Rico Board
Linux 开发手册

版本 V1.0

2015 年 2 月 5 日

版本记录

版本号	说明	时间
V1.0	初始版本	2015/2/5

目 录

第 1 章 概述及软件资源介绍

1.1 概述

Rico Board 提供了丰富的系统资源和软件资源，本手册将从环境搭建开始，一步步介绍如何进行 Rico Board Linux 开发。本手册中开发主机上的命令以 Ubuntu 为例进行介绍。

1.2 软件资源

类别	名称	备注	源码
引导程序	MLO（SPL）	一阶引导	yes
	U-boot	二阶引导	yes
Linux 内核	Linux 3.12.0	专为 Rico Board 硬件制定的 Linux 内核	yes
设备驱动	USB Host	USB Host 驱动	yes
	USB Device	USB Device 驱动	yes
	I2C	i2c-dev 驱动	yes
	Ethernet	千兆以太网驱动	yes
	MMC	MMC/SD/TF 卡驱动	yes
	eMMC	eMMC 驱动	yes
	LCD	DSS 驱动，支持 7 寸液晶屏	yes
	RTC	实时时钟驱动	yes
	HDMI	SIL9022A 驱动	yes
	Touch	电容触摸屏驱动	yes
	Button	按键驱动	yes
	UART	串口驱动	yes
	LED	LED 驱动	yes
	GPIO	GPIO 驱动	yes

类别	名称	备注	源码
	WDI	看门狗驱动	yes
	Camera	双摄像头驱动	yes
	QSPI	QSPI Flash 驱动	yes
	EERPOM	EERPOM 驱动	yes
文件系统	Ramdisk	带系统更新工具的轻量级文件系统	bin
	Matrix	TI 官方提供的 Matrix 演示系统	bin
	Buildroot	带 Qt 4.8.5 库的文件系统	bin
应用程序	KEY&LED	按键指示灯测试程序	yes
	NET	TCP/IP Sokect C/S 测试程序	yes
	RTC	实时时钟测试程序	yes
	I2C	i2c-dev 应用接口演示程序	yes
	EEPROM	EEPROM 应用接口演示程序	yes
	Framebuffer	显示设备演示程序	yes
	Dual Camera	双摄像头演示程序	yes
工具	Cross compiler	Linaro GCC 4.7	bin

表 1-1

第 2 章 Linux 开发环境搭建

2.1 建立工作目录

拷贝产品光盘中的源码到 Linux 开发主机中：

```
$ mkdir -p <WORKDIR>
$ cp -a <DVDROM>/04-Linux_Source/* <WORKDIR>
```

2.2 设置交叉编译工具

```
$ cd <WORKDIR>/Toolchain
$ tar -xvjf \
gcc-linaro-arm-linux-gnueabihf-4.7-2013.03-20130313_linux.tar.bz2
$ export PATH=$PATH:<WORKDIR>/Toolchain/\
gcc-linaro-arm-linux-gnueabihf-4.7-2013.03-20130313_linux/bin
$ export CROSS_COMPILE=arm-linux-gnueabihf-
```

执行完“export”命令后输入 arm 按 Tab 键来检查是否设置成功，该设置只对当前终端有效，如需永久修改，请修改用户配置文件。

2.3 安装工具

此外还需安装一些必要工具，以 ubutnu 系统为例：

```
$ sudo apt-get install build-essential git-core libncurses5-dev
$ sudo apt-get install flex bison texinfo zip unzip zlib1g-dev gettext
$ sudo apt-get install gperf libsdl-dev libesd0-dev libwxgtk2.6-dev
$ sudo apt-get install uboot-mkimage
$ sudo apt-get install g++ xz-utils
```

第 3 章 Linux 系统编译

3.1 编译 Bootloader

进入 Bootloader 目录，解压 U-boot 源码：

```
$ cd <WORKDIR>/Bootloader
$ tar -xvjf u-boot-2013.10-ti2013.12.01.tar.bz2
$ cd u-boot-2013.10-ti2013.12.01
```

开始编译：

```
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- distclean
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- <config>
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf-
```

编译完成后会在当前目录生成所需要的 MLO（只有 myir_ricoboard_config 配置才会生成）、u-boot.bin 和 u-boot.img。

这里的<config>是配置选项名称，不同的启动模式需使用不同的配置选项。

有如下两种情况：

启动模式	编译选项	输出文件
Q-SPI Flash	myir_ricoboard_qspiboot_config	u-boot.bin
Micro SD	myir_ricoboard_config	u-boot.img 和 MLO

表 3-1

3.2 编译 Linux 内核

进入 Kernel 目录，解压内核源码：

```
$ cd <WORKDIR>/Kernel
$ tar -xvjf linux-3.12.10-ti2013.12.01.tar.bz2
$ cd linux-3.12.10-ti2013.12.01
```

开始编译：

```
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- distclean
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- \
myir_ricoboard_defconfig
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- zImage dtbs
```

编译完成后在 arch/arm/boot 目录会生成内核镜像 zImage，在 arch/arm/boot/dts 目录

会生成设备树的二进制文件 myir_ricoboard.dtb。

第 4 章 Linux 系统烧写

Rico Board 开发板目前提供了 3 种引导加载方案：

- 方案 1 ：TF 卡启动（Ramdisk 文件系统）
- 方案 2 ：TF 卡启动（EXT4 文件系统）
- 方案 3 ：QSPI Flash 启动（挂载 eMMC 的 EXT4 文件系统）

其中方案 3 为出厂默认设置，下面将分别对这几种方式的机制和烧写方法进行说明。

4.1 TF 卡启动（Ramdisk 文件系统）

⑴ TF 卡格式化

请使用产品光盘 Tools 目录下的 HP USB Disk Storage Format Tool 2.0.6 工具来格式化 TF 卡。

① 把 TF 卡通过 USB 读卡器，接入到 Windows 主机中；

② 打开 HP USB Disk Storage Format Tool，出现类似提示如下：

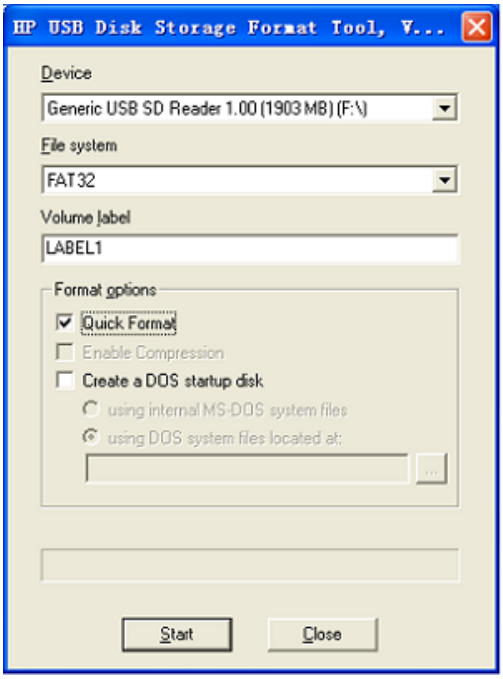

图 4-1

③ 选择“FAT32”系统格式

④ 点击“Start”

⑤ 等待格式化完成，点击“OK”

注意：HP USB Disk Storage Format Tool 会将清除 TF 卡上的所有分区和数据，格式化之前请做好数据备份。

⑵ 将产品光盘 02-Images 目录内的文件拷贝到 TF 卡的根目录，包含如下文件：

```
├──── matrix-rootfs.tar.gz              # matrix 文件系统
├──── MLO                               # 一级引导（SPL）
├──── myir_ricoboard.dtb                # 设备树文件
├──── myir-rootfs.tar.gz                # 带 Qt 库的 Buildroot 文件系统
├──── ramdisk.gz                        # Ramdisk 文件系统
├──── u-boot.bin                        # QSPI Bootloader 文件
├──── u-boot.img                        # MMC 启动 u-boot 镜像
├──── uEnv                              # 环境变量
│     ├──── uEnv_Ext4_LCD7inch.txt      # 挂载 EXT4 文件系统，LCD 显示
│     ├──── uEnv_Ext4.txt               # 挂载 EXT4 文件系统，HDMI 显示
│     ├──── uEnv_Ramdisk_LCD7inch.txt   # 挂载 Ramdisk 文件系统，LCD 显示
│     └──── uEnv_Ramdisk.txt            # 挂载 Ramdisk 文件系统，HDMI 显示
├──── uEnv.txt                          # 默认环境变量（Ramdisk，LCD）
└──── zImage                            # 内核镜像
```

⑶ 其中 MLO、u-boot.img、zImage、myir_ricoboard.dtb、ramdisk.gz 本方案必须的文件，rootfs.tar.gz、u-boot.bin 是方案 3 更新需要的文件。

⑷ 将开发板引导模式跳线设置为 MMC 模式，插入 TF 卡，给开发板上电，即可从 TF 卡加载进入系统。

4.2 TF 卡启动（EXT4 文件系统）

该启动方案结构为，Bootloader 和 Linux 内核存放在 TF 的第一个分区，根文件系统在 TF 卡的第二个分区，第一个分区为 FAT 格式，第二个分区为 EXT4 格式。制作该方案的 TF 卡的操作在 Linux 主机中完成，步骤如下：

```
$ cd <WORKDIR>/Tools
$ tar -xvjf mkmmc.tar.bz2
$ cd mkmmc
```

将文件系统打包或重命名为 rootfs.tar.gz，并拷贝到 mkmmc 的 rootfs 目录。

将编译的或光盘中的，MLO、u-boot.img、zImage、myir_ricoboard.dtb、uEnv.txt 拷贝到 mkmmc 的 boot/目录。

其中 uEnv.txt 可以使用 02-Images/uEnv 的 uEnv_Ext4.txt 或 uEnv_Ext4_LCD7inch.txt 重命名。

方案 2 的环境变量文件 uEnv.txt，与方案 1 中的不同，因为一个是挂载 FAT 分区中的 Ramdisk 文件系统，一个是挂载 EXT4 分区中的根文件系统。

执行脚本开始制作 TF 卡：

```
$ sudo ./mkmmc-linux.sh <device>
```

其中<device>为系统识别到的 TF 卡设备节点，如：/dev/sdb 等。请确保所选的设备节点确实为新插入 Linux 主机的 TF 卡设备节点，否则有可能导致 Linux 主机系统被破坏！

脚本制作完成之后，将开发板引导模式跳线设置为 MMC 模式，插入 TF 卡，给开发板上电，即可从 TF 卡加载进入系统，将加载的是 Ext4 分区中的根文件系统。

4.3 默认方案 QSPI Flash 启动（EXT4 文件系统）

该方案结构为，QSPI Flash 上只包含 Bootloader，内核和设备树文件在 eMMC 的第 1 个分区上，EXT4 文件系统将从 eMMC 的第二个分区挂载。

eMMC 一共包含 3 个分区，第一和第三分区为 FAT 格式，第二分区为 EXT4 格式。

这里烧写 QSPI Flash 需要借助方案 1 中制作的 Ramdisk 系统来完成，烧写过程如下：

⑴ 按照 4.1 节中介绍的步骤，制作带有 Ramdisk 的 TF 卡；

⑵ 将需要烧写的目标文件系统打包或重命名为 rootfs.tar.gz：

```
$ cd <WORKDIR>/Filesystem/<Your Rootfs>
$ tar -zcvf ../rootfs.tar.gz <Your Rootfs>
```

或将光盘中提供的文件系统重命名

```
$ cp matrix-rootfs.tar.gz rootfs.tar.gz
```

⑶ 将 rootfs.tar.gz 拷贝到 TF 卡的 FAT 分区中；

⑷ 跳线设置为 MMC 启动模式，使用 TF 卡进入 Ramdisk 系统；

⑸ 使用 Ramdisk 系统中的工具烧写 QSPI Flash：

```
# cd /root
# ./updatesys.sh
```

⑹ 烧写完成后，关闭电源，拔出 TF 卡，将跳线设置为 QSPI 启动模式，开发板上电后将从 QSPI Flash 启动，挂载 eMMC 第 2 分区中的 EXT4 文件系统。

4.4 uEnv.txt

uEnv.txt 是使用 MMC 启动时，U-boot 环境变量配置文件。U-boot 会读取 MMC 第一个 FAT 分区内的 uEnv.txt 文件来配置环境变量，为方便用户使用，光盘目录 02-Images/uEnv 下提供了几个常用的 uEnv 文件可供参考，将对应的 uEnv 文件复制到 SD 卡第一个分区并重命名为 uEnv.txt，便可使用对应的配置。这些文件主要配置了显示模式和所用的文件系统：

文件名	配置为
uEnv_Ext4.txt	挂载 SD 卡第二个分区的 Ext4 文件系统 使用 HDMI 显示
uEnv_Ext4_LCD7inch.txt	挂载 SD 卡第二个分区的 Ext4 文件系统 使用 7 寸 LCD 显示
uEnv_Ramdisk.txt	挂载 Ramdisk 文件系统 使用 HDMI 显示
uEnv_Ramdisk_LCD7inch.txt	挂载 Ramdisk 文件系统 使用 7 寸 LCD 显示

表 4-1

第 5 章 Linux 应用程序开发

5.1 嵌入式开发环境

嵌入式开发，通常先在 PC 上交叉编译后，再通过某种方式（如以太网或串口），将文件复制或传输到目标开发板上执行。为了方便调试，进行应用程序开发前，应该先配置好一个文件共享环境。

5.1.1 配置 TFTP 服务

TFTP 简单文件传输协议，可以实现不同主机间的文件发送和接收，另外，U-boot 启动时，还可以使用 TFTP 加载内核。

安装 tftp 服务端，以 ubuntu 为例：

```
$ sudo apt-get install tftp-hpa tftpd-hpa
```

创建 tftp 服务器工作目录

```
$ mkdir <WORKDIR>/tftpboot
$ chmod 777 <WORKDIR>/tftpboot
```

配置 tftp 服务器：

```
$ sudo vi /etc/default/tftpd-hpa
```

修改或添加如下字段：

```
TFTP_DIRECTORY="<WORKDIR>/tftpboot"
TFTP_OPTIONS="-l -c -s"
```

重启 TFTP 服务：

```
$ sudo service tftpd-hpa restart
```

假设 PC 机的 IP 地址为：192.168.1.111，运行以下命令可以将 PC 主机 tftpboot 目录中的文件传输到开发板：

```
# tftp -l <file-name> -r <file-name> -g 192.168.1.111 69
```

在开发板的控制终端运行下面的命令可以将开发板上的文件发送到 PC 主机的 tftpboot 目录：

```
# tftp -l <file-name> -r <file-name> -p 192.168.1.111 69
```

在 U-boot 命令行，可以使用以下方式加载 PC 机 tftpboot 目录中的 zImage 内核文件：

```
# setenv serverip 192.168.1.111
```

```
# setenv ipaddr 192.168.1.222
# setenv ethaddr 00:01:02:03:04:05
# tftp zImage <mem-addr>
# bootz <mem-addr>
```

5.1.2 配置 NFS 服务

NFS 即网络文件系统，允许主机直接通过网络实现文件共享。另外，除了挂载普通目录外，还可以在 Linux 启动时将 NFS 上的目录挂载为开发板根文件系统。

在开发主机上安装 NFS 服务程序，以 ubuntu 为例：

```
$ sudo apt-get install nfs-kernel-server
```

编辑 exports 文件，添加 nfs 文件夹目录：

```
$ sudo vi /etc/exports
```

例如设置/home/myir/nfs 为 NFS 目录，将如下内容添加到 exports 文件中：

```
/home/myir/nfs *(rw,subtree_check,no_root_squash,no_all_squash,sync)
```

设置目录权限：

```
$ chmod 777 -R /home/myir/nfs
```

修改设置 nfs，绑定端口

```
$ sudo vi /etc/default/nfs-kernel-server
```

将 RPCMOUNTDOPTS 修改为：

```
#RPCMOUNTDOPTS=--manage-gids
RPCMOUNTDOPTS="-p 13100"
```

重启 NFS 服务：

```
$ sudo service nfs-kernel-server restart
```

在本机上测试 NFS 服务：

```
$ sudo mount -t nfs 127.0.0.1:/home/myir/nfs /mnt
```

若本机 NFS 挂载成功，接下来在开发板上挂载 NFS，将开发板和主机接入同一网络，设置主机服务端 IP，例如：

```
$ sudo ifconfig eth0 192.168.1.111
```

设置开发板 IP，例如：

```
# ifconfig eth0 192.168.1.222
```

使用 ping 命令测试开发板与 PC 机网络是否连通：

```
# ping 192.168.1.111
```

在开发板上新建挂载目录：

```
# mkdir -p /mnt/nfs
```

挂载 NFS 文件夹：

```
# mount -t nfs -o nolock,rw 192.168.1.111:/home/myir/nfs /mnt/nfs
```

至此，NFS 共享文件夹挂载成功，通过 NFS 共享文件夹可以方便的实现主机和开发板间的文件共享，并且可以直接在共享文件夹/mnt/nfs 下运行目标程序，而免去了文件拷贝的操作。另外，除了挂载目录外，还可以在启动时将 NFS 上的目录挂载为开发板根文件系统，详细步骤可参考以下网页：

http://wiki.emacinc.com/wiki/Booting_with_an_NFS_Root_Filesystem

5.1.3 SSH 登录

使用 SSH 连接可以在通过网络，像串口终端一样，远程操作 Linux 命令行系统。

系统启动的时候已经默认启动 dropbear ssh 服务，只要使用 SSH 协议的软件，输入 IP 进行连接即可，如下图所示：

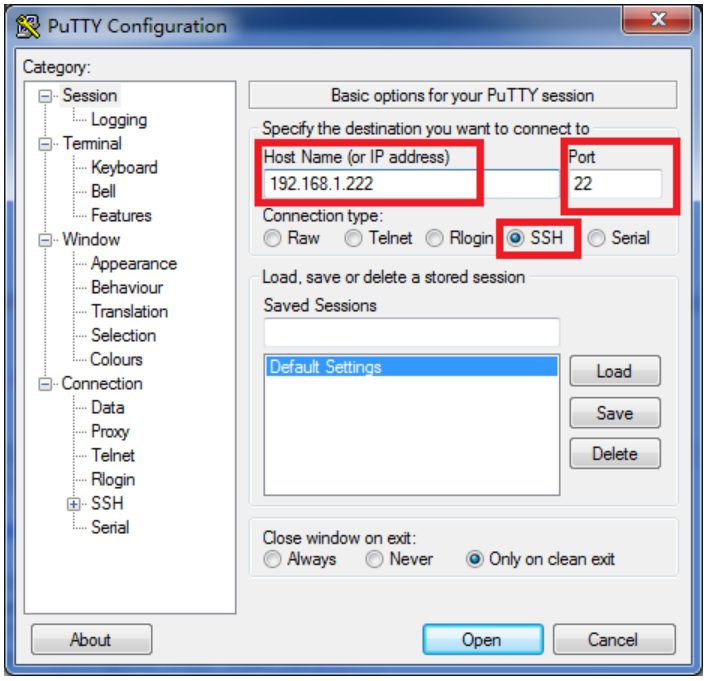

图 5-1

在指定位置输入开发板 IP，端口号，然后连接，如果弹出安全提示对话框点击 YES 忽略。在命令行界面输入登录和密码，默认状态登录名为 root，密码为空，但某些文件系统密码不能为空，需要先在开发板上设置新密码才能进行 SSH 登录。

5.2 应用实例

Rico Board 提供了常用外设的演示程序，程序以及源码都位于*“<WORKDIR>/Examples/”*，请根据目录内的 Makefile 或 README 文件进行编译：

```
$ cd <WORKDIR>/Examples/<app dir>
$ export CROSS_COMPILE=arm-linux-gnueabihf-
$ make
```

在开发板上运行程序时需要注意权限，如果无法执行请使用如下命令：

```
# chmod +x <program-to-be-execute>
```

5.2.1 GPIO

本例介绍如何使用 Linux 命令行操作 GPIO。

这里以 GPIO5_9 为例，对应 Rico Board 开发板扩展接口 J11 的 Pin-12。

GPIO5_9 对应的 Linux GPIO 编号为 GPIO169，即 5×32+9=169。

导出 GPIO 169：

```
# echo 169 > /sys/class/gpio/export
```

查看新导出的 GPIO 端口：

```
# ls /sys/class/gpio/gpio169
```

设置 GPIO 方向为输出：

```
# echo "out" > /sys/class/gpio/gpio169/direction
```

使 GPIO 口输出高电平：

```
# echo 1 > /sys/class/gpio/gpio169/value
```

使 GPIO 口输出低电平：

```
# echo 0 > /sys/class/gpio/gpio169/value
```

释放 GPIO 口：

```
# echo 169 > /sys/class/gpio/unexport
```

5.2.2 Key&LED

本例程演示如何使用 Linux API 操作开发板上的 USER 按键与 LED，详情请参考源码。

将目录*“<WORKDIR>/Examples/key_led”*中的可执行程序 key_led 拷贝至开发板，执行程序，程序将点亮 LED0～LED3，按下开发板上的 SW3、SW4 按键将打印出相关的

按键信息。

```
# ./key_led
status led0 on
status led0 off
status led1 on
status led1 off
status led2 on
status led2 off
status led3 on
status led3 off
Hit any key on board ......
key 102 Released
key 102 Pressed
key 158 Released
key 158 Pressed
```

5.2.3 NET

本例使用 TCP/IP sokect API，实现一个简单的 C/S 程序，详情请参考源码。将目录 *“<WORKDIR>/Examples/network”* 中的可执行程序 arm_client 拷贝至开发板，pc_server 拷贝至 PC，将开发板和 PC 接入网络，设置 IP：

```
$ sudo ifconfig eth0 192.168.1.111
# ifconfig eth0 192.168.1.222
```

在 PC 上运行服务程序：

```
$ ./pc_server
```

再在开发板上运行客户程序。将看到所发送的信息：

```
# ./arm_client 192.168.1.111
form server: Make Your idea Real!
```

同时可以看到 PC 端看到连接的客户端 IP：

```
$ ./pc_server
REC FROM:  192.168.1.222
```

5.2.4 RTC

本例程演示如何使用 Linux API 对开发板上的 RTC 进行时间设置与读取，详情请参考源码。

将目录 *“<WORKDIR>/Examples/rtc”* 中的可执行程序 rtc_test 拷贝至开发板。执行程

序设置当前时间，例如：

```
# ./rtc_test -s 11 03 40 2014 08 14
      date/time is updated to:  14-8-2014, 11:03:40.
```

然后读取时间，观察时间变化：

```
# ./rtc_test
  Current RTC date/time is 14-8-2014, 11:03:50.
# ./rtc_test
  Current RTC date/time is 14-8-2014, 11:03:51.
```

5.2.5 I2C

这里使用的是 i2c-dev 驱动的 ioctl，发送多开始信号控制信息，对 16 位地址的 EEPROM 进行读写操作（代码中已包含禁用 EEPROM 写保护的操作），详情请参考源码。

将目录 *“<WORKDIR>/Examples/i2c”* 中的可执行程序 i2c_test 拷贝至开发板，执行程序，将向 EEPROM 写入三个字节，然后读回：

```
# ./i2c_test
Read back:0x55
Read back:0x66
Read back:0x77
```

5.2.6 EEPROM

本例程演示如何使用 EEPROM 驱动，对开发板上的 EERPOM 进行读写操作，详情请参考源码。

禁用 EEPROM 写保护：

```
# echo 103 > /sys/class/gpio/export"
# echo "out" > /sys/class/gpio/gpio103/direction"
# echo 0 > /sys/class/gpio/gpio103/value"
```

注：假如 gpio103 已经被 export，则以上第一条 export 命令将会失败，但不影响程序运行，继续往下操作即可。

执行程序：

```
# ./eeprom_test
eeprom size: 32 KB

write 'eeprom write/read test!' to eeprom
```

```
get the following string from eeprom:
eeprom write/read test!
```

5.2.7 FrameBuffer

本例演示对 Linux 的 FrameBuffer 操作，实现颜色和彩色栅格测试。

拷贝<WORKDIR>\Examples\framebuffer 目录下的测试程序开发板，如果使用的是 matrix 图形系统，使用下面的命令关闭 matrix 界面：

```
# /etc/init.d/matrix-gui-2.0 stop
```

执行程序：

```
# ./framebuffer_test
```

终端将显示屏幕信息，显示设备会先后出现 8 种背景色，然后显示颜色栅栏，以下是 HDMI 输出信息：

```
The framebuffer device was opened successfully.
vinfo.xres=1024
vinfo.yres=768
vinfo.bits_per_bits=32
vinfo.xoffset=0
vinfo.yoffset=0
finfo.line_length=4096
The framebuffer device was mapped to memory successfully.
......
```

使用下面的命令可重启 matrix 界面：

```
# /etc/init.d/matrix-gui-2.0 start
```

5.2.8 USB Device

本例通过开发板的 USB mini 口连接电脑，使用开发板上的存储空间虚拟为一个 U 盘，的 USB Device 应用。

eMMC 在烧写系统的时候已经被分为三个分区，这里以使用 eMMC 的第三个分区（FAT32 格式）作用为 USB 存储设备为例说明 USB Device 的使用。

⑴ 使用 USB mini 线一端接电脑，一端接开发板的 USB mini 接口（J7）；

⑵ 加载 g_mass_storage 驱动：

```
# modprobe g_mass_storage stall=0 file=/dev/mmcblk0p3 removable=1
```

之后 PC 主机上将弹出一个 U 盘的盘符，可对该盘进行文件操作。

5.2.9 led_play

本例演示如何在用户空间读取按键以及如何控制用户 led 灯。

拷贝<WORKDIR>\Examples\led_play 目录下的测试程序 led_play 到开发板，在开发板的控制终端上执行程序：

```
# ./led_play
```

长按按键 SW3 三秒即可触发，触发后三个 LED 灯：D24，D25 和 D26 开始闪烁。详细操作过程请见源码。

注：led_play 已嵌入到 matrix 图形系统并设置为自启动，如果使用该文件系统则可跳过拷贝和执行步骤，直接长按按键 SW3 三秒即可触发。如需取消 led_play 的自启动，删除 matrix 文件系统下的/etc/rc5.d/S99led_play 文件即可。

第 6 章 Qt 开发

本小节描述在 Rico Board 上使用 Qt 进行 GUI 程序开发的方法和步骤，包括两大部分，第一部分讲述光盘中提供的 Qt SDK 的使用方法，一般的 Qt 程序开发使用光盘中提供的 Qt SDK 即可；第二部分讲述如何从 Qt 源码中编译生成 Qt 开发环境，当光盘中提供的 Qt 库不能满足 Qt 开发程序需求时才需要自己制定 Qt 开发环境。

6.1 使用光盘提供的 Qt SDK

⑴ 解压编译好的 tslib 到 PC：

```
$ sudo tar xvjf \
/media/cdrom/05-Linux_Source/Qt_Arm/tslib-prebuild.tar.bz2 -C /opt
```

⑵ 解压 Qt SDK 到 PC：

```
$ sudo tar xvjf \
/media/cdrom/05-Linux_Source/Qt_Arm/qt-4.8.5-sdk.tar.bz2 -C /opt
```

⑶ 配置环境变量和交叉编译 Qt 应用程序

配置环境变量和交叉编译 Qt 应用程序请参考：6.4 交叉编译 Qt 应用程序。

6.2 交叉编译 Qt 开发环境

⑴ 参考本文之前章节安装交叉编译工具，设置环境变量。

⑵ 安装 automake、libtool、autoconf 包，编译安装 tslib：

```
$ sudo apt-get install automake libtool autoconf
$ cd  <WORKDIR>
$ cp  /media/cdrom/05-Linux_Souce/Qt_Arm/tslib-1.4.tar.bz2  ./
$ tar  -jxf  tslib-1.4.tar.bz2
$ cd  tslib
$ ./ts-build
$ make install
```

注意：若执行 ts-build 出错时，编辑 ts-build 修改其中的交叉编译工具路径。

编译完成后，tslib 将被安装到*/usr/local/tslib* 目录，此时需要将此目录下 *tslib/etc/ts.conf* 文件第二行 *“#module_raw input ”* 的注释去掉，变为 *“module_raw input ”*，注意一定要顶格，如图 6-1 所示：

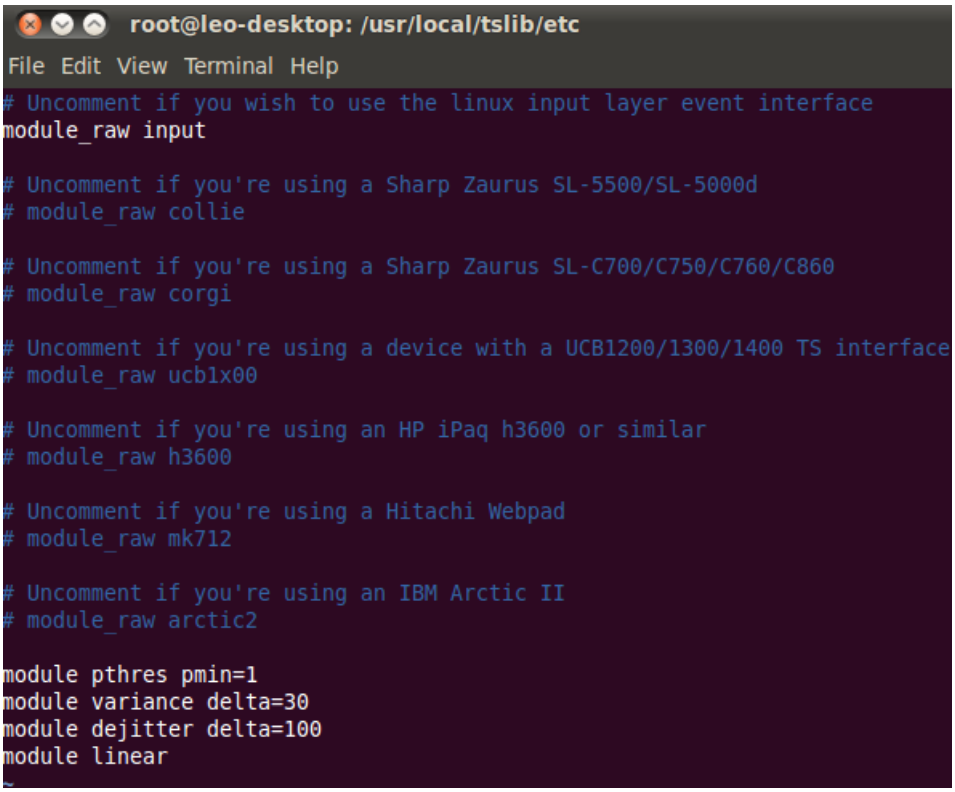

图 6-1

⑶ 编译 qt

拷贝 Qt 源码到工作目录：

```
$ cd <WORKDIR>
$ cp  \
/media/cdrom/05-Linux_Source/Qt_Arm/qt-everywhere-opensource-src-4.8.5
.tar.bz2\  ./
```

安装必要的工具包：

```
$ sudo apt-get install xorg-dev libfontconfig1-dev
libfreetype6-dev libx11-dev libxcursor-dev libxext-dev
libxfixes-dev libxft-dev libxi-dev libxrandr-dev libxrender-dev
```

开始编译：

```
$ tar  -jxf  qt-everywhere-opensource-src-4.8.5.tar.bz2
$ cd  qt-everywhere-opensource-src-4.8.5
$ ./qt-build
$ sudo make install
```

编译完成后，qt-4.8.5 将被安装到/opt/qt-4.8.5 目录。

6.3 移植 Qt 到开发板

⑴ 把安装到/usr/local 中的 tslib 目录下的文件拷贝到用于编译开发板文件系统的/usr/local/目录下：

```
$ cp -r /usr/local/tslib/bin <WORKDIR>/Filesystem/rootfs/usr/local/tslib
$ cp -r /usr/local/tslib/etc <WORKDIR>/Filesystem/rootfs/usr/local/tslib
$ cp -r /usr/local/tslib/lib <WORKDIR>/Filesystem/rootfs usr/local/tslib
```

⑵ 把安装到/opt 中的 qt-4.8.5 目录文件拷贝到用于编译开发板文件系统的 rootfs 文件夹的 opt/目录下：

```
$ cp -r /opt/qt-4.8.5 <WORKDIR>/Filesystem/rootfs/opt/
```

⑶ 进入<WORKDIR>/Filesystem/rootfs/etc/init.d/目录，编写脚本 qt.sh，如下：

```
export TSLIB_CONSOLEDEVICE=none
export TSLIB_FBDEVICE=/dev/fb0
export TSLIB_TSDEVICE=/dev/input/touchscreen0

export
LD_LIBRARY_PATH=/lib:/usr/lib:/usr/local/tslib/lib:/opt/qt-4.8.5/lib
export QT_QWS_FONTDIR=/opt/qt-4.8.5/lib/fonts
export QWS_USB_KEYBOARD=/dev/input/event2
export PATH=/bin:/sbin:/usr/bin/:/usr/sbin:/usr/local/tslib/bin

if grep "display1" /proc/cmdline > /dev/null ; then
   export                QWS_MOUSE_PROTO="Tslib:/dev/input/touchscreen0
MouseMan:/dev/input/mice"
else
   export QWS_MOUSE_PROTO="Tslib:/dev/input/touchscreen0"
fi

export QWS_DISPLAY=:1
```

修改<WORKDIR>/Filesystem/rootfs/etc/init.d/目录下的 rcS 文件，在该文件最后加上：

```
if [ -e /etc/init.d/qt.sh ];then
   /etc/init.d/qt.sh
fi
```

⑷ 然后重新制作烧写包含 Qt 库和 tslib 的文件系统。

6.4 交叉编译 Qt 应用程序

⑴ 配置 PC 机的 Qt 编译环境：

```
$ export QT_PREFIX=/opt/qt-4.8.5
$ export PATH=${QT_PREFIX}/bin:$PATH
```

```
$ export QMAKESPEC=${QT_PREFIX}/mkspecs/qws/linux-arm-g++
```

⑵ 创建代码：

```
$ mkdir hellomyir
$ cd hellomyir
$ gedit hellomyir.cpp
```

输入如下代码：

```
#include <QApplication>
#include <QLabel>
int main(int argc, char **argv)
{
    QApplication app(argc,argv);
    QLabel label("Make Your idea Real!");
    label.show();
    return app.exec();
}
```

⑶ 编译：

```
$ qmake -project
$ qmake
$ make
```

⑷将生成的 hellomyir 拷贝到开发板，并在开发板上执行：

```
# ./hellomyir
```

将编译生成的可执行文件拷贝到开发板中运行，将会在 LCD 屏幕上看到 *“Make Your idea Real!”* 的 Qt 窗口。