实验一 简单模块实验（1）

【实验内容】

编写一个最基本的模块，加载模块，观察结果

【实验目的】

掌握模块的基本要素及加载、卸载、查看工具的使用

【实验平台】

主机：Ubuntu 12.04

主机内核版本: 3.2.0-29-generic-pae

【实验步骤】

注意：在实验过程中″$″后的操作在主机上，″#″后的操作在开发板上

1、 拷贝模块源码目录

将实验代码中的 ex1-hello-world 拷贝到/home/linux 目录下并进入 ex1-hello-world 目录。

2、 编译模块

$ make

3、 插入模块

$ sudo insmod hello.ko

4、 查看内核打印信息

$ dmesg

可以看到 init_module 函数中的打印语句。

5、 查看模块

$ lsmod | grep hello

结果：

hello 578 0

6、 卸载模块

$ sudo rmmod hello

7、 查看内核打印信息

$ dmesg

可以看到 cleanup_module 函数中的打印语句。

实验二 简单模块实验（2）

【实验内容】

编写一个最基本的模块，加载模块，观察结果

【实验目的】

掌握 module_init 和 module_exit 的使用

【实验平台】

主机：Ubuntu 12.04

主机内核版本: 3.2.0-29-generic-pae

【实验步骤】

注意：在实验过程中"$"后的操作在主机上，"#"后的操作在开发板上

1、 拷贝模块源码目录

将实验代码中的 ex2-init-exit 拷贝到/home/linux 目录下并进入 ex2-init-exit 目录。

2、 编译模块

$ make

3、 插入模块

$ sudo insmod hello.ko

4、 查看内核打印信息

$ dmesg

可以看到 module_init 对于函数中的打印语句。

5、 查看模块

$ lsmod | grep hello

结果：

hello 607 0

6、 卸载模块

$ sudo rmmod hello

7、 查看内核打印信息

$ dmesg

可以看到 module_exit 函数中的打印语句。

实验三 简单模块实验（3）

【实验内容】

编写一个最基本的模块，加载模块，观察结果

【实验目的】

掌握宏 MODULE_LICENSE 、MODULE_AUTHOR、MODULE_DESCRIPTION 和 MODULE_SUPPORTED_DEVICE 的使用及 modinfo 的使用

【实验平台】

主机：Ubuntu 12.04

主机内核版本: 3.2.0-29-generic-pae

【实验步骤】

注意：在实验过程中"$"后的操作在主机上，"#"后的操作在开发板上

1、 拷贝模块源码目录

将实验代码中的 ex3-doc-license 拷贝到/home/linux 目录下并进入 ex3-doc-license 目录。

2、 编译模块

$ make

3、 插入模块

$ sudo insmod hello.ko

4、 查看内核打印信息

$ dmesg

可以看到 module_init 对于函数中的打印语句。

5、 查看模块

$ lsmod | grep hello

结果：

hello 607 0

6、 卸载模块

$ sudo rmmod hello

7、 查看内核打印信息

$ dmesg

可以看到 module_exit 函数中的打印语句。

8、 查看模块信息

$ modinfo hello.ko

结果：

filename: hello.ko
description: A sample driver
author: Foo bar
license: GPL
srcversion: 72EA5679DAA4302DB9F9F41
depends:
vermagic: 3.2.0-29-generic-pae SMP mod_unload modversions 686

实验四 简单模块实验（4）

【实验内容】

编写一个最基本的模块，加载模块，观察结果

【实验目的】

掌握向模块传递参数的方法

【实验平台】

主机：Ubuntu 12.04

主机内核版本: 3.2.0-29-generic-pae

【实验步骤】

注意：在实验过程中″$″后的操作在主机上，″#″后的操作在开发板上

1、 拷贝模块源码目录

将实验代码中的 ex4-param 拷贝到/home/linux 目录下并进入 ex4-param 目录。

2、 编译模块

$ make

3、 插入模块

$ sudo insmod hello.ko myshort=55 myint=456 mylong=4567 mystring="foobar" array=2,3

4、 查看内核打印信息

$ dmesg

结果：

[2619.717248] myshort is a short integer: 55

[2619.717251] myint is an integer: 456

[2619.717252] mylong is a long integer: 4567

[2619.717254] mystring is a string: foobar

[2619.717254]

[2619.717255] array[0] = 2

[2619.717256] array[1] = 3

[2619.717257] Got 2 arguments in array

5、 查看模块

$ lsmod | grep hello

结果：

hello 1470 0

6、 卸载模块

$ sudo rmmod hello

7、 查看内核打印信息

$ dmesg

可以看到 module_exit 函数中的打印语句。

8、 查看模块信息

$ modinfo hello.ko

filename: hello.ko

description: A sample driver

```
author:         Foobar
license:        GPL
srcversion:     DE2B92CA388A3F4EE7915E0
depends:
vermagic:       3.2.0-29-generic-pae SMP mod_unload modversions 686
parm:           myshort:A short integer (short)
parm:           myint:An integer (int)
parm:           mylong:A long integer (long)
parm:           mystring:A character string (charp)
parm:           array:An array of integers (array of int)
```

实验五 简单模块实验（5）

【实验内容】

编写一个最基本的模块，加载模块，观察结果

【实验目的】

掌握对文件编译

【实验平台】

主机：Ubuntu 12.04

主机内核版本: 3.2.0-29-generic-pae

【实验步骤】

注意：在实验过程中″$″后的操作在主机上，″#″后的操作在开发板上

1、 拷贝模块源码目录

将实验代码中的 ex5-multi-file 拷贝到/home/linux 目录下并进入 ex5-multi-file 目录。

2、 编译模块

$ make

3、 插入模块

$ sudo insmod final.ko

4、 查看内核打印信息

$ dmesg

可以看到 module_init 对于函数中的打印语句。

5、 查看模块

$ lsmod | grep final

结果：

final 607 0

6、 卸载模块

$ sudo rmmod final

7、 查看内核打印信息

$ dmesg

可以看到 module_exit 函数中的打印语句。

8、 查看 Makefile 比较与前面实验 Makefile 的差别

实验六 简单字符驱动实验（1）

【实验内容】

添加字符设备设备号的注册

【实验目的】

掌握字符驱动设备号的注册

【实验平台】

主机：Ubuntu 12.04

主机内核版本: 3.2.0-29-generic-pae

【实验步骤】

注意：在实验过程中"$"后的操作在主机上，"#"后的操作在开发板上

1、 拷贝模块源码目录

将实验代码中的 ex6-char-register 拷贝到/home/linux 目录下并进入 ex6-char-register 目录。

2、 编译模块

$ make

3、 插入模块

$ sudo insmod char-reg.ko

4、 查看设备号注册信息

$ cat /proc/device

结果：

character devices:

……

250 hello

……

5、 卸载模块

$ sudo rmmod char_reg

6、 查看设备号注册信息

$ cat /proc/device

结果没有 hello 的注册信息

实验七 简单字符驱动实验（2）

【实验内容】

添加字符设备的注册

【实验目的】

掌握字符驱动设备的注册

【实验平台】

主机：Ubuntu 12.04

主机内核版本: 3.2.0-29-generic-pae

【实验步骤】

注意：在实验过程中"$"后的操作在主机上，"#"后的操作在开发板上

1、 拷贝模块源码目录

将实验代码中的 ex7-char-dev-register 拷贝到 /home/linux 目录下并进入 ex7-char-dev-register 目录。

2、 编译模块

$ make

3、 插入模块

$ sudo insmod char-dev-reg.ko

4、 查看设备号注册信息

$ cat /proc/device

结果：

character devices:

……

250 hello

……

5、 卸载模块

$ sudo rmmod char_dev_reg

6、 查看设备号注册信息

$ cat /proc/device

结果没有 hello 的注册信息

实验八 简单字符驱动实验（3）

【实验内容】

添加 open，release 功能，并编写应用程序调用驱动相关代码

【实验目的】

掌握 open，release 的添加

【实验平台】

主机：Ubuntu 12.04

主机内核版本: 3.2.0-29-generic-pae

【实验步骤】

注意：在实验过程中"$"后的操作在主机上，"#"后的操作在开发板上

1、 拷贝模块源码目录

将实验代码中的 ex8-open 拷贝到/home/linux 目录下并进入 ex8-open 目录。

2、 编译模块

$ make

3、 编译应用程序

$ gcc -o test test.c

4、 插入模块

$ sudo insmod char-open.ko

5、 创建设备节点

$ mknod /dev/hello c 250 0

6、 运行可执行程序

$ ./test

7、 查看内核信息

$ dmesg

可以发现在 open 和 release 中的打印语句分别被打印出来

8、 卸载模块

$ sudo rmmod char_open

实验九 简单字符驱动实验（4）

【实验内容】

添加 read 功能，并编写应用程序操作驱动

【实验目的】

掌握字符驱动 write 功能的添加

【实验平台】

主机：Ubuntu 12.04

主机内核版本: 3.2.0-29-generic-pae

【实验步骤】

注意：在实验过程中"$"后的操作在主机上，"#"后的操作在开发板上

1、 拷贝模块源码目录

将实验代码中的 ex9-read 拷贝到/home/linux 目录下并进入 ex9-read 目录。

2、 编译模块

$ make

3、 编译应用程序

$ gcc -o test test.c

4、 插入模块

$ sudo insmod char-read.ko

5、 创建设备节点

$ mknod /dev/hello c 250 0

6、 运行可执行程序

$ ./test

7、 查看内核信息

$ dmesg

可以发现在 write 中的打印语句分别被打印出来

8、 卸载模块

$ sudo rmmod char_read

实验十 简单字符驱动实验（5）

【实验内容】

添加 write 功能，并编写应用程序操作驱动

【实验目的】

掌握字符驱动 write 功能的添加

【实验平台】

主机：Ubuntu 12.04

主机内核版本: 3.2.0-29-generic-pae

【实验步骤】

注意：在实验过程中″$″后的操作在主机上，″#″后的操作在开发板上

1、 拷贝模块源码目录

将实验代码中的 ex10-write 拷贝到/home/linux 目录下并进入 ex10-write 目录。

2、 编译模块

$ make

3、 编译应用程序

$ gcc -o test test.c

4、 插入模块

$ sudo insmod char-write.ko

5、 创建设备节点

$ mknod /dev/hello c 250 0

6、 运行可执行程序

$ ./test

7、 查看内核信息

$ dmesg

可以发现在 write 中的打印语句分别被打印出来

8、 卸载模块

$ sudo rmmod char_write

实验十一 简单字符驱动实验（6）

【实验内容】

添加 ioctl 功能，并编写应用程序操作驱动

【实验目的】

掌握字符驱动 ioctl 功能的添加

【实验平台】

主机：Ubuntu 12.04

主机内核版本: 3.2.0-29-generic-pae

【实验步骤】

注意：在实验过程中"$"后的操作在主机上，"#"后的操作在开发板上

1、 拷贝模块源码目录

将实验代码中的 ex11-ioctl 拷贝到/home/linux 目录下并进入 ex11-ioctl 目录。

2、 编译模块

$ make

3、 编译应用程序

$ gcc -o test test.c

4、 插入模块

$ sudo insmod hello-ioctl.ko

5、 创建设备节点

$ mknod /dev/hello c 250 0

6、 运行可执行程序

$ ./test

7、 查看内核信息

$ dmesg

可以发现在 write 中的打印语句分别被打印出来

8、 卸载模块

$ sudo rmmod hello_ioctl

实验十二 简单字符驱动实验（7）

【实验内容】

添加 debug 控制功能

【实验目的】

掌握驱动调试技巧

【实验平台】

主机：Ubuntu 12.04

主机内核版本: 3.2.0-29-generic-pae

【实验步骤】

注意：在实验过程中"$"后的操作在主机上，"#"后的操作在开发板上

1、 拷贝模块源码目录

将实验代码中的 ex12-debug 拷贝到/home/linux 目录下并进入 ex12-debug 目录。

2、 编译模块

$ make

3、 插入模块

$ sudo insmod hello.ko

4、 查看内核信息

$ dmesg

可以发现在 module_init 函数中的打印语句被打了出来

5、 卸载模块

$ sudo rmmod hello

6、 修改 Makefile

将：DEBUG = y

改为：DEBUG = n

7、 重新编译模块

$ make

8、 插入模块

$ sudo rmmod hello

9、 查看内核信息

$ dmesg

可以发现在 module_init 函数中的打印语句没有打印出来

实验十三 简单字符驱动实验（8）

【实验内容】

添加识别多设备功能

【实验目的】

掌握驱动调试技巧

【实验平台】

主机：Ubuntu 12.04

主机内核版本: 3.2.0-29-generic-pae

【实验步骤】

注意：在实验过程中″$″后的操作在主机上，″#″后的操作在开发板上

1、 拷贝模块源码目录

将实验代码中的 ex13-multi-dev 拷贝到/home/linux 目录下并进入该目录。

2、 编译模块

$ make

3、 编译应用程序

$ gcc -o test test.c

4、 插入模块

$ sudo insmod char-multi-dev.ko

5、 创建设备节点

$ mknod /dev/hello0 c 250 0

$ mknod /dev/hello1 c 250 1

6、 运行可执行程序

$ ./test

7、 卸载模块

$ sudo rmmod char_multi_dev

实验十四 waitqueue 实验

【实验内容】

编写一个字符设备驱动，包含等待队列、信号量等功能

【实验目的】

掌握待队列、信号量等功能的实现方法

【实验平台】

主机：Ubuntu 12.04

主机内核版本: 3.2.0-29-generic-pae

【实验步骤】

注意：在实验过程中″$″后的操作在主机上，″#″后的操作在开发板上

1、 拷贝模块源码目录

将实验代码中的 waitqueue 拷贝到/home/linux 目录下并进入该目录。

2、 编译模块

$ make

3、 插入模块

$ sudo insmod waitqueue.ko

4、 创建设备节点

$ sudo mknod /dev/hello c 250 0

5、 测试 waitqueue 功能

$ su root

$ cat /dev/hello

打开另外一个终端

$ su root

$ echo 1234 > /dev/hello

可以发现在前面终端会打印出 1234

退出 root 模式

6、 卸载模块

$ sudo rmmod waitqueue

实验十五 poll 实验

【实验内容】

编写一个字符设备驱动，包含等待队列、信号量、fifo、poll 等功能

【实验目的】

掌握待 poll 功能的实现方法

【实验平台】

主机：Ubuntu 12.04

主机内核版本: 3.2.0-29-generic-pae

【实验步骤】

注意：在实验过程中"$"后的操作在主机上，"#"后的操作在开发板上

1、 拷贝模块源码目录

将实验代码中的 poll 目录拷贝到/home/linux 目录下并进入该目录。

2、 编译模块

$ make

3、 编译应用程序

$ gcc test_poll.c -o test_poll

4、 插入模块

$ sudo insmod poll.ko

5、 创建设备节点

$ sudo mknod /dev/hello c 250 0

6、 测试 poll 功能

$ su root

$ ./test_poll

测试程序会同时监控标准输入和/dev/hello

若当前前终端有输入，则打印输入的内容；若/dev/hello 有数据，则读取内容并显示

打开另外一个终端

$ su root

$ echo hello > /dev/hello

7、 卸载模块

$ sudo rmmod poll

实验十六 异步通知实验

【实验内容】

编写一个字符设备驱动，包含等待队列、信号量、poll、fasync 等功能

【实验目的】

掌握待 poll 功能的实现方法

【实验平台】

主机：Ubuntu 12.04

主机内核版本: 3.2.0-29-generic-pae

【实验步骤】

注意：在实验过程中"$"后的操作在主机上，"#"后的操作在开发板上

1、 拷贝模块源码目录

将实验代码中的 fasync 目录拷贝到/home/linux 目录下并进入该目录。

2、 编译模块

$ make

3、 编译应用程序

$ gcc asyncmonitor.c -o asyncmonitor

4、 插入模块

$ sudo insmod fasync.ko

5、 创建设备节点

$ sudo mknod /dev/hello c 250 0

6、 测试 fasync 功能

$ su root

$ ./asyncmonitor

终端没有任何现象

打开另外一个终端

$ su root

$ echo 1234 > /dev/hello

退出 root 模式

7、 卸载模块

$ sudo rmmod fasync

实验十七 tasklet 实验

【实验内容】

在 globalfifo 的实验中加入 tasklet 的功能

【实验目的】

掌握 tasklet 使用

【实验平台】

主机：Ubuntu 12.04

主机内核版本: 3.2.0-29-generic-pae

【实验步骤】

注意：在实验过程中"$"后的操作在主机上，"#"后的操作在开发板上

1、 拷贝模块源码目录

将实验代码中的 tasklet 拷贝到/home/linux 目录下并进入 tasklet 目录。

2、 编译模块

$ make

3、 插入模块

$ sudo insmod tasklet.ko

4、 创建设备节点

$ sudo mknod /dev/hello c 250 0

5、 测试 tasklet 功能

$ su root

$ echo 1234 > /dev/hello

通过 dmesg 查看内核信息

```
[18397.657734] written 6 bytes(s),current_len:6
[18397.657740] in write jiffies=4526777
[18397.657745] in jit_tasklet_fn  jiffies=4526777
[18398.362106] written 6 bytes(s),current_len:12
[18398.362111] in write jiffies=4526953
[18398.362114] in jit_tasklet_fn  jiffies=4526953
```

退出 root 模式

6、 卸载模块

$ sudo rmmod tasklet

实验十八 工作队列实验

【实验内容】

在 globalfifo 的实验中加入 work_queue 的功能

【实验目的】

掌握 work_queue 使用

【实验平台】

主机：Ubuntu 12.04

主机内核版本: 3.2.0-29-generic-pae

【实验步骤】

注意：在实验过程中"$"后的操作在主机上，"#"后的操作在开发板上

1、 拷贝模块源码目录

将实验代码中的 workqueue 拷贝到/home/linux 目录下并进入 workqueue 目录。

2、 编译模块

$ make

3、 插入模块

$ sudo insmod workqueue.ko

4、 创建设备节点

$ sudo mknod /dev/hello c 250 0

5、 测试 workqueue 功能

$ su root

$ echo 1234 > / dev/hello

通过 dmesg 查看内核信息

```
[18571.999270] written 6 bytes(s),current_len:6
[18571.999276] in write jiffies=4570385
[18571.999834] in my_do_work  jiffies=4570385
[18572.942459] written 6 bytes(s),current_len:12
[18572.942465] in write jiffies=4570621
[18572.942979] in my_do_work  jiffies=4570621
```

退出 root 模式

6、 卸载模块

$ sudo rmmod workqueue

实验十九 内核定时器实验

【实验内容】

使用内核定时器编写一个秒表的字符设备驱动

【实验目的】

掌握内核定时器的使用

【实验平台】

主机：Ubuntu 12.04

主机内核版本: 3.2.0-29-generic-pae

【实验步骤】

注意：在实验过程中"$"后的操作在主机上，"#"后的操作在开发板上

1、 拷贝模块源码目录

将实验代码中的 second 拷贝到/home/linux 目录下并进入 second 目录。

2、 编译模块

$ make

3、 编译应用程序

$ gcc second_test.c -o second_test

4、 插入模块

$ sudo insmod second.ko

5、 创建设备节点

$ sudo mknod /dev/second c 250 0

6、 测试 second 功能

$ su root

$ ./second_test

终端会每秒打印一个信息如下：

```
seconds after open /dev/second :1
seconds after open /dev/second :2
seconds after open /dev/second :3
seconds after open /dev/second :4
seconds after open /dev/second :5
```

通过 dmesg 查看内核信息

```
[17881.755785] current jiffies is 4397734
[17882.753781] current jiffies is 4397984
[17883.753288] current jiffies is 4398234
[17884.754710] current jiffies is 4398484
[17885.753284] current jiffies is 4398734
```

退出 root 模式

7、 卸载模块

$ sudo rmmod second

实验二十 平台设备驱动实验

【实验内容】

添加一个 platform_driver 和一个 platform_devive 并完成这个结构的匹配

【实验目的】

掌握平台设备驱动的编写

【实验平台】

主机：Ubuntu 12.04

主机内核版本: 3.2.0-29-generic-pae

【实验步骤】

注意：在实验过程中"$"后的操作在主机上，"#"后的操作在开发板上

1、 拷贝模块源码目录

将实验代码中的 s5pc100_platform 拷贝到/home/linux 目录下并进入 s5pc100_platform 目录。

2、 编译模块

$ make

3、 插入模块

$ sudo insmod s5pc100_platform.ko

4、 查看内核信息

$ dmesg

```
[ 2242.791878] platform: match ok!
[ 2242.793585] platform: driver installed
```

5、 卸载模块

$ sudo rmmod s5pc100_platform

6、 查看内核信息

$ dmesg

```
[ 2258.951942] platform: driver remove
[ 2258.955708] platform: device remove
[ 2258.955740] platform: driver uninstalled!
```

实验代码可以拆分成两个程序，大家可以尝试把 platform_device 和 platform_driver 分别放在两个程序中进行注册

实验二十一 LED 灯驱动编写及测试

【实验内容】

编写一个字符驱动程序操作目标板的 LED 的亮灭

【实验目的】

掌握 GPIO 的操作

【实验平台】

主机：Ubuntu 12.04

目标机：FS4412

目标机内核版本：3.14.0

交叉编译器版本：arm-none-linux-gnueabi-gcc-4.6.4

【实验步骤】

1、 拷贝模块源码目录

将实验代码中的 fs4412_led 目录拷贝到/home/linux 目录下并进入该目录。

2、 编译模块

$ make

3、 编译应用程序

$ arm-none-linux-gnueabi-gcc test.c -o test

4、 将 ko 文件和测试程序拷贝到根文件系统中

$ cp *.ko test /source/rootfs

5、 板子上插入模块

sudo insmod fs4412_led.ko

6、 创建设备节点

mknod /dev/led c 500 0

7、 测试功能

./test

查看灯的状态

实验二十二 PWM 驱动编写

【实验内容】

编写一个字符驱动程序操作蜂鸣器发声

【实验目的】

掌握 GPIO 的操作及 linux 下 pwm timer 的使用

【实验平台】

主机：Ubuntu 12.04

目标机：FS4412

目标机内核版本：3.14.0

交叉编译器版本：arm-none-linux-gnueabi-gcc-4.6.4

【实验步骤】

1、 拷贝模块源码目录

将实验代码中的 fs4412_led 目录拷贝到/home/linux 目录下并进入该目录。

2、 编译模块

$ make

3、 编译应用程序

$ arm-none-linux-gnueabi-gcc pwm_music.c -o pwm_music

4、 将 ko 文件和测试程序拷贝到根文件系统中

$ cp *.ko pwm_music /source/rootfs

5、 板子上插入模块

sudo insmod fs4412_pwm.ko

6、 创建设备节点

mknod /dev/pwm c 500 0

7、 测试功能

./pwm_music

实验二十三 键盘驱动编写

【实验内容】

编写一个字符驱动程序捕捉键盘的动作

【实验目的】

掌握 GPIO 的操作及 linux 下中断的使用

【实验平台】

主机：Ubuntu 12.04

目标机：FS4412

目标机内核版本：3.14.0

交叉编译器版本：arm-none-linux-gnueabi-gcc-4.6.4

【实验步骤】

1、 进入内核源码修改 arm/arm/boot/dts/exynos4412-fs4412.dts 文件,添加如下内容：

```
fs4412-key {
        compatible = "fs4412,key";
        interrupt-parent = <&gpx1>;
        interrupts = <1 2>, <2 2>;
};
```

2、 重新编译 dts 文件并拷贝到/tftpboot 目录下

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

3、 拷贝模块源码目录

将实验代码中的 fs4412_key 目录拷贝到/home/linux 目录下并进入该目录。

4、 编译模块

```
$ make
```

5、 将 ko 文件和测试程序拷贝到根文件系统中

```
$ cp *.ko   /source/rootfs
```

6、 板子上插入模块

```
# sudo   insmod   fs4412_key.ko
```

7、 测试功能

按 key2 或 key3 查看现象

实验二十五 ADC 驱动编写及测试

【实验内容】

编写一个字符驱动程序获取 ADC 通道 1 的电压值

【实验目的】

掌握 ADC 设备驱动的编写及平台设备驱动的编写

【实验平台】

主机：Ubuntu 12.04

目标机：FS4412

目标机内核版本：3.14.0

交叉编译器版本：arm-none-linux-gnueabi-gcc-4.6.4

【实验步骤】

1、 进入内核源码修改 arm/arm/boot/dts/exynos4412-fs4412.dts 文件,添加如下内容：

```
fs4412-adc@126c0000{
        compatible = "fs4412,adc";
        reg = <0x126c0000 0x20>;
        interrupt-parent = <&combiner>;
        interrupts = <10 3>;
};
```

2、 重新编译 dts 文件并拷贝到/tftpboot 目录下

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

3、 拷贝模块源码目录

将实验代码中的 fs4412_adc 目录拷贝到/home/linux 目录下并进入该目录。

4、 编译模块

```
$ make
```

5、 编译应用程序

```
$ arm-none-linux-gnueabi-gcc  test.c  -o  test
```

6、 将 ko 文件和测试程序拷贝到根文件系统中

```
$ cp *.ko test /source/rootfs
```

7、 板子上插入模块

```
# sudo  insmod  fs4412_adc.ko
```

8、 创建设备节点

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

9、 测试功能

```
# ./test
```

旋转 VR1 电位计，查看终端显示

实验二十六 I2C 驱动编写及测试

【实验内容】

编写一个基于 i2c 总线的 E2PROM 的驱动

【实验目的】

掌握 I2C 总线的操作方法

【实验平台】

主机：Ubuntu 12.04

目标机：FS4412

目标机内核版本：3.14.0

交叉编译器版本：arm-none-linux-gnueabi-gcc-4.6.4

【实验步骤】

注意：在实验过程中"$"后的操作在主机上，"#"后的操作在开发板上

1、 进入内核源码修改 arm/arm/boot/dts/exynos4412-fs4412.dts 文件,添加如下内容：

```
i2c@138B0000 {
        samsung,i2c-sda-delay = <100>;
        samsung,i2c-max-bus-freq = <20000>;
        pinctrl-0 = <&i2c5_bus>;
        pinctrl-names = "default";
        status = "okay";

        mpu6050-3-axis@68 {
            compatible = "invensense,mpu6050";
            reg = <0x68>;
            interrupt-parent = <&gpx3>;
            interrupts = <3 2>;
        };
    };
```

2、 重新编译 dts 文件并拷贝到/tftpboot 目录下

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

3、 拷贝模块源码目录

将实验代码中的 mpu6050 目录拷贝到/home/linux 目录下并进入该目录。

4、 编译模块

```
$ make
```

5、 编译应用程序

```
$ arm-none-linux-gnueabi-gcc    test.c    -o    test
```

6、 将 ko 文件和测试程序拷贝到根文件系统中

```
$ cp *.ko test /source/rootfs
```

7、 板子上插入模块

```
# sudo    insmod    mpu6050.ko
```

8、 创建设备节点

```
# mknod    /dev/mpu6050    c    500    0
```

9、 测试功能

./test

旋转板子，查看终端显示

实验二十七 sbull 驱动

【实验内容】

在 Ubantu12.04 系统上编写一个 sbull 驱动，将一段 ram 空间模拟为 disk 使用。并在这个 disk 上建立文件系统。

【实验目的】

掌握块设备的编写方法。

【实验平台】

主机：Ubuntu 12.04

【实验步骤】

注意：在实验过程中"$"后的操作在主机上，"#"后的操作在开发板上

1、拷贝模块到虚拟机下

将实验代码中的 sbull 拷贝到驱动目录下并进入 sbull 目录

2、编译

$ make

3、通过 insmod 命令将模块加入内核

$sudo insmod sbull.ko

$cat /proc/partitions

4、分区

$ sudo fdisk /dev/sbulla

出现磁盘分区界面，选择 m 出现帮助信息

```
linux@ubuntu:~/workdir/test/sbull_farsight_2.6.35$ sudo fdisk /dev/sbulla
Device contains neither a valid DOS partition table, nor Sun, SGI or OSF disklabel
Building a new DOS disklabel with disk identifier 0xbbc6ee88.
Changes will remain in memory only, until you decide to write them.
After that, of course, the previous content won't be recoverable.

Warning: invalid flag 0x0000 of partition table 4 will be corrected by w(rite)

WARNING: DOS-compatible mode is deprecated. It's strongly recommended to
         switch off the mode (command 'c') and change display units to
         sectors (command 'u').

Command (m for help):
```

选择 n 添加新的分区

```
Command (m for help): n
Command action
   e   extended
   p   primary partition (1-4)
```

选择 p 建立一个新的主分区

选择 w，保存分区信息

```
Command action
   e   extended
   p   primary partition (1-4)
p
Partition number (1-4): 1
First cylinder (1-400, default 1): 1
Last cylinder, +cylinders or +size{K,M,G} (1-400, default 400): 400

Command (m for help): w
The partition table has been altered!

Calling ioctl() to re-read partition table.
Syncing disks.
```

5、格式化 disk

$ sudo mkfs.ext3 /dev/sbulla1

6、挂载文件系统

$ sudo mkdir /mnt/disk

$ sudo mount -t ext3 /dev/sbulla1 /mnt/disk

拷贝任意文件到/mnt/disk 下

7、卸载文件系统

$ sudo umount /mnt/disk

8、卸载模块

$ sudo rmmod sbull