SkyEye模拟器的LCD及TouchScreen模拟模块的设计、实现与验证*)

尹文超[1] 陈 渝[2] 康 烁[1] 杨 晔[3] 曾 益[4]

(清华大学软件学院 北京100084)[1] (清华大学计算机系 北京100084)[2]

(英特尔上海软件实验室 上海200233)[3] (复旦大学计算机系 上海200433)[4]

摘 要 本文首先描述了开放源码的嵌入式硬件仿真环境SkyEye的总体架构,然后对SkyEye模拟器的LCD及TouchScreen模拟模块的设计、实现与验证过程进行了深入的阐述,并且分析比较了LCD模拟的不同实现方式。开源嵌入式GUI系统—MiniGUI和Linux操作系统在SkyEye模拟器上的成功运行说明了SkyEye模拟器的LCD及TouchScreen模拟模块的设计实现是正确和可靠的。

关键词 嵌入式系统,指令级模拟器,SkyEye,FrameBuffer,MiniGUI

Design, Implementation and Verification of SkyEye Simulator's LCD and TouchScreen Simulation Modules

YIN Wen-Chao[1] CHEN Yu[2] KANG Shuo[1] YANG Ye[3] ZENG Yi[4]

(School of Software, TsingHua University, Beijing 100084)[1]

(Department of Computer Science and Technology, TsingHua University, Beijing 100084)[2] (Intel Shanghai Software Lab, Shanghai 200233)[3]

(Department of Computer Science and Technology, Fudan University, Shanghai 200433)[4]

Abstract This paper first discribes the framework of SkyEye, which is an open source simulator for embedded system. And then gives detailed ideals of the design, implementation and verification process of SkyEye simulator's LCD and TouchScreen modoles. MiniGUI, an open source embedded GUI system, and Linux Operating System could run successfully on SkyEye simulator, which proves that the design and implementation of SkyEye simulator's LCD and TouchScreen modoles is correct and reliable.

Keywords Embedded system, Instruction-level simulator, SkyEye, FrameBuffer, MiniGUI

1 引言

SkyEye是一个开源软件项目,中文名字是"天目"。SkyEye的目标是在通用的Linux和Windows平台实现一个纯软件集成开发环境,模拟常见的嵌入式计算机系统;可在SkyEye上运行μCLinux以及μC/OS-II等多种嵌入式操作系统和各种系统软件(如TCP/IP,图形子系统,文件子系统等),并可对它们进行源码级的分析和测试。目前SkyEye已经在嵌入式系统教学和嵌入式系统开发中得到很好的应用。

SkyEye是一个指令级模拟器[1],可以模拟多种嵌入式开发板,可支持多种CPU指令集,在SkyEye上运行的操作系统意识不到它是在一个虚拟的环境中运行,而且开发人员可以通过SkyEye调试操作系统和系统软件。由于SkyEye的目标不是验证硬件逻辑,而是协助开发、调试和学习嵌入式系统软件,所以在实现上SkyEye与真实的硬件环境相比还是有一定差别的。SkyEye在时钟节拍的时序上不保证与硬件完全相同,对软件透明的一些硬件仿真进行了一定的简化,这样带来的好处是SkyEye的执行效率更高。

SkyEye从总体上分为四个层次:

用户接口模块:包括命令行用户界面和图形用户界面,完成处理用户的输入命令,并把相关调试数据输出给用户的任务。这一部分基本上直接利用了GDB的用户接口模块,并在此基础上有一定的扩充。

符号处理模块:主要处理执行文件的头信息,解释执行文件中内嵌的调试信息,对符号表的管理,对源代码表达式的解析,定位源代码中的语句位置和机器码的位置关系等。这一部分也是直接利用了GDB的符号处理模块,这也是有了这个模块支持,SkyEye可以支持源码级调试。

目标控制模块:主要完成执行控制(如中断程序的执行,设置中断条件等),程序栈结构分析,对具体目标硬件的控制(如本地调试、远程调试和模拟调试的控制)。这一部分完成对SkyEye上运行的软件的控制,提供了多种调试手段。

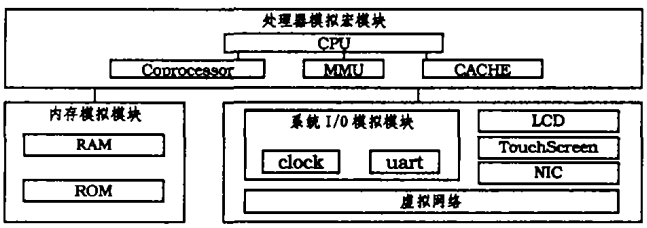

图1 SkyEye硬件模拟逻辑结构图

目标模拟模块:这一部分是SkyEye的核心。它的功能是模仿计算机系统中的主要硬件(包括CPU、内存、I/O寄存器和LCD,TouchScreen等各种硬件外设)的执行,对执行文件的机器指令进行解释,并模拟执行每一条机器指令,产生相应

*)本文受国家863项目基金(2003AA1Z2090)和2003自然科学基金(60203024)资助。尹文超 硕士研究生,研究方向为嵌入式系统。陈 渝 博士,讲师,研究方向为嵌入式系统、嵌入式操作系统、高性能并行通信、并行编译。

的硬件响应等。

SkyEye 起源于 GDB/ARMulator，并进行了全面的改进和扩展。SkyEye 建立在 GNU GDB 的底层，可以模仿多种完整的嵌入式计算机系统，目前模拟的硬件包括 CPU、内存、I/O 寄存器、时钟、UART、网络芯片、MMU、CACHE 等。

LCD 和 TouchScreen 已经成为嵌入式系统普遍使用的人机交互设备，在 SkyEye 模拟器中加入 LCD 和 TouchScreen 模拟模块，一方面可以使 SkyEye 完整地模拟带有 LCD 和 TouchScreen 的开发板，例如基于 Cirrus Logic EP7312 CPU 的嵌入式开发板，基于 Intel PXA250/255 CPU 的 Lubbock 开发板等；另一方面可以提供一个嵌入式 GUI 系统的研究开发与测试的平台，这对于嵌入式系统的学习研究无疑是一个积极的促进和推动。

2 LCD 模拟模块的设计、实现与验证

2.1 LCD 模拟模块的设计与实现

LCD 模拟模块的设计思路是，使用 GTK+图形系统在 X Window 系统和 Win32 系统上实现一个 LCD 屏幕模拟，在 SkyEye 上运行的嵌入式操作系统中的 LCD 驱动程序象驱动真正的 LCD 控制器一样发送控制命令或对 LCD 显示内存进行访问操作，而 SkyEye 解释这些控制命令，并根据这些命令对 LCD 屏幕窗口进行相应的 GTK+图形操作，完成对不同灰度或颜色图形的绘制。

在 SkyEye 模拟器中，如果嵌入式操作系统要执行 I/O 地址访问，具体的处理过程由特定 CPU 和开发板 I/O 模拟模块中的 read/write_byte/halfword/word 函数处理。所以 LCD 模拟模块关注的主要是内存模拟模块模拟出来的 LCD 显示内存中存储的数据。

LCD 的显示内存映射到内存 RAM 中，代表了要在 LCD 屏幕上显示的图像。显示内存必须足够大，以处理显示屏幕上所有的像素。应用程序通过直接或间接地存取显示内存中的数据来进行图形操作，改变屏幕显示的内容。

LCD 模拟模块对 GTK+的使用目前仅限于根据分辨率（例如320×240，640×480）创建相应大小的窗口以及根据显示内存中的数据逐点在该窗口进行绘制，因为画点是 LCD 屏幕最基本的动作，所有其它的相对复杂工作如图形绘制，嵌入式 GUI 系统的实现都应该由基于 LCD 驱动程序的应用程序（包括基于 FrameBuffer 驱动程序的嵌入式 GUI 系统，例如 MiniGUI）通过对 LCD 显示内存的读写操作来实现，SkyEye "看到"的只是显存中对应于屏幕上各个点的像素值，而不关心这些像素值组成的是什么样的图像。

下图就是 SkyEye 模拟器的 LCD 模拟的流程图（包括与真实硬件的比较）。

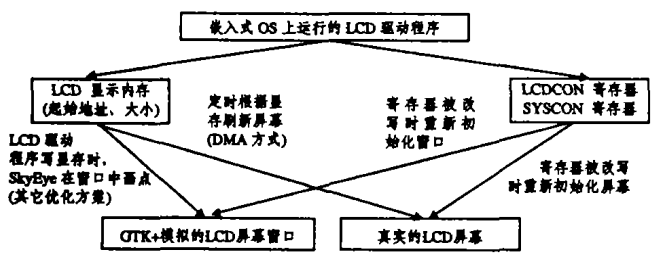

图2 SkyEye 模拟器的 LCD 模拟流程图

LCD 模拟模块的实现先后采用了两种方案，在第一种方案中，在 SkyEye 的内存模拟模块中，在每一次的写内存操作之后判断其地址是否属于 LCD 显示内存的地址范围，如果在该范围之内则调用 LCD 模拟模块中的 GTK+画点函数 gdk_draw_point()，根据由像素值查找彩色查找表 CLUT 得到的 RGB 值（对于真彩色，颜色深度为16，24，32时，RGB 值可以直接由像素值得到），在模拟屏幕窗口的相应位置画一个相应灰度或颜色的点。

该方案的优点在于实现起来简单，且模拟了真实的 LCD 最基本的画点动作，对于图像随时间流逝而只有小范围变化的情况具有一定的优势，因为对显存有写操作时才有画点操作，但是也有两个方面的缺点，其一，与 SkyEye 模拟器的内存模拟模块耦合紧密，破坏了模块间的独立性；其二，对于图像随时间流逝而大范围变化的情况，本方案效率低下，在 LCD 驱动程序连续的每两次写显存操作中，都要经历一个单位延迟时间，其长度等于一次地址范围的判断，一次 CLUT 查找及一次 GTK+画点函数的调用所耗费的时间，对于一次全屏操作，以320×240×8为例，若以字节为单位写显存，则额外的时间延迟将320×240×8/8=76800倍于单位延迟时间。

而第二种方案则直接定时（时间间隔可调，例如设置成200ms）调用 GTK+的绘图函数 gdk_draw_rgb_image()将显存中的数据一次性绘制到窗口中。该方案模拟了 DMA 的定时扫描方式，与真实的 DMA 方式不同的是，在真实的硬件上，DMA 方式无须 CPU 参与，可与 CPU 并行工作，而用软件模拟的硬件无法做到这一点，只能串行地定时扫描显示内存，其时间延迟不可避免地比真实硬件大。

第二方案降低了 LCD 模拟模块与内存模拟模块之间的耦合度，其缺点是不能实时地反映显存的快速变化。如果将定时间隔设置得过大，则增大了窗口内容刷新时的闪烁；如果定时间隔设置得过小，定时扫描过于频繁地发生，对系统资源是一种浪费。

2.2 LCD 模拟模块的验证

2.2.1 *采用普通的 LCD 驱动程序* 为了验证 LCD 模拟模块的正确性和有效性，作者首先采用了一个可以在真实的硬件上运行的普通 LCD 驱动程序，编译进 armlinux 内核，并基于该驱动程序编写了一个 demo，这个 demo 首先打开 LCD 设备，再打开一幅320×240×8的非压缩的 BMP 文件，然后将位图中的数据按字节写入显存。交叉编译该 demo，连同位图一起放入文件系统镜像文件。结果是内核启动过程中装载 LCD 驱动程序模块时，在 X Window 上生成一个大小为320×240的窗口，内核启动完成后，在嵌入式操作系统的控制台提示符下运行该 demo 程序，则该位图在 GTK+实现的 LCD 屏幕模拟窗口上正确显示。

2.2.2 *定制 FrameBuffer 驱动程序* FrameBuffer 是出现在 2.2.x 以上内核的版本当中的一种驱动程序接口。这种接口将显示设备抽象为帧缓冲区。用户可以将它看成是显示内存的一个映像，将其映射到进程地址空间之后，就可以直接进行读写操作，而写操作可以立即反映在屏幕上。

考虑到目前已有成熟的基于 FrameBuffer 的嵌入式 GUI 系统（例如，MiniGUI 等），如果有能在 SkyEye 运行的 FrameBuffer 驱动程序，那么就可以在 SkyEye 上移植嵌入式 GUI 系统。

在 linux 源代码中与 FrameBuffer 驱动程序相关的文件有两种，一种是与具体显示设备无关的文件，向应用程序提供抽象的文件操作接口，另一种是与具体显示设备相关的文件，完成底层硬件的初始化和启动，如果要让 FrameBuffer 支持

一种新的显示设备，就需要在 Linux 源代码的 driver/video/ 下添加针对这种设备的文件。

SkyEye 模拟器的 LCD 模拟模块首先是在模拟 Cirrus Logic EP7312开发板时实现的，而 Linux 的源代码中只有支持 EP7212的 LCD 设备的 FrameBuffer 驱动程序。作者对该驱动程序进行了修改，以支持 EP7312的 LCD 设备。

验证的结果是，基于 FrameBuffer 的应用程序正常运行，为 MiniGUI 的移植奠定了基础。本文的第4节将详细介绍 MiniGUI 在 SkyEye 上的移植。

2.2.3 LCD 模拟模块的两种实现方案的性能比较及分析 在 SkyEye 模拟器中，为了模拟外围设备的执行，在解释执行 ARM 指令过程的每一个周期[1]，会执行一个 io_do_cycle 函数，它会调用特定 CPU 和开发板的 I/O 模拟模块中的 *_io_do_cycle 函数(例如 ep7312_io_do_cycle)，完成对时钟、网络输入输出或 UART 输入输出等的处理。我们在 *_io_do_cycle 中调用 LCD 模拟模块中的 lcd_cycle 函数：

```
void lcd_cycle(ARMul_State * state)
{
  if(!skyeye_config.no_lcd)
    gtk_main_iteration_do(FALSE);
}
```

该函数完成 GTK 主事件循环的一次迭代，检查 GTK 窗口是否有事件需要处理，没有则直接返回。其运行时间对 LCD 模拟模块的效率有重要影响。

在 SkyEye 中可以通过设定 io.tc_prescale 的值来调整 io_do_cycle 的调用的频率，当 io.tc_prescale=50时，每解释执行50条 ARM 指令，就调用一次 io_do_cycle。

为了对比两种 LCD 模拟模块的实现方案的性能，我们编写了一个测试程序，使用 memset，按字，或按字节等三种刷屏方式。表1，表2，表3分别记录和比较了两种方案的总时间及调用 lcd_cycle 的时间。

表1 io.tc_prescale=50和100时两种方案的运行时间比较(单位：秒)

	io.tc_prescale=50			io.tc_prescale=100		
	第一方案	第二方案	比较	第一方案	第二方案	比较
memset	0.226648	0.060940	3.7倍	0.159694	0.051899	3.1倍
按字	1.297727	0.318033	4.1倍	0.788647	0.233865	3.4倍
按字节	13.011391	3.063382	4.2倍	7.564231	2.219035	3.4倍

表2 io.tc_prescale=50时两种方案中 lcd_cycle 函数调用的时间比较(单位：秒)

	第一方案(画点方案)			第二方案(定时扫描方案)		
	时间	次数	平均时间	时间	次数	平均时间
memset	0.110252	874	0.000126	0.010362	874	0.000012
按字	0.880891	6802	0.000130	0.087492	6802	0.000013
按字节	8.681378	60453	0.000144	0.801475	60453	0.000013

表3 io.tc_prescale=100时两种方案中 lcd_cycle 函数调用的时间比较(单位：秒)

	第一方案(画点方案)			第二方案(定时扫描方案)		
	时间	次数	平均时间	时间	次数	平均时间
memset	0.057857	442	0.000131	0.005435	442	0.000012
按字	0.461943	3427	0.000135	0.041628	3427	0.000012
按字节	4.561015	30496	0.000150	0.419021	30496	0.000014

为了分析这些数据，我们令刷屏的总时间为 T，测试程序完成计算和写显存的时间为累计 Tw，LCD 模拟模块完成窗口内容绘制的时间累计为 Td，lcd_cycle 调用的累计时间为 Tc，它们满足关系 $T=Tw+Td+Tc$。对应于表中数据，可以看出，对于画点方案，Tc 在 T 中占决定性的比重，且随着 lcd_cycle 调用次数的增加，其所占比重也在增加。而对于定时扫描方案，Tc 的值只相当于画点方案中的 Tc 的十分之一，对 T 的影响不显著。

在定时扫描方案中，只有当到了定时间隔才会对 LCD 屏幕窗口有一次绘图操作，而在画点方案中，每一次写显存的操作都对应于一次 LCD 屏幕窗口内容的改变，导致在每次 lcd_cycle 调用中浪费更多的时间，这一时间若经过数万次累加则相当可观，例如表2中画点方案的 Tc 的值就已经到达了8.681378秒。

在这两种方案中导致 Tc 如此大的差异的根本原因是，在画点方案中，用 gdk_draw_point ( )函数复制图像数据到服务器中是非常慢的，因为每一个点都要求一个服务器请求(也许还不止一个，因为还需要为每个点更改其 GC(图形设备上下文))。而在定时扫描方案中，GdkRGB 用一个称为 GdkImage 的对象在一个请求内将图像数据复制到服务器。虽然这样还是有点慢(大量的数据需要复制)，但是 GdkRGB 是已经对此进行了优化，并且如果客户和服务器碰巧是在同一台机器上，它还会使用共享内存。所以在 X Window 结构中，这是完成这个任务最快的方法[6]。

3 TouchScreen 模拟模块的设计、实现与验证

3.1 TouchScreen 模拟模块的设计与实现

TouchScreen 模拟模块的设计思路，将与 LCD 模拟窗口同样大小的 GTK+组件置于 LCD 组件容器中，并为该组件注册鼠标键按下，释放及移动三种事件，当鼠标在组件窗口有键按下，释放或移动的动作，则在相应的事件回调函数中记录其在窗口上的坐标及键的状态，并产生修改中断寄存器中的相应位置1，在 SkyEye 上运行的嵌入式 OS 检测到中断寄存器的数据变化就产生中断，TouchScreen 驱动程序中注册了该中断的中断服务程序 ISR 则复制所记录的数据供应用程序使用，这一思路简单说来就是，完成 GTK+的鼠标事件到 TouchScreen 事件的映射。

因此 TouchScreen 模拟模块只需要关注 GTK+鼠标事件的发生，记录事件数据并在 *_io_do_cycle 函数中对 I/O 模拟模块所模拟的中断状态寄存器进行置数操作，即为嵌入式操作系统内核产生中断信号的条件。

图3就是 SkyEye 模拟器的 TouchScreen 模拟的流程图(包括与真实硬件的比较)。

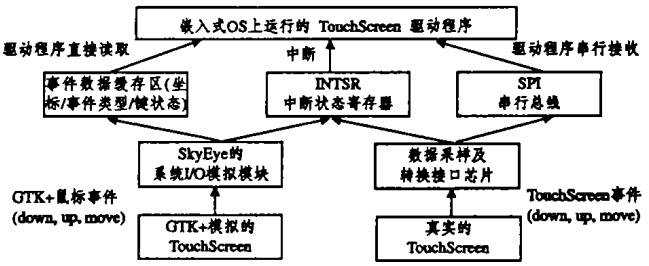

图3 SkyEye 模拟器的 TouchScreen 模拟流程图

TouchScreen 模拟模块的实现采用了与模拟采用 DragonBall 开发板的 Xcopilot 模拟器相类似的简化方式。在实际

的 TouchScreen 硬件中，为了定位动作发生的坐标，要先经过一个12位的 A/D 转换器分别转换 X，Y 坐标对应12位数字量，然后由驱动程序通过 SPI 串行总线串行接收。SkyEye 作为一个指令级的模拟器，无需保证与真实时钟节拍在时序上的一致，因此允许对 TouchScreen 这样的外设的模拟进行简化。

3.2 TouchScreen 模拟模块的验证

为了驱动简化的 TouchScreen 虚拟设备，其驱动程序相对于真实硬件的驱动程序也要进行简化。只需保证向应用程序提供的接口相同即可。因此作者定制了这样的 TouchScreen 驱动程序。并基于该驱动程序编写了简单的 demo，以验证 TouchScreen 模拟模块的有效性。

验证的运行结果是，当鼠标在 LCD+TouchScreen 的窗口上有键按下，释放或移动这样的动作发生时，该应用程序能够获得正确的坐标及事件类型。

4 移植嵌入式 GUI 系统-MiniGUI 到 SkyEye 模拟器

LCD 和 TouchScreen 模拟模块的实现，创建了嵌入式 GUI 系统在模拟器上运行所必需的基础设施，为了验证其正确性和有效性，作者将目前在嵌入式 GUI 领域有重要影响的开源项目 MiniGUI 移植到 SkyEye 模拟器上。选择 MiniGUI 是因为 MiniGUI 具有轻量级、占用资源少，高性能，高可靠性，可配置的优点[3]。作者移植的 MiniGUI 的版本是1.3.0。

4.1 定制 MiniGUI 的输入引擎(IAL)

在 libminigui-1.3.0/src/ial/ial.c 文件中包含所用到的 TouchScreen 驱动程序的头文件：

```
#include "skyeye_ts_drv.h"
```

并在 input 结构数组中为定制的输入引擎添加入口项。

```
static INPUT inputs[]=
{
    {"SkyEye4EP7312", InitSkyEye4EP7312Input, TermSkyEye4EP7312Input},
}
```

由于 MiniGUI 默认的输入设备是 PS2鼠标，如果交叉编译 MiniGUI 的函数库时采用内建式资源，则应修改 libminigui-1.3.0/src/include/incoreres.h

```
#define IAL_ENGINE " SkyEye4EP7312"
#define IAL_MDEV "/dev/ts"
#define IAL_MTYPE "def"
```

4.2 移植采用 FrameBuffer 图形引擎(GAL)的 MiniGUI

作者成功移植了能够验证 LCD 及 TouchScreen 模拟效果的基于 MiniGUI -thread 版的两个游戏程序-same 和 housekeeper 到 SkyEye 模拟器上，移植的详细步骤：

(1)安装和设置交叉编译环境，这一步应该在交叉编译内核之前就已完成。

(2)交叉编译并安装 MiniGUI 的函数库 libminigui-1.3.0 到指定路径，注意 configure 时的选项：

```
--disable-lite
--disable-newgal
--enable-fblin8
--enable-tinyscreen
```

其它的选项可根据需要决定是否选择。

对 MiniGUI.cfg 作如下修改，拷贝到文件系统目录下的 /etc/或/usr/local/etc/目录下。

```
[system]
#GAL engine
gal_engine=fbcon
# IAL engine
ial_engine= SkyEye4EP7312
mdev=/dev/ts
mtype=def
[fbcon]
defaultmode=320×240-8bpp
```

(3)安装 MiniGUI 的资源文件 minigui-res-1.3.0。

(4)编译 MiniGUI 的演示程序，去掉调试信息并将 housekeeper 和 same 目录下的可执行文件及相应得资源文件目录 res 拷贝到文件系统目录下，并生成文件系统镜像文件。

```
#genromfs -d romfs/ -f romfs.img
```

(5)在 SkyEye 上运行 MiniGUI 演示程序

```
#skyeye
(SkyEye)file vmlinux
(SkyEye)target sim
(SkyEye)load
(SkyEye)run
... ...
/>/bin/same/same
或
/>/bin/housekeeper/housekeeper
```

以下是其运行时的截图：

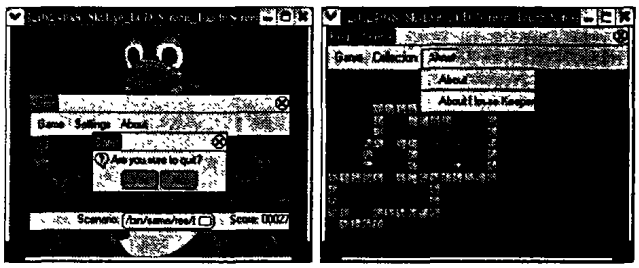

图4 基于 MiniGUI 的应用程序在 SkyEye 运行的效果截图

进一步的工作 1)增加对显示卡的硬件加速功能的支持，增加对其他颜色深度的支持，增加对高级图形功能的支持(例如，Alpha 混和、透明位块传输、光栅操作、YUV 覆盖、Gamma 校正等)。

2)增加对 MiniGUI 窗口拖拽的支持，模拟一些控制键，例如鼠标右键。

3)移植其他的嵌入式 GUI 系统，例如 QT/Embedded，PicoGUI，Tiny-X，ucGUI，GTK/FB 到 SkyEye 上。

4)移植 LCD 及 TouchScreen 模拟模块到 SkyEye 模拟的其他的开发板上。

5)对 SkyEye 的解释执行模块进行系统级的优化，加快 SkyEye 的模拟速度。

参考文献

1 陈渝，李明，杨晔. 源码开放的嵌入式系统软件分析与实践-基于 SkyEye 和 ARM 开发平台. 北京：北京航空航天大学出版社，2004

2 陈渝. SkyEye 核心实现技术报告. 2004

3 陈章龙，唐志强，涂时亮. 嵌入式技术与系统--Intel Xscale 结构与开发. 北京：北京航空航天大学出版社. 2004

4 魏永明. MiniGUI 用户手册. 北京飞漫软件技术有限公司，2003

5 魏永明. MiniGUI 编程指南. 北京飞漫软件技术有限公司，2003

6 于明俭，陈向阳，方汉. Linux 程序设计权威指南. 北京：机械工业出版社，2000

7 许宏松，吴明行，廖世恩. Linux 应用程序开发指南：使用 Gtk+/Gnome 库. 北京：机械工业出版社，2000

8 Rubini A，等著. Linux 设备驱动程序(第二版). 北京：中国电力出版社，2002