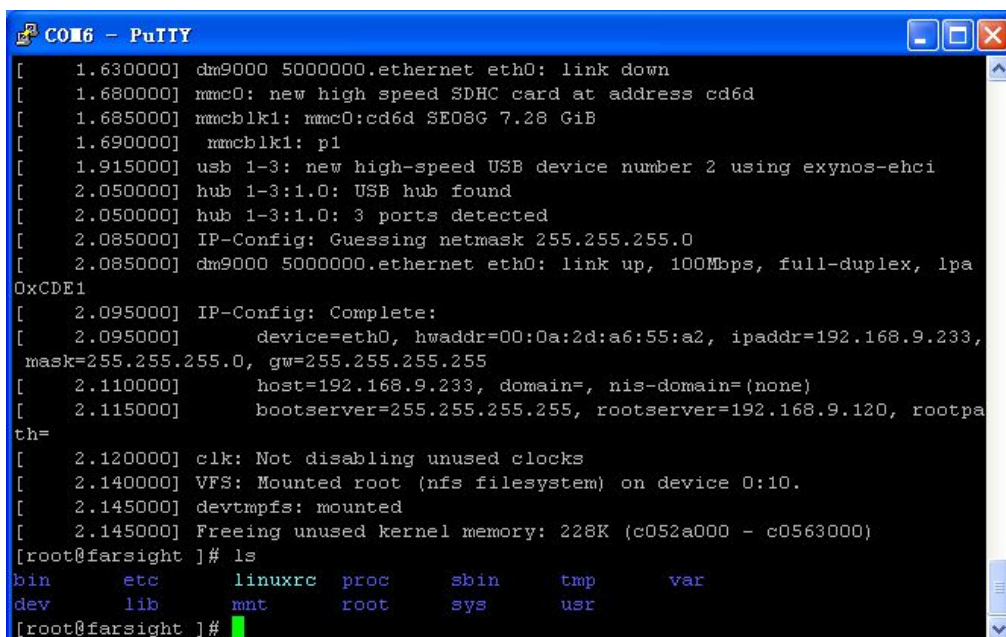
这两个 ip 应该根据自己的实际情况适当修改

g) 启动开发板看到如下现象表示成功通过网络挂载：



```
COM6 - PuTTY
[ 1.550000] mmcblk0boot1: unknown partition table
[ 1.550000] mmcblk0boot0: unknown partition table
[ 1.605000] drivers/rtc/hctosys.c: unable to open rtc device (rtc0)
[ 1.630000] dm9000 5000000.ethernet eth0: link down
[ 1.680000] mmc0: new high speed SDHC card at address cd6d
[ 1.685000] mmcblk1: mmc0:cd6d SE08G 7.28 GiB
[ 1.690000] mmcblk1: p1
[ 1.915000] usb 1-3: new high-speed USB device number 2 using exynos-ehci
[ 2.050000] hub 1-3:1.0: USB hub found
[ 2.050000] hub 1-3:1.0: 3 ports detected
[ 2.085000] IP-Config: Guessing netmask 255.255.255.0
[ 2.085000] dm9000 5000000.ethernet eth0: link up, 100Mbps, full-duplex, lpa
0xCDE1
[ 2.095000] IP-Config: Complete:
[ 2.095000] device=eth0, hwaddr=00:0a:2d:a6:55:a2, ipaddr=192.168.9.233,
mask=255.255.255.0, gw=255.255.255.255
[ 2.110000] host=192.168.9.233, domain=, nis-domain=(none)
[ 2.115000] bootserver=255.255.255.255, rootserver=192.168.9.120, rootpa
th=
[ 2.120000] clk: Not disabling unused clocks
[ 2.140000] VFS: Mounted root (nfs filesystem) on device 0:10.
[ 2.145000] devtmpfs: mounted
[ 2.145000] Freeing unused kernel memory: 228K (c052a000 - c0563000)
[root@farsight]#
```

这是可以输入一些 Linux 的命令测试



```
COM6 - PuTTY
[ 1.630000] dm9000 5000000.ethernet eth0: link down
[ 1.680000] mmc0: new high speed SDHC card at address cd6d
[ 1.685000] mmcblk1: mmc0:cd6d SE08G 7.28 GiB
[ 1.690000] mmcblk1: p1
[ 1.915000] usb 1-3: new high-speed USB device number 2 using exynos-ehci
[ 2.050000] hub 1-3:1.0: USB hub found
[ 2.050000] hub 1-3:1.0: 3 ports detected
[ 2.085000] IP-Config: Guessing netmask 255.255.255.0
[ 2.085000] dm9000 5000000.ethernet eth0: link up, 100Mbps, full-duplex, lpa
0xCDE1
[ 2.095000] IP-Config: Complete:
[ 2.095000] device=eth0, hwaddr=00:0a:2d:a6:55:a2, ipaddr=192.168.9.233,
mask=255.255.255.0, gw=255.255.255.255
[ 2.110000] host=192.168.9.233, domain=, nis-domain=(none)
[ 2.115000] bootserver=255.255.255.255, rootserver=192.168.9.120, rootpa
th=
[ 2.120000] clk: Not disabling unused clocks
[ 2.140000] VFS: Mounted root (nfs filesystem) on device 0:10.
[ 2.145000] devtmpfs: mounted
[ 2.145000] Freeing unused kernel memory: 228K (c052a000 - c0563000)
[root@farsight]# ls
bin      etc      linuxrc  proc     sbin     tmp      var
dev      lib      mnt     root     sys      usr
[root@farsight]#
```


4、从 EMMC 加载内核和文件系统

a) 拷贝第一天/镜像文件/ramdisk.img 拷贝到虚拟机/tftpboot 目录下

b) 烧写内核镜像到 EMMC 上

```
# tftp 41000000 uImage
# movi write kernel 41000000
```

c) 烧写设备树文件到 EMMC 上

```
# tftp 41000000 exynos4412-fs4412.dtb
# movi write dtb 41000000
```

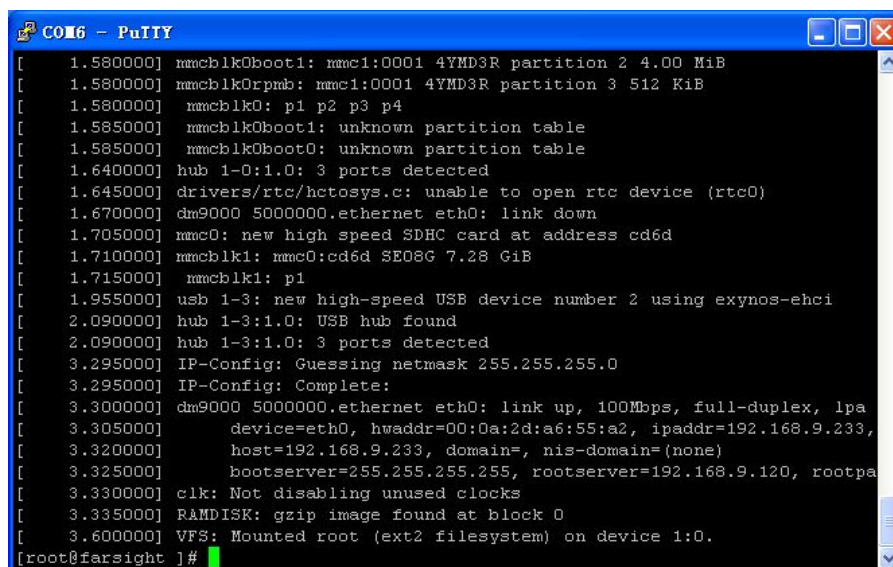
d) 烧写文件系统镜像到 EMMC 上

```
# tftp 41000000 ramdisk.img
# movi write rootfs 41000000 300000
```

e) 设置启动参数

```
# setenv bootcmd movi read kernel 41000000\;movi read dtb 42000000\;movi read rootfs
43000000 300000\;bootm 41000000 43000000 42000000
# saveenv
```

f) 重新启动开发板，u-boot 自动加载、执行内核



```
COM6 - PuTTY
[ 1.580000] mmcblk0boot1: mmc1:0001 4YMD3R partition 2 4.00 MiB
[ 1.580000] mmcblk0rpmb: mmc1:0001 4YMD3R partition 3 512 KiB
[ 1.580000] mmcblk0: p1 p2 p3 p4
[ 1.585000] mmcblk0boot1: unknown partition table
[ 1.585000] mmcblk0boot0: unknown partition table
[ 1.640000] hub 1-0:1.0: 3 ports detected
[ 1.645000] drivers/rtc/hctosys.c: unable to open rtc device (rtc0)
[ 1.670000] dm9000 5000000.ethernet eth0: link down
[ 1.705000] mmc0: new high speed SDHC card at address cd6d
[ 1.710000] mmcblk1: mmc0:cd6d SE08G 7.28 GiB
[ 1.715000] mmcblk1: p1
[ 1.955000] usb 1-3: new high-speed USB device number 2 using exynos-ehci
[ 2.090000] hub 1-3:1.0: USB hub found
[ 2.090000] hub 1-3:1.0: 3 ports detected
[ 3.295000] IP-Config: Guessing netmask 255.255.255.0
[ 3.295000] IP-Config: Complete:
[ 3.300000] dm9000 5000000.ethernet eth0: link up, 100Mbps, full-duplex, lpa
[ 3.305000] device=eth0, hwaddr=00:0a:2d:a6:55:a2, ipaddr=192.168.9.233,
[ 3.320000] host=192.168.9.233, domain=, nis-domain=(none)
[ 3.325000] bootserver=255.255.255.255, rootserver=192.168.9.120, rootpa
[ 3.330000] clk: Not disabling unused clocks
[ 3.335000] RAMDISK: gzip image found at block 0
[ 3.600000] VFS: Mounted root (ext2 filesystem) on device 1:0.
[root@farsight]#
```


5、其他命令练习

```
FS4412 # ?
?      - alias for 'help'
base    - print or set address offset
bdfinfo - print Board Info structure
boot    - boot default, i.e., run 'bootcmd'
bootd    - boot default, i.e., run 'bootcmd'
bootelf  - Boot from an ELF image in memory
bootm    - boot application image from memory
bootp    - boot image via network using BOOTP/TFTP protocol
bootvx   - Boot vxWorks from an ELF image
cmp      - memory compare
coninfo  - print console devices and information
cp       - memory copy
crc32    - checksum calculation
dhcp     - boot image via network using DHCP/TFTP protocol
echo     - echo args to console
editenv  - edit environment variable
emmc     - Open/Close eMMC boot Partition
env      - environment handling commands
erase    - erase FLASH memory
exit     - exit script
false    - do nothing, unsuccessfully
fatinfo  - print information about filesystem
fatload  - load binary file from a dos filesystem
fatls    - list files in a directory (default /)
fdisk    - fdisk - fdisk for sd/mmc.

fdt      - flattened device tree utility commands
flinfo   - print FLASH memory information
go       - start application at address 'addr'
help     - print command description/usage
iminfo   - print header information for application image
imxtract - extract a part of a multi-image
itest    - return true/false on integer compare
loadb    - load binary file over serial line (kermit mode)
loads    - load S-Record file over serial line
loady    - load binary file over serial line (ymodem mode)
```

6、交叉编译和交叉调试

a) 将 u-boot 启动参数修改为网络启动

```
#setenv serverip 192.168.9.120

#setenv ipaddr 192.168.9.233

#setenv bootcmd tftp 41000000 uImage;tftp 42000000 exynos4412-fs4412.dtb;bootm 41000000
- 42000000

#setenv      bootargs      root=/dev/nfs      nfsroot=192.168.9.120:/source/rootfs      rw
console=ttySAC2,115200      init=/linuxrc      ip=192.168.9.233
```

```
# saveenv
```

注意：192.168.9.120 对应 Ubuntu 的 ip

192.168.9.233 对应板子的 ip

这两个 ip 应该根据自己的实际情况适当修改

b) 编辑程序源码 myapp.c(自己写一个简单的 c 程序)

c) 交叉编译后复制到/source/rootfs (编译时添加选项-g)

```
$arm-none-linux-gnueabi-gcc myapp.c -o myapp -g
```

```
$ cp myapp /source/rootfs
```

d) 复制 gdbserver 到/source/rootfs/bin (gdbserver 在交叉工具链中找，路径为：
/home/linux/toolchain/gcc-4.6.4/arm-arm1176jzfssf-linux-gnueabi/debug-root/bin)

e) 在开发板上如下运行

```
# gdbserver 192.168.9.233:12345 myapp &
```

f) 在主机上运行交叉调试器

```
$ arm-none-linux-gnueabi-gdb myapp
```

g) 在交叉调试器下和开发板 gdbserver 建立连接

```
(gdb) target remote 192.168.9.233:12345
```

设置断点，输入命令 c 开始调试程序(注意观察串口输出)

实验三 u-boot-2013.01 的移植

【实验目的】

了解 u-boot 的代码结构及移植方法。

【实验环境】

- 1、ubuntu 10.10 发行版
- 2、u-boot-2013.01
- 3、FS_4412 平台（EXYNOS 4412）
- 4、交叉编译器 arm-none-linux-gnueabi-gcc

【实验步骤】

一、建立自己的平台

1、 下载源码

我们可以在下面这个网站上下载最新的和以前任一版本的 uboot

<ftp://ftp.denx.de/pub/u-boot/>

2、 解压 uboot 源码并进入目录

```
$ tar xvf u-boot-2013.01.tar.bz2
$ cd u-boot-2013.01
```

3、 指定交叉编译工具链

```
$ vim Makefile
```

把

```
ifeq ($(HOSTARCH),$(ARCH))

    CROSS_COMPILE ?=

#endif
```

下添加

```
ifeq arm,$(ARCH)

    CROSS_COMPILE ?= arm-none-linux-gnueabi-

#endif
```

4、指定产品 CPU

我们产品用的 CPU 是 exynos 4412

查看 u-boot 源码该 CPU 是否已支持

U-boot 已支持，见 arch/arm/cpu/armv7/exynos/

5、指定产品 BOARD

找一个最类似的 board 配置修改，这里我们参考的是 board/samsung/origen/

```
$ cp -rf board/samsung/origen /board/samsung/fs4412
```

```
$ mv board/samsung/fs4412/origen.c board/samsung/fs4412/fs4412.c
```

```
$ vim board/samsung/fs4412/Makefile
```

修改 origen.o 为 fs4412.o

```
$ cp include/configs/origen.h include/configs/fs4412.h
```

```
$ vim include/configs/fs4412.h
```

修改

```
#define CONFIG_SYS_PROMPT "ORIGEN #"
```

为

```
#define CONFIG_SYS_PROMPT "fs4412 #"
```

修改

```
#define CONFIG_IDENT_STRING for ORIGEN
```

为

```
#define CONFIG_IDENT_STRING for fs4412
```

```
#vim boards.cfg
```

参考

```
origen arm armv7 origen samsung exynos
```

并在后面新增

```
fs4412 arm armv7 fs4412 samsung exynos
```

6、编译 u-boot

7、\$ make distclean

```
$ make fs4412_config
```

```
$ make
```

编译完成后生成的 u-boot.bin 就是可执行的镜像文件。

烧写 uboot 命令：

```
tftp 41000000 u-boot.bin
```

```
movi write uboot 41000000
```

但是该文件还不能在我们板子上运行，我们需要对 u-boot 源代码进行相应的修改。

二、实现能看到串口终端信息

1、确认第一条指令有运行到（点灯法）

- 在 arch/arm/cpu/armv7/start.S 134 行后添加点灯程序

```
#if 1

    ldr r0, =0x11000c40 @GPK2_7 led2

    ldr r1, [r0]

    bic r1, r1, #0xf0000000

    orr r1, r1, #0x10000000

    str r1, [r0]


    ldr r0, =0x11000c44

    mov r1, #0xff

    str r1, [r0]

#endif
```

- 添加三星加密方式

exynos 需要三星提供的初始引导加密后，我们的 u-boot,才能被引导运行

将 sdfuse_q 目录拷贝到 u-boot-2013.01 源码目录下

注：sdfuse_q 三星提供的加密处理

将 CodeSign4SecureBoot 目录拷贝到 u-boot-2013.01 源码目录下

注：CodeSign4SecureBoot 三星提供的安全启动方式

- 修改 Makefile

```
$vim Makefile
```

修改实现 sdfuse_q 的编译

在

```
$(obj)u-boot.bin: $(obj)u-boot

    $(OBJCOPY) ${OBJCFLAGS} -O binary $< $@

    $(BOARD_SIZE_CHECK)
```

下添加

```
@#./mkuboot

@split -b 14336 u-boot.bin bl2      此处 bl2 是字母'l'， 不是数字 1

@+make -C sdfuse_q/

@#cp u-boot.bin u-boot-4212.bin

@#cp u-boot.bin u-boot-4412.bin

@#./sdfuse_q/add_sign

@./sdfuse_q/chksum

@./sdfuse_q/add_padding

@rm bl2a*

@echo
```

注意是 tab 键缩进的，否则 makefile 编译报错

注意如果执行了 make distclean 需重新拷贝 CodeSign4SecureBoot

- 拷贝编译脚本

```
$cp build.sh u-boot-2013.01  
$chmod 777 u-boot-2013.01/build.sh  
$ ./buildsh
```

注：build.sh 脚本方式完成自动添加加密方式，

编译生成所需文件 u-boot_fs4412.bin

烧写新的 u-boot_fs4412.bin

复位，发现灯有点亮，说明我们的 u-boot 有运行到

2、实现串口输出

修改 lowlevel_init.S 文件

```
$vim board/samsung/fs4412/lowlevel_init.S
```

- 添加临时栈

在

```
lowlevel_init:
```

后添加

```
ldr sp,=0x02060000 @use iRom stack in bl2
```

- 添加关闭看门狗代码

在

```
beq wakeup_reset
```

后添加

```
#if 1 /*for close watchdog */
```

```
/* PS-Hold high */
```

```
    ldr r0,=0x1002330c
```

```
    ldr r1,[r0]
```

```
    orr r1,r1,#0x300
```

```
    str r1,[r0]
```



```

        ldr    r0, =0x11000c08

        ldr r1, =0x0

        str r1, [r0]

/* Clear  MASK_WDT_RESET_REQUEST  */

        ldr r0, =0x1002040c

        ldr r1, =0x00

        str r1, [r0]

#endif

```

- 添加串口初始化代码

在 uart_asm_init: 的

```

        str    r1, [r0, #EXYNOS4_GPIO_A1_CON_OFFSET]

```

后添加

```

ldr    r0, =0x10030000

        ldr    r1, =0x666666

        ldr    r2, =CLK_SRC_PERIL0_OFFSET

        str    r1, [r0, r2]

        ldr    r1, =0x777777

        ldr    r2, =CLK_DIV_PERIL0_OFFSET

        str    r1, [r0, r2]

```

注释掉 trustzone 初始化

注释掉

```

        bl    uart_asm_init

```

下的

```

        bl    tzpc_init

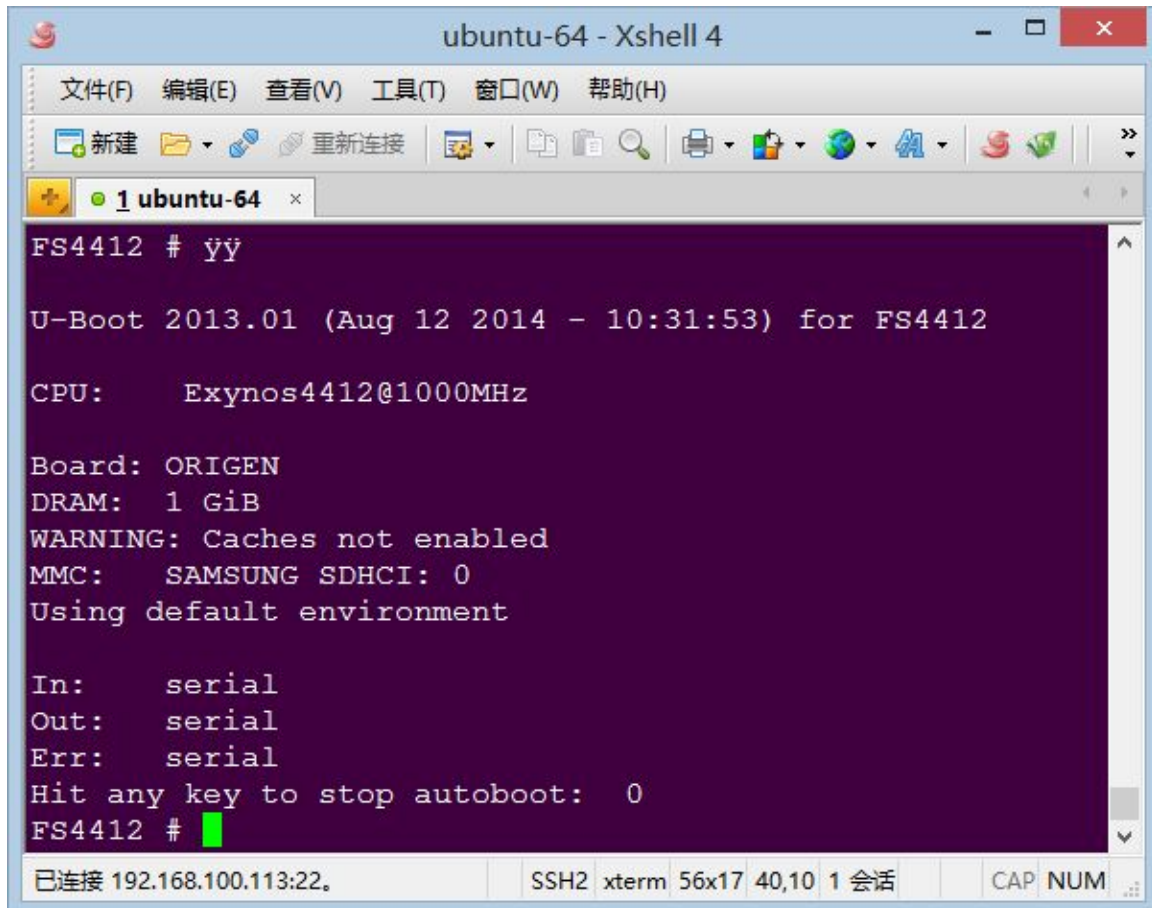
```

重新编译 u-boot

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

烧写新的 u-boot_fs4412.bin

复位会看到串口信息



```
ubuntu-64 - Xshell 4
文件(F) 编辑(E) 查看(V) 工具(T) 窗口(W) 帮助(H)
新建 重新连接
1 ubuntu-64 x
FS4412 # yy
U-Boot 2013.01 (Aug 12 2014 - 10:31:53) for FS4412
CPU:      Exynos4412@1000MHz
Board: ORIGIN
DRAM: 1 GiB
WARNING: Caches not enabled
MMC:      SAMSUNG SDHCI: 0
Using default environment
In:       serial
Out:      serial
Err:      serial
Hit any key to stop autoboot:  0
FS4412 #
```

三、网卡移植

1、添加网络初始化代码

```
$vim board/samsung/fs4412/fs4412.c
```

在 struct exynos4_gpio_part2 *gpio2; 后添加

```
#ifndef CONFIG_DRIVER_DM9000

#define EXYNOS4412_SROMC_BASE 0X12570000

#define DM9000_Tacs      (0x1)
```

```

#define DM9000_Tcos      (0x1)

#define DM9000_Tacc      (0x5)

#define DM9000_Tcoh      (0x1)

#define DM9000_Tah      (0xC)

#define DM9000_Tacp      (0x9)

#define DM9000_PMC       (0x1)


struct exynos_sromc {

unsigned int bw;

unsigned int bc[6];

};


/*

 * s5p_config_sromc() - select the proper SROMC Bank and configure the
 * band width control and bank control registers

 * srom_bank      - SROM

 * srom_bw_conf   - SMC Band width reg configuration value

 * srom_bc_conf   - SMC Bank Control reg configuration value

 */

void exynos_config_sromc(u32 srom_bank, u32 srom_bw_conf, u32 srom_bc_conf)

{

unsigned int tmp;

struct exynos_sromc *srom = (struct exynos_sromc *) (EXYNOS4412_SROMC_BASE);


    /* Configure SMC_BW register to handle proper SROMC bank */

tmp = srom->bw;

tmp&= ~(0xF << (srom_bank * 4));

```

```

tmp |= srom_bw_conf;

srom->bw = tmp;


/* Configure SMC_BC register */

srom->bc[srom_bank] = srom_bc_conf;
}

static void dm9000aep_pre_init(void)
{
unsigned int tmp;

unsigned char smc_bank_num = 1;

unsigned int    smc_bw_conf=0;
unsigned int    smc_bc_conf=0;


/* gpio configuration */

writel(0x00220020, 0x11000000 + 0x120);
writel(0x00002222, 0x11000000 + 0x140);

/* 16 Bit bus width */

writel(0x22222222, 0x11000000 + 0x180);
writel(0x0000FFFF, 0x11000000 + 0x188);
writel(0x22222222, 0x11000000 + 0x1C0);
writel(0x0000FFFF, 0x11000000 + 0x1C8);
writel(0x22222222, 0x11000000 + 0x1E0);
writel(0x0000FFFF, 0x11000000 + 0x1E8);

smc_bw_conf &= ~(0xf<<4);

smc_bw_conf |= (1<<7) | (1<<6) | (1<<5) | (1<<4);

smc_bc_conf = ((DM9000_Tacs << 28)

| (DM9000_Tcos << 24)

```

```

        | (DM9000_Tacc << 16)

        | (DM9000_Tcoh << 12)

        | (DM9000_Tah << 8)

    | (DM9000_Tacp << 4)

    | (DM9000_PMC));

    exynos_config_sromc(smc_bank_num,smc_bw_conf,smc_bc_conf);
}

#endif

```

在 `gd->bd->bi_boot_params = (PHYS_SDRAM_1 + 0x100UL);` 后添加

```

#ifdef CONFIG_DRIVER_DM9000

    dm9000aep_pre_init();

#endif

```

在文件末尾添加

```

#ifdef CONFIG_CMD_NET

int board_eth_init(bd_t *bis)

{

    int rc = 0;

    #ifdef CONFIG_DRIVER_DM9000

    rc = dm9000_initialize(bis);

    #endif

    return rc;

}

#endif

```

2、 修改配置文件添加网络相关配置

```
$ vim include/configs/fs4412.h
```

修改

```
#undef CONFIG_CMD_PING
```

为

```
#define CONFIG_CMD_PING
```

修改

```
#undef CONFIG_CMD_NET
```

为

```
#define CONFIG_CMD_NET
```

在文件末尾

```
#endif /* __CONFIG_H */
```

前面添加

```
#ifndef CONFIG_CMD_NET
#define CONFIG_NET_MULTI
#define CONFIG_DRIVER_DM9000 1
#define CONFIG_DM9000_BASE 0x05000000
#define DM9000_IO CONFIG_DM9000_BASE
#define DM9000_DATA (CONFIG_DM9000_BASE + 4)
#define CONFIG_DM9000_USE_16BIT
#define CONFIG_DM9000_NO_SROM 1
#define CONFIG_ETHADDR 11:22:33:44:55:66
#define CONFIG_IPADDR 192.168.9.200
#define CONFIG_SERVERIP 192.168.9.120
#define CONFIG_GATEWAYIP 192.168.9.1
#define CONFIG_NETMASK 255.255.255.0
```

```
#endif
```

3、重新编译 u-boot

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

烧写新的 u-boot_fs4412.bin

复位后

```
# ping 192.168.9.120
```

```
FS4412 # ping 192.168.9.120
dm9000 i/o: 0x5000000, id: 0x90000a46
DM9000: running in 16 bit mode
MAC: 11:22:33:44:55:66
operating at 100M full duplex mode
Using dm9000 device
host 192.168.9.120 is alive
```

四、FLASH 移植 (EMMC)

1、初始化 EMMC

```
$cp movi.c arch/arm/cpu/armv7/exynos/
```

```
$vim arch/arm/cpu/armv7/exynos/Makefile
```

在 pinmux.o 后添加 movi.o

修改板级文件

```
$vim board/samsung/fs4412/fs4412.c
```

在

```
#include <asm/arch/mmc.h>
```

后面添加

```
#include <asm/arch/clk.h>
```

```
#include "origen_setup.h"
```

在


```
#ifndef CONFIG_GENERIC_MMC
```

后面添加

```
u32 sclk_mmc4;  /*clock source for emmc controller*/

#define __REGMY(x) (((volatile u32 *)(x)))

#define CLK_SRC_FSYS __REGMY(EXYNOS4_CLOCK_BASE + CLK_SRC_FSYS_OFFSET)

#define CLK_DIV_FSYS3 __REGMY(EXYNOS4_CLOCK_BASE + CLK_DIV_FSYS3_OFFSET)


int emmc_init()
{
    u32 tmp;

    u32 clock;

    u32 i;

    /* setup_hsmmc_clock */

    /* MMC4 clock src = SCLKMPLL */

    tmp = CLK_SRC_FSYS & ~(0x000f0000);

    CLK_SRC_FSYS = tmp | 0x00060000;

    /* MMC4 clock div */

    tmp = CLK_DIV_FSYS3 & ~(0x0000ff0f);

    clock = get_pll_clk(MPLL)/1000000;

    for(i=0 ; i<=0xf; i++)  {

sclk_mmc4=(clock/(i+1));


        if(sclk_mmc4 <= 160) //200

        {

            CLK_DIV_FSYS3 = tmp | (i<<0);

            break;
        }
    }
}
```

```

    }

}

emmcdbg("[mjdbg] sclk_mmc4:%d MHZ; mmc_ratio: %d\n",sclk_mmc4,i);

sclk_mmc4 *= 1000000;

/*

* MMC4 EMMC GPIO CONFIG
*
* GPK0[0]    SD_4_CLK
* GPK0[1]    SD_4_CMD
* GPK0[2]    SD_4_CDn
* GPK0[3:6]  SD_4_DATA[0:3]
*/

writel(readl(0x11000048)&~(0xf),0x11000048); //SD_4_CLK/SD_4_CMD pull-down enable

writel(readl(0x11000040)&~(0xff),0x11000040); //cdn set to be output


writel(readl(0x11000048)&~(3<<4),0x11000048); //cdn pull-down disable

writel(readl(0x11000044)&~(1<<2),0x11000044); //cdn output 0 to shutdown the emmc power

writel(readl(0x11000040)&~(0xf<<8)|(1<<8),0x11000040); //cdn set to be output

udelay(100*1000);

writel(readl(0x11000044)|(1<<2),0x11000044); //cdn output 1


writel(0x03333133, 0x11000040);


writel(0x00003FF0, 0x11000048);

writel(0x00002AAA, 0x1100004C);

```

```

#ifdef CONFIG_EMMC_8Bit

writel(0x04444000, 0x11000060);

writel(0x00003FC0, 0x11000068);

writel(0x00002AAA, 0x1100006C);

#endif


#ifdef USE_MMC4

smdk_s5p_mshc_init();

#endif

}

```

将 int board_mmc_init(bd_t *bis)函数内容改写为

```

int board_mmc_init(bd_t *bis)

{

    int i, err;

#ifdef CONFIG_EMMC

    err = emmc_init();

#endif

    return err;

}

```

在末尾添加

```

#ifdef CONFIG_BOARD_LATE_INIT

#include <movi.h>

int  chk_bootdev(void)//mj for boot device check

{

```

```

char run_cmd[100];

struct mmc *mmc;

int boot_dev = 0;

int cmp_off = 0x10;

ulong  start_blk, blkcnt;


mmc = find_mmc_device(0);


if (mmc == NULL)
{
    printf("There is no eMMC card, Booting device is SD card\n");

    boot_dev = 1;

    return boot_dev;
}

start_blk = (24*1024/MOVI_BLKSIZE);

blkcnt = 0x10;


sprintf(run_cmd,"emmc open 0");

run_command(run_cmd, 0);


sprintf(run_cmd,"mmc read 0 %lx %lx %lx",CFG_PHY_KERNEL_BASE,start_blk,blkcnt);
run_command(run_cmd, 0);


/* switch mmc to normal paritition */

sprintf(run_cmd,"emmc close 0");

run_command(run_cmd, 0);

```

```

        return 0;
    }

int board_late_init (void)
{
    int boot_dev = 0 ;

    char boot_cmd[100];

        boot_dev = chk_bootdev();

    if(!boot_dev)
    {
        printf("\n\nChecking Boot Mode ... EMMC4.41\n");
    }

    return 0;
}

#endif

```

2、 添加相关命令

```

$ cp    cmd_movi.c    common/
$ cp    cmd_mmc.c    common/
$ cp    cmd_mmc_fdisk.c    common/

```

修改 Makefile

```
$ vim    common/Makefile
```

在

```
COBJS-$(CONFIG_CMD_MMC) += cmd_mmc.o
```

后添加

```
COBJS-$(CONFIG_CMD_MMC) += cmd_mmc_fdisk.o
```

```
COBJS-$(CONFIG_CMD_MOVINAND) += cmd_movi.o
```

添加驱动

```
$ cp mmc.c drivers/mmc/  
$ cp s5p_mshc.c drivers/mmc/  
$ cp mmc.h include/  
$ cp movi.h include/  
$ cp s5p_mshc.h include/
```

修改 Makefile

```
$vim drivers/mmc/Makefile
```

添加

```
COBJS-$(CONFIG_S5P_MSHC) += s5p_mshc.o
```

3、 添加 EMMC 相关配置

```
$vim include/configs/fs4412.h
```

添加

```
#define CONFIG_EVT1 1 /* EVT1 */  
  
#ifndef CONFIG_EVT1  
  
#define CONFIG_EMMC44_CH4 //eMMC44_CH4 (OMPIN[5:1] = 4)  
  
#ifndef CONFIG_SDMMC_CH2  
  
#define CONFIG_S3C_HSMMMC  
  
#undef DEBUG_S3C_HSMMMC  
  
#define USE_MMC2  
  
#endif
```

```

#ifdef CONFIG_EMMC44_CH4

#define CONFIG_S5P_MSHC

#define CONFIG_EMMC          1

#define USE_MMC4

/* #define CONFIG_EMMC_8Bit */

#define CONFIG_EMMC_EMERGENCY

/* #define emmcdbg(fmt,args...) printf(fmt ,##args) */ //for emmc debug

#define emmcdbg(fmt,args...)

#endif

#endif /*end CONFIG_EVT1*/

#define CONFIG_CMD_MOVINAND

#define CONFIG_CLK_1000_400_200

#define CFG_PHY_UBOOT_BASE    CONFIG_SYS_SDRAM_BASE + 0x3e00000

#define CFG_PHY_KERNEL_BASE   CONFIG_SYS_SDRAM_BASE + 0x8000

#define BOOT_MMCSDB           0x3

#define BOOT_EMMC43           0x6

#define BOOT_EMMC441          0x7

#define CONFIG_BOARD_LATE_INIT

```

4、 重新编译 u-boot

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

烧写新的 u-boot_fs4412.bin

复位后

```
# mmcinfo
```


CPU: Exynos4412@1000MHz

Board: FS4412

DRAM: 1 GiB

WARNING: Caches not enabled

MMC: MMC0: 3728 MB

In: serial

Out: serial

Err: serial

MMC read: dev # 0, block # 48, count 16 ...16 blocks read: OK
eMMC CLOSE Success.!!

Checking Boot Mode ... EMMC4.41

Net: dm9000

Hit any key to stop autoboot: 0

FS4412 # mmcinfo

Device: S5P_MSHC4

Manufacturer ID: 15

OEM: 100

Name: 4YMD3

Tran Speed: 0

Rd Block Len: 512

MMC version 4.0

High Capacity: Yes

Capacity: 7.3 MiB

Bus Width: 2-bit

FS4412 #

实验四内核的配置和编译

【实验目的】

了解内核的编译过程及配置选项的内容

说明：在本系统移植课程实验中命令行提示符“\$”表示是在主机上执行，“#”表示在目标板执行

【实验环境】

- 主机：ubuntu 12.04 发行版
- 目标机：FS4412 平台
- 交叉编译工具：arm-none-linux-gnueabi-gcc

【实验步骤】

- 解压内核

将 linux-3.14.tar.xz 拷贝到/home/linux 下并解压

```
$tar xvf linux-3.14.tar.xz
```

```
$ cd linux-3.14
```

- 修改内核顶层目录下的 Makefile

```
$ vim Makefile
```

修改：

```
ARCH      ?= $(SUBARCH)
```

```
CROSS_COMPILE ?= $(CONFIG_CROSS_COMPILE:"%"=%)
```

为：

```
ARCH      ?= arm
```

```
CROSS_COMPILE ?= arm-none-linux-gnueabi-
```

- 导入默认配置

```
$ make exynos_defconfig
```

- 配置内核

```
$ make menuconfig
```

```
System Type --->
```

(2) S3C UART to use for low-level messages

该命令执行时会弹出一个菜单，我们可以对内核进行详细的配置。这里我们先查看一下，内核都提供了那些功能！

- 编译内核

```
$ make uImage
```

通过上述操作我们能够在 arch/arm/boot 目录下生成一个 uImage 文件，这就是经过压缩的内核镜像。

如果编译过程中提示缺少 mkimage 工具，需将第二天编译的 uboot 源码中的 tools/mkimage 拷贝到 ubuntu 的 /usr/bin 目录下

```
$ cp u-boot-2013.01/tools/mkimage /usr/bin
```

修改设备树文件

生成设备树文件，以参考板 origen 的设备数文件为参考。

```
$ cp arch/arm/boot/dts/exynos4412-origen.dts arch/arm/boot/dts/exynos4412-fs4412.dts
```

添加新文件需修改 Makefile 才能编译

```
$ vim arch/arm/boot/dts/Makefile
```

在

```
exynos4412-origen.dtb \
```

下添加如下内容

```
exynos4412-fs4412.dtb \
```

- 编译设备树文件

```
$ make dtbs
```

- 拷贝内核和设备树文件到/tftpboot 目录下

```
$ cp arch/arm/boot/uImage /tftpboot
```

```
$ cp arch/arm/boot/dts/exynos4412-fs4412.dtb/tftpboot/
```

- 修改 uboot 启动参数

重启板子在系统倒计时是按任意键结束启动，输入如下内容修改 uboot 环境变量：

```
#setenv serverip 192.168.9.120
```

```
#setenv ipaddr 192.168.9.233

#setenv bootcmd tftp 41000000 uImage\;tftp 42000000 exynos4412-fs4412.dtb\;bootm 41000000
- 42000000

#setenv      bootargs      root=/dev/nfs      nfsroot=192.168.9.120:/source/rootfs      rw
console=ttySAC2,115200  init=/linuxrc  ip=192.168.9.233

# saveenv
```

注意：192.168.9.120 对应 Ubuntu 的 ip

192.168.9.233 对应板子的 ip

这两个 ip 应该根据自己的实际情况适当修改

重启开发板查看现象

实验五网卡驱动的移植

【实验目的】

通过上面的实验我们能够获得一个内核，但是这个内核只是一个最基本的配置，很多的功能并没有包含。

网卡是嵌入式产品最常用的设备，这里我们需要完成网卡驱动的移植。FS4412 使用的是 DM9000 网卡，我们通过这个实验能够了解如何在内核中添加网卡驱动及网络功能的基本配置。

说明：在本系统移植课程实验中命令行提示符“\$”表示是在主机上执行，“#”表示在目标板执行

【实验环境】

- 主机：ubuntu 12.04 发行版
- 目标机：FS4412 平台
- 交叉编译工具：arm-none-linux-gnueabi-gcc

【实验步骤】

- 设备树文件修改：

```
$ vim arch/arm/boot/dts/exynos4412-fs4412.dts
```

添加如下内容：

```
srom-cs1@5000000 {      此处的 cs1 是数字 1，不是字母'I'

    compatible = "simple-bus";

    #address-cells = <1>;

    #size-cells = <1>;

    reg = <0x5000000 0x1000000>;

    ranges;

    ethernet@5000000 {

        compatible = "davicom,dm9000";

        reg = <0x5000000 0x2 0x5000004 0x2>;

        interrupt-parent = <&gpx0>;

        interrupts = <6 4>;
```

```

        davicom,no-eprom;

        mac-address = [00 0a 2d a6 55 a2];

    };

};

```

- 修改文件 driver/clock/clock.c

修改

```
static bool clock_ignore_unused;
```

为

```
static bool clock_ignore_unused = true;
```

- 配置内核：

```

make menuconfig

[*] Networking support --->

Networking options --->

    <*> Packet socket

    <*> Unix domain sockets

    [*] TCP/IP networking

    [*] IP: kernel level autoconfiguration

Device Drivers --->

[*] Network device support --->

    [*] Ethernet driver support (NEW) --->

        <*> DM9000 support

File systems --->

[*] Network File Systems (NEW) --->

    <*> NFS client support

    [*] NFS client support for NFS version 3

    [*] NFS client support for the NFSv3 ACL protocol extension

    [*] Root file system on NFS

```

- 编译内核和设备树

```
$ make uImage
```

```
$ make dtbs
```

- 测试:

拷贝内核和设备树文件到/tftpboot 目录下

```
$ cp arm/arm/boot/uImage /tftpboot
```

```
$ cp arch/arm/boot/dts/exynos4412-fs4412.dtb /tftpboot/
```

启动开发板，修改内核启动参数，通过 NFS 方式挂载根文件系统

实验六 LED 驱动的移植

【实验目的】

1. 驱动程序的编译
2. 应用程序如何打开/操作设备

【实验环境】

- 主机：ubuntu 12.04 发行版
- 目标机：FS4412 平台
- 交叉编译工具：arm-none-linux-gnueabi-gcc

【实验步骤】

1. 添加驱动文件

将实验代码 Led_test/fs4412_led_drv.c 拷贝到 drivers/char 下

2. 修改 drivers/char/Kconfig

在 menu "Character devices"下面
添加如下内容：

```
config FS4412_LED

tristate "FS4412LED Device Support"

depends on ARCH_EXYNOS4

help

    support leddevice on FS4412develop board
```

3. 修改 drivers/char/Makefile

在文件最后添加如下代码

```
obj-$(CONFIG_FS4412_LED) += fs4412_led_drv.o
```

4. 将 s5pv210_led_app.c 拷贝到 Linux 任意目录下并交叉编译测试程序

```
$ arm-none-linux-gnueabi-gcc fs4412_led_app.c -o fs4412_led_app
sudo cp fs4412_led_app /source/rootfs
```

5. 编译 LED 驱动到内核中

- 配置内核时按“空格”选择，配置完成后保存退出

```
$ make menuconfig  
  
Device Drivers  --->  
  
    Character devices  --->  
  
        <*>FS4412  LED Device Support
```

- 保存退出，重新编译后把 uImage 拷贝到 tftpboot 下

```
$ make uImage  
  
$ cp  arch/arm/boot/uImage/tftpboot
```

重新启动开发板，加载内核并运行。在终端下执行下面操作

- 创建设备节点

```
# mknod  /dev/led  c  500  0
```

- 运行测试程序并观察现象

```
# ./fs4412_led_app
```

6. 编译 LED 驱动为模块

- 配置内核时按“空格”选择，配置完成后保存退出

```
$ make menuconfig  
  
Device Drivers  --->  
  
    Character devices  --->  
  
        <M>FS4412  LED Device Support
```

- 保存退出，重新编译后把 zImage 拷贝到 tftpboot 下，把驱动模块拷贝到/source/rootfs 下

```
$ make uImage modules  
  
$ cp  arch/arm/boot/uImage  /tftpboot/  
  
$ cp  drivers/char/fs4412_led_drv.ko  /source/rootfs/
```

重新启动开发板，linux 运行起来后在终端下操作

- 创建设备节点

```
# mknod /dev/led c 500 0
```

- 加载 LED 驱动模块

```
# insmod fs4412_led_drv.ko
```

- 运行测试程序并观察现象

```
# ./fs4412_led_app
```

实验七 SD 卡驱动移植

【实验目的】

SD 卡是嵌入式系统最常用的外部扩展存储设备，这里介绍 SD 驱动移植的过程。

说明：在系统移植课程实验中命令行提示符“\$”表示是在主机上执行，“#”表示在目标板执行

【实验环境】

- 主机：ubuntu 12.04 发行版
- 目标机：FS4412 平台
- 交叉编译工具：arm-none-linux-gnueabi-gcc

【实验步骤】

1. 修改设备树文件

```
$ vim arch/arm/boot/dts/exynos4412-fs4412.dts
```

修改

```
sdhci@12530000 {  
    bus-width = <4>;  
  
    pinctrl-0 = <&sd2_clk &sd2_cmd &sd2_bus4 &sd2_cd>;  
    pinctrl-names = "default";  
    vmmc-supply = <&mmc_reg>;  
    status = "okay";  
};
```

为:

```
sdhci@12530000 {  
    bus-width = <4>;  
  
    pinctrl-0 = <&sd2_clk &sd2_cmd &sd2_bus4>;  
    cd-gpios = <&gpx0 7 0>;  
    cd-inverted = <0>;  
    pinctrl-names = "default";
```

```
/*vmmc-supply = <&mmc_reg>*/  
  
status = "okay";  
  
};
```

2. 配置内核

```
$ make menuconfig  
  
Device Drivers  --->  
  
    <*> MMC/SD/SDIO card support  --->  
  
        <*>    Secure Digital Host Controller Interface support  
  
        <*>    SDHCI support on Samsung S3C SoC  
  
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  
  
    *- Native language support  --->  
  
        <*>    Codepage 437 (United States, Canada)  
  
        <*>    Simplified Chinese charset (CP936, GB2312)  
  
        <*>    ASCII (United States)  
  
        <*>    NLS ISO 8859-1  (Latin 1; Western European Languages)  
  
        <*>    NLS UTF-8
```

3. 编译内核和设备树

```
$ make uImage  
  
$ make dtbs
```

4. 测试:

拷贝内核和设备树文件到/tftpboot 目录下

```
$ cp arm/arm/boot/uImage /tftpboot
```

```
$ cp arch/arm/boot/dts/exynos4412-fs4412.dtb /tftpboot/
```

启动开发板会有如下内容显示：

```
[ 1.620000] mmc0: new high speed SDHC card at address cd6d
```

```
[ 1.625000] mmcblk1: mmc0:cd6d SE08G 7.28 GiB
```

```
[ 1.630000] mmcblk1: p1(mmcblk1 为设备名 p1 为分区名)
```

挂载，注意不要挂在 EMMC 的分区

```
#mount -t vfat /dev/mmcblk1p1 /mnt
```

查看/mnt/目录即可看到 sd 卡中内容

实验八 USB 驱动的移植

【实验目的】

USB 接口是现在计算机系统中最通用的一种接口，

说明：在本系统移植课程实验中命令行提示符“\$”表示是在主机上执行，“#”表示在目标板执行

【实验环境】

- 主机：ubuntu 12.04 发行版
- 目标机：FS4412 平台
- 交叉编译工具：arm-none-linux-gnueabi-gcc

【实验步骤】

1. 修改设备树文件

```
$ vim arch/arm/boot/dts/exynos4412-fs4412.dts
```

添加如下内容：

```
usbphy: usbphy@125B0000 {
#address-cells = <1>;
#size-cells = <1>;

compatible = "samsung,exynos4x12-usb2phy";

reg = <0x125B0000 0x100>;

ranges;

clocks = <&clock 2>, <&clock 305>;

clock-names = "xusbxti", "otg";

usbphy-sys {

reg = <0x10020704 0x8 0x1001021c 0x4>;

};
```

```
};

ehci@12580000 {
    status = "okay";
    usbphy = <&usbphy>;
};

usb3503@08 {
    compatible = "smc,usb3503";
    reg = <0x08 0x4>;
    connect-gpios = <&gpm3 3 1>;
    intrn-gpios = <&gpx2 3 1>;
    reset-gpios = <&gpm2 4 1>;
    initial-mode = <1>;
};
```

2. 配置内核

```
make menuconfig

Device Drivers  --->

  [*] USB support  --->

    <*>      EHCI HCD (USB 2.0) support

    <*>EHCI support for Samsung S5P/EXYNOS SoC Series

    <*>      USB Mass Storage support

    <*>      USB3503 HSIC to USB20 Driver

    USB Physical Layer drivers  --->

      <*> Samsung USB 2.0 PHY controller Driver

    SCSI device support  --->

      <*> SCSI device support

      <*> SCSI disk support
```



```
<*> SCSI generic support
```

3. 编译内核和设备树

```
$ make uImage
```

```
$ make dtbs
```

4. 测试:

拷贝内核和设备树文件到/tftpboot 目录下

```
$ cp arm/arm/boot/uImage /tftpboot
```

```
$ cp arch/arm/boot/dts/exynos4412-fs4412.dtb /tftpboot/
```

启动目标板并在目标板上完成如下操作:

插入 U 盘显示如下

```
[ 72.695000] usb 1-3.2: USB disconnect, device number 3
[ 74.435000] usb 1-3.2: new high-speed USB device number 4 using exynos-ehci
[ 74.555000] usb-storage 1-3.2:1.0: USB Mass Storage device detected
[ 74.560000] scsi1 : usb-storage 1-3.2:1.0
[ 75.645000] scsi 1:0:0:0: Direct-Access      Kingston DataTraveler 160 PMAP PQ: 0 ANSI:
4
[ 75.660000] sd 1:0:0:0: Attached scsi generic sg0 type 0
[ 76.695000] sd 1:0:0:0: [sda] 15556608 512-byte logical blocks: (7.96 GB/7.41 GiB)
[ 76.700000] sd 1:0:0:0: [sda] Write Protect is off
[ 76.705000] sd 1:0:0:0: [sda] No Caching mode page found
[ 76.710000] sd 1:0:0:0: [sda] Assuming drive cache: write through
[ 76.725000] sd 1:0:0:0: [sda] No Caching mode page found
[ 76.730000] sd 1:0:0:0: [sda] Assuming drive cache: write through
[ 76.760000] sda: sda1 (sda 是设备名 sda1 是分区名)
[ 76.770000] sd 1:0:0:0: [sda] No Caching mode page found
[ 76.770000] sd 1:0:0:0: [sda] Assuming drive cache: write through
[ 76.780000] sd 1:0:0:0: [sda] Attached SCSI removable disk
```

在终端上执行挂载的设备与上边显示相关

```
# mount -t vfat /dev/sda1 /mnt
```

```
# ls
```

可以查看到 U 盘内容，即完成实验。

实验九内存调试

【实验目的】

段错误和内存错误是 C 语言编程经常遇到的问题，使用 memwatch 是由 johan lindh 编写，是一个开发源代码 C 语言内存错误检测工具。能检测双重释放，错误释放，没有释放内存，溢出等等情况。

【实验环境】

- 1、ubuntu 12.04 发行版
- 2、FS4412 平台
- 3、交叉编译器 arm-none-linux-gnueabi-gcc

【实验步骤】

- 1、解压 memwatch.-2.7.1.tar.gz，在解开的目录下添加代码 memtest.c

```
#include <stdlib.h>

#include <stdio.h>

#include "memwatch.h"

int main(int argc, char **argv)

{

    char *ptr1;

    char *ptr2;

        ptr1 = malloc(512);

        ptr2 = malloc(512);

ptr1[512]='A';

        ptr2 = ptr1;

    free(ptr2);

    free(ptr1);

    return 0;

}
```

- 2、修改 Makefile

Makefile 文件为：

```
memtest:
```

```
$(CC) -DMEMWATCH -DMW_STDIO memtest.cmemwatch.c
```

5、运行 make 并在主机上执行 a.out，执行后会生成一个记录文件 memwatch.log，内容如下：

```
===== MEMWATCH 2.71 Copyright (C) 1992-1999 Johan Lindh =====
```

```
Started at Thu Jan 1 00:08:33 1970
```

```
Modes: __STDC__ 32-bit mwDWORD==(unsigned long)
```

```
mwROUNDALLOC==4 sizeof(mwData)==32 mwDataSize==32
```

```
overflow: <3> memtest.c(12), 512 bytes alloc'd at <1> memtest.c(8)
```

```
double-free: <4> memtest.c(13), 0x1a1b4 was freed from memtest.c(12)
```

```
Stopped at Thu Jan 1 00:08:33 1970
```

```
unfreed: <2> test.c(9), 512 bytes at 0x1a3e4 {FE FE FE FE FE FE FE FE FE FE FE FE FE FE FE FE FE FE FE FE .....}
FE FE FE .....
```

```
Memory usage statistics (global):
```

```
N)umber of allocations made: 2
```

```
L)argest memory usage : 1024
```

```
T)otal of all alloc() calls: 1024
```

```
U)nfreed bytes totals : 512
```

//overflow: <3> memtest.c(12)缓冲区溢出，当程序执行到第 12 行 free(ptr2)才检测到的；
512 bytes alloc'd at <1>memtest.c(8)

//表示出错缓冲区的大小为 512 字节，是在 memtest.c 的第 8 行分配的。很容易发现代码的 ptr1[512]='A'出现错误。

```
double-free: <4> memtest.c(13), 0x1a7f4 was freed from memtest.c(12)
```

//double-free: <4> memtest.c(13)是一个双重释放的错误，表示程序执行到 13 行的时候才检测到。
0x1a7f4 was freed from memtest.c(12)

//表示首地址为 0x1a7f4 的内存 12 行已经被释放。

```
Stopped at Wed Dec 31 19:00:38 1969
```

```
unfreed: <2> test.c(9), 512 bytes at 0x1a3e4      {FE FE FE FE FE FE FE FE
```

//表示一块内存没有释放，表示这块内存是在 memtest.c 的第 9 行分配，大小为 512 字节，首地址为 0x1a3e4。

```
Memory usage statistics (global):
```

```
N)umber of allocations made: 2
```

```
L)argest memory usage      : 1024
```

//程序结束时能够使用的最大动态内存

```
T)otal of all alloc() calls: 1024 //总共分配的动态内存
```

```
U)nfreed bytes totals      : 512
```

//表示未释放的内存

实验十内核调试

【实验目的】

由于在驱动开发中经常看到内核崩溃的问题，最常见的就是 OOPS 错误，本实验要求学员掌握这种调试方法。

【实验环境】

- 1、ubuntu 12.04 发行版
- 2、FS4412 平台
- 3、交叉编译器 arm-none-linux-gnueabi-gcc

【实验步骤】

1、通过 OOPS 信息中 PC 寄存器的值可以知道出错指令的地址，通过栈回溯信息可以知道出错时的函数调用的关系，根据这两点可以很快定位错误。

2、修改 drivers/net/ethernet/davicom/dm9000.c，在 dm9000_probe 函数中 u32 id_val;下增加下面语句：

```
int *ptr =NULL;

*ptr=0xff;
```

3、编译内核下载到开发板上，内核启动会出现如类似下信息：

```
Unable to handle kernel NULL pointer dereference at virtual address 00000000

pgd = c0004000

[00000000] *pgd=00000000

Internal error: Oops: 805 [#1] PREEMPT SMP ARM

Modules linked in:

CPU: 1 PID: 1 Comm: swapper/0 Not tainted 3.14.0 #11

task: ee8a0000 ti: ee8a4000 task.ti: ee8a4000

PC is at dm9000_probe+0x1c/0x8f0

LR is at platform_drv_probe+0x18/0x48

pc : [<c0277cc8>]   lr : [<c0247f7c>]   psr: 60000153

sp : ee8a5e48   ip : 00000000   fp : 00000000

r10: c052a4fc   r9 : 00000000   r8 : c0591e98
```

r7 : 00000000 r6 : ee97c810 r5 : ee97c800 r4 : 00000000

r3 : 000000ff r2 : 00000000 r1 : ee8a5de8 r0 : ee97c800

Flags: nZCv IRQs on FIQs off Mode SVC_32 ISA ARM Segment kernel

Control: 10c5387d Table: 4000404a DAC: 00000015

Process swapper/0 (pid: 1, stack limit = 0xee8a4240)

Stack: (0xee8a5e48 to 0xee8a6000)

5e40: ee975cf0 00000000 ee1503a8 00000001 c0561afc ee150438
5e60: 00000000 ee97c810 c0591e98 ee97c810 00000000 c0591e98 c0561afc c052a4fc
5e80: 00000000 c0247f7c c0247f64 c05d931c c0591e98 c0246668 ee97c810 c0591e98
5ea0: ee97c844 00000000 c054332c c0246804 c0591e98 c0246778 00000000 c0244fbc
5ec0: ee805478 ee9771c0 c0591e98 eeb73a00 c0590028 c0245e28 c04c3128 c0591e98
5ee0: 00000000 c0591e98 00000000 c054e2ac c059f280 c0246e1c 00000000 ee8a4000
5f00: 00000000 c00087b4 ee903b00 c05c3d50 60000153 c0571c00 60000100 c0571c00
5f20: 00000000 00000000 c0571bfc 00000000 c0505bc8 ef7fc918 00000089 c0034c6c
5f40: c04ca680 c0505338 00000006 00000006 00000000 c054e2c8 c054e2cc 00000006
5f60: c054e2ac c059f280 00000089 c052a4fc 00000000 c052ac4c 00000006 00000006
5f80: c052a4fc c003e0dc 00000000 c03b46ec 00000000 00000000 00000000 00000000
5fa0: 00000000 c03b46f4 00000000 c000e4b8 00000000 00000000 00000000 00000000
5fc0: 0000000000000000 00000000 00000000 00000000 00000000 00000000 00000000
5fe0: 00000000 00000000 00000000 00000000 00000013 00000000 ffffffff ffffffff

[<c0277cc8>] (dm9000_probe) from [<c0247f7c>] (platform_drv_probe+0x18/0x48)

[<c0247f7c>] (platform_drv_probe) from [<c0246668>] (driver_probe_device+0x100/0x210)

[<c0246668>] (driver_probe_device) from [<c0246804>] (__driver_attach+0x8c/0x90)

[<c0246804>] (__driver_attach) from [<c0244fbc>] (bus_for_each_dev+0x58/0x88)

[<c0244fbc>] (bus_for_each_dev) from [<c0245e28>] (bus_add_driver+0xd8/0x1cc)

[<c0245e28>] (bus_add_driver) from [<c0246e1c>] (driver_register+0x78/0xf4)

[<c0246e1c>] (driver_register) from [<c00087b4>] (do_one_initcall+0x30/0x144)

```
[<c00087b4>] (do_one_initcall) from [<c052ac4c>] (kernel_init_freeable+0xfc/0x1c8)
```

```
[<c052ac4c>] (kernel_init_freeable) from [<c03b46f4>] (kernel_init+0x8/0xe4)
```

错误分析：

1、第一行

Unable to handle kernel NULL pointer dereference at virtual address 00000000

说明是空指针造成的错误

2、寄存器信息主要是 PC 的值

```
PC is at dm9000_probe+0x1c/0x8f0
```

和

```
pc : [<c0277cc8>]
```

错误定位：

```
$ arm-none-linux-gnueabi-objdump -D vmlinux > vmlinux.dis
```

文件 vmlinux.dis 非常大打开需要一定时间

```
662841 c0277cac <dm9000_probe>:
662842 c0277cac: e92d4ff0 push {r4, r5, r6, r7, r8, r9, sl, fp, lr}
662843 c0277cb0: e5909074 ldr r9, [r0, #116] ; 0x74
662844 c0277cb4: e3a04000 mov r4, #0
662845 c0277cb8: e3a030ff mov r3, #255 ; 0xff
662846 c0277cbc: e1590004 cmp r9, r4
662847 c0277cc0: e24dd01c sub sp, sp, #28
662848 c0277cc4: e1a05000 mov r5, r0
662849 c0277cc8: e5843000 str r3, [r4]
```

对于大多数情况，从反汇编代码定位到 C 代码并不会如此容易，需要有较强的阅读汇编代码的能力。

另外一种方法是通过 addr2line 去定位

```
$ arm-none-linux-gnueabi-addr2line 0xc0277cc8-e vmlinux -f
```

重要：该实验完成后不要忘记恢复被修改的代码

实验十一文件系统的移植

【实验目的】

熟悉 Linux 文件系统目录结构，创建自己的文件系统，通过 NFS 方式测试；用文件系统工具生成 ramdisk 文件系统映象文件。

【实验环境】

- 1、ubuntu 12.04 发行版
- 2、FS4412 平台
- 3、交叉编译器 arm-none-linux-gnueabi-gcc

【实验步骤】

一、根文件系统制作

1、源码下载

我们选择的版本是 busybox-1.22.1.tar.bz2 下载路径为：

<http://busybox.net/downloads/>

2、解压源码

```
$ tar xvf busybox-1.22.1.tar.bz2
```

3、进入源码目录

```
$ cd busybox-1.22.1
```

4、配置源码

```
$ make menuconfig
```

```
Busybox Settings --->
```

```
Build Options --->
```

```
[*] Build BusyBox as a static binary (no shared libs)
```

```
[ ] Force NOMMU build
```

```
[ ] Build with Large File Support (for accessing files > 2 GB)
```

```
(arm-none-linux-gnueabi-) Cross Compiler prefix
```

```
() Additional CFLAGS
```

5、编译

```
$ make
```

6、安装

busybox 默认安装路径为源码目录下的_install

```
$ make install
```

7、进入安装目录下

```
$ cd _install
```

```
$ ls
```

```
bin  linuxrc  sbin  usr
```

8、创建其他需要的目录

```
$ mkdir dev etc mnt proc var tmp sys root
```

9、添加库

- 将工具链中的库拷贝到_install 目录下

```
$ scp /home/linux/toolchain/gcc-4.6.4/arm-arm1176jzfssf-linux-gnueabi/lib/ . -a
```

- 删除静态库和共享库文件中的符号表

```
$ sudo rm lib/*.a
```

```
$ arm-none-linux-gnueabi-strip lib/*
```

- 删除不需要的库，确保所有库大小不超过 8M

```
$ du -mh lib/
```

10、添加系统启动文件

在 etc 下添加文件 inittab，文件内容如下：

```
#this is run first except when booting in single-user mode.

::sysinit:/etc/init.d/rcS

# /bin/sh invocations on selected ttys

# start an "askfirst" shell on the console (whatever that may be)

::askfirst:/bin/sh

# stuff to do when restarting the init process

::restart:/sbin/init

# stuff to do before rebooting
```

```
::ctrlaltdel:/sbin/reboot
```

在 etc 下添加文件 fstab，文件内容如下：

#device	mount-point	type	options	dump	fsck order
proc	/proc	proc	defaults	0	0
tmpfs	/tmp	tmpfs	defaults	0	0
sysfs	/sys	sysfs	defaults	0	0
tmpfs	/dev	tmpfs	defaults	0	0

这里我们挂载的文件系统有三个 proc、sysfs 和 tmpfs。在内核中 proc 和 sysfs 默认都支持，而 tmpfs 是没有支持的，我们需要添加 tmpfs 的支持

修改内核配置：

```
$ make menuconfig

File systems --->

    Pseudo filesystems --->

        [*] Virtual memory file system support (former shm fs)

        [*] Tmpfs POSIX Access Control Lists
```

重新编译内核

在 etc 下创建 init.d 目录，并在 init.d 下创建 rcS 文件，rcS 文件内容为：

```
#!/bin/sh

# This is the first script called by init process

/bin/mount -a

echo /sbin/mdev > /proc/sys/kernel/hotplug

/sbin/mdev -s
```

为 rcS 添加可执行权限：

```
$ chmod +x init.d/rcS
```

在 etc 下添加 profile 文件，文件内容为：

```
#!/bin/sh

export HOSTNAME=farsight

export USER=root
```

```
export HOME=root

export PS1="[$USER@$HOSTNAME \W]# "

PATH=/bin:/sbin:/usr/bin:/usr/sbin

LD_LIBRARY_PATH=/lib:/usr/lib:$LD_LIBRARY_PATH

export PATH LD_LIBRARY_PATH
```

重要：新制作的文件系统尺寸若超出 8M，删除不需要的库文件

二、NFS 测试

- 1、删除原先的/source/rootfs

```
$ sudo rm -rf /source/rootfs
```

- 2、将我们新建的根文件系统拷贝到/source/rootfs 目录下

```
$sudo mkdir /source/rootfs
```

```
$ sudo cp _install/* /source/rootfs -a
```

- 3、设置 uboot 环境变量

```
#setenv serverip 192.168.9.120
```

```
#setenv ipaddr 192.168.9.233
```

```
#setenv bootcmd tftp 41000000 uImage;tftp 42000000 exynos4412-fs4412.dtb;bootm 41000000
- 42000000
```

```
#setenv bootargs root=/dev/nfs nfsroot=192.168.9.120:/source/rootfs rw
console=ttySAC2,115200 init=/linuxrc ip=192.168.9.233
```

```
# saveenv
```

重新启动开发板，查看是否能够正常挂载，功能是否正常

```
[ 6.890000] VFS: Mounted root (nfs filesystem) on device 0:10.
[ 6.895000] devtmpfs: mounted
[ 6.895000] Freeing unused kernel memory: 228K (c052a000 - c0563000)
```

三、制作 ramdisk 文件系统

通过 NFS 测试以后，就可以制作 ramdisk 文件系统了，具体如下：

- 1、制作一个大小为 8M 的镜像文件

```
$ cd ~
```

```
$ dd if=/dev/zero of=ramdisk bs=1k count=8192 (ramdisk 为 8M)
```

- 2、格式化这个镜像文件为 ext2

```
$ mkfs.ext2 -F ramdisk
```

- 3、在 mount 下面创建 initrd 目录作为挂载点

```
$ sudo mkdir /mnt/initrd
```

- 4、将这个磁盘镜像文件挂载到/mnt/initrd 下

注意这里的 ramdisk 不能存放在 rootfs 目录中

```
$ sudo mount -t ext2 ramdisk /mnt/initrd
```

- 5、将我们的文件系统复制到 ramdisk 中

将测试好的文件系统里的内容全部拷贝到 /mnt/initrd 目录下面

```
$ sudo cp/source/rootfs/* /mnt/initrd -a
```

- 6、卸载 initrd

```
$ sudo umount /mnt/initrd
```

- 7、压缩 ramdisk 为 ramdisk.gz 并拷贝到/tftpboot 下

```
$ gzip --best -c ramdisk>ramdisk.gz
```

- 8、格式化为 uboot 识别的格式

```
$ mkimage -n "ramdisk" -A arm -O linux -T ramdisk -C gzip -d ramdisk.gz ramdisk.img
```

```
$ cp ramdisk.img /tftpboot
```

- 9、配置内核支持 RAMDISK

制作完 ramdisk.img 后，需要配置内核支持 RAMDISK 作为启动文件系统

```
make menuconfig
```

```
File systems --->
```

```
<*> Second extended fs support
```

```
Device Drivers
```

```
SCSI device support --->
```

```
<*> SCSI disk support
```

```
Block devices --->
```

```
<*>RAM block device support
```

```
(16)Default number of RAM disks
```

```
(8192) Default RAM disk size (kbytes) (修改为 8M)
```

```
General setup --->
```

```
[*] Initial RAM filesystem and RAM disk (initramfs/initrd) support
```

重新编译内核，复制到/tftpboot

10、在 U-BOOT 命令行重新设置启动参数：

```
#setenv bootcmd tftp 41000000 uImage;tftp 42000000 exynos4412-fs4412.dtb;tftp 43000000  
ramdisk.img;bootm 41000000 43000000 42000000  
  
# saveenv
```

重新启动开发板查看能否正常启动

四、ext4 文件系统制作

1、格式化工具制作

拷贝 e2fsprogs-1.42.5.tar.xz 到 Linux 下

解压

```
$ tar xvf e2fsprogs-1.42.5.tar.xz
```

进入工程

```
$ cd e2fsprogs-1.42.5
```

创建脚本 build.sh 并添加如下内容：

```
$ vim build.sh  
  
#!/bin/sh  
  
CC=arm-none-linux-gnueabi-gcc  
  
./configure --enable-elf-shlibs --host=arm-none-linux-gnueabi \\  
--prefix=/home/linux/tools  
  
make  
  
make install
```

执行脚本

```
./build.sh
```

在编译到最后可能会有个错误如下：

```
make[1]: [libext2fs.dvi] Error 1 (ignored)
```

这个不要紧其实我们需要的文件都编译好了

拷贝可执行文件和库到文件系统中

```
$ cp /home/linux/tools/sbin/mkfs.ext3 /source/rootfs/sbin
```

```
$ cp /home/linux/tools/lib/* /source/rootfs/lib
```

2、修改 u-boot 启动参数

```
setenv bootargs root=/dev/nfs rwnfsroot=192.168.9.120:/source/rootfs init=/linuxrc
console=ttySAC2,115200 ip=192.168.9.233
```

注意：192.168.9.120 对应 Ubuntu 的 ip

192.168.9.233 对应板子的 ip

这两个 ip 应该根据自己的实际情况适当修改

3、分区

启动开发板在倒计时期间按任意键结束启动，执行如下命令

```
# fdisk -c 0
```

4、格式化

重新启动开发板进入系统后执行

```
# mkfs.ext3 -F /dev/mmcblk0p2
```

5、修改我们/source/rootfs/etc/fstab

在最后添加如下内容

```
/dev/mmcblk0p2 /mnt ext3 defaults 0 0
```

重新启动系统系统在启动最后会挂载 mmcblk0p2 作为用户文件系统