构建基于 M P 的嵌入式 L x系统 u x

齐传兵 李昊
(无锡江南信息安全工程技术中心 2) 1 4 0 7 2

摘 要：本文基于 M P 和 8L 5i xn内核，介绍了构建嵌入式 L i系统的主要过程和关键技术，包括：建立交叉开发环境、U - 移植、内核定制和编译、创建根文件系统。

关键词： E L D K L Ui内核 Bu根文件系统 B u s y B o x
中图分类号 :T P 3 1 6 文献标识码 :A 文章编号 :1 6 7 3 - 0 5 3 4 (2 0 0 7) 0 2 (b) - 01

1硬件环境

在进行嵌入式系统硬件设计前，应先选择一款成熟的目标板作参考，尽量选择与参考板相同型号的器件，如 F L型号、网络接口芯片、 D 等，这样就可以利用现成的驱动，大大减少软件移植的工作量。本文采用自主设计的目标板。目标板上集成了摩托罗拉 M P C处理器，5 42 1的 6D、MD 1R的6 M F L以及其他专用芯片，另外还提供了二个三速以太网接口、二个 R S 串口2 -3个 P 槽和一个 J 调试接口。系统体积小、功耗低、处理能力强，能够装载和运行嵌入式 L i操作系统。x下面基于该目标板，讲述利用开放资源构造嵌入式 L 系统的主要过程和技术。

2建立交叉编译环境

我们工作用的 P机一般是 x架构的处理器，而目标板的处理器是 P 架构，因此，在进行开发前需要建立一个运行在 x 架构上，能够编译 P处理器程序的跨平台编译工具链，又叫交叉编译工具链。嵌入式 L i交叉编译工具链由编译器、链接器和 l 库等组成。建立一个交叉编译工具链是一个相当复杂的过程，如果不愿自行建立，可以从网上免费下载一些编译好的可用的交叉编译工具链，如 E L(E K b L i n u x D)。本文使用的是 E L 嵌入式 4 L . i开发套件，该套件包括 G 交叉开发工具，如编译器、b i n ug等，以及一些已经编译好的目标系统工具和目标系统链接库。用 E L 建立交叉开发环境的过程如下：

2 建立宿主机环境

首先在宿主机上安装 L i操作系统。x为了保证 E 能够正确安装，建议安装较新版本的 L i，如 F e d o。我安装的是 R e d。H a t 9 . 0

2 获取2 E 软件包 K

从以下站点下载 E 的 DI 映像文件。

f t p : / / m i r r o r . s w . n e t / p u b / e l d k / 或者 h t t p : / / m i r r o r . s w . n e t / e l d k / e l d k /

2 安装3 E L D K

m o u n t - o
1 1 _ f r e e s c a l e . i s o
. / i n s t a l l -

2 配置4 E L D K

通过如下命令增加 E 的工作路径和交叉编译前缀：

P A T H = $ P A T H : / o p t / e l d k / u s r / b i n : / o p t / e l d k / b i n
e x
C R O S S _ C O M P I L E = p p c _ 8 5 x x -

2 使用5 E L D K

完成上述工作以后，就可以用 p p c _ 编译程序了。如： g c c
p p c _ 8 5 x x - g c c

关于 E 更详细的使用说明，请参见 2光盘根目录下的 R E A D。M E . h t m l

3 U -移植 o o t

B o o 是系统复位后进入操作系统之前执行的一段代码，主要用于完成由硬件启动到操作系统启动的过渡，从而为操作系统提供基本的运行环境，其功能类似于 P C 机的 B 程序。 L i有许多 B o o t l o a d e r 可用，但就目前来看， U - 对 P o w e r P C 系列处理器支持最为丰富，对 L i的支持也最完善。 U - 是开放源码的项目，用户可以根据目标板的硬件，修改和配置 U - B ，定制专用的 B o o t l o a d e r 。

下面以我的 M P C主板为例，说明 U - 的移植过程。

3 获取 U - 源码。

建议从 h t t p : / / s o u r c e f o r g e . n e t / 下载最新的 U - 源码。下载的 U - 源码为 t a r 格式，使用命令 " t a r - j x v " 解压。

3 建立自己的工程

U - 套件中有大量预定义的目标板配置，请查看 R E A 文件，找一个与自己目标板相近的配置作为参考。我使用的是 M P 处理器，所以参考了 m p c 8 5 4 1 c d s 目标板。建立工程的过程如下：

① 首先，修改 u - b o o t / M a k e f i l e 文件，在其中找到：

M P C 8 5 4 1 C D S _ c o n f i g :
@ . / m k c o n f i g $ (@ : _ c o n f i g =) p p c m p c 8 5 x x m p c 8 5 4 1 c d s

在其后加两行：

M P C 8 5 4 1 J N S _ c o n f i g :
@ . / m k c o n f i g $ (@ : _ c o n f i g =) p p c m p c 8 5 x x j n s

其中 p p c 是 C P U 的架构， m p c 8 5 x x 是 C P U 对应的目录， j n s 是目标板对应的目录。

② 第二步，拷贝源码，作为移植基础。在 / u - b o o t / b o a r d 下新建目录 j n s ，把 / u - b o o t / b o a r d / m p c 8 5 4 1 c d s 并将 / u - b o o t / b o a r d / m p c 8 5 4 1 c d s 目录拷贝到 / u - b o o t / b o a r d / j n s 下，然后将 u - b o o t / b o a r d / j n s 下的 m p c 8 5 4 1 c d s . c 所有文件拷贝到 / u - b o o t / b o a r d / j n s 下，并将 m p c 8 5 4 1 c d s . c 改名为 j n s . c

③ 第三步，在 / u - b o o t / i n c l u d e / c o n f i g s 下添加对应的头文件。 / M P C 8 5 4 1 对应的头文件是 _ c o n f i g s / M P C 8 5 4 1 C D S . h 可以将 M P C 8 5 4 1 C D S . h 拷贝一份，命名为 M P C 8 5 4 1 J N S . h

④ 第四步，修改 b o a r d / j n s / u - b o o t . l d s
将 : b o a r d / c d s / m p c 8 5 4 1 c d s / i n i t . o (. b o o t p g)
改为 : b o a r d / j n s / m p c 8 5 4 1 j n s / i n i t . o (. b o o t p g)
将 : b o a r d / c d s / m p c 8 5 4 1 c d s / i n i t . o (. t e x t)
改为 : b o a r d / j n s / m p c 8 5 4 1 j n s / i n i t . o (. t e x t)

⑤ 最后，试编译一下：

m a k e M P C 8 5 4 1 J N S _ c o n f i g
m a k e

这时编译出来的目标代码和 M P C 8 5 4 1 C D S 是一模一样的，还需要根据自己的目标板进行修改。

3 文件修改

移植过程中，需要修改的主要文件有：

① u - b o o t / b o a r d / j n s / t l b . c 。该文件定义了系统存储空间的分配和映射，包括 D D R 、L B C 、 F L A S H 、 B C S R 、 P C I 寄存器、 C C S R 映射地址等。

② u - b o o t / b o a r d / j n s / m p c 8 5 4 1 j n s . c 。该文件完成系统 D D R 、 S D R A M 、 L B C 和 P C I 总线的初始化。

③ u - b o o t / i n c l u d e / c o n f i g s / M P C 8 5 4 1 J N S . h 。大多数的系统配置参数和寄存器参数都是在这个文件中设置的。

④ / u - b o o t / c p u / m p c 8 5 x x / s t a r t . S 修改此文件，可以设置在 M P C 8 5 4 1 J N S . h 没有定义的 C C S R 寄存器参数。

3 编译与安装

执行如下命令完成 U - 的编译：

m a k e c l e a n 清除以前的编译结果
m a k e M P C 8 5 4 1 J N S _ c o n f i g 配置编译选项
m a k e

编译生成的文件有：

s y s t e m . m a p 符号表
u - b o o t E L F 文件格式的 U - 映像
u - b o o t . b i n 原始二进制映像，可写入引导存储设备
u - b o o t . s r e c S - R e c o r d 格式的 U - 映像

最后用硬件编程设备将 U - 映像加载到目标板的存储设备即可。

4内核定制和编译

4 取 1L 内核源码 x

安装好 E 后，在 K/ o p t / p p c _ 8 5下有 xL / i内核源码 x / s 建议从 h t t p : 下载最新的内核源码。我使用的内核是 L i n u x。- 2 . 6 b. j1 - 8 m. 1 : = m y

4 修改源码

① 修改 a r c h / p p c / p f l i a e t 3 f . o o r m s / 8 5 x x e / l d k / m p c 8 5 x x。此文件包含针对目标板特定硬件的初始化代码，因此通常需要修改。例如中断分配、网络设备初始化等。

② 修正一些内核编译期间出现的错误，这样的错误较少也较易修改。

4 .配置内核

内核的配置选项相当多，想搞清楚每个选项的意义得花相当的时间和精力。建议先参考某个预定义目标板配置文件，然后在此基础上进行修改。 P 预定义目标板配置文件存放在 l i n u x / a r 目录下的 / p

首先，执行如下命令，清除以前的配置和编译结果。注意要先备份自己的配置文件 。

```
# m a k e A R C H = p p c _ 8 5 x x
```

然后再执行如下命令配置内核选项：

```
# m a k e A R C H = p p c _ 8 5 x x
```

其中 A 变量用来选择处理器架构，A 取值为 H l i n 下的目录名 a 不同的处理器架构具有不同的配置菜单。C R O S S 变量用来选择工具程序的 L E 内核源码树里的 M a k 会根据 C R O S S 来产生工具程序的名称，例如上述命令产生的 C 编译器名称为 p p c _ 8 5 x x - g c c

4 编译内核

配置好内核后，就可以使用如下命令编译内核映像，这与 L i n 内核需要先执行 m a k 建立源码依赖关系有所不同。

```
# m a k e C R O S S _ C O M P I L E = p p c _ 8 5 x x -
```

生成的内核文件位于 l i n u x / b o o t 目录下，包括二个文件：

u l : m a k 用 e 产生的内核，能被 U - 加载 o o t

v m l i n 用 u g x 压缩的内核

v m l i : n 未压缩的内核 i n

4 编译模块

使用如下命令编译内核模块：

```
# m a k e A R C H = p p c _ 8 5 x x
```

如果要编译自己的驱动模块， i n u 与 i n 有所不同。. L4 i n 只需内核头 . 足文件即可，而 L i n 需要事先准备一个 6 内核源代码树，并配置内核编译选项。可参考如下步骤将自己的驱动代码加入内核源代码树：

① 在内核源代码树下建立自己的驱动程序目录： l i n u x / d 并将自己 v e 的驱动程序拷贝到该目录下。

② 在 l i n u x / d 中加一 v 行：

```
o b j - m + =
```

这样在进行模块编译时就会进入 l i n 目录。/ s / m y d r v P R 指定，F 例如 “ X m a k e P R

③ 在 l / i n u x / d r i e m s n / t m / y i d h ” i v t / r M d a k i e n f s i t l a 写入如下两行 g

```
m y d r v _ m o d u l e
```

m f o i n d e t 是 f 文件的名字，2 可以有许多个。

④ 在 l 目录下执行

```
m a k e A R C H P I L E = p p c _ 8 5 x x
```

即可生成自己的驱动模块 m y d r d u . l e . k o

5创建根文件系统

5 建立系统目录

根文件系统的顶层目录包括 g s b / i n d e v e t c h o m e s b i n ,t 其中 p b i n u s r l i b p r 是必不可少的，其余的目录可以省略。建议保留所有目录，除非你确信不需要，因为在嵌入式系统中即使保留不用的空目录也没有太大的影响，而过度简化目录，可能危害到某些软件的运行。可以使用如下脚本建立系统目录 o n f i g l i b

```
m k d i r / m n t / i
c h O b j P a t h = / m n t
c d $ O b j P a t h
m k d e v d i r n d e v
m k d e v c o n s o l e c 5 1 2 2 1
m k d e v t t y 0 c 4 4 1
m k d e v t t y S 0 c 4 4 6 4
m k d e v n u l l c 1 3 3
m k d i r p r o c s y s t m p # / u s r / e l d k / u s r / b i n
c h m o d 1 7 7 7 $ O b j P a t h / t m p p c _ 8 5 x x / l i b / l i b
m k d i r / m n t / u s r / l i b
m k d i r / m n t / v a r / l i b
m k d i r / m n t / v a r / r u n
c h m o d 1 7 7 7 $ O b j P a t h / t m p p c _ 8 5 x x / l i b / l i b
```

4 建立好目录后，就可以将其他组件安装到相应的位置。

5A 定制系统命令 H = p p c _ 8 5 x x -

Linux提供的命令非常多，尽管你可以从网上找到所有命令的源码，进行交叉编译，生成所需的命令，但这个过程相当冗长，并且生成的命令集会占用较大的存储空间。建议从 h t t p : / 下载 B u 套件，以 B u s y b o x 为基础构建 L i 命令集。u x

B u 是一个极小型的应用程序，却提供了大多数 L i 命令，并且用户还可以根据需要对命令进行定制，删除不需要的命令，进一步减小应用程序体积。s B u s y B o x 并不支持所有的命令选项，但它提供的子集足以满足大多数应用的要求。 B u 的用法与 L i 内核十分相似，只需下面三步：

```
# m a k e m e n u c o n f i g
# m a k e
# m a k e i n s t a l l
```

执行 “ m a k e ” 可以对项目进行配置，设定交叉编译工具，定制需要的命令，最后生成 “ . c ” 配置文件。r 执行 “/ M a k 可以生成 “ i b u s y b o x ” 可执行文件。执行 “ m a k e ” 可以将生成的命令安装到/ _ i n 目录。如果要 l l 将生成的命令安装到指定目录，可以用变量 P R 指定，例如 “ m a k e P R = / t m p / y i d h ” 。

如果 B u 不能提供需要的功能，可以从网上下载相应的套件，进行交叉编译，生成所需命令。另外，还可以到 l / e d . p o f f 目录下找找，看看 B u s y B o x 有没有提供这些命令。例如 M 工具，既可以从 / o p t / e l d k 下直接拷贝，也可以从 h t t p : / / 下载 G C C 源码，进行交叉编译，生成 m k d 工具。D e s 5 v 拷贝链接库。

因为我们使用 E 开发套件，所以可以直接拷贝 E 提供的链接库。 E 将 D K 常用的链接库放在 / o p t / e l d k / 目录下，而将不常用的链接库放在 / o p t / p p c _ 8 5 x x / 下。/ 8 5 x x / 下包括动态链接库及其符号链接，还有静态链接库和其他类型的文件，实际上我们只需拷贝动态链接库及其符号链接即可。执行如下命令可以拷贝所有链接库及其符号链接：

```
# c p - d p R * . s o * $ O b j P a t h / l i b
```

当然，你可能并不需要所有的库文件。如果只想拷贝需要的库文件，可以使用 E 提供的 l 工具，例出目标板二进制文件依赖哪些链接库。例如执行如下命令：

```
u s b s p y b o x
```

会显示：

```
v a r / l o c k / s u b s y s / l o c k . g
l i b m . s o . 6 = > / l i b / l i b m . s o . 6
l i b c . s o . 6 = > / l i b / l i b c . s o . 6
l d . s o . 1 = > / o p t / e l d k / u s r / l i b / l d . s o . 1
```

这样就可以知道 b u s y b o x 依赖的库文件 。

u s 为了减小链接库的尺寸，可以使用 s t r i p 工具对链接库进行处理，对应用程序也可以用该工具进行处理。例如：

```
# / o p t / e l d k / u s r / b i n / a r m - l i n u x - s t r i p $ O b j P a t h / l i b / * . s o
```

5 建立 d 条目 v

B u s y B o x 的 l i n u x / D o c u m e n t a t i o n / 文档提供了设备条目的有关信息。你可以通过此文档查阅某个设备条目的类型、主设备号和次设备号。

o n f i g 的 / d e v 目录下有非常多的条目，但对于嵌入式系统来说，只需要建立让系统正常工作的必要条目即可。我的嵌入式系统中只包含如下条目：

以上条目，除了 mtd 和 mtdblock 外，其他的在 Linux 的 /dev 目录下都有，直接拷贝即可，注意拷贝要使用 -d 选项。例如：

```
# cp -dpR /dev/console
```

mtd 和 mtdblock 条目，可以由 MTD 工具建立，也可以用 mknod 手工建立，例如：

```
mknod -m 644 c 90 0
```

5 系统初始化

Linux 内核的最后一个初始化动作就是启动 init 程序。BusyBox 提供的 Bio 功能特别适合在嵌入式系统中使用。下面了解一下 BusyBox 的 init 工作流程：

① 为 init 设置信号处理程序。

② 初始化控制台。

③ 剖析 /etc/inittab 文件。

④ 执行系统初始化命令行。缺省情况下使用 /etc/init.d/rcS 命令行。

⑤ 执行所有会导致 init 暂停的 inittab 命令（动作类型：wait）。

⑥ 执行所有仅执行一次的 inittab 命令（动作类型：once）。

完成上述工作后，init 进程便会循环执行以下工作：

⑦ 执行所有终止时必须重新启动的 inittab 命令（动作类型：respawn）。

⑧ 执行所有终止时必须重新启动，但启动前必须先询问用户的 inittab 命令（动作类型：askfirst）。

由上可知，init 的初始化过程主要是执行 /etc/ 和 /inittab 这两个文件中的命令行。因此，我们的主要任务是编写 /etc/ 和 /inittab 这两个文件。

先看一下 inittab 文件的语法格式，inittab 文件中每一行的格式如下：

```
<id>:<runlevels>:<action>:<process>
```

其中 id 字段用来指定启动进程的控制台，对于 BusyBox 的 init 而言，该字段被忽略，可以不填。action 字段指明动作类型，BusyBox 支持的动作类型有：sysinit、respawn、askfirst、wait、once、restart、ctrlaltdel 和 shutdown，其含义如表 1 所示。process 字段指明要执行的进程及其命令行。

我的 inittab 是这样写的：

```
::sysinit:/etc/init.d/rcS
::respawn:/sbin/getty -L ttyS0 115200 vt100
::restart:/sbin/init
::ctrlaltdel:/sbin/reboot
::shutdown:/bin/umount -a -r
```

关于 inittab 更详细的说明，可以参考 busybox/examples/inittab 文档。

/etc/init.d/rcS 一般是一个脚本文件，用于初始化需要特别处理的系统组件。例如安装文件系统、启动网络接口、加载驱动模块等等。以下是我的 rcS 文件内容：

```
/bin/mount -a
/sbin/ifconfig eth0 192.168.1.30 netmask 255.255.255.0
```

最后，应为文件 /etc/init.d/rcS 增加可执行属性。

5 生成根文件系统映像

嵌入式系统中通常使用 initrd 机制为内核提供根文件系统。在宿主机上制作目标板根文件系统压缩映像的过程如下：

首先为根文件系统建立一个空的文件系统映像：

```
# dd if=/dev/zero of=/tmp/ramdisk.img bs=1k count=32768
# /sbin/mke2fs -F -v -m0 /tmp/ramdisk.img
# mount -o loop /tmp/ramdisk.img /mnt/initrd
```

上述 dd 命令首先建立了一个 32768 的 8KB 文件系统映像 ramdisk，并用 /dev/zero 对它进行初始化，然后 mke2fs 在其上建立文件系统，最后将其 mount 到 /mnt/initrd 上。

表 1 .BusyBox init 支持的 inittab 动作类型

sysinit	init 首先执行的进程。只有所有 sysinit 进程执行完成，才会执行其他进程。
wait	init 要等该进程执行完成，才会继续执行。
once	init 只执行该进程一次，并且不必等待进程执行完成，就会继续执行。
restart	当 init 重新启动时执行的进程，通常就是 init 本身。
ctrlaltdel	当按下 CTRL-ALT-DEL 组合键时，init 执行的进程。
shutdown	init 退出前执行的进程。
respawn	进程退出时，init 会自动重新启动该进程。
askfirst	此类进程与 respawn 进程类似，只是重新启动时 init 会要求用户先按回车键。

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

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

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

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

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

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

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