VxWorks 上的一种 GUI 系统的设计与实现

赵 甫， 李 跃， 李 芳

(中国航天科工集团 第二研究院 706 所， 北京 100854)

摘 要：VxWorks 操作系统原有的图形支持 WindML 对于开发高级图形应用较困难。CompactGUI 系统是采用面向对象、层次化的设计思路，利用 SiliconMotion 公司的 Lynx3DM+芯片的显卡，实现的一套包括 2D 硬件加速在内的支持基本图元绘制，位图和矢量字体和基本 Windows 风格控件的 GUI 系统。

关键词：VxWorks； Lynx3DM+； 图形用户界面； WindML； 2D 加速

中图法分类号：TP311.52 **文献标识码**：A **文章编号**：1000-7024 (2006) 15-2839-04

Design and implementation of VxWorks GUI

ZHAO Fu, LI Yue, LI Fang

(Institude 706, Second Academy, China Aerospace Science and Industry Corporation, Beijing 100854, China)

Abstract：It's difficulty to develop graphic applications on VxWorks using WindML. CompactGUI is a object-oriented and layered designed GUI system including full featured hardware accelerates and preliminary draws, bitmap, Chinese fonts, and some controls of Windows style based on Lynx3DM+ video chips of Silicon Motion Inc.

Key words：GUI; Lynx3DM+; VxWorks; WindML; 2D accelerate

0 引 言

VxWorks 操作系统在航天、汽车等领域获得广泛应用，随着各种应用的图形界面要求越来越复杂，其自带的图形支持库 WindML 显得力不从心，影响了开发进度和效率。Lynx3DM+芯片是 siliconmotion 公司的一款全功能 2D 加速的图形加速产品，本文利用该芯片设计了它的 VxWorks 上的驱动程序，并完成了一套适用于各种高级图形开发的 GUI 系统。

1 背景介绍

1.1 VxWorks 及其图形支持

VxWorks 是美国风河公司推出的嵌入式实时操作系统，具有高效的 Wind 微内核，灵活的任务间通信机制，先进全面的网络支持，功能强大的文件系统和 I/O 管理，并且支持 POSIX 标准的实时扩展，但是图形方面的支持略差，仅通过 WindML 支持了一般的基本图元绘制，位图和英文字体，使图形应用编程困难，容易产生错误。本系统的主要目的就是替代 WindML 成为 VxWorks 图形应用的控件和加速图形 API 库。

1.2 Lynx3DM+系芯片和显卡

Lynx3DM+芯片及基于该芯片的显卡是 siliconmotion 公司主打产品。其性能远优于工业控制领域和航空航天领域广泛应用的 Chips&Technology 公司的 ct69000 及 ct69030 系列芯片。在图形方面主要有以下特色：①8M 显存；②双 VGA 显示出口，最大支持 1 280×1 024；③双路独立的视频窗口和运动补偿；④全功能的 2D 加速；⑤IEEE 的浮点引擎和全功能的 3D 渲染。

这些特点都促使其替代 ct69000 系列显卡占领市场。本系统就采用 Lynx3DM+系显卡全方位的加速了 2D 图元绘制，使整个系统的性能在硬件上得到了保障。

1.3 GUI 系统

目前 GUI 系统种类繁多，主要有 MS 的 Windows 系统，X Windows 系统，MicroWindows，Qt/Embeded，MiniGUI 等。这些 GUI 系统大都采用消息机制实现了按钮、编辑框、列表框、进度条、工具栏等控件，但是这些系统中的不确定性强的消息传递机制和大量的系统维护和管理使这些软件不能更充分的发挥 VxWorks 的特点，进而在对数据显示要求更高的领域得到更好的应用。

2 CompactGUI 系统的设计

整个 GUI 系统采用层次化的设计思路，清晰的分为以下几个层次。

2.1 Lynx3DM+显卡驱动

包括显卡的初始化，显示模式设置，全功能 2D 加速，包括加速的水平线，垂直线，斜线绘制，矩形填充，屏幕拷贝，硬件屏幕剪切等，都在这一层次实现。采用统一的结构 SMIDrv 表示该驱动对于上层的接口，接口的主要函数为

SMIHwInit();

SMIModeInit();
SMIHwLine();
SMIRectFill();
SMIPatternFill();
SMIImageWrite();
SMISetClipRect();
SMIDisableClip();
SMIScreenCopy();

2.2 基本图元绘制层

我们知道，显存映射到系统内存空间后，程序可以和访问内存一样直接访问，而内容就可以显示在屏幕上。该层次就采用通常的软件方法，实现显存映射后的 8 位，16 位颜色模式的点线和其它基本图元函数。这些函数主要是

SetPixel();
GetPixel();
CopyBox();
DrawRectangle();
DrawLine();
DrawCircle();
DrawEllipse();
DrawArc();
FillRect();
FillEllipse ();
FillSector();

2.3 中文字体支持和位图

WindML 中采用专用的 bmf 字体格式支持多种英文字体，本系统也采用了类似 bmf 字体格式的光栅字体，支持多种大小的中文，并采用著名的 freetype 引擎支持 TTF 矢量字体。涉及利用 SMI722 所提供的硬件功能对字体进行优化。支持 bmp 位图文件的显示。

设计了以下函数支持这些基本功能

GetCurrentFont();
SetCurrentFont();
DrawText();
SetTextPosition();
GetTextPosition();
CreateBMP();
DrawBMP();
CopyBMP();

2.4 通用控件和特殊控件

通用控件包括按钮，对话框，窗口，进度条等。这些控件主要是为了界面风格和编程方便，把应用程序员从使用基本画线函数一点点绘出某种风格的界面的简单任务中解脱出来。因此，这些控件都提供了简单的绘制函数，调用就可以在屏幕上绘出相应的图形。

CreateDownButton();
CreateUpButton();
CreateWindow();
CreateDialog();
CreateMenu();
CreateStatusBar();
CreateProgressBar();

总的系统层次如图 1 所示。

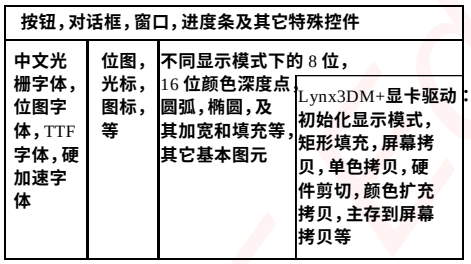

图 1 CompactGUI 系统结构

3 CompactGUI 系统的实现

3.1 Lynx3DM+显卡驱动的实现

Lynx3DM+显卡驱动的程序结构图如图 2 所示。

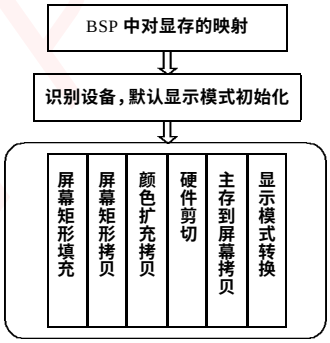

图 2 Lynx3DM+驱动结构

BSP 对现存的映射可以通过两种方式实现，一种是在 BSP 的 sysLib.c 文件管理静态设备映射的 PHYS_MEM_DESC sysPhysMemDesc []结构中加入如下代码

```
{
(void *) 0xe4000000,
(void *) 0xe4000000,
0x4000000,
VM_STATE_MASK_FOR_ALL,
VM_STATE_FOR_IO
}
```

本例说明该显卡将被映射在地址为 0xe4000000 开始的位置上，并且显存大小为 0x4000000 字节。

另一种是改写 BSP 动态映射显卡，每一次启动系统时，即使映射地址发生改变也可确保正确，对用户是透明的。该动态映射函数的伪代码表示

```
if (pciFindDevice (GRAPHIC_VENDOR_ID, GRAPHIC_DEV_ID, instance, &pciBus, &pciDevice, &pciFunc) == OK)
    {
        if(pciConfigInLong (pciBus, pciDevice, pciFunc, PCI_CFG_BASE_ADDRESS_0, &mapBaseAddr) != OK)
        goto error;
```

```
/*计算长度*/
GetVideoMemSize(pciBus,pciDevice, pciFunc, tmpAddr, &videoMemSize);
/*动态映射*/
if(sysMmuMapAdd((void *)(mapBaseAddr & 0xfffffff0), videoMemSize, VM_STATE_MASK_VALID|VM_STATE_MASK_WRITABLE|VM_STATE_MASK_CACHEABLE, VM_STATE_VALID | VM_STATE_WRITABLE | VM_STATE_CACHEABLE_NOT ) == ERROR)
goto error;
...
}
```

识别设备和默认模式的初始化实现流程为：①选择默认的显示模式；②检测设备是否存在；③必要的空间分配（将保存寄存器的初值）；④按照显示模式设置寄存器的初始值；⑤把初始值写入寄存器。

同时，在设置模式和寄存器操作时，将 Lynx3DM+所有寄存器分为 VGA 兼容和 SMI 扩展的两部分，分别实现。与 VGA 兼容的寄存器主要包括 5 类：CRTC 控制寄存器，Sequencer 寄存器，Graphics 寄存器，Attribute 寄存器以及颜色查找表寄存器。SMI 扩展寄存器包括平板寄存器，电源管理寄存器，2D 引擎寄存器，3D 引擎寄存器，视频控制寄存器。

初始值写入寄存器的顺序为：

(1) 关闭屏幕，关闭时序；

(2) 等待显卡空闲；

(3) 设置一般图形控制命令寄存器（不使用显卡作为 PCI 主设备，关闭 PCI 脉冲模式，使用 SMI 扩展 Aperture 进行加速，设置显存的直线访问模式）；

(4) 设置显卡的内部电源管理寄存器和功能管理寄存器（使能 2D 引擎）；

(5) 设置平板控制寄存器(使用两个平板显示出口的一个)；

(6) 写入 VGA 兼容寄存器（按照总控，Sequencer，CRTC，Graphics，Attribute，颜色表寄存器顺序写入初值）；

(7) 写入颜色表掩码；

(8) 写入 SMI 映射的掩 CRTC 功能寄存器（用来实际控制屏幕的宽高模式）；

(9) 写入 2D 引擎寄存器的初始值；

(10) 打开屏幕，使能时序；

(11) 调整屏幕中心焦点；

这个步骤完成后，我们所选择好的默认显示模式就已经设置完毕了。接下来的工作是实现图形加速功能。

所谓全功能 2D 硬件加速，主要包括如表 1 所示的几类。

本驱动实现了上述全功能 2D 加速，以矩形填充为例，介绍实现过程如下：

矩形填充是以上种类中应用最广、实现最简单的一种加速功能。显卡不一定支持所有的硬件加速，但是一般都支持矩形填充。芯片内部一般设置 2 条或 4 条流水线来完成加速操作和 3D 的纹理设置，我们在进行某一种加速时，需要读出当前有几条流水线空闲，需要的流水线空闲时，才进行相应操作。

首先介绍 Lynx3DM+芯片对 2D 加速支持的寄存器概况。

表 1　常用加速种类和功能描述

加速种类	功能描述
Screen to Screen Copies	屏幕到屏幕的拷贝，显存内操作
Solid Fills	实体填充，也就是矩形填充，显存内操作
Solid Lines	实线，包括水平，垂直，斜线
Dashed Lines	虚线，包括水平，垂直，斜线
Screen to Screen Color Expansion	屏幕内颜色扩充填充，显存内操作
CPU to Screen Color Expansion	内存到屏幕颜色扩充，内存到显存
8×8 Mono Pattern Fills	8×8 单色模式填充，显存内
8×8 Color Pattern Fills	8×8 颜色模式填充，显存内
Image Writes	内存到屏幕的连续拷贝，内存到显存
Clipping	硬件剪切

Lynx3DM+芯片使用了 27 个寄存器来支持 2D 加速，有 16 位寄存器和 32 位寄存器两种。主要包括：

(1) 控制和命令寄存器：设置触发绘制引擎所使用的命令和数据操作种类（与，或等）。

(2) 操作的源位置和目标位置寄存器：设置源和目标的 x、y 坐标。

(3) 操作的宽度和高度寄存器：设置进行操作的宽高信息。

(4) 要进行的操作的源和目标地址寄存器：设置源和目标的起始基准地址，假如为零，表示前面的源和目标 x、y 位置都从显存开始处计算。

(5) 前景色和背景色以及颜色比较寄存器：设置要进行操作的颜色选项。

(6) Pattern 操作的 Pattern 内容寄存器：设置要进行的 Pattern 操作的内容。

(7) 剪切寄存器：4 个寄存器分别控制了操作的可见范围。

以矩形填充为例，矩形填充操作的流程为：首先设置 4 个剪切寄存器设置好剪切矩形，这将决定在某一个矩形范围内才能看到我们绘制的矩形；再设置待填充的颜色；再设置填充模式为 Pattern 实填充，即待填充的 88 源数据模块值均为 1；然后设置要填充矩形的位置，左上方顶点坐标和宽度，高度；最后写入操作命令为打开引擎，原样拷贝。这样，屏幕上就看到我们所期望的位置出现填充后的颜色矩形区域。

其它几种加速的实现方法类似，不再赘述。

3.2　基本图元绘制层的实现

基本图元绘制的实现，采用了统一的数据结构描述绘图上下文内容的信息。主要结构如下

```
// Graphic Context
struct GraphicContext
{
  SMIDrvPtr    pSmi;    //SMI722 驱动，提供了硬件加速
  int    clipLeftTopx;
  int    clipLeftTopy;
  int    clipRightDownx;
  int    clipRightDowny;                    //剪切矩形坐标
  int    screenWidth;
  int    screenHeight;                      //屏幕宽、高
  int    bpp;                               //颜色深度
  void * videoMemAddr;                      //显存开始位置
```

```
    int     videoMemSize;                    //显存大小
    UINT32  DBufOffset;                      //双缓冲偏移
    COLOR  foreground;                       //前景色
    COLOR  background;                       //背景色
    COLOR  pencolor;                         //画笔色
    COLOR  brushcolor;                       //画刷色
    COLOR  textcolor;                        //字体色
    POINT BrushPos;                          //画刷位置
    POINT CurPenPos;                         //当前画笔位置
    POINT CurTextPos;                        //当前字体位置
    PFONT pFont;                             //当前所使用的字体
    int rop;                                 //当前光栅操作种类
};
```

使用当前上下文实现的高级图形操作主要包括：点、线、圆、椭圆、圆弧、有宽度的曲线，以及特殊多边形填充，填充模式，画笔和画刷等。显然，该层次上的图形元素绘制会使用到SMI驱动中的加速功能。调用驱动程序的函数进行加速，可以充分的改善单纯由软件实现的高级图形操作的速度，尤其是硬件的直线，拷贝，填充，大大加快了图形显示的速度，把CPU从繁重的跨PCI总线传送和计算中解脱出来。

3.3 位图和字体的实现

位图和字体文件的解析，无论是压缩位图还是普通的bmp格式，无论是光栅字体还是矢量字体，都有格式可循，实现是不复杂的。在本产品中，则同时充分应用SMI722所提供的硬件加速来优化位图和字体的实现。

以光栅字体的实现为例，字体从解析到显示到屏幕某处的过程如图3所示。

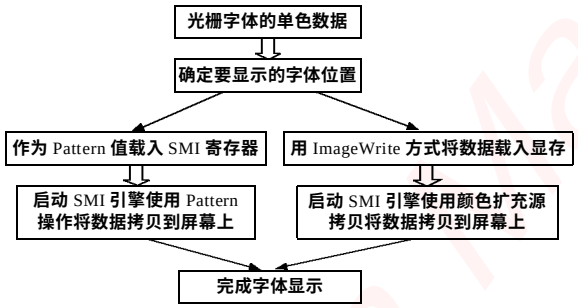

图3 字体显示流程

首先需要采用某种格式存储光栅字体的单色数据，需要存储的项目包括：该字的宽、高、绘画起点、确切单色数据位置等。相关详细知识请参考光栅字体有关文献。

这些数据被提取出来以后，可以分两种路径来显示到屏幕上。一是当作Pattern数据填入SMI提供的两个Pattern数据寄存器，这些数据可以作为源数据使用，经过硬件图形引擎运算后将结果输出到屏幕上。然后设置好字体要显示的位置，设置好要进行的操作为颜色扩充的以Pattern寄存器数据为源的扩充拷贝，启动引擎后就可以见到字体显示到指定的位置上。另一条路是将这些单色数据通过SMI提供的显存与内存通信的专门空隙采用ImageWrite的命令将单色数据快速传递到显存不可见部分，然后设置字体要显示的位置，设置要进行的操作为颜色扩充的以现存某处数据为源的扩充拷贝，启动引擎后就可以见到字体显示出来。

这两种情况均充分使用了硬件的特性，以最快的速度完成了字体的显示。而对于TTF字体，则可以通过分配显存空间，完成字体解析和计算后拷贝到显示位置的方式进行优化。TTF文件的解析依靠freetype引擎实现，这里不再赘述。

3.4 控件的实现

每种控件都实现专门的消息处理函数以完成相关的操作。以按钮为例，按钮的回调函数DefaultButtonProc()就实现了关于绘制某种状态的按钮所要进行的详细绘制的组合。以及不同消息改变绘制状态的响应，这种相应用于完成该按钮状态的转换，如按下，弹起等。

在进行某种空间的绘制时，通过调用下面的函数SendDrawMessage进行绘制。其中iMsg包含了进行绘制的种类，如绘制普通按钮，还是绘制按下状态的按钮等，绘制控件的位置，宽高等信息。

```
int SendDrawMessage(int CtrlName, MSG iMsg)
{
  WNDPROC WndProc;
  if (CtrlName == NULL) return -1;
  WndProc = GetDefaultProcOfControl (CtrlName);
  return (*WndProc)(NULL, iMsg);
}
```

这样，控件就可以当作一个可见的复合图形进行显示。还可以通过向该控件发送不同的消息而改变它的显示状态。

其余的控件实现思路都是一样的。同时，要想让控件活动起来，就需要有输入和消息的响应。这里介绍输入和消息的实现机制。

键盘和鼠标输入通过两种方式实现。一是采用VxWorks本身的WindML图形库所带的鼠标和键盘驱动，设计循环等待从WindML事件服务传递上来的鼠标和键盘事件，再传递给本系统进行分析和处理。该循环结构如下

```
void GetUglEvent(GC *gc, INPUTEVENT ie)
{
    UGL_EVENT event;
    UGL_STATUS status;
    UGL_FOREVER
    {
    status = uglEventGet (gc->qId, &event, sizeof (event),
UGL_WAIT_FOREVER);
    if (status != UGL_STATUS_Q_EMPTY)
        uglConvertEvent(event, ie);
    }
}
```

uglConvertEvent()负责将UGL的事件形式转换成Compact-GUI所识别的事件。

另一种方式是当系统不启动WindML的输入事件服务时，就采用本系统自己的输入循环在输入设备上使用select功能调用等待输入事件，获得事件之后向上层传递。这个流程是：

(下转第2865页)

从中加载的 URL。即 webserver 提供类加载的路径。

当动态加载的类发送到另一个客户时，因为该类是使用 URLClassLoder 加载的，因此第 2 个客户可以使用正确的 URL 信息加载该类。

当远程方法的参数或者返回值是导出后的远程对象，RMI 用该远程对象的 stub 替换，序列化的是 stub。

3.3.2 移动代理移动性和自主性

客户端通过访问 RMI 注册，获得代理主机的 stub，随后客户端使用该 stub 与远程对象通信。

客户端创建移动代理后，并没将移动代理导出（export），因此当调用代理主机的远程方法，如 Agent addAgent (Agent agent) 时，虽然参数 agent 是远程对象，由于 agent 没有导出 (export)，和普通 Java 对象一样，序列化的是移动代理实现本身，而不是其 stub；接着在服务器端 agent 被反序列化，移动代理实现类并不在服务器端，这时 RMI 使用动态类加载从 WebServer 加载需要的移动代理实现类。然后代理主机将移动代理导出到 RMI 注册，RMI 将用该对象的 stub 替换它，并返回给客户端。系统基于此特性实现移动代理的移动性。

在服务器端安装安全管理器，并赋予移动代理一定的访问主机资源的权限，移动代理，执行动作，访问主机资源－实现移动代理的自主性。

客户端示例性代码如下

```
//获取代理主机 stub
AgentHost stub = (AgentHost) new InitialContext (). lookup ("agenthost")；
//创建移动代理对象
ServiceAgent  agent = new ServieAgentImpl ()；
//注册，agent 被序列化，并传到服务器端，返回的是 stub。
agent = (ServiceAgent)host.addAgent(agent)；
```

服务器端示例代码为：①导出代理主机到 RMI；②在 addAgent (Agent agent) 方法中导出 agent：Remote stub = UnicastRemoteObject.exportObject(agent)；③将返回移动代理实现类的 stub。

4 实验及验证

实验基于图 3 中提供的 ServieAgent 及 ServieAgentImpl 验证了移动代理的移动性，自主性。过程为：①启动 webserve，配置需要动态加载的类；②启动服务器，启动客户端；③客户创建移动代理，通过 RMI 注册，取得代理主机 stub，与代理主机通讯发送移动代理到服务器，并在服务器上完成任务。

5 结束语

系统基于 RMI 和动态类加载技术实现了移动代理系统，并进行了验证，主要关注于移动代理的移动性。移动代理的自主性，与 Java 的安全模型关系紧密，本文并没有过多关注，利用 Java 提供的安全管理器可以实现对移动代理自主性的授权。

参考文献：

[1] Liu M L.分布式计算原理与应用[M].北京:清华大学出版社, 2004.

[2] 李治,任波,王乘.基于 Java 技术的分布式计算环境研究[J].计算机工程与设计, 2004,26(6):912-920.

[3] Rickard Oberg.精通 RMI-Java 与 EJB 企业级应用开发[M].北京:机械工业出版社, 2003.

[4] Bill Venners: Inside the java virtual machine [M]. Second Edition.北京:机械工业出版社, 2003.

[5] 郑晓东,王志坚,周晓峰,等. 一种基于 Web Service 的分布式计算模型研究及其实现[J].计算机工程与应用,2004,40(1):144-147.

[6] 刘文红,罗友平.J2EE 开发使用手册[M].北京:电子工业出版社, 2004.

(上接第 2842 页)

select 鼠标和键盘设备的输入，获得输入后读出鼠标和键盘的输入信息（鼠标的某个健按下。键盘的某个字符按下或松开），将这些信息封装成消息的形式传递给系统的消息队列等待处理，继续 select 等待输入。

消息处理机制：CompactGUI 借鉴其它通用 GUI 中的消息方式，内部维护一套消息机制，这个消息机制包括消息队列，该消息队列用于采集和暂存系统得到的输入消息，以及绘图等系统消息，而相应的控件实现其响应函数，在得到消息时进行响应。编写应用软件时，多个任务分别负责数据的接收，数据的计算，图形的显示，计算完毕的数据被封装成消息，使用 SysSendMessage 传递给负责图形显示的任务，实际上是直接调用待绘控件的绘制函数，这样可以保证实时性，而不会由于系统需要处理其它消息而必须让当前消息按顺序排队。

4 结束语

本系统基于 SiliconMotion 的 Lynx3DM+芯片实现了 VxWorks 上使用的使用简单，风格富有亲和力的图形界面系统，充分利用了硬件加速功能，速度快，可靠性和可维护性高，编程简单。是一种既可配合 WindML 使用，也可以单独使用完成图形系统初始化，图形和控件绘制的 GUI 系统。

参考文献：

[1] Lynx3DM+数据手册[S]. SiliconMotion 公司, 2002.

[2] MiniGUI[OL]. 2004-07-01.*www.minigui.org*.

[3] 魏永明.实时嵌入式 Linux 系统上 GUI 的发展与展望[J].微电脑世界, 2000,278(47):11-13.

[4] MicroWindows [OL]. 2004-07-01.*http://microwindows.censoft.com*.

[5] OpenGUI[OL]. 2004-07-01.*http://www.tutok.sk/fastgl/*.

[6] TrollTech[OL]. 2004-07-01.*http://www.trolltech.com/*.

[7] Freetype[OL]. 2004-07-21.*http://www.freetype.org*.

嵌入式资源免费下载

总线协议：

1. 基于 PCIe 驱动程序的数据传输卡 DMA 传输
2. 基于 PCIe 总线协议的设备驱动开发
3. CANopen 协议介绍
4. 基于 PXI 总线 RS422 数据通信卡 WDM 驱动程序设计
5. FPGA 实现 PCIe 总线 DMA 设计
6. PCI Express 协议实现与验证
7. VPX 总线技术及其实现
8. 基于 Xilinx FPGA 的 PCIE 接口实现
9. 基于 PCI 总线的 GPS 授时卡设计
10. 基于 CPCI 标准的 6U 信号处理平台的设计
11. USB30 电路保护
12. USB30 协议分析与框架设计
13. USB 30 中的 CRC 校验原理及实现
14. 基于 CPLD 的 UART 设计
15. IPMI 在 VPX 系统中的应用与设计
16. 基于 CPCI 总线的 PMC 载板设计
17. 基于 VPX 总线的工件台运动控制系统研究与开发
18. PCI Express 流控机制的研究与实现
19. UART16C554 的设计
20. 基于 VPX 的高性能计算机设计
21. 基于 CAN 总线技术的嵌入式网关设计
22. Visual C 串行通讯控件使用方法与技巧的研究
23. IEEE1588 精密时钟同步关键技术研究
24. GPS 信号发生器射频模块的一种实现方案
25. 基于 CPCI 接口的视频采集卡的设计
26. 基于 VPX 的 3U 信号处理平台的设计
27. 基于 PCI Express 总线 1394b 网络传输系统 WDM 驱动设计
28. AT89C52 单片机与 ARINC429 航空总线接口设计
29. 基于 CPCI 总线多 DSP 系统的高速主机接口设计
30. 总线协议中的 CRC 及其在 SATA 通信技术中的应用
31. 基于 FPGA 的 SATA 硬盘加解密控制器设计
32. Modbus 协议在串口通讯中的研究及应用
33. 高可用的磁盘阵列 Cache 的设计和实现
34. RAID 阵列中高速 Cache 管理的优化

35. 一种新的基于 RAID 的 CACHE 技术研究与实现
36. 基于 PCIE-104 总线的高速数据接口设计
37. 基于 VPX 标准的 RapidIO 交换和 Flash 存储模块设计
38. 北斗卫星系统在海洋工程中的应用
39. 北斗卫星系统在远洋船舶上应用的研究
40. 基于 CPCI 总线的红外实时信号处理系统
41. 硬件实现 RAID 与软件实现 RAID 的比较
42. 基于 PCI Express 总线系统的热插拔设计
43. 基于 RAID5 的磁盘阵列 Cache 的研究与实现

VxWorks：

1. 基于 VxWorks 的多任务程序设计
2. 基于 VxWorks 的数据采集存储装置设计
3. Flash 文件系统分析及其在 VxWorks 中的实现
4. VxWorks 多任务编程中的异常研究
5. VxWorks 应用技巧两例
6. 一种基于 VxWorks 的飞行仿真实时管理系统
7. 在 VxWorks 系统中使用 TrueType 字库
8. 基于 FreeType 的 VxWorks 中文显示方案
9. 基于 Tilcon 的 VxWorks 简单动画开发
10. 基于 Tilcon 的某武器显控系统界面设计
11. 基于 Tilcon 的综合导航信息处理装置界面设计
12. VxWorks 的内存配置和管理
13. 基于 VxWorks 系统的 PCI 配置与应用
14. 基于 MPC8270 的 VxWorks BSP 的移植
15. Bootrom 功能改进经验谈
16. 基于 VxWorks 嵌入式系统的中文平台研究与实现
17. VxBus 的 A429 接口驱动
18. 基于 VxBus 和 MPC8569E 千兆网驱动开发和实现
19. 一种基于 vxBus 的 PPC 与 FPGA 高速互联的驱动设计方法
20. 基于 VxBus 的设备驱动开发
21. 基于 VxBus 的驱动程序架构分析
22. 基于 VxBus 的高速数据采集卡驱动程序开发
23. Vxworks 下的冗余 CAN 通讯模块设计
24. WindML 工业平台下开发 S1d13506 驱动及显示功能的实现

Linux：

1. Linux 程序设计第三版及源代码
2. NAND FLASH 文件系统的设计与实现
3. 多通道串行通信设备的 Linux 驱动程序实现
4. Zsh 开发指南-数组
5. 常用 GDB 命令中文速览
6. 嵌入式 C 进阶之道
7. Linux 串口编程实例
8. 基于 Yocto Project 的嵌入式应用设计
9. Android 应用的反编译
10. 基于 Android 行为的加密应用系统研究
11. 嵌入式 Linux 系统移植步步通
12. 嵌入式 CC++语言精华文章集锦
13. 基于 Linux 的高性能服务器端的设计与研究
14. S3C6410 移植 Android 内核
15. Android 开发指南中文版
16. 图解 Linux 操作系统架构设计与实现原理（第二版）
17. 如何在 Ubuntu 和 Linux Mint 下轻松升级 Linux 内核
18. Android 简单 mp3 播放器源码
19. 嵌入式 Linux 系统实时性的研究
20. Android 嵌入式系统架构及内核浅析
21. 基于嵌入式 Linux 操作系统内核实时性的改进方法研究
22. Linux TCP IP 协议详解
23. Linux 桌面环境下内存去重技术的研究与实现
24. 掌握 Android 7.0 新增特性 Quick Settings
25. Android 应用逆向分析方法研究
26. Android 操作系统的课程教学
27. Android 智能手机操作系统的研究
28. Android 英文朗读功能的实现
29. 基于 Yocto 订制嵌入式 Linux 发行版
30. 基于嵌入式 Linux 的网络设备驱动设计与实现
31. 如何高效学习嵌入式
32. 基于 Android 平台的 GPS 定位系统的设计与实现

Windows CE:

1. Windows CE.NET 下 YAFFS 文件系统 NAND Flash 驱动程序设计
2. Windows CE 的 CAN 总线驱动程序设计
3. 基于 Windows CE.NET 的 ADC 驱动程序实现与应用的研究

4. 基于 Windows CE.NET 平台的串行通信实现
5. 基于 Windows CE.NET 下的 GPRS 模块的研究与开发
6. win2k 下 NTFS 分区用 ntldr 加载进 dos 源代码
7. Windows 下的 USB 设备驱动程序开发
8. WinCE 的大容量程控数据传输解决方案设计
9. WinCE6.0 安装开发详解
10. DOS 下仿 Windows 的自带计算器程序 C 源码
11. G726 局域网语音通话程序和源代码
12. WinCE 主板加载第三方驱动程序的方法
13. WinCE 下的注册表编辑程序和源代码
14. WinCE 串口通信源代码
15. WINCE 的 SD 卡程序[可实现读写的源码]
16. 基于 WinCE 的 BootLoader 研究
17. Windows CE 环境下无线网卡的自动安装
18. 基于 Windows CE 的可视电话的研究与实现

PowerPC：

1. Freescale MPC8536 开发板原理图
2. 基于 MPC8548E 的固件设计
3. 基于 MPC8548E 的嵌入式数据处理系统设计
4. 基于 PowerPC 嵌入式网络通信平台的实现
5. PowerPC 在车辆显控系统中的应用
6. 基于 PowerPC 的单板计算机的设计
7. 用 PowerPC860 实现 FPGA 配置
8. 基于 MPC8247 嵌入式电力交换系统的设计与实现
9. 基于设备树的 MPC8247 嵌入式 Linux 系统开发
10. 基于 MPC8313E 嵌入式系统 UBoot 的移植
11. 基于 PowerPC 处理器 SMP 系统的 UBoot 移植
12. 基于 PowerPC 双核处理器嵌入式系统 UBoot 移植
13. 基于 PowerPC 的雷达通用处理机设计
14. PowerPC 平台引导加载程序的移植

ARM：

1. 基于 DiskOnChip 2000 的驱动程序设计及应用
2. 基于 ARM 体系的 PC-104 总线设计
3. 基于 ARM 的嵌入式系统中断处理机制研究
4. 设计 ARM 的中断处理
5. 基于 ARM 的数据采集系统并行总线的驱动设计
6. S3C2410 下的 TFT LCD 驱动源码
7. STM32 SD 卡移植 FATFS 文件系统源码
8. STM32 ADC 多通道源码
9. ARM Linux 在 EP7312 上的移植
10. ARM 经典 300 问
11. 基于 S5PV210 的频谱监测设备嵌入式系统设计与实现
12. Uboot 中 start.S 源码的指令级的详尽解析
13. 基于 ARM9 的嵌入式 Zigbee 网关设计与实现
14. 基于 S3C6410 处理器的嵌入式 Linux 系统移植
15. CortexA8 平台的μC-OSⅡ及 LwIP 协议栈的移植与实现
16. 基于 ARM 的嵌入式 Linux 无线网卡设备驱动设计
17. ARM S3C2440 Linux ADC 驱动
18. ARM S3C2440 Linux 触摸屏驱动
19. Linux 和 Cortex-A8 的视频处理及数字微波传输系统设计
20. Nand Flash 启动模式下的 Uboot 移植
21. 基于 ARM 处理器的 UART 设计

Hardware:

1. DSP 电源的典型设计
2. 高频脉冲电源设计
3. 电源的综合保护设计
4. 任意波形电源的设计
5. 高速 PCB 信号完整性分析及应用
6. DM642 高速图像采集系统的电磁干扰设计
7. 使用 COMExpress Nano 工控板实现 IP 调度设备
8. 基于 COM Express 架构的数据记录仪的设计与实现
9. 基于 COM Express 的信号系统逻辑运算单元设计
10. 基于 COM Express 的回波预处理模块设计
11. 基于 X86 平台的简单多任务内核的分析与实现
12. 基于 UEFI Shell 的 PreOS Application 的开发与研究
13. 基于 UEFI 固件的恶意代码防范技术研究
14. MIPS 架构计算机平台的支持固件研究
15. 基于 UEFI 固件的攻击验证技术研究

16. 基于 UEFI 的 Application 和 Driver 的分析与开发

Programming:

1. 计算机软件基础数据结构 - 算法