目 录

1. LDD6410 硬软件特性.....3
1.1 LDD6410 的电路板组成和结构.....3
1.2 LDD6410 的启动跳线设置.....6
1.3 LDD6410 的软件特性.....7
2. LDD6410 Linux 开发完全剖析.....8
2.1 建立工具链和开发环境.....8
2.1.1 S3C6410XToolChain4.2.2EABI.....8
2.1.2 strace、gdbserver 和 armlinuxgdb.....9
2.1.3 gdb 调试器的用法.....9
2.1.4 主机端 tftp 和 nfs 服务安装.....11
2.2 UBOOT.....11
2.2.1 U-BOOT 的移植.....11
2.2.2 NAND 版 UBOOT.....12
2.2.3 SD 卡 版 UBOOT.....13
2.2.4 从 tftp 服务器引导 Linux.....13
2.3 Linux 内核和驱 动.....14
2.3.1 LDD6410 内核和 BSP.....14
2.3.2 按键驱动.....16
2.3.3 LED 驱动.....17
2.3.4 LCD 和 VGA 驱动.....18
2.3.5 WM9714 声 卡 ASoC 驱 动.....23
2.3.6 DM9000 网 卡 驱 动.....23
2.3.7 USB 驱动.....24
2.3.8 驱动学习实例：helloworld、globalmem、globalfifo.....26
2.4 根文件系统.....27
2.4.1 根文件系统的组成.....27
2.4.2 使用 nfs 作为根文件系统.....32
2.5 应用程序.....32
2.5.1 按键测试程序.....32
2.5.2 USB 鼠标测试程序.....36
2.5.3 触摸屏测试程序.....38
2.5.4 framebuffer 测 试 程 序.....41
2.5.4 jpegview 图片浏览器.....42
2.5.5 mplayer 媒体播放器.....43
2.5.6 appweb 服 务 器.....44
2.6 Android.....44
2.6.1 内核 Android 补丁.....44
2.6.2 Android 文件系统.....46
2.7 Qt Embedded.....48
2.7.1 Tslib 和 ts_calibration.....48
2.7.2 Qt.....50
3. LDD6410 Linux 软件包的烧录与更新.....52
3.1 SD 卡烧录.....52
3.2 NAND 烧录.....53

3.2.1 更新 NAND 版 UBOOT.....53
3.2.2 更新 NAND 分区中的 Linux 内核.....53
3.2.3 更新 NAND 分区中的文件系统.....54

LDD6410 开发板用户手册

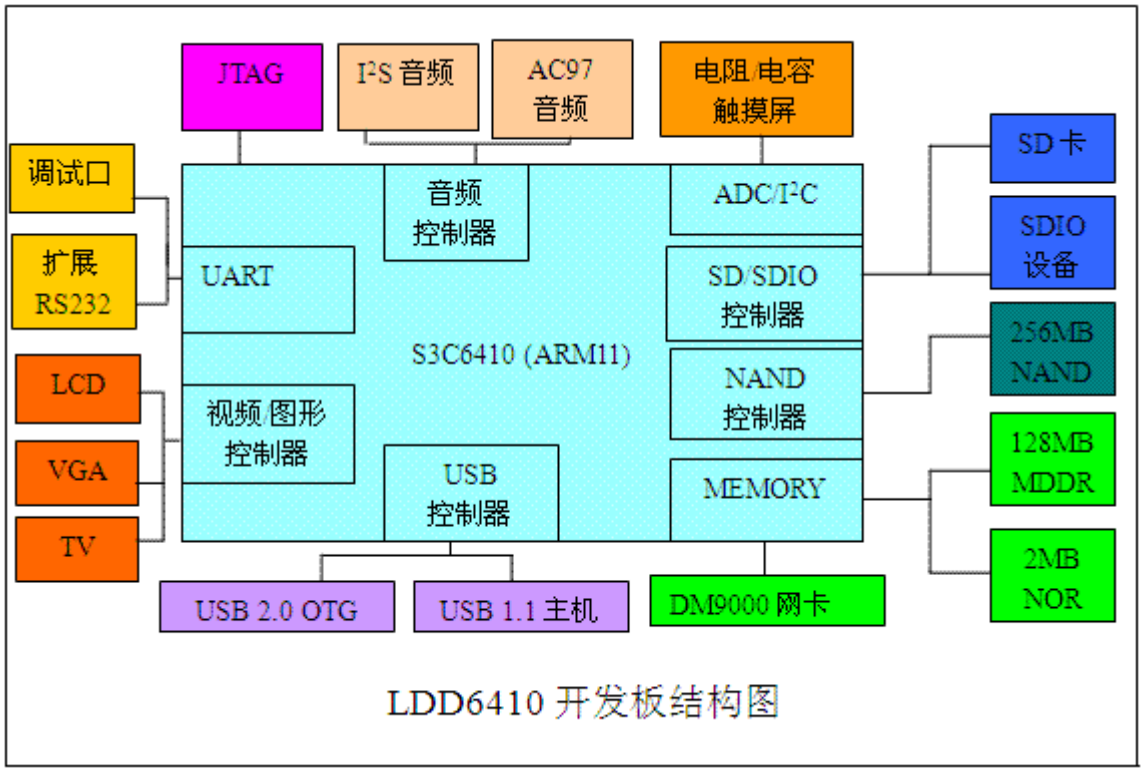
1. LDD6410 硬软件特性

1.1 LDD6410 的电路板组成和结构

LDD6410 是一款高端 ARM11 处理器开发板，采用三星公司最新推出 S3C6410 处理器，芯片拥有强大的内部资源和视频处理能力，板上集成了丰富的外围接口，其主要特点如下：

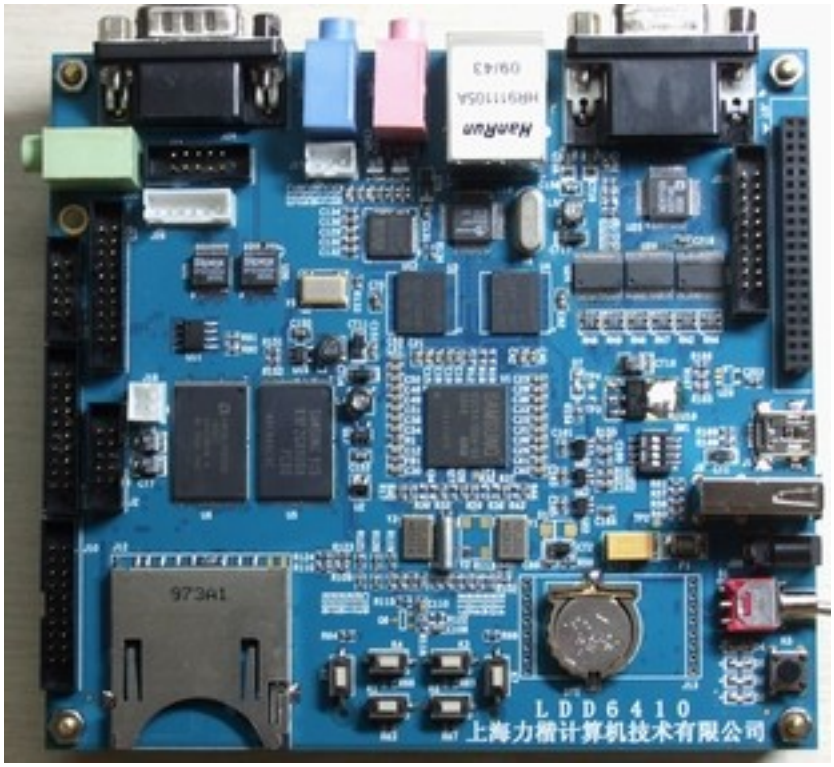
1. 运行于 533MHz 的 ARM11 处理器（最高主频可达到 667MHz）
2. 运行于 266MHz 的 DDR 内存，128MB
3. 1MB NOR FLASH
4. 256MB NAND FLASH
5. WM9714 AC97 声卡
6. VGA 输出接口（可达 1024*768@60Hz）
7. TV 输出接口
8. USB 2.0 OTG 接口及 USB 1.1 host 接口
9. SD/SDIO 接口，支持 SD 卡和 SDIO 设备
9. DM9000 百兆网卡
10. LCD、触摸屏
11. S3C6410 芯片内嵌图形加速，JPEG、多媒体编解码
12. 6 个 GPIO 按键
13. 可扩展 Camera、WiFi、3G modem 等模块
14. 可扩张外部矩阵键盘

LDD6410 电路板的结构如下图：

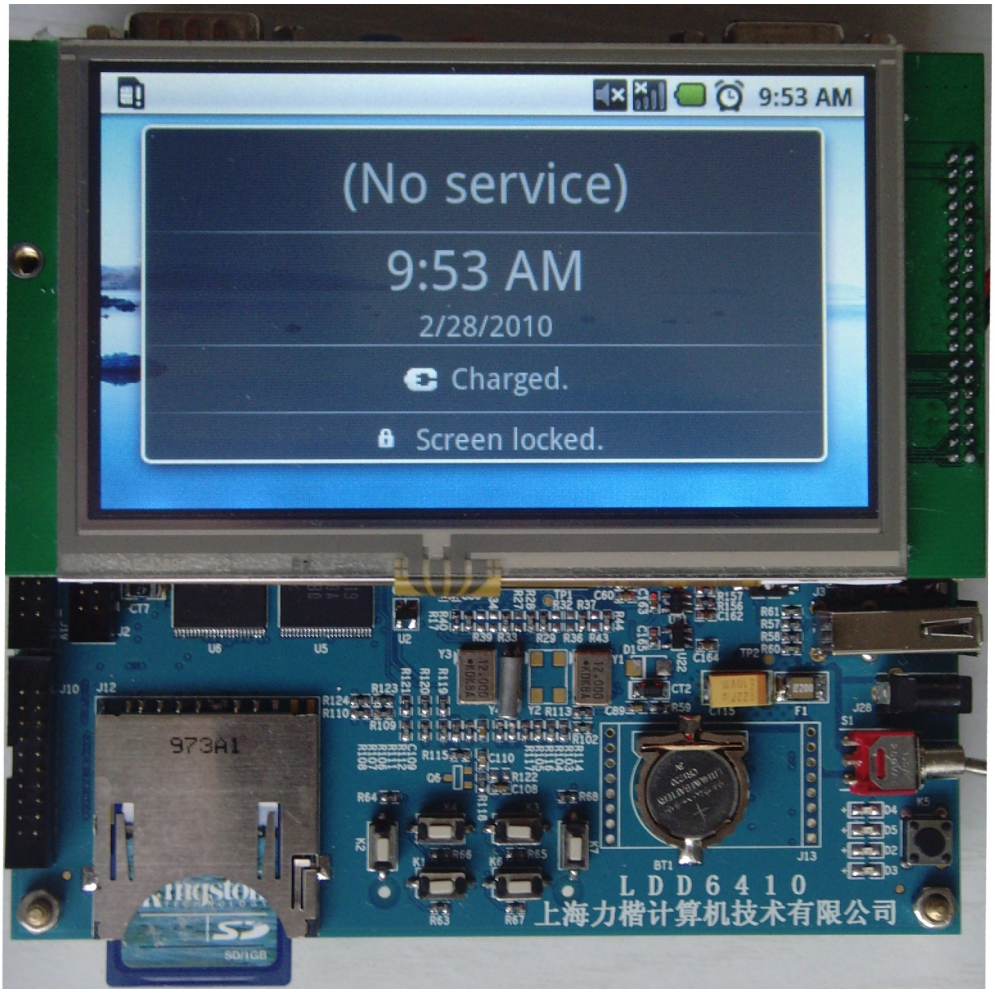


LDD6410 开发板结构图

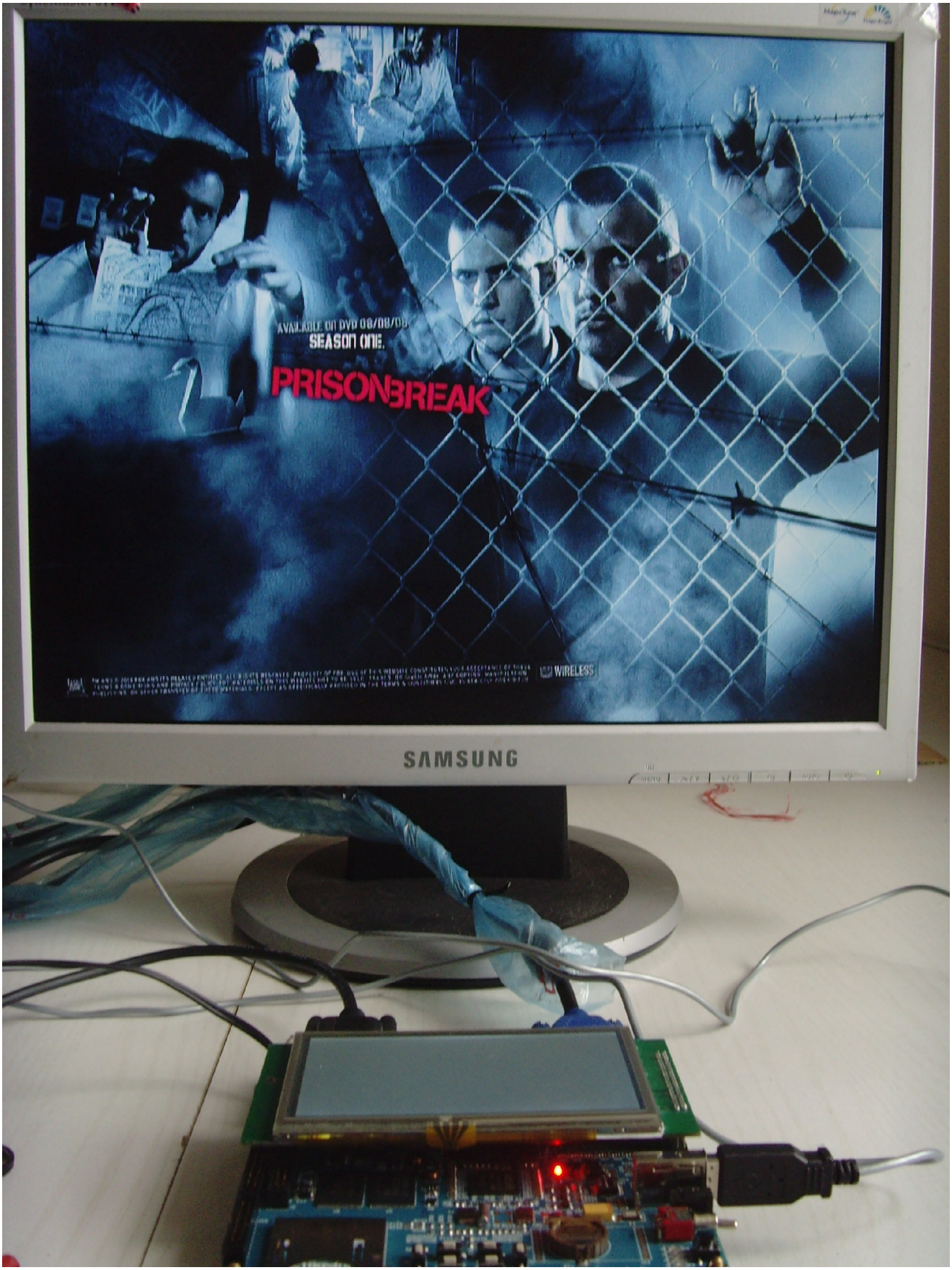
其实物图如下：



LDD6410 的 LCD 输出显示效果如下图：



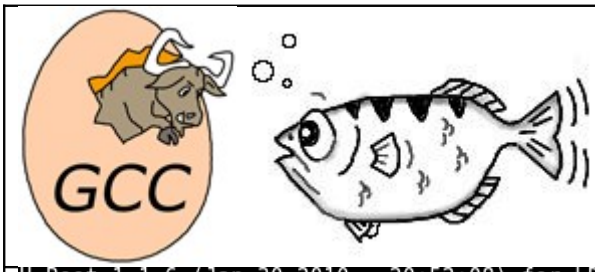
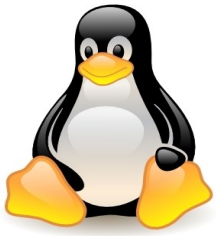
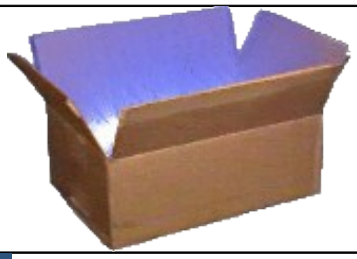
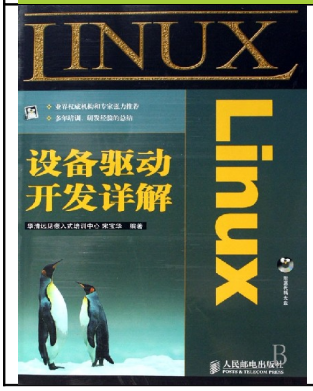
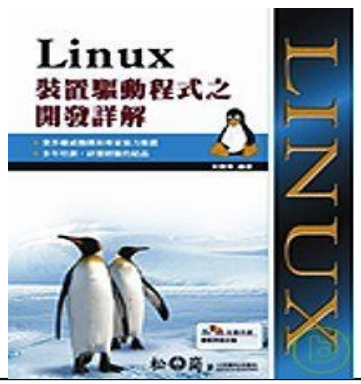
LDD6410 的 VGA 输出显示效果如下图：


SEASON ONE.
PRISONBREAK
WIRELESS
SAMSUNG

1.2 LDD6410 的启动跳线设置

通过电路板上的 SW1 可设置 LDD6410 的启动模式，从 SD 卡启动设备为全 ON，从 NAND 启动时，将 1、2 设置为 ON，3、4 设置为 OFF。

1.3 LDD6410 的软件特性

	armlinuxgcc armlinuxgdb gdbserver strace
	源代码包含独立的 LDD6410 板文件 支持从 SD 卡、NAND 启动 支持 DM9000 网卡引导
	Linux 2.6.28 内核独立 源代码包含独立的 LDD6410 BSP 完整的设备驱动
 	最新版 Busybox 1.15.1 文件系统集成 jpegview, mplayer, appweb 等大量集成 keytest, lcdtest, micetest, tstest 等大量开发案例
	Android 1.6 内核 power management 补丁源代码 内核 android 驱动源代码 Android 1.6 SDK 和文件系统 支持按键、触摸屏、鼠标 支持外接 VGA 显示器
	Qt Embedded 4.5.3 Tslib, ts_calibration
 	配套教材与大量 Linux 设备驱动开发案例 * 2008 年度 chinapub 10 大畅销经典 * 2008 年度 51cto、chinapub、中华读书报 10 大原创精品 * 繁体中文版在宝岛台湾热销

SKYEYE	u-boot、内核、文件系统对 skyeye 模拟器进行完整支持 可模拟 CPU、内存、LCD 等 支持内核、内核模块、应用程序 gdb 调试

2. LDD6410 Linux 开发完全剖析

2.1 建立工具链和开发环境

2.1.1 S3C6410XToolChain4.2.2EABI

S3C6410 处理器最常使用的交叉编译工具链为 S3C6410XToolChain4.2.2EABIV0.0cross4.2.2eabi.tar。

下载地址： http://ldd6410.googlecode.com/files/cross4.2.2eabi.tar.bz2

光盘目录： toolchains/cross4.2.2eabi.tar.bz2

安装步骤:

1. 解压上述工具链获得文件夹：4.2.2eabi/

2. 在/usr/local/下面创建目录 arm/ (注意，最好是放到这个目录，不然在以后的编译过 程中可能出现一些错误)

3. 将目录 4.2.2eabi/移动到/usr/local/arm/下面

4. 设置环境变量:

编辑/etc/profile 文件，在文件末尾添加:

```
PATH="$PATH:/usr/local/arm/4.2.2eabi/usr/bin"
export PATH
```

使环境变量生效，在终端输入命令:

```
source /etc/profile
```

另外，也可以通过修改 home 目录的.bashrc 来将/usr/local/arm/4.2.2eabi/usr/bin 添加到 PATH:

```
export PATH=/usr/local/arm/4.2.2eabi/usr/bin/:$PATH
```

5. 测试环境变量是否设置成功:

在终端输入：echo $PATH，如果输出的路径中包含了/usr/local/arm/4.2.2eabi/usr/bin 则说明环境变量设置成功。

6. 测试交叉编译工具链

在终端输入 armlinuxgcc v 显示如下:

```
Using builtin specs.

Target: armunknownlinuxgnueabi

Configured with: /home/scsuh/workplace/coffee/buildroot20071011/toolchain_build_arm

/gcc4.2.2/configure prefix=/usr build=i386pclinuxgnu host=i386pclinuxgnu
```

```
target=armunknownlinuxgnueabi enablelanguages=c,c++ withsysroot=/usr/local

/arm/4.2.2eabi/ withbuildtimetools=/usr/local/arm/4.2.2eabi//usr/armunknownlinux

gnueabi/bin disable   cxa_atexit enabletargetoptspace withgnuld enableshared

withgmp=/usr/local/arm/4.2.2eabi//gmp withmpfr=/usr/local/arm/4.2.2eabi//mpfr

disablenls enablethreads disablemultilib disablelargefile witharch=armv4t

withfloat=soft enablecxxflags=msoftfloat

Thread model: posix gcc version 4.2.2
```

说明交叉编译工具链已经安装成功。

2.1.2 strace、gdbserver 和 armlinuxgdb

力楷为 S3C6410 整合了 ldd6410debugtools.tar.gz 调试工具包，包含 strace、gdbserver 和 armlinuxgdb，其中 strace、gdbserver 用于目标板文件系统，arm linuxgdb 运行于主机端对目标板上的内核、内核模块应用程序进行调试。

下载地址：http://ldd6410.googlecode.com/files/ldd6410debugtools.tar.gz

光盘目录：toolchains/ ldd6410debugtools.tar.gz

解压 ldd6410debugtools.tar.gz，将其中的 armlinuxgdb 放入主机上 armlinuxgcc 所在的目录：/usr/local/arm/4.2.2eabi/usr/bin/。

而 strace、gdbserver 则可根据需要放入目标机根文件系统的/usr/sbin 目录。

在目标板上运行“strace 应用程序”可以跟踪应用程序所进行的所有系统调用，譬如我们 strace 一下 echo 命令：

```
# strace echo "hello, linux"

execve("/bin/echo", ["echo", "hello, linux"], [/* 7 vars */]) = 0

uname({sys="Linux", node="lihacker", ...}) = 0 brk(0)     = 0x145000 brk(0x145d02)
   = 0x145d02 set_tls(0x1454a0, 0x13f4c4, 0, 0x1, 0x1454a0) = 0 brk(0x166d02)                    =
0x166d02 brk(0x167000)                          = 0x167000 getuid32()          = 0
fcntl64(1, F_GETFL)       = 0x2 (flags O_RDWR)

fstat64(1, {st_mode=S_IFCHR|0644, st_rdev=makedev(204, 64), ...}) = 0

ioctl(1, SNDCTL_TMR_TIMEBASE or TCGETS, {B115200 opost isig icanon echo ...}) = 0

mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, 1,
0) = 0x40000000

write(1, "hello, linux\n"..., 13hello, linux)= 13

exit_group(0)      = ?
```

2.1.3 gdb 调试器的用法

GDB 是 GNU 开源组织发布的一个强大的 UNIX 下的程序调试工具，GDB 主要可帮助工程师完成下面 4 个方面的功能：

* 启动程序，可以按照工程师自定义的要求随心所欲的运行程序。
* 让被调试的程序在工程师指定的断点处停住，断点可以是条件表达式。
* 当程序被停住时，可以检查此时程序中所发生的事，并追索上文。
* 动态地改变程序的执行环境。

为了调试 LDD6410 上的应用程序，可以在目标板上开启 gdbserver，在主机端使用 armlinuxgdb 对其进行调试。下面我们编写一个简单的应用程序并演示如何通过 gdbserver 和 armlinuxgdb 调试之。

程序源代码如下：

```
/*
* gdb_example.c: program to show how to use armlinuxgdb
* Copyright 2009 LiHacker Computer Technology Inc.
*/

void increase_one(int *data)
{
*data = *data + 1;
}

int main(int argc, char *argv[])
{
int dat = 0;
int *p = 0;

increase_one(&dat);

/* program will crash here */
increase_one(p);
return 0;
}
```

通过 debug 方式编译它：

```
armlinuxgcc g o gdb_example gdb_example.c
```

将程序下载到目标板后，在目标板上运行：

```
# gdbserver 192.168.1.20:1234 gdb_example
Process gdb_example created; pid = 1096
Listening on port 1234
```

其中 192.168.1.20 为目标板的 IP，1234 为 gdbserver 的侦听端口。

在主机上运行：

```
lihacker@lihackerlaptop:~/ldd6410/tests/gdbexample$ armlinuxgdb gdb_example
GNU gdb 6.6
Copyright (C) 2006 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are welcome to change it
and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB. Type "show warranty" for details.
This GDB was configured as "host=/usr/local/arm/4.2.2eabi/usr/bin/ target=armlinux"... (gdb)
```

主机的 armlinuxgdb 中运行如下命令连接目标板：

```
(gdb) target remote 192.168.1.20:1234
Remote debugging using 192.168.1.20:1234
...
0x400007b0 in ?? ()
```

运行如下命令将断点设置在 increase_one(&dat);这一行：

```
(gdb) b gdb_example.c:16
Breakpoint 1 at 0x8390: file gdb_example.c, line 16.
```

通过“c”命令继续运行目标板上的程序，发生断点：

```
(gdb) c
Continuing.
...
Breakpoint 1, main (argc=1, argv=0xbead4eb4) at gdb_example.c:16
16 increase_one(&dat);
```

运行“n”命令执行完 increase_one(&dat);

再查看 dat 的值：

```
(gdb) n
19 increase_one(p); (gdb) p dat
$1 = 1
```

发现 dat 变成 1。继续运行“c”命令，由于即将访问空指针，gdb_example 将崩溃：

```
(gdb) c
Continuing.
Program received signal SIGSEGV, Segmentation fault.
0x0000834c in increase_one (data=0x0) at gdb_example.c:8
8   *data = *data + 1;
```

我们通过“bt”命令可以拿到 backtrace：

```
(gdb) bt
#0  0x0000834c in increase_one (data=0x0) at gdb_example.c:8
#1  0x000083a4 in main (argc=1, argv=0xbead4eb4) at gdb_example.c:19
```

通过“info reg”命令可以查看当时的寄存器值：

```
(gdb) info reg
r0  0x0 0
r1  0xbead4eb4      3199028916
r2  0x1 1
r3  0x0 0
r4  0x4001e5e0      1073866208
r5  0x0 0
r6  0x826c 33388
r7  0x0 0
r8  0x0 0
r9  0x0 0
r10 0x40025000      1073893376
r11 0xbead4d44      3199028548
r12 0xbead4d48      3199028552
sp  0xbead4d30      0xbead4d30
lr  0x83a4 33700
pc  0x834c 0x834c <increase_one+24>
fps 0x0 0
cpsr 0x60000010     1610612752
```

2.1.4 主机端 tftp 和 nfs 服务安装

LDD6410 可使用 tftp 或 nfs 文件系统与主机通过网口交互。对于 Ubuntu 或 Debian 用 户而言，在主机端可通过如下方法安装 tftp 服务：

在主机端执行：

```
sudo aptget install tftpdhpa sudo mkdir /home/tftp
sudo chmod 777 /home/tftp
```

运行“sudo vim /etc/default/tftpdhpa”或“sudo gedit /etc/default/tftpdhpa”修改文件内容为：

RUN_DAEMON="yes OPTIONS="l c s /home/tftp"

开启 tftp 服务：

```
sudo /etc/init.d/tftpdhpa start
Starting HPA's tftpd: in.tftpd.
```

对于 Ubuntu 或 Debian 用户而言，在主机端可通过如下方法安装 nfs 服务： 在主机端执行：

```
aptget install nfskernelserver sudo mkdir /home/nfs
sudo chmod 777 /home/nfs
```

运行“sudo vim /etc/exports”或“sudo gedit /etc/exports”修改该文件内容为：

/home/nfs *(sync,rw)

运行 exportfs rv 开启 NFS 服务：

```
/etc/init.d/nfskernelserver restart
```

2.2 UBOOT

2.2.1 U-BOOT 的移植

力楷为 LDD6410 开发板建立了完整独立的 UBOOT 板级支持，而不是去破坏公共代码或者对 SMDK6410 进行修改。这些工作包括：

(1)新的 s3c-u-boot-1.1.6\board\samsung\ldd6410 板文件目录及其下的 ldd6410.c、flash.c、lowlevel_init.S、u-boot.lds 等文件；

(2)新的 s3c-u-boot-1.1.6\include\configs\ldd6410.h 板配置文件；

(3)s3c-u-boot-1.1.6\Makefile 中针对 LDD6410 的配置选项：

```
ldd6410_config    :       unconfig
    @$(MKCONFIG) $(@:_config=) arm s3c64xx ldd6410 samsung s3c6410
```

(4)针对 LDD6410 的宏定义。

编译 LDD6410 U-BOOT 的方法是首先运行 make ldd6410_config，在运行 make。

LDD6410 的 UBOOT 可存放于 SD 卡或 NAND Flash 中，支持从 SD 卡、NAND 或网络 tftp 服务器引导 Linux 内核映像。

编译 NAND 版本的 U-BOOT 时需要修改 include\configs\ldd6410.h，定义 CONFIG_BOOT_NAND 而不定义 CONFIG_BOOT_MOVINAND：

```
#define CONFIG_BOOT_NAND
/* #define CONFIG_BOOT_MOVINAND */
```

反之对于 SD 卡版 U-BOOT 则需要定义 define CONFIG_BOOT_MOVINAND 而不定义 CONFIG_BOOT_NAND：

```
/* #define CONFIG_BOOT_NAND */
#define CONFIG_BOOT_MOVINAND
```

在编译出 SD 卡版本的 U-BOOT 后，需要用 mkmovi 这个脚本对编译得到的 U-BOOT 映像进行处理，这个脚本的内容为：

```
#!/bin/bash

#
# This script will create a u-boot binary for movinand/mmc boot
#

# padding to 256k u-boot
cat u-boot.bin >> u-boot-2x.bin
cat u-boot.bin >> u-boot-2x.bin
split -d -a 1 -b 256k u-boot-2x.bin u-boot-256k.bin

# make BL1 u-boot (8kb)
split -d -a 2 -b 8k u-boot.bin u-boot-8k.bin

# concat the BL1 behind of padded 256k binary
cat u-boot-8k.bin00 >> u-boot-256k.bin0

# rename and chmod
mv u-boot-256k.bin0 u-boot-movi.bin
chmod 777 u-boot-movi.bin

# remove temp files
rm -f u-boot-8k*
rm -f u-boot-256k*
rm -f u-boot-2x.bin
```

2.2.2 NAND 版 UBOOT

NAND 板 UBOOT 的启动信息为：

```
UBoot 1.1.6 (Jan  2 2010  09:43:17) for LDD6410

**************************************
**      LDDS3C6410 Nand boot v0.1    **
**   ShangHai Lihacker Computer Tec    **
**       http://www.lihacker.com       **
**************************************
CPU:       S3C6410@532MHz
Fclk = 532MHz, Hclk = 133MHz, Pclk = 66MHz, Serial = CLKUART (SYNC Mode) Board:   LDD6410
DRAM:     128 MB
Flash:   2048 kB NAND:    256 MB In:      serial
Out:       serial
Err:       serial
Hit any key to stop autoboot:  0
```

我们在 U-BOOT 中通过 printenv 命令可以查看 NAND 情况下默认的 UBOOT 环境变量设置：

```
LDD6410 # printenv
bootargs=root=/dev/mtdblock2 rootfstype=yaffs2 console=ttySAC0,115200 bootcmd=nand read 0xc0008000
0x00080000 0x00800000;bootm 0xc0008000 bootdelay=3
baudrate=115200 ethaddr=00:40:5c:26:0a:5b ipaddr=192.168.1.20 serverip=192.168.1.111
gatewayip=192.168.1.1 netmask=255.255.255.0 stdin=serial
stdout=serial
stderr=serial
Environment size: 320/16380 bytes
```

2.2.3 SD 卡版 UBOOT

```
UBoot 1.1.6 (Jan  2 2010  09:43:17) for LDD6410

**************************************
**      LDDS3C6410 SD boot v0.1     **
**   ShangHai Lihacker Computer Tec     **
**      http://www.lihacker.com      **
**************************************

CPU:       S3C6410@532MHz
Fclk = 532MHz, Hclk = 133MHz, Pclk = 66MHz, Serial = CLKUART (SYNC Mode) Board:   LDD6410
DRAM:     128 MB
Flash:   2048 kB NAND:    256 MB In:      serial
Out:       serial
Err:       serial
Hit any key to stop autoboot:  0
```

我们在 U-BOOT 中通过 printenv 命令可以查看 SD 情况下默认的 UBOOT 环境变量设置：

```
LDD6410 # printenv
bootargs=root=/dev/mmcblk0p2 rootfstype=ext3 rootwait console=ttySAC0,115200
bootcmd=movi read 32 0x00800000 0xc0008000;bootm 0xc0008000
bootdelay=3 baudrate=115200 ethaddr=00:40:5c:26:0a:5b ipaddr=192.168.1.20 serverip=192.168.1.111
gatewayip=192.168.1.1 netmask=255.255.255.0 stdin=serial
stdout=serial
stderr=serial
Environment size: 333/16380 bytes
```

我们可以看到 NAND 和 SD 版 UBOOT 的 BOOTCMD 命令分别为“nand read 0xc0008000 0x00080000 0x00800000;bootm 0xc0008000”和“movi read 32 0x00800000 0xc0008000;bootm 0xc0008000”，前者通过“nand read”指令从 NAND 读取 Linux 内核，后者通过“movi read”指令从 SD 卡读取内核。

2.2.4 从 tftp 服务器引导 Linux

LDD6410 开发板 SD 卡和 NAND 版的 UBOOT 都直接支持 DM9000，UBOOT 中可以直接使用 DM9000，如运行 ping 命令：

```
LDD6410 # ping 192.168.1.111
dm9000 i/o: 0x18000300, id: 0x90000a46
MAC: 00:40:5c:26:0a:5b
operating at 100M full duplex mode host 192.168.1.111 is alive
LDD6410 #
```

可从主机的 tftp 服务上直接下载 Linux 内核，其操作步骤如下：

1. 下载 zImage：

```
LDD6410 # tftp 0xc0008000 zImage dm9000 i/o: 0x18000300, id: 0x90000a46
MAC: 00:40:5c:26:0a:5b
operating at 100M full duplex mode
TFTP from server 192.168.1.111; our IP address is 192.168.1.20
Filename 'zImage'.
Load address: 0xc0008000
Loading: ###################################################################
#################################################################
#################################################################
#################################################################
#################################################################
#################################################################
#############################################################
```

```
done
Bytes transferred = 2279916 (22c9ec hex)
```

2.启动内核：

```
LDD6410 # bootm 0xc0008000
Boot with zImage
Starting kernel ...
```

2.3 Linux 内核和驱 动

2.3.1 LDD6410 内核和 BSP

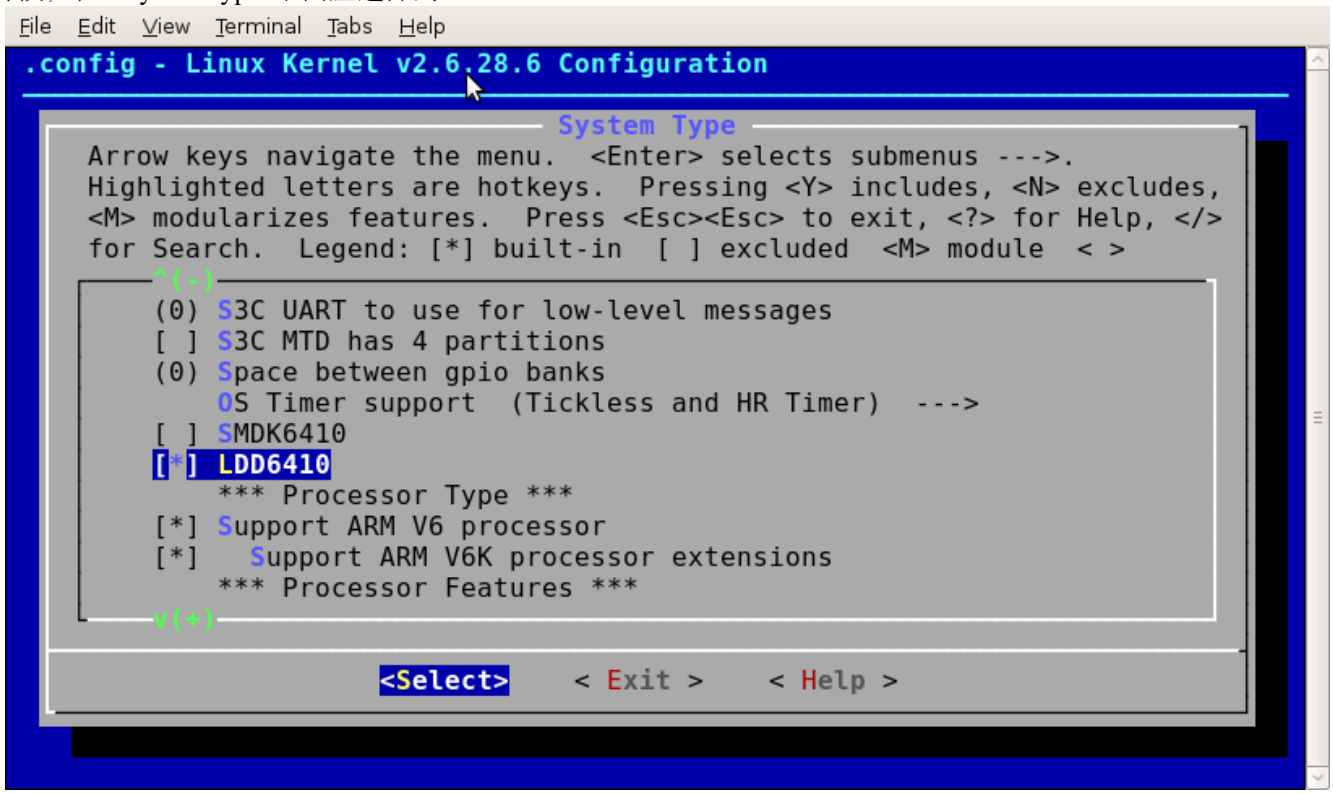
力核计算机技术有限公司为 LDD6410 开发板整合了唯一的内核版本，该内核版本直接为板上所有外设和 S3C6410 内部各个控制器提供了驱动支持，内核启动过程如下：

```
Starting kernel ...
Uncompressing
Linux.....
done, booting the kernel.
Linux version 2.6.28.6svn2dirty (root@lihackerlaptop) (gcc version 4.2.2) #41 Fri Jan 1 21:04:48
CST 2010
CPU: ARMv6compatible processor [410fb766] revision 6 (ARMv7), cr=00c5387f
CPU: VIPT nonaliasing data cache, VIPT nonaliasing instruction cache
Machine: SMDK6410
Memory policy: ECC disabled, Data cache writeback
CPU S3C6410 (id 0x36410101)
S3C24XX Clocks, (c) 2004 Simtec Electronics
S3C64XX: PLL settings, A=532000000, M=532000000, E=24000000
S3C64XX: HCLKx2=266000000, HCLK=133000000, PCLK=66500000
mout_apll: source is fout_apll (1), rate is 532000000 mout_epll: source is fout_epll (1), rate is 24000000
mout_mpll: source is mpll (1), rate is 532000000 mmc_bus: source is dout_mpll (1), rate is 266000000
mmc_bus: source is dout_mpll (1), rate is 266000000 mmc_bus: source is dout_mpll (1), rate is 266000000
usbhostbus: source is mout_epll (0), rate is 24000000 uclk1: source is dout_mpll (1), rate is 66500000
spi_epll: source is mout_epll (0), rate is 24000000
spi_epll: source is mout_epll (0), rate is 24000000 sclk_audio0: source is mout_epll (0), rate is 24000000
sclk_audio1: source is mout_epll (0), rate is 24000000 sclk_audio2: source is mout_epll (0), rate is 24000000
irdabus: source is mout_epll (0), rate is 24000000
Built 1 zonelists in Zone order, mobility grouping on.  Total pages: 32512
Kernel command line: set bootargs root=/dev/mtdblock2 rootfstype=yaffs2
console=ttySAC0,115200
PID hash table entries: 512 (order: 9, 2048 bytes) Console: colour dummy device 80x30
console [ttySAC0] enabled
Dentry cache hash table entries: 16384 (order: 4, 65536 bytes) Inodecache hash table entries: 8192 (order:
3, 32768 bytes) Memory: 128MB = 128MB total
Memory: 124956KB available (3796K code, 461K data, 524K init)
SLUB: Genslabs=12, HWalign=32, Order=03, MinObjects=0, CPUs=1, Nodes=1
Calibrating delay loop... 530.84 BogoMIPS (lpj=1327104) Mountcache hash table entries: 512
CPU: Testing write buffer coherency: ok net_namespace: 316 bytes
NET: Registered protocol family 16
S3C6410: Initialising architecture
S3C DMApl080 Controller Driver, (c) 20062007 Samsung Electronics
Total 32 DMA channels will be initialized. SCSI subsystem initialized
usbcore: registered new interface driver usbfs usbcore: registered new interface driver hub usbcore:
registered new device driver usb
NET: Registered protocol family 2
IP route cache hash table entries: 1024 (order: 0, 4096 bytes)
TCP established hash table entries: 4096 (order: 3, 32768 bytes) TCP bind hash table entries: 4096 (order:
4, 81920 bytes)
TCP: Hash tables configured (established 4096 bind 4096)
TCP reno registered
NET: Registered protocol family 1
NetWinder Floating Point Emulator V0.97 (double precision) JFFS2 version 2.2. (NAND) © 20012006 Red
Hat, Inc.
yaffs Dec 12 2009 12:30:56 Installing. alg: No test for stdrng (krng)
io scheduler noop registered
io scheduler anticipatory registered io scheduler deadline registered
io scheduler cfq registered (default)
S3C_LCD clock got enabled :: 133.000 Mhz
```

```
VGA Display will be initialized
Window[0] FB1: map_video_memory: clear ff000000:00300000
FB1: map_video_memory: dma=57400000 cpu=ff000000 size=00300000
Window[0] FB2: map_video_memory: clear ff180000:00180000
FB2: map_video_memory: dma=57580000 cpu=ff180000 size=00180000
Console: switching to colour frame buffer device 128x48
fb0: s3cfb frame buffer device
Serial: 8250/16550 driver4 ports, IRQ sharing disabled
s3c6400uart.0: s3c2410_serial0 at MMIO 0x7f005000 (irq = 16) is a S3C6400/10 s3c6400uart.1:
s3c2410_serial1 at MMIO 0x7f005400 (irq = 20) is a S3C6400/10 s3c6400uart.2: s3c2410_serial2 at MMIO
0x7f005800 (irq = 24) is a S3C6400/10 s3c6400uart.3: s3c2410_serial3 at MMIO 0x7f005c00 (irq = 28) is a
S3C6400/10 brd: module loaded
loop: module loaded
PPP generic driver version 2.4.2
dm9000 Ethernet Driver, V1.31
eth0: dm9000a at c887a000,c887e004 IRQ 108 MAC: 00:16:d4:9f:ed:a4 (platform data) Linux video
capture interface: v2.00
Driver 'sd' needs updating  please use bus_type methods
SCSI Media Changer driver v0.25
Driver 'ch' needs updating  please use bus_type methods
S3C NAND Driver, (c) 2008 Samsung Electronics
S3C NAND Driver is using hardware ECC.
NAND device: Manufacturer ID: 0xec, Chip ID: 0xda (Samsung NAND 256MiB 3,3V 8bit) Creating 3
MTD partitions on "NAND 256MiB 3,3V 8bit":
0x000000000x00080000 : "Bootloader"
0x000800000x00400000 : "Kernel"
0x004000000x10000000 : "File System"
usbmon: debugfs is not available
ohci_hcd: USB 1.1 'Open' Host Controller (OHCI) Driver s3c2410ohci s3c2410ohci: S3C24XX OHCI
s3c2410ohci s3c2410ohci: new USB bus registered, assigned bus number 1
s3c2410ohci s3c2410ohci: irq 79, io mem 0x74300000
usb usb1: configuration #1 chosen from 1 choice hub 10:1.0: USB hub found
hub 10:1.0: 2 ports detected
Initializing USB Mass Storage driver...
usbcore: registered new interface driver usbstorage
USB Mass Storage support registered.
s3cudc : S3C HS USB OTG Device Driver, (c) 20082009 Samsung Electronics s3cudc : version 15 March
2009 (DMA Mode)
mice: PS/2 mouse device common for all mice input: gpiokeys as /class/input/input0
S3C Touchscreen driver, (c) 2008 Samsung Electronics
S3C TouchScreen got loaded successfully : 12 bits input: S3C TouchScreen as /class/input/input1
S3C24XX RTC, (c) 2004,2006 Simtec Electronics
s3c2410_rtc: tick irq 34, alarm irq 92
s3c2410rtc s3c2410rtc: rtc disabled, reenabling s3c2410rtc s3c2410rtc: rtc core: registered s3c as rtc0
i2c /dev entries driver
s3c2440i2c s3c2440i2c.0: i2c0: S3C I2C adapter s3c2440i2c s3c2440i2c.1: i2c1: S3C I2C adapter at24 0-
0050: 1024 byte 24c08 EEPROM (writable) at24 10057: 16384 byte 24c128 EEPROM (writable)
S3C2410 Watchdog Timer, (c) 2004 Simtec Electronics
s3c2410wdt s3c2410wdt: watchdog inactive, reset disabled, irq enabled sdhci: Secure Digital Host
Controller Interface driver
sdhci: Copyright(c) Pierre Ossman
s3csdhci s3csdhci.0: clock source 0: hsmmc (133000000 Hz) s3csdhci s3csdhci.0: clock source 1: hsmmc
(133000000 Hz) s3csdhci s3csdhci.0: clock source 2: mmc_bus (53200000 Hz)
mmc0: SDHCI controller on samsunghsmmc [s3csdhci.0] using ADMA
usbcore: registered new interface driver usbhid usbhid: v2.6:USB HID core driver
Advanced Linux Sound Architecture Driver Version 1.0.18rc3. ASoC version 0.13.2
smdk6400_init:67
WM9713/WM9714 SoC Audio Codec 0.15
asoc: AC97 HiFi <> s3c64xxac97 mapping ok
usb 11: new low speed USB device using s3c2410ohci and address 2
ALSA device list:
#0: SMDK6410 (WM9713) TCP cubic registered
RPC: Registered udp transport module.
RPC: Registered tcp transport module.
VFP support v0.3: implementor 41 architecture 1 part 20 variant b rev 5
s3c2410rtc s3c2410rtc: setting system clock to 20000225 20:02:30 UTC (951508950)
yaffs: dev is 32505858 name is "mtdblock2"
yaffs: passed flags ""
```

```
    yaffs: Attempting MTD mount on 31.2, "mtdblock2"
    s3cnand: ECC uncorrectable error detected yaffs: restored from checkpoint yaffs_read_super:
isCheckpointed 1
    VFS: Mounted root (yaffs2 filesystem).
    Freeing init memory: 524K
    mmc0: new high speed SD card at address 1d1d mmcblk0: mmc0:1d1d SD02G 1.83 GiB mmcblk0: p1 p2
    usb 11: configuration #1 chosen from 1 choice input: USB Mouse as /class/input/input2
    genericusb 0003:15D9:0A33.0001: input: USB HID v1.10 Mouse [USB Mouse] on usbs3c24xx1/
    input0
    eth0: link down
    Welcome to
    ARMLinux for Lihacker LDD6410
    For further information please check: http://www.lihacker.com/ http://www.linuxdriver.cn/
    For technical support, please subscribe mail list linuxdriver@googlegroups.com and post there. The url is
http://groups.google.com/group/linuxdriver.
    g_file_storage gadget: Filebacked Storage Gadget, version: 7 August 2007
    g_file_storage gadget: Number of LUNs=1 g_file_storage gadgetlun0: ro=0, file: /demo/vfat.img
Registered gadget driver 'g_file_storage'
    # eth0: link up, 100Mbps, fullduplex, lpa 0x45E1
    #
```

在 BSP 方面，力核建立了独立的 LDD6410 板文件，位于内核源代码的 arch/arm/machs3c6410/mach-ldd6410.c，该板文件对 LDD6410 开发板板上设备、资源进行了整合。因此，在为 LDD6410 开发板编译内核的时候，在“system type”下面应选择的“LDD6410”。



这里要强调一点的是，很多人在进行 Linux 移植时候，喜欢直接修改参考电路板的板文件，比如 arch/arm/machs3c6410/machsmdk6410.c，这种做法是错误的。其原因在于，板文件就是对自身电路板设备和资源的描述，自身设计的电路板与 smdk 板并不相同，直接修改 smdk 既破坏了对方的 BSP，也没有体现自己的特点。

2.3.2 按键驱动

Linux 内核下的 drivers/input/keyboard/gpio_keys.c 实现了一个体系结构无关的 GPIO 按键驱动，使用此按键驱动，开发者不需要修改一行代码，只需要在 BSP 的板文件（对于 LDD6410 为 arch/arm/machs3c6410/mach-ldd6410.c）中定义相关的 platform 设备和数据。在 LDD6410 开发板上，用 GPN0~GPN5 实现了 DOWN、ENTER、HOME、POWER、TAB、MENU 六个按键，因此其对应的 platform 信息如下：

```
    static struct gpio_keys_button ldd6410_buttons[] = {
        {
                .gpio           = S3C64XX_GPN(0),
                .code           = KEY_DOWN,
                .desc           = "Down",
```

```
            .active_low     = 1,
    },
    {
            .gpio           = S3C64XX_GPN(1),
            .code           = KEY_ENTER,
            .desc           = "Enter",
            .active_low     = 1,
            .wakeup     = 1,
    },
    {
            .gpio           = S3C64XX_GPN(2),
            .code           = KEY_HOME,
            .desc           = "Home",
            .active_low     = 1,
    },
    {
            .gpio           = S3C64XX_GPN(3),
            .code           = KEY_POWER,
            .desc           = "Power",
            .active_low     = 1,
            .wakeup     = 1,
    },
    {
            .gpio           = S3C64XX_GPN(4),
            .code           = KEY_TAB,
            .desc           = "Tab",
            .active_low     = 1,
    },
    {
            .gpio           = S3C64XX_GPN(5),
            .code           = KEY_MENU,
            .desc           = "Menu",
            .active_low     = 1,
    },
};

static struct gpio_keys_platform_data ldd6410_button_data = {
    .buttons        = ldd6410_buttons,
    .nbuttons       = ARRAY_SIZE(ldd6410_buttons),
};

static struct platform_device ldd6410_device_button = {
    .name           = "gpio-keys",
    .id             = -1,
    .dev            = {
            .platform_data = &ldd6410_button_data,
    }
};
```

并将“&ldd6410_device_button,”语句填入 struct platform_device *ldd6410_devices[]数组，作为该数组的一个成员。

编译内核时选择：

```
Device Drivers  >
Input device support  >
[*]   Keyboards  >
   <*>   GPIO Buttons
```

如果要修改按键对应的 GPIO 和键值，只需要简单的修改 ldd6410_buttons[]数组中的内容。

2.3.3 LED 驱动

Linux 内核下的 drivers/leds/ledsgpio.c 实现了一个体系结构无关的 GPIO LED 驱动， 使用此 LED 驱动，开发者不需要修改一行代码，只需要在 BSP 的板文件（对于 LDD6410 为 arch/arm/machs3c6410/machldd6410.c）中定义相关的 platform 设备和数据。在 LDD6410 开发板上，用 GPM0~GPM3 实现了四个 LED，因此其对应的 platform 信息如下：

```
static struct gpio_led ldd6410_leds[] = {
[0] = {
```

```
        .name = "LED1",
        .gpio = S3C64XX_GPM(0),
    },
    [1] = {
        .name = "LED2",
        .gpio = S3C64XX_GPM(1),
    },
    [2] = {
        .name = "LED3",
        .gpio = S3C64XX_GPM(2),
    },
    [3] = {
        .name = "LED4",
        .gpio = S3C64XX_GPM(3),
    },
    };

    static struct gpio_led_platform_data ldd6410_gpio_led_pdata = {
        .num_leds = ARRAY_SIZE(ldd6410_leds),
        .leds = ldd6410_leds,
    };

    static struct platform_device ldd6410_device_led = {
        .name  = "ledsgpio",
        .id    = 1,
        .dev   = {
        .platform_data = &ldd6410_gpio_led_pdata,
        },
    };
```

并将“&ldd6410_device_led,”语句填入 struct platform_device *ldd6410_devices[]数组，作为该数组的一个成员。

编译内核时选择：

```
Device Drivers  >
    [*] LED Support  >
            <*>  LED Class Support
                <*>  LED Support for GPIO connected LEDs
```

内核启动时会打印：

```
Registered led device: LED1
Registered led device: LED2
Registered led device: LED3
Registered led device: LED4
```

LED1、LED2、LED3、LED4 分别对应板子右下角的 D2、D3、D4、D5。通过如下命令可以开亮 D2：

```
# echo 1 > /sys/class/leds/LED1/brightness
```

通过如下命令可以熄灭 D2：

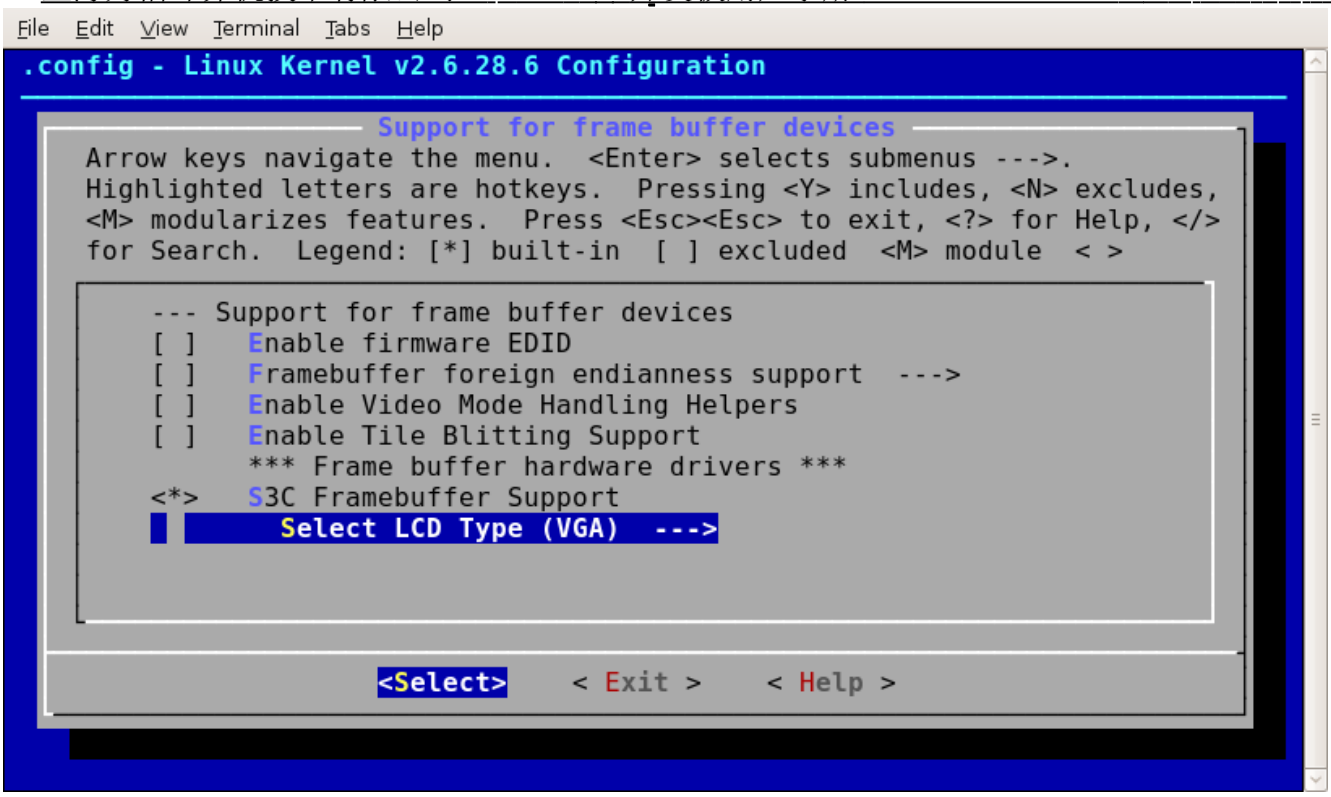
```
# echo 0 > /sys/class/leds/LED1/brightness
```

与按键类似，如果要修改 LED 对应的 GPIO 和极性，只需要简单的修改 ldd6410_buttons[]数组中的内容即可。

2.3.4 LCD 和 VGA 驱动

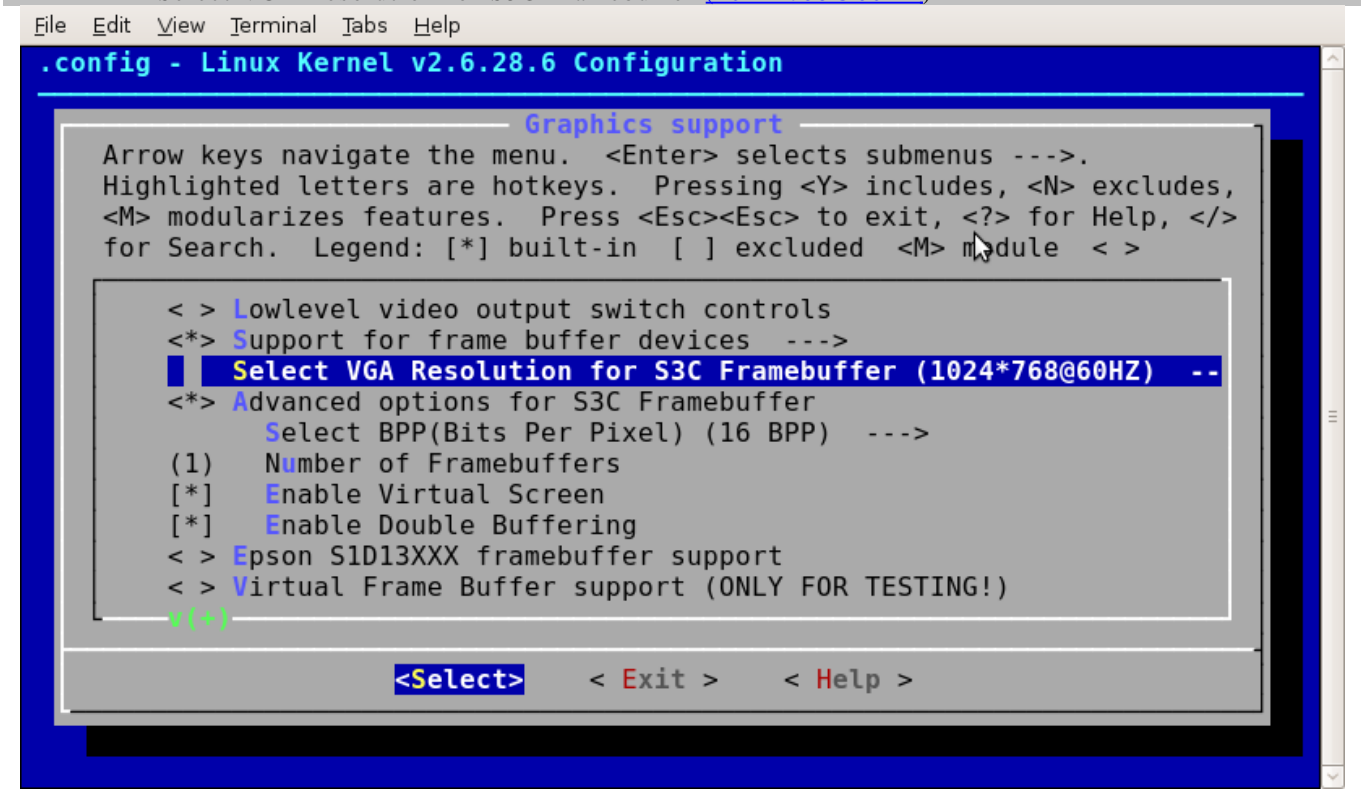
LDD6410 支持 LCD 和 VGA 两种输出方式，默认的内核支持 LCD。为编译支持 VGA 的内核，运行 make menuconfig，进入如下菜单项，并选择“VGA”。

```
Device Drivers  >
    Graphics support  >
        Support for frame buffer devices  >
            S3C Framebuffer Support
```

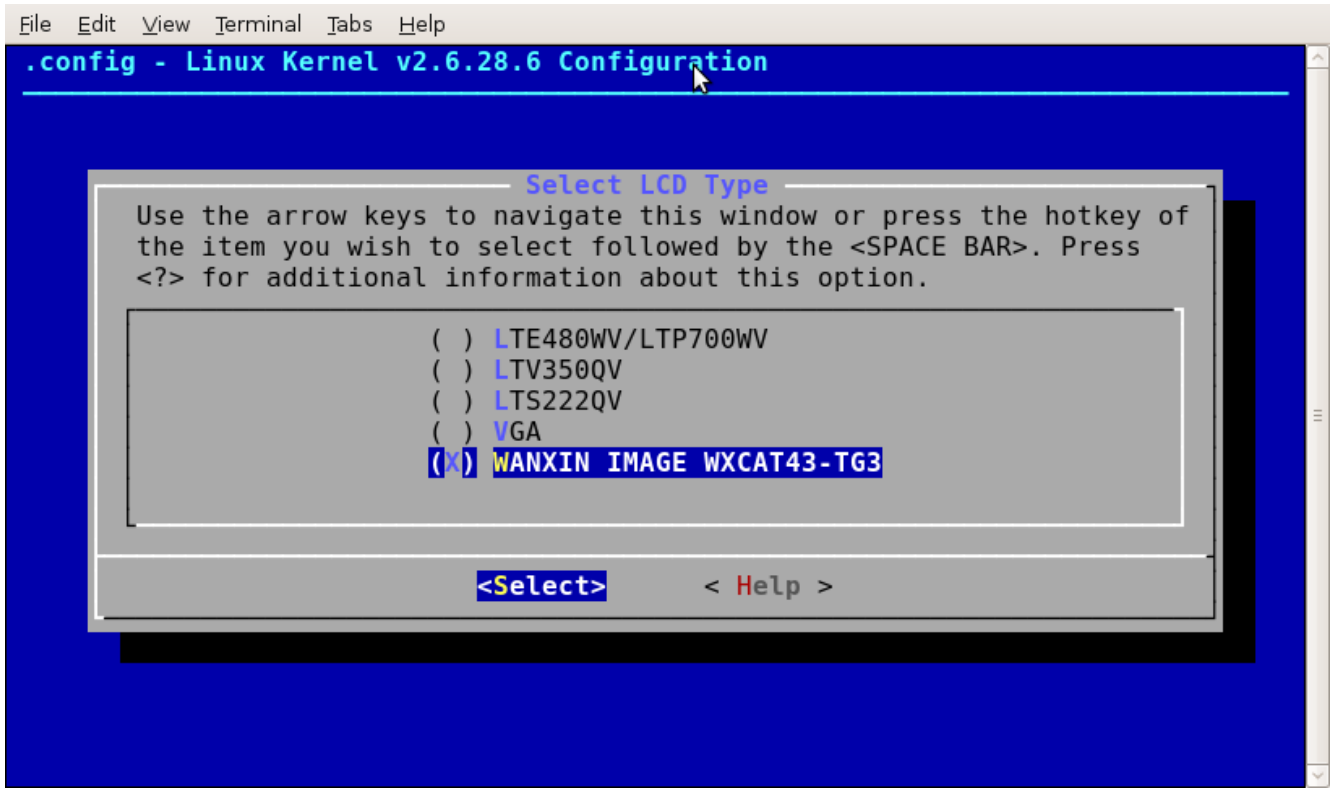


也可运行 make menuconfig 设置 VGA 的分辨率，若要设置为 1024*768@60HZ，进入如下菜单项，并选择“1024*768@60HZ”。

```
Device Drivers >
    Graphics support >
        Select VGA Resolution for S3C Framebuffer (1024*768@60HZ)
```



LCD 方面，力楷开发了独立的集成 LCD 和触摸屏的电路板，采用 WANXIN IMAGE 的 WXCAT43TG3，其分辨率为 480*272。其配置选项如图：



Linux 内核下的 drivers/video/samsung/实现了 S3C6410 内部 framebuffer 控制器的绝大多数工作，对于新的 LCD 移植，主体工作是根据 LCD 数据手册获取时序参数，并填充入驱动。譬如我们从 WANXIN IMAGE 的 WXCAT43TG3 数据手册可获得如下表:

File Edit View Go Help

7.4.2 Timing Requirement 1

Parameter	Symbol	Min.	Typ.	Max.	Unit
Clock cycle	f_{CLK}	-	9	15	MHz
Hsync cycle	1/th	-	17.14	-	KHz
Vsync cycle	1/tv	-	59.94	-	Hz
Horizontal Signal					
Horizontal cycle	th(1)	-	525	-	CLK
Horizontal display period	thd	-	480	-	CLK
Horizontal front porch	thf	2	-	-	CLK
Horizontal pulse width	thp	2	41	-	CLK
Horizontal back porch	thb	2	2	-	CLK
Vertical Signal					
Vertical cycle	tv	-	286	-	H
Vertical display period	tvd	-	272	-	H
Vertical front porch	tvf	1	2	-	H
Vertical pulse width	tvp	1	10	-	H
Vertical back porch	tvb	1	2	-	H

Note:
(1) thd=480CLK, thf=2CLK, thp=41CLK, thb=2CLK, thf + thp + thb > 44CLK. (CLK=1/ fCLK ,H=th)

据此可以得到如下的驱动 drivers/video/samsung/s3cfb_wanxin.c：

```
#include ...

#include "s3cfb.h"

/* 480*272 */
#define S3CFB_HFP     2  /* front porch */
#define S3CFB_HSW     41 /* hsync width */
#define S3CFB_HBP     2 /* back porch */

#define S3CFB_VFP     2   /* front porch */
#define S3CFB_VSW     10   /* vsync width */
#define S3CFB_VBP     2   /* back porch */

#define S3CFB_HRES    480    /* horizon pixel  x resolition */
#define S3CFB_VRES    272 /* line cnt      y resolution */

#define S3CFB_HRES_VIRTUAL 480    /* horizon pixel  x resolition */
#define S3CFB_VRES_VIRTUAL (272*2)    /* line cnt      y resolution */

#define S3CFB_HRES_OSD    480    /* horizon pixel  x resolition */
#define S3CFB_VRES_OSD    272 /* line cnt      y resolution */

#define S3CFB_VFRAME_FREQ          60      /* frame rate freq */

#define S3CFB_PIXEL_CLOCK    (S3CFB_VFRAME_FREQ * (S3CFB_HFP + S3CFB_HSW +
S3CFB_HBP + S3CFB_HRES) * (S3CFB_VFP + S3CFB_VSW + S3CFB_VBP + S3CFB_VRES))

static void s3cfb_set_fimd_info(void)
{
    s3cfb_fimd.vidcon1 = S3C_VIDCON1_IHSYNC_INVERT | S3C_VIDCON1_IVSYNC_INVERT | S3C
_VIDCON1_IVDEN_NORMAL;
    s3cfb_fimd.vidtcon0 = S3C_VIDTCON0_VBPD(S3CFB_VBP - 1) |
S3C_VIDTCON0_VFPD(S3CFB_VFP - 1) | S3C_VIDTCON0_VSPW(S3CFB_VSW - 1);
    s3cfb_fimd.vidtcon1 = S3C_VIDTCON1_HBPD(S3CFB_HBP - 1) |
S3C_VIDTCON1_HFPD(S3CFB_HFP - 1) | S3C_VIDTCON1_HSPW(S3CFB_HSW - 1);
    s3cfb_fimd.vidtcon2 = S3C_VIDTCON2_LINEVAL(S3CFB_VRES - 1) |
S3C_VIDTCON2_HOZVAL(S3CFB_HRES - 1);

    s3cfb_fimd.vidosd0a = S3C_VIDOSDxA_OSD_LTX_F(0) | S3C_VIDOSDxA_OSD_LTY_F(0);
    s3cfb_fimd.vidosd0b = S3C_VIDOSDxB_OSD_RBX_F(S3CFB_HRES - 1) |
S3C_VIDOSDxB_OSD_RBY_F(S3CFB_VRES - 1);

    s3cfb_fimd.vidosd1a = S3C_VIDOSDxA_OSD_LTX_F(0) | S3C_VIDOSDxA_OSD_LTY_F(0);
    s3cfb_fimd.vidosd1b = S3C_VIDOSDxB_OSD_RBX_F(S3CFB_HRES_OSD - 1) |
S3C_VIDOSDxB_OSD_RBY_F(S3CFB_VRES_OSD - 1);

    s3cfb_fimd.width = S3CFB_HRES;
    s3cfb_fimd.height = S3CFB_VRES;
    s3cfb_fimd.xres = S3CFB_HRES;
    s3cfb_fimd.yres = S3CFB_VRES;

#if defined(CONFIG_FB_S3C_VIRTUAL_SCREEN)
    s3cfb_fimd.xres_virtual = S3CFB_HRES_VIRTUAL;
    s3cfb_fimd.yres_virtual = S3CFB_VRES_VIRTUAL;
#else
    s3cfb_fimd.xres_virtual = S3CFB_HRES;
    s3cfb_fimd.yres_virtual = S3CFB_VRES;
#endif

    s3cfb_fimd.osd_width = S3CFB_HRES_OSD;
    s3cfb_fimd.osd_height = S3CFB_VRES_OSD;
    s3cfb_fimd.osd_xres = S3CFB_HRES_OSD;
    s3cfb_fimd.osd_yres = S3CFB_VRES_OSD;

    s3cfb_fimd.osd_xres_virtual = S3CFB_HRES_OSD;
    s3cfb_fimd.osd_yres_virtual = S3CFB_VRES_OSD;
```

```
    s3cfb_fimd.pixclock = S3CFB_PIXEL_CLOCK;

    s3cfb_fimd.hsync_len = S3CFB_HSW;
    s3cfb_fimd.vsync_len = S3CFB_VSW;
    s3cfb_fimd.left_margin = S3CFB_HFP;
    s3cfb_fimd.upper_margin = S3CFB_VFP;
    s3cfb_fimd.right_margin = S3CFB_HBP;
    s3cfb_fimd.lower_margin = S3CFB_VBP;
}

int s3cfb_wanxin_set_gpio(void)
{
    unsigned long val;
    int i;

    /* Must be '0' for Normal-path instead of By-pass */
    writel(0x0, S3C_HOSTIFB_MIFPCON);

    /* enable clock to LCD */
    val = readl(S3C_HCLK_GATE);
    val |= S3C_CLKCON_HCLK_LCD;
    writel(val, S3C_HCLK_GATE);

    /* select TFT LCD type (RGB I/F) */
    val = readl(S3C64XX_SPC_BASE);
    val &= ~0x3;
    val |= (1 << 0);
    writel(val, S3C64XX_SPC_BASE);

    /* VD */
    for (i = 0; i < 16; i++)
            s3c_gpio_cfgpin(S3C64XX_GPI(i), S3C_GPIO_SFN(2));

    for (i = 0; i < 12; i++)
            s3c_gpio_cfgpin(S3C64XX_GPJ(i), S3C_GPIO_SFN(2));

    mdelay(100);

    return 0;
}

void s3cfb_init_hw(void)
{
    printk(KERN_INFO "WANXIN LCD will be initialized\n");

    s3cfb_set_fimd_info();
    s3cfb_wanxin_set_gpio();
}
```

其中的 S3CFB_HFP、S3CFB_HSW、S3CFB_HBP、S3CFB_VFP、S3CFB_VSW、S3CFB_VBP 均直接取材于表中的数据。

而对于 VGA 驱动 drivers/video/samsung/s3cfb_vga.c 而言，这些参数直接取自 VESA 工业标准：

```
#ifdef CONFIG_FB_S3C_VGA_1024_768
/* 1024*768 */
#define S3CFB_HFP       24  /* front porch */
#define S3CFB_HSW       136 /* hsync width */
#define S3CFB_HBP       160 /* back porch */
#define S3CFB_VFP       3   /* front porch */
#define S3CFB_VSW       6   /* vsync width */
#define S3CFB_VBP       29  /* back porch */
#define S3CFB_HRES      1024    /* horizon pixel  x resolition */
#define S3CFB_VRES      768 /* line cnt  y resolution */


#define S3CFB_HRES_VIRTUAL 1024     /* horizon pixel  x resolition */
#define S3CFB_VRES_VIRTUAL (768*2) /* line cnt       y resolution */
#define S3CFB_HRES_OSD        1024    /* horizon pixel  x resolition */
#define S3CFB_VRES_OSD        768 /* line cnt  y resolution */
```

```
#elif defined(CONFIG_FB_S3C_VGA_640_480)
#define S3CFB_HFP       16  /* front porch */
#define S3CFB_HSW       96  /* hsync width */
#define S3CFB_HBP       48  /* back porch */
#define S3CFB_VFP       10  /* front porch */
#define S3CFB_VSW       2   /* vsync width */
#define S3CFB_VBP       33  /* back porch */
#define S3CFB_HRES      640 /* horizon pixel  x resolition */
#define S3CFB_VRES      480 /* line cnt  y resolution */

#define S3CFB_HRES_VIRTUAL 640 /* horizon pixel  x resolition */
#define S3CFB_VRES_VIRTUAL 480 /* line cnt  y resolution */
#define S3CFB_HRES_OSD      640 /* horizon pixel  x resolition */
#define S3CFB_VRES_OSD      480 /* line cnt  y resolution */

#endif
```

2.3.5 WM9714 声卡 ASoC 驱动

目前 Linux ALSA 驱动针对 SoC 系统有一套自己的驱动框架，即 ALSA SoC，简称 ASoC。ASoC 驱动主要由三部分组成：CPU DAI（Digital Audio Interfaces）、Codec DAI／ Codec 驱动、machine 驱动（或称板驱动，用于将 CPU DAI 和 Codec DAI／Codec 连接在一起），具体细节详见 Linux 内核源代码 Documentation/sound/alsa/soc 目录下的各个文档。

对于 S3C6410 上的 WM9714 而言，其 Codec 驱动位于 sound/soc/codecs/wm9713.c，其 CPU DAI 驱动位于 sound/soc/s3c/s3cac97.c，而 machine 驱动位于 sound/soc/s3c/smdk6410_wm9713.c。

在 LDD6410 根文件系统的/usr/lib/目录下包含了 alsa 的库文件，而在/etc 下面包含了声卡配置文件 asound.conf，其内容如下：

```
pcm.wm9713 {
    type hw
    card SMDK6410
    device 0
}

pcm_slave.sltest {
    pcm wm9713 rate 44100
}

pcm.rate_convert {
    type rate slave sl2
}
```

ALSA utility 中的 aplay 可用于播放 wav 文件，而 arecord 则用于录制 wav 文件。譬如 录制 16 bit little endian, 44100, stereo 的 wav：

```
#arecord f cd > 1.wav
Recording WAVE 'stdin' : Signed 16 bit Little Endian, Rate 44100 Hz, Stereo
```

播放它：

```
# aplay 1.wav
Playing WAVE '1.wav' : Signed 16 bit Little Endian, Rate 44100 Hz, Stereo
```

边录边放：

```
# arecord f cd | aplay
Recording WAVE 'stdin' : Signed 16 bit Little Endian, Rate 44100 Hz, Stereo
Playing WAVE 'stdin' : Signed 16 bit Little Endian, Rate 44100 Hz, Stereo
```

2.3.6 DM9000 网卡驱动

Linux 内核下的 drivers/net/dm9000.c 实现了与体系结构无关的 DM9000 网卡驱动，开 发者需要修改一行代码即可使用此驱动，只需要在 BSP 的板文件（对于 LDD6410 为 arch/arm/machs3c6410/mach-ldd6410.c）中定义相关的 platform 设备和数据：

```
/* DM9000 ethernet devices */

static struct resource ldd6410_dm9000_resource[] = {
    [0] = {
```

```
                .start = 0x18000000,
                .end   = 0x18000000 + 3,
                .flags = IORESOURCE_MEM
        },
        [1] = {
                .start = 0x18000000 + 0x4,
                .end   = 0x18000000 + 0x7,
                .flags = IORESOURCE_MEM
        },
        [2] = {
                .start = IRQ_EINT(7),
                .end   = IRQ_EINT(7),
                .flags = IORESOURCE_IRQ | IORESOURCE_IRQ_HIGHLEVEL,
        }

};

static struct dm9000_plat_data ldd6410_dm9000_platdata = {
    .flags          = DM9000_PLATF_16BITONLY | DM9000_PLATF_NO_EEPROM,
    .dev_addr   = { 0x0, 0x16, 0xd4, 0x9f, 0xed, 0xa4 },
};

static struct platform_device ldd6410_dm9000 = {
    .name           = "dm9000",
    .id             = 0,
    .num_resources= ARRAY_SIZE(ldd6410_dm9000_resource),
    .resource       = ldd6410_dm9000_resource,
    .dev            = {
            .platform_data = &ldd6410_dm9000_platdata,
    }
};
```

并将“&ldd6410_dm9000,”语句填入 struct platform_device *ldd6410_devices[]数组，作为该数组的一个成员。

为了使得 LDD6410 能支持域名解析，可修改根文件系统中的/etc/resolv.conf 文件，例如上海铁通为：

```
nameserver 10.203.104.17
nameserver 10.203.104.1
```

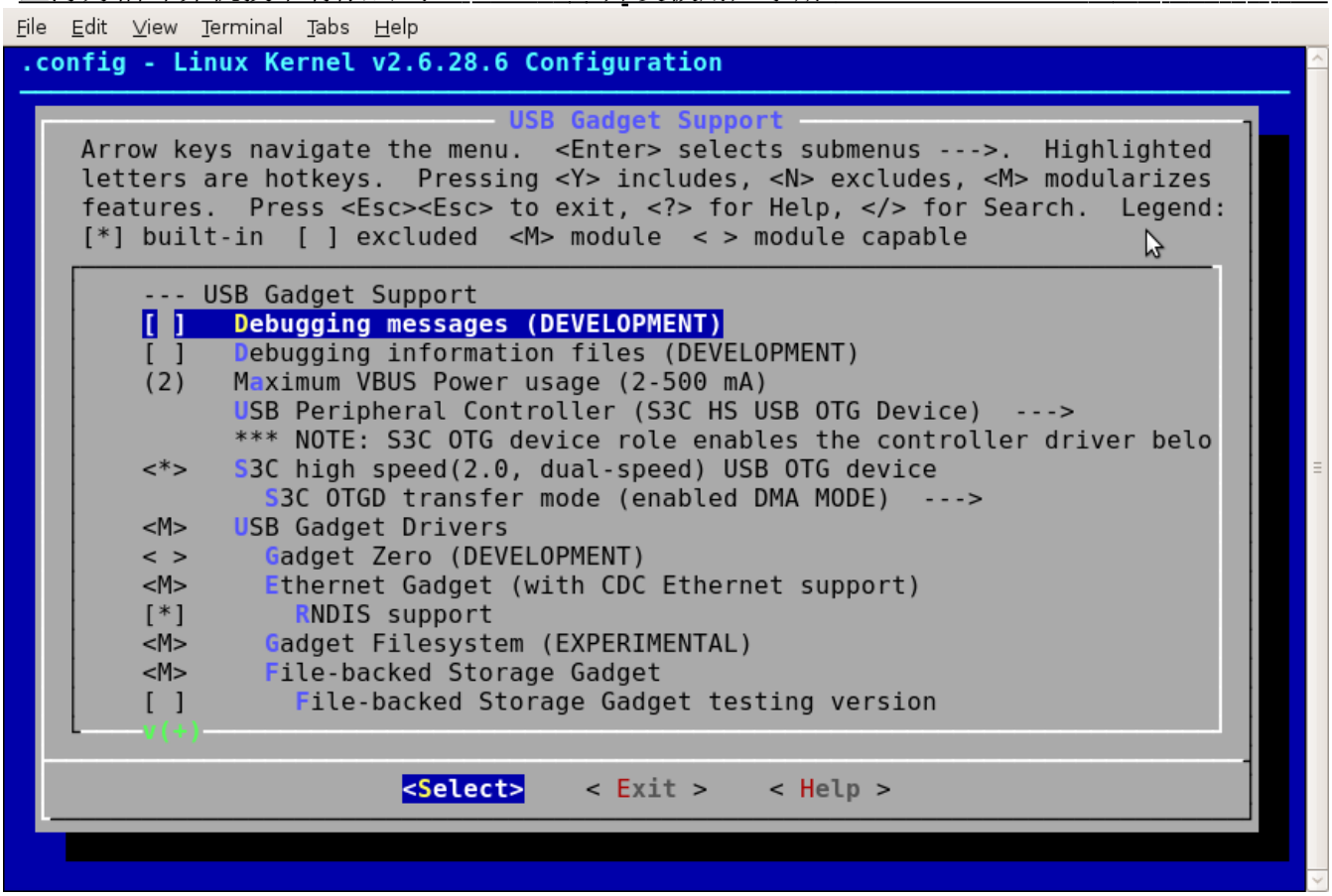
之后访问 www.lihacker.com：

```
# ping www.lihacker.com
PING www.lihacker.com (202.153.195.67): 56 data bytes
64 bytes from 202.153.195.67: seq=0 ttl=49 time=292.071 ms
64 bytes from 202.153.195.67: seq=1 ttl=49 time=295.565 ms
64 bytes from 202.153.195.67: seq=2 ttl=49 time=289.397 ms
64 bytes from 202.153.195.67: seq=3 ttl=49 time=296.995 ms
64 bytes from 202.153.195.67: seq=4 ttl=49 time=301.641 ms
```

2.3.7 USB 驱动

LDD6410 开发板包含 1 个 USB 2.0 OTG 接口和一个 USB 1.1 主机接口。

USB 2.0 OTG 的主机控制器驱动和设备控制器驱动分别位于：drivers/usb/host/s3cotg/目录和 drivers/usb/gadget/s3c_udc_otg.c。当前，USB 2.0 OTG 接口的最主要用途是用作 gadget 设备，编译内核时选择：



file storage gadget 可以使得电路板上的文件或者设备节点成为主机的一个 U 盘，譬如下面的例子将 vfat.img 这个映像作为 U 盘使用：

```
# modprobe g_file_storage file=/demo/vfat.img stall=0 removable=1
g_file_storage gadget: Filebacked Storage Gadget, version: 7 August 2007
g_file_storage gadget: Number of LUNs=1 g_file_storage gadgetlun0: ro=0, file: /demo/vfat.img
Registered gadget driver 'g_file_storage'
```

其中的 vfat.img 可以在 pc 上通过 dd 和 mkfs.vfat 命令得到，例如做一个 20MB 的 VFAT 映像：

```
$ dd if=/dev/zero of=vfat.img bs=1M count=20
20+0 records in
20+0 records out
20971520 bytes (21 MB) copied, 0.195482 s, 107 MB/s
$ sudo losetup /dev/loop0 vfat.img
$ sudo mkfs.vfat /dev/loop0
mkfs.vfat 2.11 (12 Mar 2005)
Loop device does not match a floppy size, using default hd params
$ mkdir vfat_mount_point
$ sudo mount t vfat /dev/loop0 vfat_mount_point
```

这样之后可以把需要的文件拷入 vfat_mount_point 目录，完成后 umount vfat_mount_point 目录并删除 loop0：

```
$ sudo umount vfat_mount_point
$ sudo losetup d /dev/loop0
```

S3C6410 的 USB 1.1 主机控制器驱动采用 OHCI 结构，源代码位于内核的 drivers/usb/host 目录。对于 USB 1.1 的主机而言，我们需要在内核 config 中选中

```
Device Drivers >
  [*] USB support >
      <*>     OHCI HCD support
```

为了支持 USB 鼠标，在内核 config 中选择：

```
[*] HID Devices >
  <*> USB Human Interface Device (full HID) support
```

为了支持 U 盘，在内核 config 中选择：

```
Device Drivers >
  [*] USB support >
```

```
        <*>  USB Mass Storage support
```

当插入一个 USB 鼠标，控制台会打印类似信息：

```
usb 11: configuration #1 chosen from 1 choice input: USB Mouse as /class/input/input2
genericusb 0003:1267:0201.0001: input: USB HID v1.00 Mouse [USB Mouse] on usb
s3c24xx1/input0
```

拔出 USB 鼠标后，控制台打印：

```
usb 11: USB disconnect, address 2
```

插入一个 U 盘，控制台会打印类似信息：

```
usb 11: new full speed USB device using s3c2410ohci and address 4
usb 11: configuration #1 chosen from 1 choice
scsi0 : SCSI emulation for USB Mass Storage devices
# scsi 0:0:0:0: DirectAccess      Lenovo  USB Flash Drive  1100 PQ: 0 ANSI: 0 CCS
sd 0:0:0:0: [sda] 7831552 512byte hardware sectors: (4.00 GB/3.73 GiB)
sd 0:0:0:0: [sda] Write Protect is off
sd 0:0:0:0: [sda] Assuming drive cache: write through
sd 0:0:0:0: [sda] 7831552 512byte hardware sectors: (4.00 GB/3.73 GiB)
sd 0:0:0:0: [sda] Write Protect is off
sd 0:0:0:0: [sda] Assuming drive cache: write through sda: sda1
sd 0:0:0:0: [sda] Attached SCSI removable disk sd 0:0:0:0: Attached scsi generic sg0 type 0
```

通过 mount 命令来挂载这个 U 盘：

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

2.3.8 驱动学习实例：helloworld、globalmem、globalfifo

将 helloworld、globalmem、globalfifo 放入内核代码中添加代码、Kconfig 和 Makefile 在 Linux 内核源码的 drivers/char 目录下建立子目录：

```
lihacker@lihackerlaptop:~/ldd6410/linux2.6.28samsung/drivers/char$ mkdir driver_examples
```

将三个驱动 hello.c、globalmem.c、globalfifo.c 拷入 driver_examples 目录：

```
cd driver_examples/
cp ../../../../training/kernel/drivers/hello/hello.c ./
cp ../../../../training/kernel/drivers/globalmem/globalmem.c ./
cp ../../../../training/kernel/drivers/globalmem/globalfifo.c ./
```

修改 drivers/char 下面的 Kconfig 和 Makefile 导入 driver_examples 目录：

* 在 drivers/char/Kconfig 中添加：

```
source "drivers/char/driver_examples/Kconfig"
```

* 添加 drivers/char/driver_examples/Kconfig 文件：

```
#
# driver examples configuration
#
menu
config DRIVER_EXAMPLE
tristate "driver examples in 'Explain Linux Device Drivers in detail'"
help
say Yes to buildin hello world, globalmem, globalfifo, say M to get those kernel modules
if DRIVER_EXAMPLE
config HELLO_WORLD
tristate "Hello World"
help
To compile this driver as a module, choose M here; the module will be called hello.mem

config GLOBALMEM
tristate "globalmem"
help
To compile this driver as a module, choose M here; the module will be called globalmem.

config GLOBALFIFO
tristate "globalfifo"
help
To compile this driver as a module, choose M here; the module will be called globalfifo.
endif # DRIVER_EXAMPLE
```

* 在 drivers/char/Makefile 中添加：

```
obj$(CONFIG_DRIVER_EXAMPLE)        += driver_examples/
```

* 添加 drivers/char/driver_examples/Makefile 文件：

```
obj$(CONFIG_HELLO_WORLD)            += hello.o
obj$(CONFIG_GLOBALMEM)       += globalmem.o
obj$(CONFIG_GLOBALFIFO)      += globalfifo.o
```

运行 make menuconfig 选中三个模块，在 linux2.6.28samsung 下运行“make modules”命令即可编译出所有内核模块。

这些驱动的运行方法：

```
# modprobe hello
Hello World enter
# modprobe globalmem  globalmem_major=250
# modprobe globalfifo globalfifo_major=251
```

查看加载的模块：

```
# lsmod
globalmem 3356 0 - Live 0xc0f66000
hello 1188 0 - Live 0xc1128000
globalfifo 4292 0 - Live 0xc182a000
```

查看设备：

```
# cat /proc/devices
Character devices:
  1 mem
  2 pty
  3 ttyp
  4 /dev/vc/0
  4 tty
  4 ttyS
  5 /dev/tty
  5 /dev/console
  5 /dev/ptmx
  7 vcs
 10 misc
 13 input
 21 sg
 29 fb
128 ptm
136 pts
204 s3c2410_serial
250 globalmem
251 globalfifo
```

建立结点：

```
# mknod /dev/globalmem c 250 0
# mknod /dev/globalfifo c 251 0
#
```

读写设备文件：

```
# echo "hello, lihacker" > /dev/globalmem
written 16 bytes(s) from 0
# cat /dev/globalmem
read 4096 bytes(s) from 0
hello, cisco

# cat /dev/globalfifo &
# echo "hello, lihacker, I love you" > /dev/globalfifo
written 27 bytes(s),current_len:27
# read 27 bytes(s),current_len:0
hello,lihacker, I love you
```

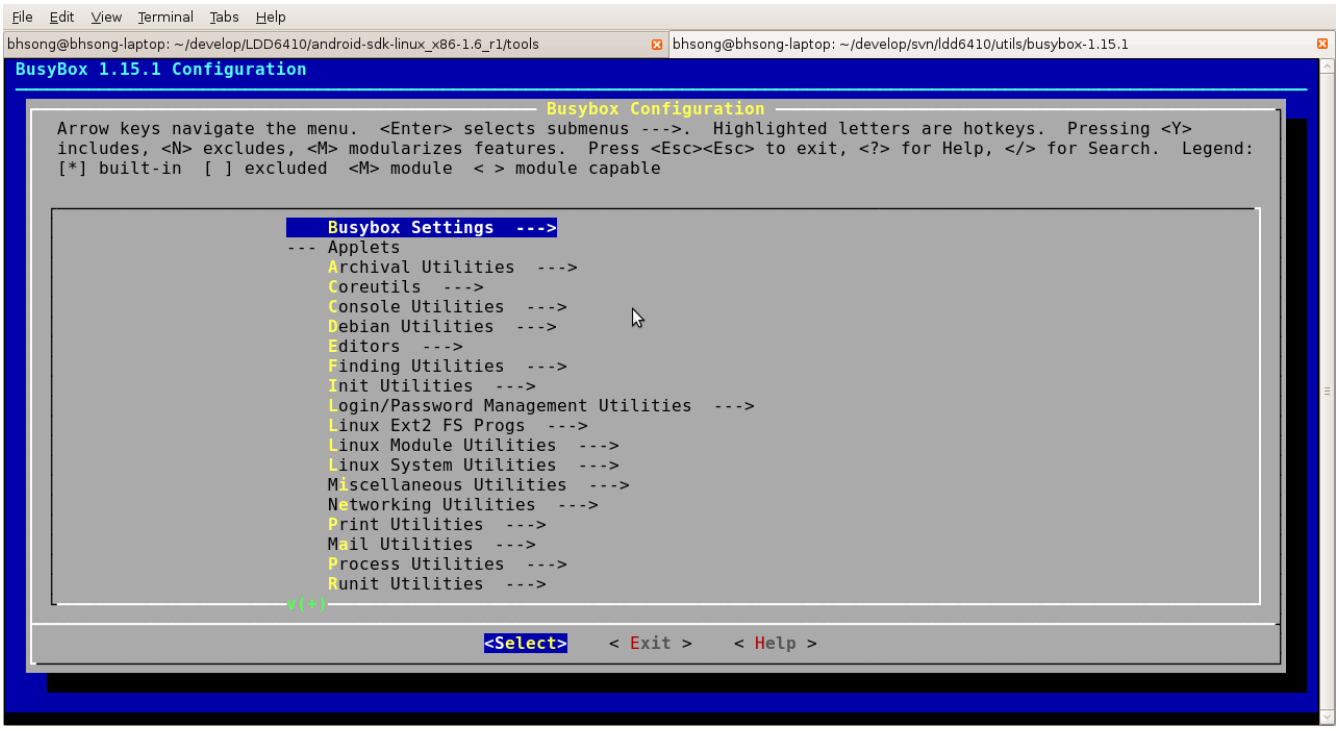
2.4 根文件系统

2.4.1 根文件系统的组成

根文件系统方面，力楷为 LDD6410 整合了唯一一个基于 BUSYBOX 的根文件系统，Android、QT 等都是文件系统的一部分。BUSYBOX 方面为最新的稳定版本：

```
# busybox
BusyBox v1.15.1 (20091212 21:04:26 CST) multicall binary
Copyright (C) 19982008 Erik Andersen, Rob Landley, Denys Vlasenko and others. Licensed under GPLv2.
See source distribution for full notice.
Usage: busybox [function] [arguments]... or: function [arguments]...
BusyBox is a multicall binary that combines many common Unix utilities into a single executable. Most people will create a
link to busybox for each function they wish to use and BusyBox will act like whatever it was invoked as!

Currently defined functions:
addgroup, adduser, adjtimex, arp, arping, ash, awk, basename, bunzip2, bzcat, bzip2, cal, cat, catv, chgrp, chmod, chown, chpasswd, chroot,
chrt, cksum, clear, cmp, comm, cp, crond, crontab, cryptpw, cut, date, dc, dd, delgroup, deluser, depmod, df, dhcprelay, diff, dirname, dmesg,
dnsdomainname, dos2unix, du, dumpleases, echo, egrep, env, expr, fdflush, fdisk, fgrep, find, flash_eraseall, free, fsck, fsync, getty,
grep, gunzip, gzip, halt, hostname, hush, ifconfig, inetd, init, insmod, ip, ipaddr, iplink, iproute, iprule, iptunnel, kill, killall, killall5, klogd, last, length, less, linuxrc, ln, logger, login,
logread, ls, lsmod, md5sum, mdev, mkdir, mkdosfs, mkfifo, mkfs.vfat,
mknod, mkpasswd, modprobe, more, mount, msh, mv, nc, netstat, nice, passwd, patch, ping, poweroff, printenv, ps, pwd, readlink, reboot,
renice, reset, resize, rm, rmdir, rmmod, route, rtcwake, rx, sed, seq, sh, sleep, sort, split, stat, stty, su, sulogin, swapoff, swapon, sync, sysctl, syslogd, tail, tar, telnet, telnetd, tftp, tftpd, time, top,
touch, tr, traceroute, true, tty, udhcpc, udhcpd, umount, uname,
unexpand, uniq, unix2dos, uptime, usleep, uudecode, uuencode, vconfig, vi, vlock, watch, wc, which, who, whoami, yes, zcat
```

编译 BUSYBOX 的方式与 Linux 内核比较相似，同样也是通过 make menuconfig 选择需要的项目：



而后在 BUSYBOX 的源代码目录运行 make 和 make install,源代码目录下会多出一个_install 的目录，其内容为:

```
lihacker@lihackerlaptop:~/develop/svn/ldd6410/utils/busybox1.15.1/_install$ tree
.
| bin
|  | addgroup > busybox
|  | adduser > busybox
|  | ash > busybox
|  | busybox
|  | cat > busybox
|  | catv > busybox
|  | chgrp > busybox
|  | chmod > busybox
|  | chown > busybox
|  | cp > busybox
|  | date > busybox
|  | dd > busybox
|  | delgroup > busybox
|  | deluser > busybox
|  | df > busybox
|  | dmesg > busybox
|  | dnsdomainname > busybox
|  | echo > busybox
|  | egrep > busybox
|  | fdflush > busybox
|  | fgrep > busybox
|  | fsync > busybox
|  | grep > busybox
|  | gunzip > busybox
|  | gzip > busybox
|  | hostname > busybox
|  | hush > busybox
|  | ip > busybox
|  | ipaddr > busybox
|  | iplink > busybox
|  | iproute > busybox
|  | iprule > busybox
|  | iptunnel > busybox
|  | kill > busybox
|  | ln > busybox
|  | login > busybox
```

```
|  | ls > busybox
|  | mkdir > busybox
|  | mknod > busybox
|  | more > busybox
|  | mount > busybox
|  | msh > busybox
|  | mv > busybox
|  | netstat > busybox
|  | nice > busybox
|  | ping > busybox
|  | printenv > busybox
|  | ps > busybox
|  | pwd > busybox
|  | rm > busybox
|  | rmdir > busybox
|  | sed > busybox
|  | sh > busybox
|  | sleep > busybox
|  | stat > busybox
|  | stty > busybox
|  | su > busybox
|  | sync > busybox
|  | tar > busybox
|  | touch > busybox
|  | true > busybox
|  | umount > busybox
|  | uname > busybox
|  | usleep > busybox
|  | vi > busybox
|  | watch > busybox
|  ` zcat > busybox
| linuxrc > bin/busybox
| sbin
|  | adjtimex > ../bin/busybox
|  | arp > ../bin/busybox
|  | depmod > ../bin/busybox
|  | fdisk > ../bin/busybox
|  | fsck > ../bin/busybox
|  | getty > ../bin/busybox
|  | halt > ../bin/busybox
|  | ifconfig > ../bin/busybox
|  | init > ../bin/busybox
|  | insmod > ../bin/busybox
|  | klogd > ../bin/busybox
|  | logread > ../bin/busybox
|  | lsmod > ../bin/busybox
|  | mdev > ../bin/busybox
|  | mkdosfs > ../bin/busybox
|  | mkfs.vfat > ../bin/busybox
|  | modprobe > ../bin/busybox
|  | poweroff > ../bin/busybox
|  | reboot > ../bin/busybox
|  | rmmod > ../bin/busybox
|  | route > ../bin/busybox
|  | sulogin > ../bin/busybox
|  | swapoff > ../bin/busybox
|  | swapon > ../bin/busybox
|  | sysctl > ../bin/busybox
|  | syslogd > ../bin/busybox
|  | udhcpc > ../bin/busybox
|  ` vconfig > ../bin/busybox
` usr
| bin
|  | arping > ../../bin/busybox
|  | awk > ../../bin/busybox
|  | basename > ../../bin/busybox
|  | bunzip2 > ../../bin/busybox
|  | bzcat > ../../bin/busybox
```

```
|   |-bzip2 -> ../../bin/busybox
|   |-cal -> ../../bin/busybox
|   |-chrt -> ../../bin/busybox
|   |-cksum -> ../../bin/busybox
|   |-clear -> ../../bin/busybox
|   |-cmp -> ../../bin/busybox
|   |-comm -> ../../bin/busybox
|   |-crontab -> ../../bin/busybox
|   |-cryptpw -> ../../bin/busybox
|   |-cut -> ../../bin/busybox
|   |-dc -> ../../bin/busybox
|   |-diff -> ../../bin/busybox
|   |-dirname -> ../../bin/busybox
|   |-dos2unix -> ../../bin/busybox
|   |-du -> ../../bin/busybox
|   |-dumpleases -> ../../bin/busybox
|   |-env -> ../../bin/busybox
|   |-expr -> ../../bin/busybox
|   |-find -> ../../bin/busybox
|   |-free -> ../../bin/busybox
|   |-killall -> ../../bin/busybox
|   |-killall5 -> ../../bin/busybox
|   |-last -> ../../bin/busybox
|   |-length -> ../../bin/busybox
|   |-less -> ../../bin/busybox
|   |-logger -> ../../bin/busybox
|   |-md5sum -> ../../bin/busybox
|   |-mkfifo -> ../../bin/busybox
|   |-mkpasswd -> ../../bin/busybox
|   |-nc -> ../../bin/busybox
|   |-passwd -> ../../bin/busybox
|   |-patch -> ../../bin/busybox
|   |-readlink -> ../../bin/busybox
|   |-renice -> ../../bin/busybox
|   |-reset -> ../../bin/busybox
|   |-resize -> ../../bin/busybox
|   |-rtcwake -> ../../bin/busybox
|   |-rx -> ../../bin/busybox
|   |-seq -> ../../bin/busybox
|   |-sort -> ../../bin/busybox
|   |-split -> ../../bin/busybox
|   |-tail -> ../../bin/busybox
|   |-telnet -> ../../bin/busybox
|   |-tftp -> ../../bin/busybox
|   |-tftpd -> ../../bin/busybox
|   |-time -> ../../bin/busybox
|   |-top -> ../../bin/busybox
|   |-tr -> ../../bin/busybox
|   |-traceroute -> ../../bin/busybox
|   |-tty -> ../../bin/busybox
|   |-unexpand -> ../../bin/busybox
|   |-uniq -> ../../bin/busybox
|   |-unix2dos -> ../../bin/busybox
|   |-uptime -> ../../bin/busybox
|   |-uudecode -> ../../bin/busybox
|   |-uuencode -> ../../bin/busybox
|   |-vlock -> ../../bin/busybox
|   |-wc -> ../../bin/busybox
|   |-which -> ../../bin/busybox
|   |-who -> ../../bin/busybox
|   |-whoami -> ../../bin/busybox
|   `-yes -> ../../bin/busybox
`-sbin
```

```
| chpasswd > ../../bin/busybox
| chroot > ../../bin/busybox
| crond > ../../bin/busybox
| dhcprelay > ../../bin/busybox
| flash_eraseall > ../../bin/busybox
| inetd > ../../bin/busybox
| telnetd > ../../bin/busybox
` udhcpd > ../../bin/busybox
5 directories, 171 files
```

可以看出/bin、/sbin、/usr/bin 下面的程序都指向 busybox 同一个 binary，这很好地体现了 BUSYBOX“瑞士军刀”的概念。

2.4.2 使用 nfs 作为根文件系统

在调试过程中，我们可以使用 nfs 文件系统，将主机上的目录挂载到目标机，以避免频繁地在二者之前拷贝程序。

内核配置

内核中可使能如下选项：

```
CONFIG_NETWORK_FILESYSTEMS=y
CONFIG_NFS_FS=y
CONFIG_NFS_V3=y
# CONFIG_NFS_V3_ACL is not set
CONFIG_NFS_V4=y
CONFIG_ROOT_NFS=y
# CONFIG_NFSD is not set
CONFIG_LOCKD=y
CONFIG_LOCKD_V4=y
CONFIG_NFS_COMMON=y
CONFIG_SUNRPC=y
CONFIG_SUNRPC_GSS=y
```

UBOOT 配置和启动启动配置信息如下：

```
root=/dev/nfs ip=192.168.1.20:192.168.1.111:192.168.1.1:255.255.255.0::eth0:off
nfsroot=192.168.1.111:/home/nfs,nolock,proto=tcp console=ttySAC0,115200
```

其中 192.168.1.20 为目标板 IP，192.168.1.111 为主机 IP，192.168.1.1 是网关 IP，255.255.255.0 为子网掩码。/home/nfs 是主机 NFS 根目录。 通过在 UBOOT 中运行如下命令完成：

```
LDD6410 # set bootargs root=/dev/nfs ip=192.168.1.20:192.168.1.111:192.168.1.1:255.255.255.0::eth0:off
nfsroot=192.168.1.111:/home/nfs,nolock,proto=tcp console=ttySAC0,115200
LDD6410 #
```

通过 NFS 启动根文件系统的启动信息最后一步就是 mount nfs，具体如下：

```
eth0: link up, 100Mbps, fullduplex, lpa 0x45E1
IPConfig: Complete:
device=eth0, addr=192.168.1.20, mask=255.255.255.0, gw=192.168.1.1,host=192.168.1.20, domain=, nis-
domain=(none), bootserver=192.168.1.111, rootserver=192.168.1.111, rootpath=
Looking up port of RPC 100003/2 on 192.168.1.111
Looking up port of RPC 100005/1 on 192.168.1.111
VFS: Mounted root (nfs filesystem). Freeing init memory: 524K
Welcome to
ARMLinux for Lihacker LDD6410
For further information please check: http://www.lihacker.com/ http://www.linuxdriver.cn/
For technical support, please subscribe mail list linuxdriver@googlegroups.com and post there.
The url is http://groups.google.com/group/linuxdriver.
#
```

2.5 应用程序

应用方面，LDD6410 针对电路板整合了大量测试程序（如按键、鼠标、触摸 屏、framebuffer、LED 测试等）、demo 程序和应用程序（如 jpegview、mplayer、appweb 等）。

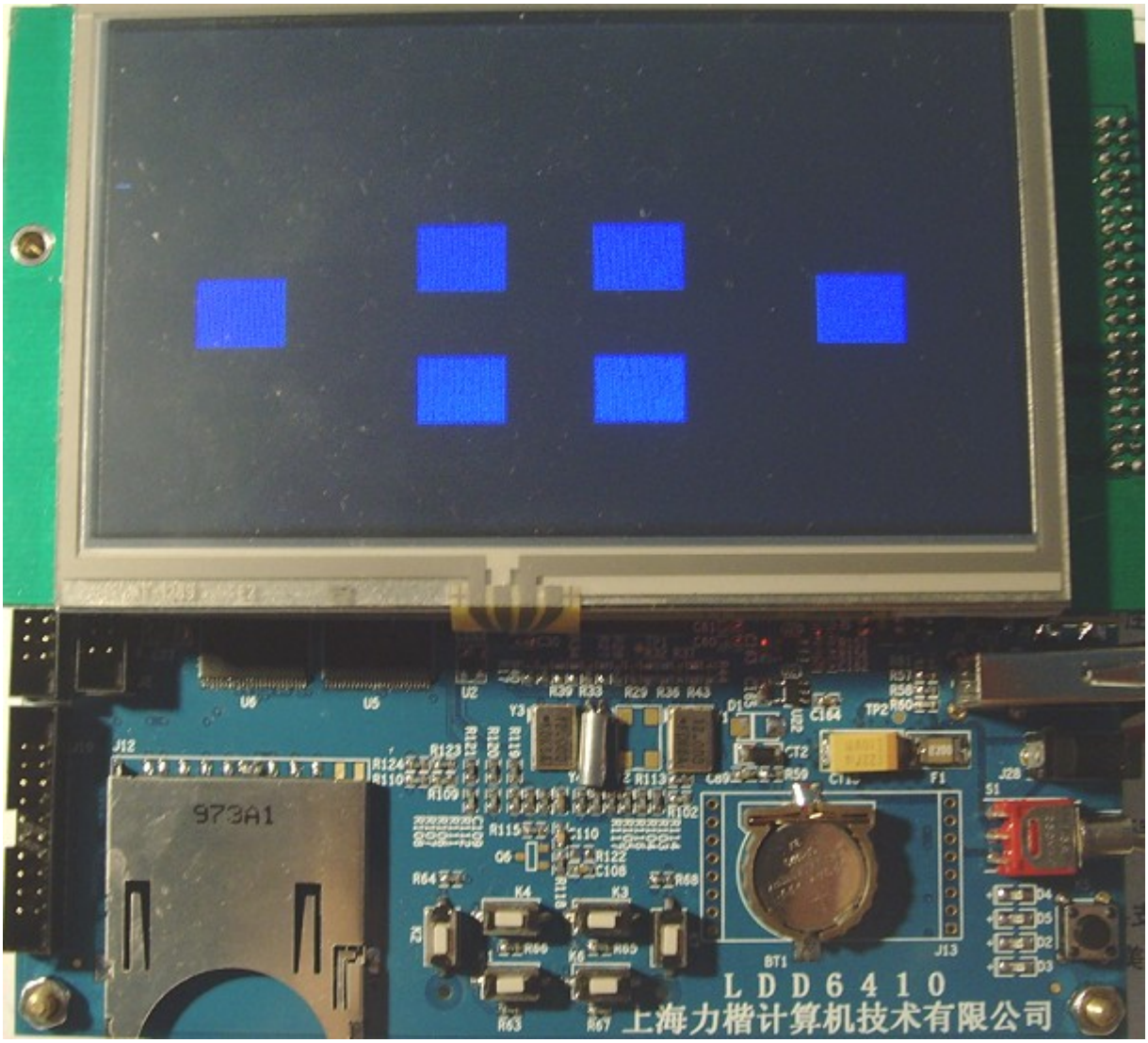
2.5.1 按键测试程序

编写 input 设备应用程序的核心工作是打开 input event 节点，通过 read()获取 input_event，解析 input_event 进行相应处理。LDD6410 的按键测试程序会在用户按下键的同时在屏幕相应位置进行显示：

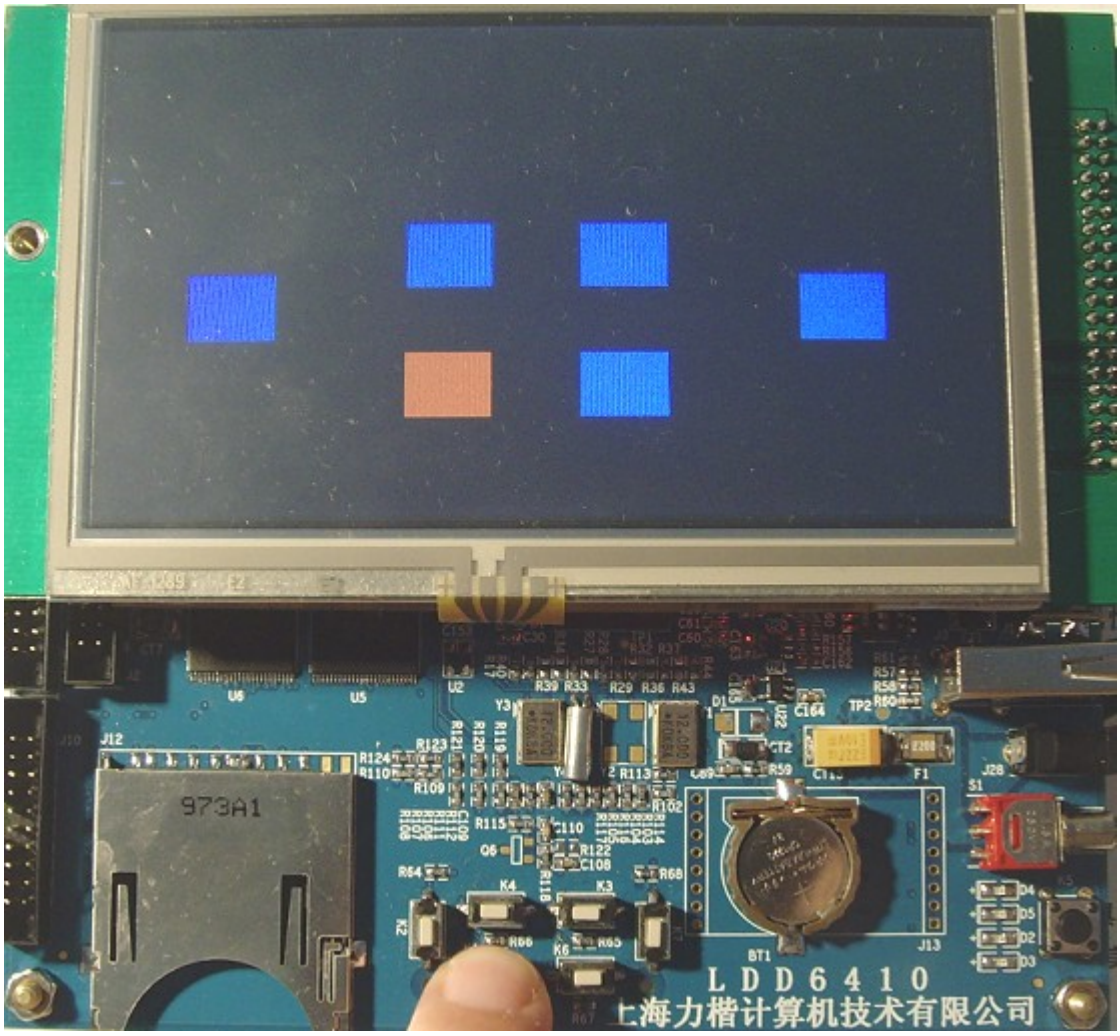
```
/*
 * LDD6410 key test programs
 * Copyright 2009 LiHacker Computer Technology Inc.
 */
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <fcntl.h>
#include <linux/fb.h>
#include <linux/input.h>
#include <sys/mman.h>
#include <sys/time.h>

char *fbp = 0;
struct fb_var_screeninfo vinfo;
struct fb_fix_screeninfo finfo;

void draw_rect(int x_s, int x_e, int y_s, int y_e,  int color)
{
    int x, y;

    /* 1024*768 is for VGA, for WANXIN LCD, let x,y smaller */

    x_s = (x_s * 1.0/1024) * vinfo.xres;
    x_e = (x_e * 1.0/1024) * vinfo.xres;
    y_s = (y_s * 1.0/768.0) * vinfo.yres;
    y_e = (y_e * 1.0/768.0) * vinfo.yres;

    for (y = y_s; y < y_e; y++) {
            for (x = x_s; x < x_e; x++) {

                    unsigned long location = (x+vinfo.xoffset) * (vinfo.bits_per_pixel/8) +
                            (y+vinfo.yoffset) * finfo.line_length;

                    *((unsigned short int*)(fbp + location)) = color;
            }
    }
}

int main()
{
    int fbfd = 0;
    int keyfd = 0;
    struct input_event event;

    long int screensize = 0;
    int x = 0, y = 0;

    // Open the file for reading and writing
    fbfd = open("/dev/fb0", O_RDWR);
    if (!fbfd) {
            printf("Error: cannot open framebuffer device.\n");
            exit(1);
    }
    printf("The framebuffer device was opened successfully.\n");

    keyfd = open("/dev/event0", O_RDWR);
    if (!keyfd) {
            printf("Error: cannot open key input device.\n");
            exit(1);
    }
    printf("The key device was opened successfully.\n");


    // Get fixed screen information
    if (ioctl(fbfd, FBIOGET_FSCREENINFO, &finfo)) {
            printf("Error reading fixed information.\n");
            exit(2);
```

```
    }

    // Get variable screen information
    if (ioctl(fbfd, FBIOGET_VSCREENINFO, &vinfo)) {
            printf("Error reading variable information.\n");
            exit(3);
    }

    printf("%dx%d, %dbpp\n", vinfo.xres, vinfo.yres, vinfo.bits_per_pixel);

    // Figure out the size of the screen in bytes
    screensize = vinfo.xres * vinfo.yres * vinfo.bits_per_pixel / 8;

    // Map the device to memory
    fbp = (char *)mmap(0, screensize, PROT_READ | PROT_WRITE, MAP_SHARED,
            fbfd, 0);
    if ((int)fbp == -1) {
            printf("Error: failed to map framebuffer device to memory.\n");
            exit(4);
    }
    printf("The framebuffer device was mapped to memory successfully.\n");

    draw_rect(0, 1024, 0, 768, 0);
    draw_rect(100,200,400,500,0x1f);
    draw_rect(350,450,320,420,0x1f);
    draw_rect(550,650,320,420,0x1f);
    draw_rect(350,450,520,620,0x1f);
    draw_rect(550,650,520,620,0x1f);
    draw_rect(800,900,400,500,0x1f);

    int left = 0,right = 0;

    while(1) {
            int ret = read(keyfd, &event, sizeof(struct input_event));
            if (ret == sizeof(struct input_event))
                    printf("type:%d code:%d value:%d\n", event.type, event.code, event.value);
            else
                    continue;

            if( event.type == 1) {
                    if (event.code == KEY_ENTER) {
                            draw_rect(100,200,400,500,event.value > 0 ? (0x1f << 11):0x1f);
                            left = event.value;
                    }
                    if (event.code == KEY_POWER)
                            draw_rect(350,450,320,420,event.value > 0 ? (0x1f << 11):0x1f);
                    if (event.code == KEY_DOWN)
                            draw_rect(350,450,520,620,event.value > 0 ? (0x1f << 11):0x1f);
                    if (event.code == KEY_HOME)
                            draw_rect(550,650,320,420,event.value > 0 ? (0x1f << 11):0x1f);
                    if (event.code == KEY_TAB)
                            draw_rect(550,650,520,620,event.value > 0 ? (0x1f << 11):0x1f);
                    if (event.code == KEY_MENU) {
                            draw_rect(800,900,400,500,event.value > 0 ? (0x1f << 11):0x1f);
                            right = event.value;
                    }
                    if (left && right)
                            exit(0);
            }
    }
    munmap(fbp, screensize);
    close(fbfd);
    return 0;
}
```

该程序位于 LDD6410 工程源代码的 tests/keypadmice/key_test.c，其最开始的运行效果如下图：



现在按下一个键，看看屏幕的变化：



同时按下最左和最右的按键将退出该程序。

2.5.2 USB 鼠标测试程序

鼠标通过 input event 结点报告相对坐标事件，编写鼠标设备应用程序的核心工作是打开 input event 节点，通过 read()获取 input_event，解析 input_event 进行相应处理。LDD6410 的鼠标测试程序会在用户移动鼠标的同时在屏幕显示鼠标位置，当按下鼠标左、中、右键的时候相应的矩形窗口变色：

```
/*
 * LDD6410 USB mice test programs
 * Copyright 2009 LiHacker Computer Technology Inc.
 */

#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <fcntl.h>
#include <linux/fb.h>
#include <linux/input.h>
#include <sys/mman.h>

char *fbp = 0;
struct fb_var_screeninfo vinfo;
struct fb_fix_screeninfo finfo;

void draw_rect(int x_s, int x_e, int y_s, int y_e,  int color)
{
    int x, y;

    /* for WANXIN LCD, let x,y smaller */

    x_s = (x_s * 1.0/1024) * vinfo.xres;
    x_e = (x_e * 1.0/1024) * vinfo.xres;
```

```
    y_s = (y_s * 1.0/768.0) * vinfo.yres;
    y_e = (y_e * 1.0/768.0) * vinfo.yres;

    for (y = y_s; y < y_e; y++) {
            for (x = x_s; x < x_e; x++) {

                    unsigned long location = (x+vinfo.xoffset) * (vinfo.bits_per_pixel/8) +
                            (y+vinfo.yoffset) * finfo.line_length;

                    *((unsigned short int*)(fbp + location)) = color;
            }
    }
}

int main()
{
    int fbfd = 0;
    int keyfd = 0;
    struct input_event event;

    long int screensize = 0;
    int x = 0, y = 0;

    // Open the file for reading and writing
    fbfd = open("/dev/fb0", O_RDWR);
    if (!fbfd) {
            printf("Error: cannot open framebuffer device.\n");
            exit(1);
    }
    printf("The framebuffer device was opened successfully.\n");

    keyfd = open("/dev/event2", O_RDWR);
    if (!keyfd) {
            printf("Error: cannot open mice input device.\n");
            exit(1);
    }
    printf("The mice device was opened successfully.\n");


    // Get fixed screen information
    if (ioctl(fbfd, FBIOGET_FSCREENINFO, &finfo)) {
            printf("Error reading fixed information.\n");
            exit(2);
    }

    // Get variable screen information
    if (ioctl(fbfd, FBIOGET_VSCREENINFO, &vinfo)) {
            printf("Error reading variable information.\n");
            exit(3);
    }

    printf("%dx%d, %dbpp\n", vinfo.xres, vinfo.yres, vinfo.bits_per_pixel);

    // Figure out the size of the screen in bytes
    screensize = vinfo.xres * vinfo.yres * vinfo.bits_per_pixel / 8;

    // Map the device to memory
    fbp = (char *)mmap(0, screensize, PROT_READ | PROT_WRITE, MAP_SHARED,
                    fbfd, 0);
    if ((int)fbp == -1) {
            printf("Error: failed to map framebuffer device to memory.\n");
            exit(4);
    }
    printf("The framebuffer device was mapped to memory successfully.\n");

    draw_rect(0,1024,0,768,0xffff);
    draw_rect(300,400,400,500,0x1f);
    draw_rect(500,600,400,500,0x1f);
```

```
    draw_rect(700,800,400,500,0x1f);

    int screen_x = 100, screen_y = 100;
    int left=0,right=0;

    while(1) {
            int ret = read(keyfd, &event, sizeof(struct input_event));
            if (ret == sizeof(struct input_event))
                    printf("type:%d code:%d value:%d\n", event.type, event.code, event.value);
            else {
                    printf("read len:%d\n", ret);
                    continue;
            }

            if(event.type == EV_KEY) {
                    if (event.code == BTN_LEFT) {
                            draw_rect(300,400,400,500,event.value > 0 ? (0x1f << 11):0x1f);
                            left = event.value;
                    }
                    if (event.code == BTN_MIDDLE)
                            draw_rect(500,600,400,500,event.value > 0 ? (0x1f << 11):0x1f);
                    if (event.code == BTN_RIGHT) {
                            draw_rect(700,800,400,500,event.value > 0 ? (0x1f << 11):0x1f);
                            right = event.value;
                    }
            }
            if (left && right)
                    exit(0);

            if(event.type == EV_REL) {
                    if(event.code == REL_X)
                            x = event.value;
                    if(event.code == REL_Y)
                            y = event.value;
            }
            if(event.type == 0) {
                    screen_x += x;
                    screen_y += y;
                    if(screen_x>=1014) screen_x = 1014;
                    if(screen_y>=758) screen_y = 758;

                    if(screen_x<0) screen_x = 0;
                    if(screen_y<0) screen_y = 0;
                    draw_rect(screen_x,screen_x+10,screen_y,screen_y+10, 0);
                    x = y =0;
            }
    }
    munmap(fbp, screensize);
    close(fbfd);
    return 0;
}
```

该程序位于 LDD6410 工程源代码的 tests/keypadmice/mice_test.c，同时按下鼠标的左键和右键将退出该程序。

2.5.3 触摸屏测试程序

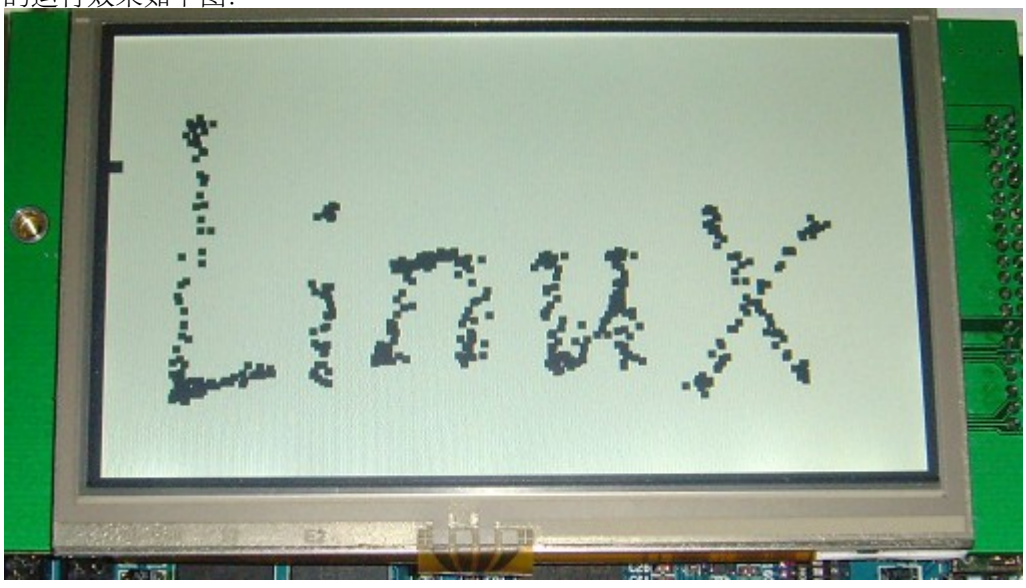
触摸屏通过 input event 结点报告绝 对 坐标事件，编写触摸屏设备应用程序的核心工作 是打开 input even 节点，通过 read()获取 input_event，解析 input_event 进行相应处理。LDD6410 的触摸屏测试程序会在用户触摸屏幕的时候显示触摸位置：

```
/*
 * LDD6410 touchpad test programs
 * Copyright 2009 LiHacker Computer Technology Inc.
 */

#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
```

```
#include <fcntl.h>
#include <linux/fb.h>
#include <linux/input.h>
#include <sys/mman.h>

char *fbp = 0;
struct fb_var_screeninfo vinfo;
struct fb_fix_screeninfo finfo;

void draw_rect(int x_s, int x_e, int y_s, int y_e,  int color)
{
    int x, y;

    for (y = y_s; y < y_e; y++) {
            for (x = x_s; x < x_e; x++) {

                    unsigned long location = (x+vinfo.xoffset) * (vinfo.bits_per_pixel/8) +
                            (y+vinfo.yoffset) * finfo.line_length;

                    *((unsigned short int*)(fbp + location)) = color;
            }
    }
}

int main()
{
    int fbfd = 0;
    int keyfd = 0;
    struct input_event event;

    long int screensize = 0;
    int x = 0, y = 0;
    int screen_x = 0 , screen_y = 0;

    // Open the file for reading and writing
    fbfd = open("/dev/fb0", O_RDWR);
    if (!fbfd) {
            printf("Error: cannot open framebuffer device.\n");
            exit(1);
    }
    printf("The framebuffer device was opened successfully.\n");

    keyfd = open("/dev/event1", O_RDWR);
    if (!keyfd) {
            printf("Error: cannot open mice input device.\n");
            exit(1);
    }
    printf("The mice device was opened successfully.\n");


    // Get fixed screen information
    if (ioctl(fbfd, FBIOGET_FSCREENINFO, &finfo)) {
            printf("Error reading fixed information.\n");
            exit(2);
    }

    // Get variable screen information
    if (ioctl(fbfd, FBIOGET_VSCREENINFO, &vinfo)) {
            printf("Error reading variable information.\n");
            exit(3);
    }

    printf("%dx%d, %dbpp\n", vinfo.xres, vinfo.yres, vinfo.bits_per_pixel);

    // Figure out the size of the screen in bytes
    screensize = vinfo.xres * vinfo.yres * vinfo.bits_per_pixel / 8;

    // Map the device to memory
```

```
    fbp = (char *)mmap(0, screensize, PROT_READ | PROT_WRITE, MAP_SHARED,
                       fbfd, 0);
    if ((int)fbp == -1) {
            printf("Error: failed to map framebuffer device to memory.\n");
            exit(4);
    }
    printf("The framebuffer device was mapped to memory successfully.\n");

    draw_rect(0,vinfo.xres,0,vinfo.yres,0xffff);

    while(1) {
            int ret = read(keyfd, &event, sizeof(struct input_event));
            if (ret == sizeof(struct input_event))
                    printf("type:%d code:%d value:%d\n", event.type, event.code, event.value);
            else {
                    printf("read len:%d\n", ret);
                    continue;
            }

            if(event.type == EV_ABS) {
                    if(event.code == ABS_X)
                            x = event.value;
                    if(event.code == ABS_Y)
                            y = event.value;
            }

            if(event.type == 0) {
                    screen_x = x;
                    screen_y = y;

                    if(screen_x >= vinfo.xres - 5) screen_x = vinfo.xres - 5;
                    if(screen_y >= vinfo.yres - 5) screen_y = vinfo.yres - 5;
                    if(screen_x<0) screen_x = 0;
                    if(screen_y<0) screen_y = 0;

                    draw_rect(screen_x,screen_x+5,screen_y,screen_y+5, 0);
            }
    }
    munmap(fbp, screensize);
    close(fbfd);
    return 0;
}
```

该程序位于 LDD6410 工程源代码的 tests/keypadmice/touchpad_test.c，譬如我们用笔在屏幕上书写“Linux”字符串，程序的运行效果如下图：



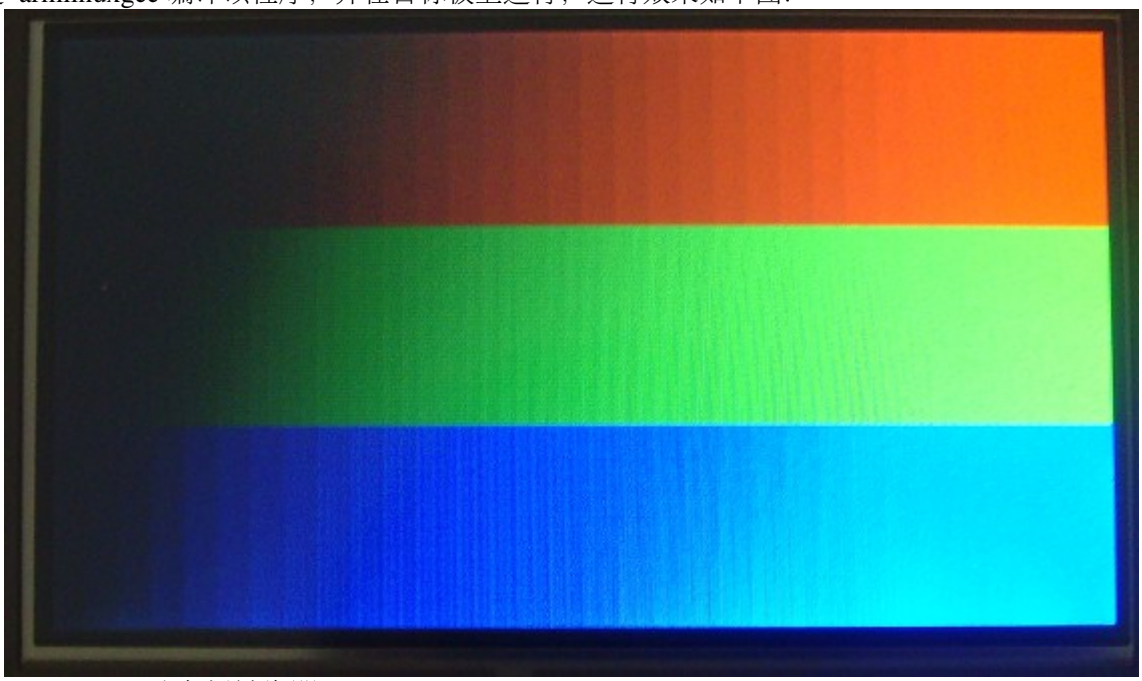
2.5.4 framebuffer 测 试 程 序

编写 framebuffer 应用程序的核心工作是打开/dev/fb0 节点，通过 ioctl()获取屏幕参数，再通过 mmap()映射显示缓冲区到用户空间，之后通过 write()写映射缓冲区。LDD6410 的 framebuffer 测试程序演示了如何在 LDD6410 的显示屏上绘制渐变的红黄蓝三色：

```
/*
 * LDD6410 framebuffer test programs
 * Copyright 2009 LiHacker Computer Technology Inc.
 */
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <fcntl.h>
#include <linux/fb.h>
#include <sys/mman.h>

int main()
{
    int fbfd = 0;
    struct fb_var_screeninfo vinfo;
    unsigned long screensize = 0;
    char *fbp = 0;
    int x = 0, y = 0;
    int i = 0;

    // Open the file for reading and writing
    fbfd = open("/dev/fb0", O_RDWR);
    if (!fbfd) {
            printf("Error: cannot open framebuffer device.\n");
            exit(1);
    }
    printf("The framebuffer device was opened successfully.\n");

    // Get variable screen information
    if (ioctl(fbfd, FBIOGET_VSCREENINFO, &vinfo)) {
            printf("Error reading variable information.\n");
            exit(1);
    }

    printf("%dx%d, %dbpp\n", vinfo.xres, vinfo.yres, vinfo.bits_per_pixel);
    if (vinfo.bits_per_pixel != 16) {
            printf("Error: not supported bits_per_pixel, it only supports 16 bit color\n");
            exit(1);
    }

    // Figure out the size of the screen in bytes
    screensize = vinfo.xres * vinfo.yres * 2;

    // Map the device to memory
    fbp = (char *)mmap(0, screensize, PROT_READ | PROT_WRITE, MAP_SHARED,
            fbfd, 0);
    if ((int)fbp == -1) {
            printf("Error: failed to map framebuffer device to memory.\n");
            exit(4);
    }
    printf("The framebuffer device was mapped to memory successfully.\n");

    // Draw 3 rect with graduated RED/GREEN/BLUE
    for (i = 0; i < 3; i++) {
            for (y = i * (vinfo.yres / 3); y < (i + 1) * (vinfo.yres / 3); y++) {
                    for (x = 0; x < vinfo.xres; x++) {
                            long location = x * 2 + y *  vinfo.xres * 2;
                            int r = 0, g = 0, b = 0;
                            unsigned short rgb;

                            if (i == 0)
```

```
                                r = ((x * 1.0) / vinfo.xres) * 32;
                        if (i == 1)
                                g = ((x * 1.0) / vinfo.xres) * 64;
                        if (i == 2)
                                b = ((x * 1.0) / vinfo.xres) * 32;

                        rgb = (r << 11) | (g << 5) | b;
                        *((unsigned short*)(fbp + location)) = rgb;
                }
        }
    }

    munmap(fbp, screensize);
    close(fbfd);
    return 0;
}
```

通过 armlinuxgcc 编译该程序，并在目标板上运行，运行效果如下图：



2.5.4 jpegview 图片浏览器

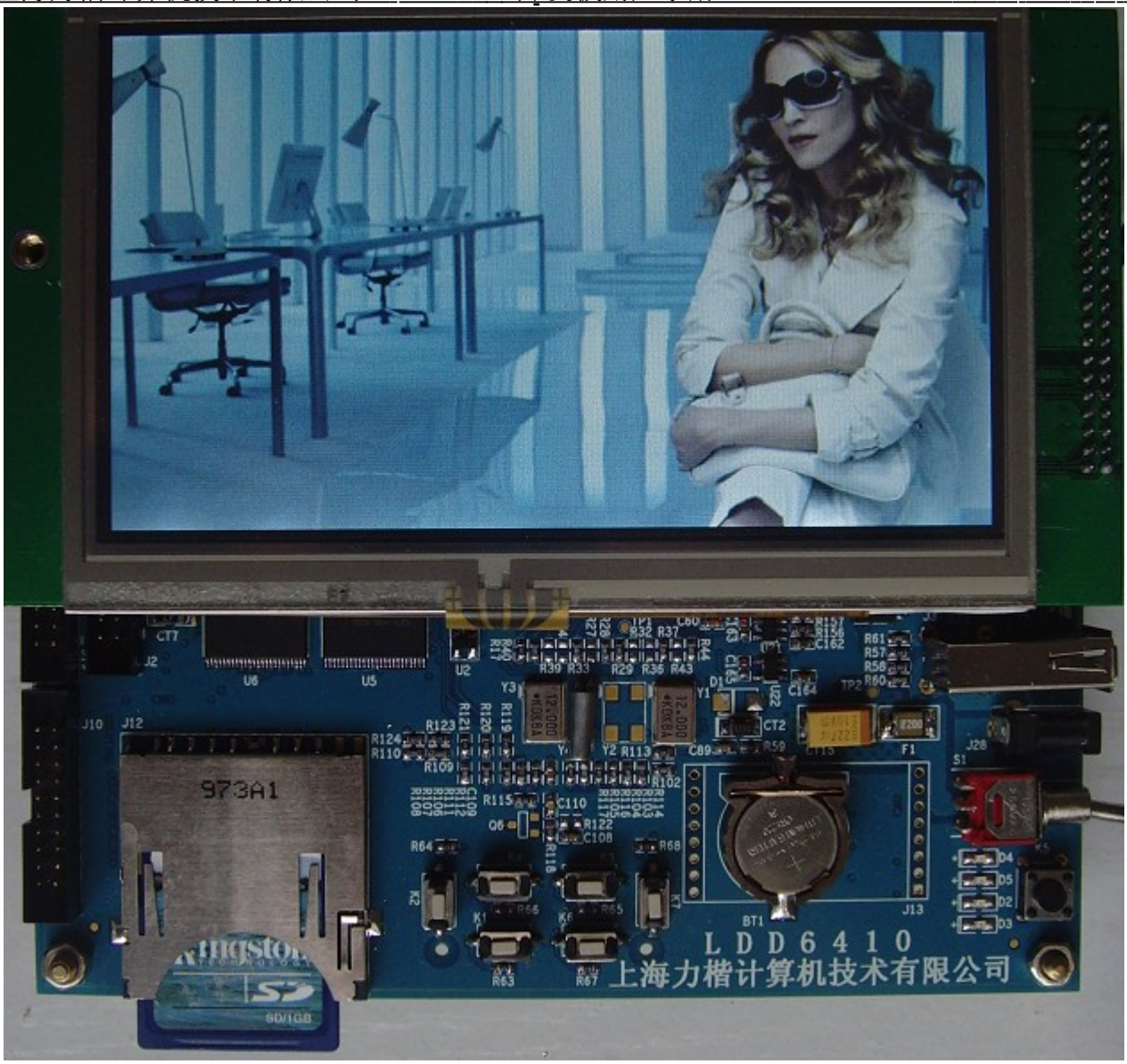
力核为 LDD6410 整合了基于文本界面的 jpegview 浏览器，其源代码位于源码目录的 utils/jpegview/。在目标板的控制台上通过运行命令即可浏览图片，该命令的格式为：

```
# jpegview
Usage: jpegview <s[Seconds]> <f|o> file_1.jpg file_2.jpg ...
s[Seconds] : Time to show this picture
f : Fixed Scale
o : NoneFixed Scale
```

例如，运行如下命令可浏览 1.jpg 2.jpg 3.jpg 4.jpg 四副 JPEG 图片：

```
# jpegview 1.jpg 2.jpg 3.jpg 4.jpg
```

运行效果如下图：



2.5.5 mplayer 媒体播放器

力核为 LDD6410 整合了基于文本界面的 mplayer 浏览器，其源代码位于源码目录的 utils/mplayer/。该命令的格式为：

```
# mplayer
MPlayer 1.0rc24.2.2 (C) 20002007 MPlayer Team
CPU: ARM
Usage:   mplayer [options] [url|path/]filename
Basic options: (complete list in the man page)


vo <drv>   select video output driver ('vo help' for a list)
ao <drv>   select audio output driver ('ao help' for a list)
ss <position>   seek to given (seconds or hh:mm:ss) position
nosound    do not play sound
```

```
fs  fullscreen playback (or vm, zoom, details in the man page)
x <x> y <y>         set display resolution (for use with vm or zoom)
sub <file> specify subtitle file to use (also see subfps, subdelay)
playlist <file> specify playlist file
vid x aid y select video (x) and audio (y) stream to play
fps x srate y  change video (x fps) and audio (y Hz) rate
pp <quality>        enable postprocessing filter (details in the man page)
framedrop enable frame dropping (for slow machines)


Basic keys: (complete list in the man page, also check input.conf)
<  or  >    seek backward/forward 10 seconds down or up          seek backward/forward  1 minute
pgdown or pgup   seek backward/forward 10 minutes
< or >       step backward/forward in playlist
p or SPACE          pause movie (press any key to continue)
q or ESC    stop playing and quit program
+ or         adjust audio delay by +/ 0.1 second
o   cycle OSD mode:  none / seekbar / seekbar + timer
* or /       increase or decrease PCM volume
x or z       adjust subtitle delay by +/ 0.1 second
r or t       adjust subtitle position up/down, also see vf expand

* * * SEE THE MAN PAGE FOR DETAILS, FURTHER (ADVANCED) OPTIONS AND KEYS *
* *
```

例如，运行如下命令可播放电影 1.avi：

```
# mplayer 1.avi
```

2.5.6 appweb 服务器

LDD6410 的 Linux 软件包集成了 appweb 服务器，在根文件系统的启动脚本中包含了 如下语句以启动 web 服务：

```
/usr/bin/appweb config /usr/share/appweb/appweb.conf &
```

根文件系统的/usr/share/appweb 目录包含了 appweb 的配置文件与脚本文件，默认的网页位于/usr/share/appweb/defaultweb 目录，在 PC 上通过浏览器访问：

```
http://192.168.1.20:4000/ldd6410.html
```

为了进行 web 服务定制，用户只需要在/usr/share/appweb/defaultweb 目录存放自己的网页文件即可。

2.6 Android

2.6.1 内核 Android 补丁

力楷为 LDD6410 整合了完整的 Android 驱动（位于 drivers/android 下的 binder、lowmemorykiller 等）、内核电源管理（位于 kernel/power 下的 wakelock、userwakelock 等）、ashmem 补丁（位于 mm/ashmem.c）和虚拟电池（drivers/power/fake_battery.c）等。

下图显示了 drivers/android 下驱动的配置：

```
File  Edit  View  Terminal  Tabs  Help
.config - Linux Kernel v2.6.28.6 Configuration
                          Android
  Arrow keys navigate the menu.  <Enter> selects submenus --->.
  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes,
  <M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </>
  for Search.  Legend: [*] built-in  [ ] excluded  <M> module  < >

      [*] Android Drivers
      [*] Android Binder IPC Driver
      <*> Android log driver
      [*] Android RAM buffer console
      [*]   Enable verbose console messages on Android RAM console
      [ ]   Android RAM Console Enable error correction  --->
      [ ]   Start Android RAM console early
      [*] Timed output class driver
      <*>   Android timed gpio driver
      [*] Android Low Memory Killer

                <Select>    < Exit >    < Help >
```

下图显示了 kernel/power 下 Android 电源管理的配置：

```
File  Edit  View  Terminal  Tabs  Help
.config - Linux Kernel v2.6.28.6 Configuration
                      Power management options
  Arrow keys navigate the menu.  <Enter> selects submenus --->.
  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes,
  <M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </>
  for Search.  Legend: [*] built-in  [ ] excluded  <M> module  < >

      [*] Power Management support
      [ ]   Power Management Debug Support
      [*] Suspend to RAM and standby
      [*] Wake lock
      [*]   Wake lock stats
      [*]   Userspace wake locks
      [*]   Early suspend
          User-space screen access (Sysfs interface)  --->
      <*> Advanced Power Management Emulation

                <Select>    < Exit >    < Help >
```

2.6.2 Android 文件系统

力楷为 LDD6410 整合了 Android 1.6，Android 本身作为 Linux 文件系统的一部分进行管理。在系统启动后，运行根目录下的 android，系统将进入 Android。

LDD6410 Android 支持按键、触摸屏和鼠标操作，显示设备可以是 LCD（480*272）和 VGA（1024*768@60HZ）。

同时在光盘中提供了 Android 1.6 SDK androidsdklinux_x861.6_r1.tgz，使用该 SDK 就可以生成支持 LDD6410 的 Android，其步骤是：

（一）在 PC 上创建 LDD6410 虚拟机

```
lihacker@lihackerlaptop:~/develop/LDD6410/androidsdklinux_x861.6_r1/tools$ ./android create avd n LDD6410 t 2
Android 1.6 is a basic Android platform.
Do you wish to create a custom hardware profile [no]y
Device ram size: The amount of physical RAM on the device, in megabytes. hw.ramSize [96]:128
Touchscreen support: Whether there is a touch screen or not on the device.
hw.touchScreen [yes]:
Trackball support: Whether there is a trackball on the device. hw.trackBall [yes]:n
Keyboard support: Whether the device has a QWERTY keyboard.
hw.keyboard [yes]:n
DPad support: Whether the device has DPad keys hw.dPad [yes]:y
GSM modem support: Whether there is a GSM modem in the device.
hw.gsmModem [yes]:n
Camera support: Whether the device has a camera. hw.camera [no]:n
Maximum horizontal camera pixels
hw.camera.maxHorizontalPixels [640]: Maximum vertical camera pixels hw.camera.maxVerticalPixels [480]:
GPS support: Whether there is a GPS in the device.
hw.gps [yes]:n
Battery support: Whether the device can run on a battery. hw.battery [yes]:n
Accelerometer: Whether there is an accelerometer in the device.
hw.accelerometer [yes]:n
Audio recording support: Whether the device can record audio hw.audioInput [yes]:
Audio playback support: Whether the device can play audio
hw.audioOutput [yes]:
SD Card support: Whether the device supports insertion/removal of virtual SD Cards. hw.sdCard [yes]:
Cache partition support: Whether we use a /cache partition on the device.
disk.cachePartition [yes]:n
Cache partition size disk.cachePartition.size [66MB]:
Abstracted LCD density: Must be one of 120, 160 or 240. A value used to roughly describe the density of the LCD screen for automatic resource/asset selection.
hw.lcd.density [160]:
Created AVD 'LDD6410' based on Android 1.6, with the following hardware config:
hw.gps=no hw.dPad=yes hw.accelerometer=no hw.lcd.density=160 disk.cachePartition=no hw.keyboard=no hw.trackBall=no hw.ramSize=128 hw.gsmModem=no hw.camera=no hw.battery=no
```

（二）在主机上创建一个 sd card 的 image

```
sudo ./mksdcard 128M sdcard.img
```

（三）在主机上启动 Android 模拟器运行如下命令启动 LDD6410 虚拟机：

```
sudo ./emulator sdcard ./sdcard.img @LDD6410
```

启动 adb shell 连接 LDD6410 虚拟机，查看模拟器目标机上文件系统的挂载情况：

```
# mount
rootfs / rootfs ro 0 0
tmpfs /dev tmpfs rw,mode=755 0 0
devpts /dev/pts devpts rw,mode=600 0 0
proc /proc proc rw 0 0
sysfs /sys sysfs rw 0 0
tmpfs /sqlite_stmt_journals tmpfs rw,size=4096k 0 0
/dev/block/mtdblock0 /system yaffs2 ro 0 0
/dev/block/mtdblock1 /data yaffs2 rw,nosuid,nodev 0 0
/dev/block//vold/179:0 /sdcard vfat
rw,dirsync,nosuid,nodev,noexec,uid=1000,gid=1015,fmask=0702,dmask=0702,allow_utime=0020,c
odepage=cp437,iocharset=iso88591,shortname=mixed,utf8 0 0
```

（四） 提取 Android 1.6

把 busybox 放入模拟器目标机文件系统中：

```
./adb push ~/develop/svn/ldd6410/utils/busybox1.15.1/_install/bin/busybox /data
```

我们现在把/system、/data、/sbin 目录以及根目录下的 init、init.rc 等都放入 sdcard 的 image 中：

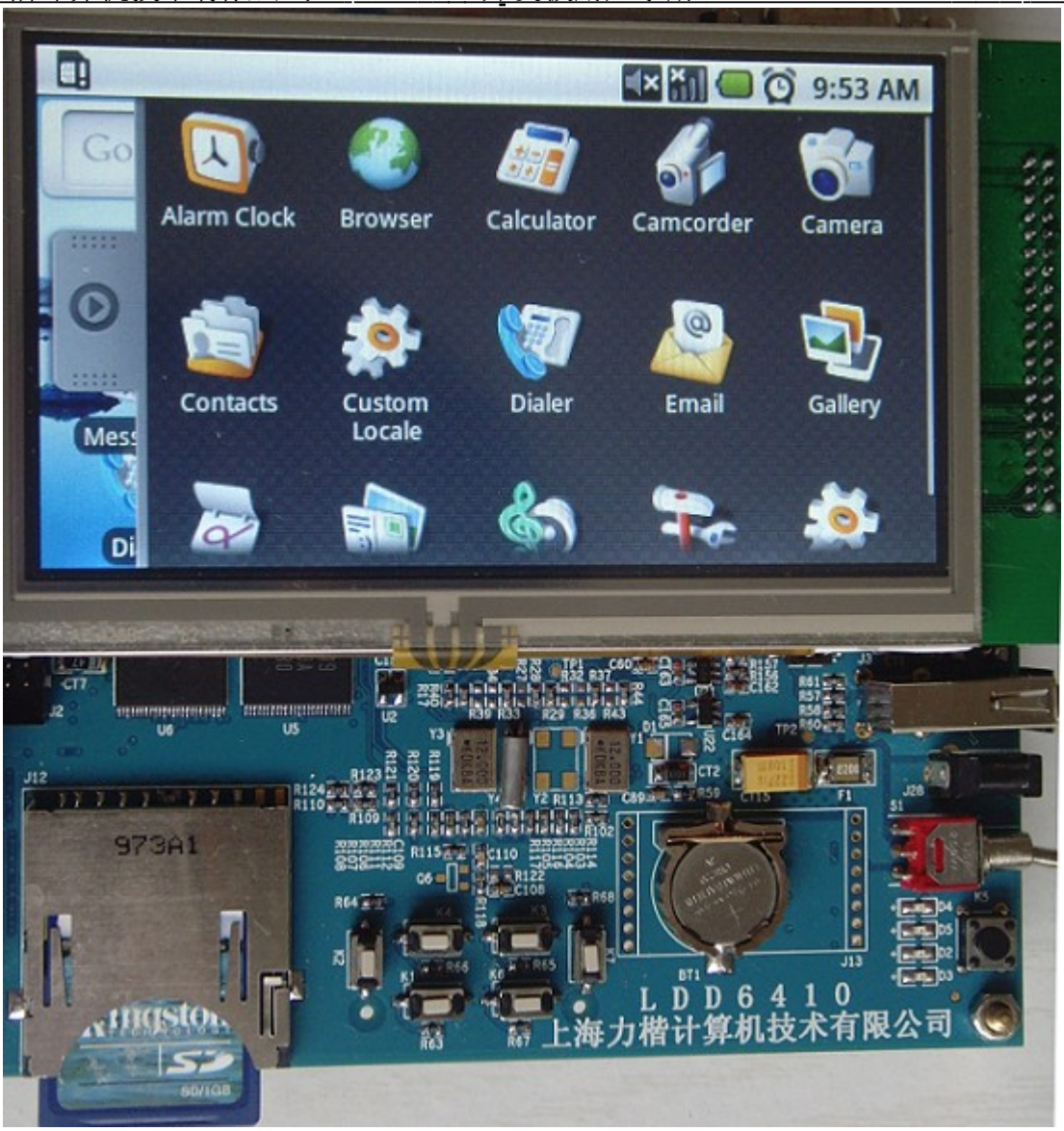
```
# /data/busybox tar cvf /sdcard/android.tar /data /system /sbin /sqlite_stmt_journals /init.rc
/init.goldfish.rc /init
```

进入/sdcard 目录看看得到的压缩文件：

```
# cd /sdcard
# ls l
rwxrx system   sdcard_rw 78487552 20100130 12:37 android.tar
```

在主机上以 loop 方式 mount sdcard 的 image，并将里面的文件放到 LDD6410 目标电路板的根文件系统。

对于 LDD6410 而言，运行根目录下的/android 程序就可以直接启动 android。



2.7 Qt Embedded

力楷为 LDD6410 开发板整合了 Tslib 和 Qt Embedded 4.5.3。前者用于触摸屏坐标的 calibration，后者依赖于前者才能工作。

2.7.1 Tslib 和 ts_calibration

Tslib 的源代码位于 LDD6410 源代码工程的 libs/tslib 目录。其编译方法为：

第一步，运行：

```
./configure host=armlinux target=armlinux prefix=/home/lihacker
```

第二步，运行 make，发现有如下 link 错误：

```
libtool: link: armlinuxgcc DGCC_HASCLASSVISIBILITY O2 Wall W o .libs/ts_test ts_test.o fbutils.o font_8x8.o font_8x16.o ../src/.libs/libts.so ldl
ts_test.o: In function `main':
ts_test.c:(.text+0x1e4): undefined reference to `rpl_malloc' fbutils.o: In function `open_framebuffer':
fbutils.c:(.text+0x588): undefined reference to `rpl_malloc' collect2: ld returned 1 exit status
```

解决方法是注释掉 config.h 中的“#define malloc rpl_malloc”一行，编译即可通过。

第二步，运行 make install，在/home/lihacker 下面得到了如下目录：

```
.

| bin
|  | ts_calibrate
|  | ts_harvest
|  | ts_print
```

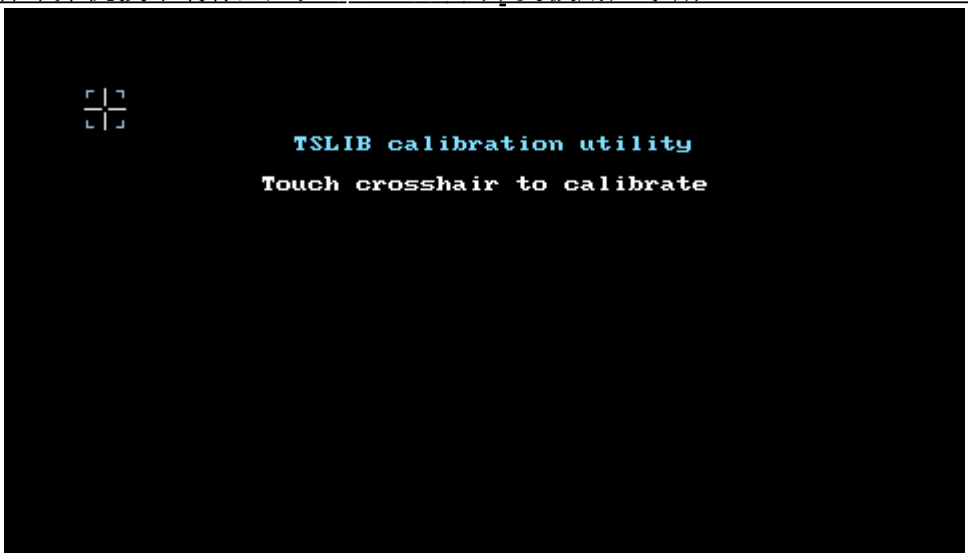
```
|  |ts_print_raw
|  `ts_test
|etc
|  `ts.conf
|include
|  `tslib.h
`lib
|libts0.0.so.0 > libts0.0.so.0.1.1
|libts0.0.so.0.1.1
|libts.la
|libts.so > libts0.0.so.0.1.1
|pkgconfig
|  `tslib0.0.pc
`ts
|arctic2.la
|arctic2.so
|collie.la
|collie.so
|corgi.la
|corgi.so
|dejitter.la
|dejitter.so
|h3600.la
|h3600.so
|input.la
|input.so
|linear.la
|linear.so
|linear_h2200.la
|linear_h2200.so
|mk712.la
|mk712.so
|pthres.la
|pthres.so
|ucb1x00.la
|ucb1x00.so
|variance.la
`variance.so
```

将 bin 下面的 ts_calibration 等可执行程序拷入 LDD6410 根文件系统的/usr/bin 目录，将 lib 下 面的所有文件和目录拷入 LDD6410 根文件系统的/lib 目录。

在运行 ts_calibration 之前我们需要编写 LDD6410 根文件系统中的 touchscreen 配置文件/etc/ts.conf，内容如下：

```
module_raw input module pthres pmin=1
module variance delta=30 module dejitter delta=100 module linear
```

从下节 QT 的启动脚本可以看出，运行 ts_calibration 之前需 export 一系列环境变量如 TSLIB_TSDEVICE、TSLIB_CONFFILE、TSLIB_PLUGINDIR、TSLIB_CALIBFILE 等。在 LDD6410 上，ts_calibration 的运行效果如下图：



2.7.2 Qt

解压缩 qtembedded1inuxopensourcesrc4.5.3 源代码，并配置：

```
./configure \
noqt3support \
qtzlib \
qtlibtiff \
qtlibpng \
qtlibmng \
qtlibjpeg \
make libs \
nomake examples \
nomake demos \
nomake docs \
nonis \
norpath \
nocups \
noxmlpatterns \
nophonon \
nosvg \
nolargefile \
noiconv \
nowebkit \
noopenssl \
noscripttools \
nommx \
no3dnow \
nosse \
nosse2 \
nofeatureQWS_CURSOR \
xplatform qws/linuxarmg++ \
embedded arm \
littleendian \
qtfreetype \
depths 16 \
qtgfxlinuxfb \
nogfxtransformed \
nogfxqvfb \
nogfxvnc \
nogfxmultiscreen \
qtkbdusb \
qtkbdtty \
qtmousepc \
noglib \
noopengl \
qtmousetslib I/home/lihacker/tslib_build/include L/home/lihacker/tslib_build/lib
```

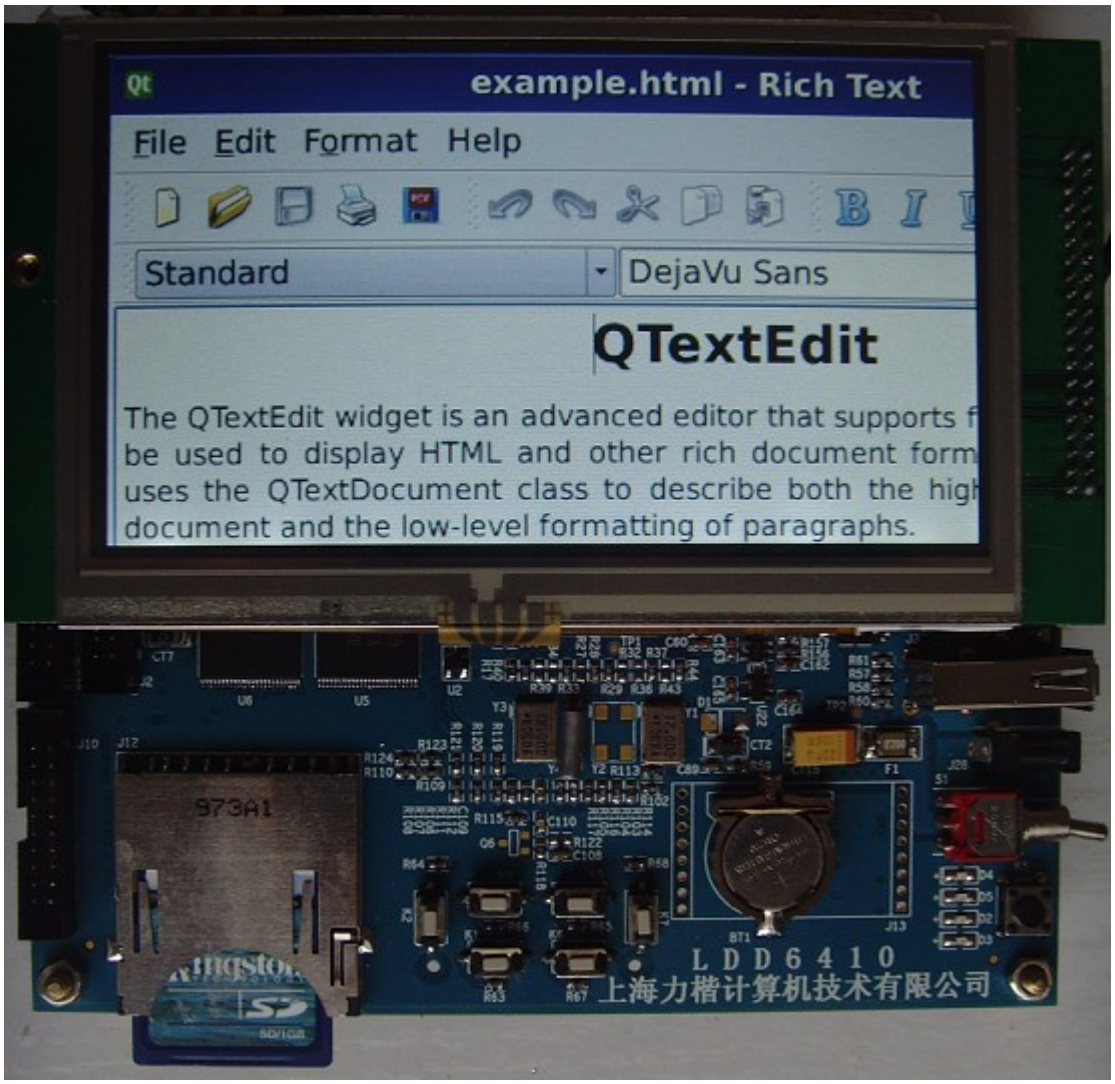
运行 make 和 make install 得到/usr/local/Trolltech/QtEmbedded4.5.3arm 目录，将该目录放入目标机文件系统。如 LDD6410 开发板将编译出来的 lib 和 plugins 拷入了/opt/qt 目录，而将 QT 实例程序

basicdrawing、framebuffer、menus 和 textedit 等拷入了/opt/qtopia/bin 目录。编译 Qt 实例程序的方法是进行对应实例的源代码目录，先运行/usr/local/Trolltech/QtEmbedded4.5.3arm 中的“qmake –project”，再运行“qmake”，最后运行 make。

在目标机根文件系统的根目录下建立如下脚本文件 qtopia：

```
#!/bin/sh
export set TSLIB_TSDEVICE=/dev/event1 export set TSLIB_CONFFILE=/etc/ts.conf export set
TSLIB_PLUGINDIR=/lib/ts
export set TSLIB_CALIBFILE=/etc/pointercal
export set HOME=/root export set QTDIR=/opt/qt
export set QPEDIR=/opt/qtopia export set KDEDIR=/opt/qt
#export QWS_DISPLAY="LinuxFB:/dev/fb0"
export QWS_DISPLAY="LinuxFB:mmWidth115:mmHeight65:0"
export QWS_SIZE=480x272
#export set QWS_KEYBOARD="USB:/dev/event2"
export set QWS_KEYBOARD="None"
export set QWS_MOUSE_PROTO="Tslib:/dev/event1"
export set QT_PLUGIN_PATH=$QTDIR/plugins/ export set QT_QWS_FONTDIR=$QTDIR/lib/fonts/
export set PATH=$QPEDIR/bin:$PATH
export set LD_LIBRARY_PATH=$QTDIR/lib:$QPEDIR/plugins/imageformats:$LD_LIBRARY_P
[ f /etc/pointercal ] || ts_calibrate
$QPEDIR/bin/textedit qws
```

运行这个脚本，会看到 textedit 的界面：



3. LDD6410 Linux 软件包的烧录与更新

3.1 SD 卡烧录

第一步，在安装了 Linux 的 PC 机上通过 fdisk 给一张空的 SD 卡分为 2 个区（如果 SD 卡中本身已经包含 ，请通过 fdisk 的“d”命令全部删除），得到如下的分区表：

```
Command (m for help): p

Disk /dev/sdb: 1030 MB, 1030225920 bytes
32 heads, 62 sectors/track, 1014 cylinders
Units = cylinders of 1984 * 512 = 1015808 bytes
Disk identifier: 0x6f20736b

   Device Boot      Start         End      Blocks   Id  System
/dev/sdb1   *           1          20       19809   83  Linux
/dev/sdb2              21        1014      986048   83  Linux
```

注意第 1 个分区制作的命令为：

```
Command (m for help): n
Command action
   e   extended
   p   primary partition (1-4)
p
Partition number (1-4): 1
First cylinder (1-1014, default 1):
Using default value 1
Last cylinder, +cylinders or +size{K,M,G} (1-1014, default 1014): 20M
```

第 2 个分区制作的命令是：

```
Command (m for help): n
Command action
   e   extended
   p   primary partition (1-4)
p
Partition number (1-4): 2
First cylinder (21-1014, default 21):
Using default value 21
Last cylinder, +cylinders or +size{K,M,G} (21-1014, default 1014):
Using default value 1014

Command (m for help):
```

我们还要通过“a”命令标记第 1 个分区：

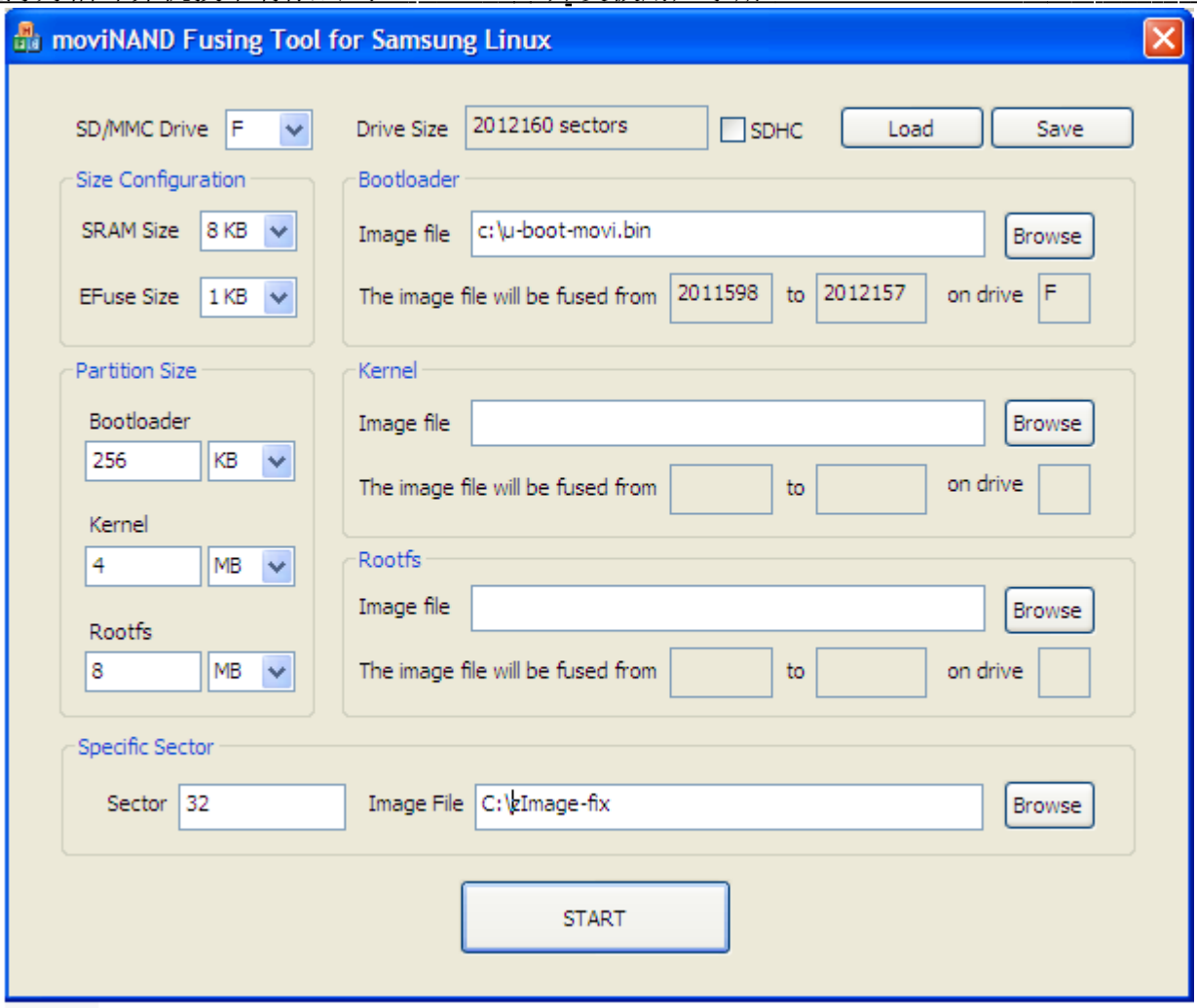
```
Command (m for help): a
Partition number (14): 1
```

最后要通过“w”命令把建好的分区表写入 SD 卡。

第二步，格式化 SD 卡的分区 1 和分区 2：

```
mkfs.vfat /dev/sdb1 mkfs.ext3 /dev/sdb2 fsck.ext3 /dev/sdb2
```

第三步，通过 moviNAND_Fusing_Tool.exe 烧写 SD 卡 UBOOT 和 zImage：



更新 SD 卡根文件系统的方法很简单，在 PC 机器上 mount /dev/sdb2 后，直接通过

```
cp fa <yourrootfs> <sdb2mountpoint>
```

的方式就可以替换根文件系统了。<yourrootfs>是你的根文件系统的目录，<sdb2mountpoint>是你的/dev/sdb2 挂载的目录。

特别要注意的是，SD 的设备节点不一定是/dev/sdb，应该视用户电脑的硬盘情况而 言，可能是/dev/sdc，/dev/sdd 等。

3.2 NAND 烧录

3.2.1 更新 NAND 版 UBOOT

通过 tftp 或 nfs 等方式获取新的 UBOOT，如：

```
# tftp r ubootmovi.bin g 192.168.1.111
```

运行：

```
# flashcp ubootmovi.bin /dev/mtd0
```

若/dev/mtd0 结点不存在，请先运行：

```
# mdev s
```

3.2.2 更新 NAND 分区中的 Linux 内核

通过 tftp 或 nfs 等方式获取新的 zImage，如：

```
# tftp r zImagefix g 192.168.1.111
```

运行：

```
# flashcp zImagefix /dev/mtd1
```

若/dev/mtd1 结点不存在，请先运行：

```
# mdev s
```

3.2.3 更新 NAND 分区中的文件系统

在 PC 上将做好的新的根文件系统拷贝入 SD 卡或 NFS 的某目录，下面我们以<new_rootfs_dir>指代该目录。

以 SD 卡或 NFS 为根文件系统启动系统，运行如下命令擦除/dev/mtd2 分区：

```
# flash_eraseall /dev/mtd2
```

然后将 NAND 的该分区 mount 到/mnt：

```
# mount /dev/mtdblock2 t yaffs2 /mnt/
```

将新的文件系统拷贝到/mnt：

```
# cp fa <new_rootfs_dir> /mnt
```

若/dev/mtd2 或/dev/mtdblock2 不存在，请先运行：

```
# mdev s
```