基于 S3C6410 的 u-boot 分析与移植

周健昌，李振兴
（暨南大学 信息科学与技术学院，广东 广州 510632）

摘要：Bootloader 的设计是整个嵌入式系统中开发中一个非常重要步骤，u-boot 是当前比较流行、功能强大的开源 Bootloader。文中分析了 u-boot 的启动流程并详细阐述了将其移植到当前应用十分广泛的基于高性能处理器 ARM11 的 tq6410 板的过程，为从更高抽象层使用目标板作充分准备。

关键词：ARM11；S3C6410；TQ6410；u-boot；移植
中图分类号：PT33　　**文献标识码**：A　　**文章编号**：1674-6236(2012)17-0053-04

U-boot analysis and transplantation based on S3C6410

ZHOU Jian-chang，LI Zhen-xing
（*Jinan University School of Information and Technique*，*Guanzhou* 510632，*China*）

Abstract: Development of Bootloader is the very important part of the embeded system，u-boot is a popular and powerful open-source Bootloader. This article analyses the booting process of u-boot and the porting process of which to the board tq6410 that is a wide range of application and based on the high performance processor ARM11，make a full preparation for the use of the target board from a higher abstract layer.

Key words: ARM11；S3C6410；TQ6410；u-boot；transplant

Bootloader 是指系统加电启动后，在操作系统内核运行以前运行的一段小程序，它是系统加电运行的第一段软件代码[1]。通过 Bootloader，我们可以初始化硬件设备，给操作系统提供所在载体的硬件信息，从而将系统的软硬件环境带到一个合适状态，以便为最终调用操作系统内核准备好正确的环境。

由于 Bootloader 的设计依赖于 CPU 的体系结构和具体硬件配置，其设计需满足量体裁衣、定身制作的原则，以满足最小化代码的要求，也正因这种强依赖关系决定了开发通用 Bootloader 是几乎不可能的[2]。从可用资源和实际产品开发周期角度考虑，移植已有的 Bootloader 源码来解决这一问题符合大多数项目的开发要求。

S3C6410 是韩国三星推出的一款 ARM11 处理器，与较上一代的基于 ARM9 的 S3C2440 相比，其性能更强，其广泛应用于通信领域，通用视频处理，智能终端等领域，是近年来十分流行的一款嵌入式处理器。广州天嵌公司利用 S3C6410 制成的嵌入式开发板 TQ6410 则是目前国内公认的性价比较高的开发板。笔者结合 u-boot 源码及 TQ6410 嵌入式开发板，介绍了 u-boot 的运行流程并详细介绍其移植到 S3C6410 的步骤，为后续嵌入式平台开发提供良好的基础。

1 u-boot 简介

如前述，由于 Bootloader 的实现依赖于 CPU 的体系结构，大多数 Bootloader 分为两个阶段，u-boot[3]也采用这种阶段两分的方法来实现引导，目的在于最大限度地将对硬件的依赖性减至最低：

阶段一：

执行与 CPU 体系结构紧密相关的代码，同时也因为此阶段还未建立堆栈（准确而言，是在本阶段最后一步建立堆栈后就跳到第二个阶段），要求代码短小精悍，因而这一阶段的代码几乎都要使用汇编代码。它依次完成以下工作[4]：

1）硬件设备初始化：通常而方是屏蔽所有中断，设置 CPU 速度和时钟频率、RAM 初始化、LED 初始化、关闭 CPU 内部的指令、数据缓存；

2）为加载 Bootloader 的第二个阶段代码准备 RAM 空间；

3）拷贝 Bootloader 的第二个阶段代码到 RAM 空间中；

4）建立堆栈，为以后的 C 语言代码的运行作好准备；

5）通过修改 PC 寄存器，跳转到第二阶段的 C 程序入口。

阶段二：

在此阶段执行的代码主要都是用 C 语言编写的，不但可以实现更复杂的功能，而且具有更好的可读性和可移植性。它基本依次完成如下工作[4]：

1）初始化本阶段要用到的硬件；

2）检测系统内存映射；

3）将 kernel 映像和根文件系统映像从 Flash 上读到 RAM 空间中；

4）为内核设置启动参数；

5）直接跳转到内核的第一条指令处，以调用内核。

从用到的源文件来看来，第一阶段主要涉及到 2 个文件：

arch/arm/cpu/arm1176/start.S

board/{开发板目录}/lowlevel_init.S

前者与平台相关，后者与开发板相关。

第二阶段则主要涉及 lib_arm/board.c，它从 start_armboot 函数开始，进行一系列设置后，最终进入 main_loop 函数的死循环，等待输入命令。

2 移植 u-boot 的软硬件环境

笔者的目标平台，采用 ARM11 系列架构，版本为 ARM1176JZF-S 内核的 S3C6410 芯片。S3C6410 支持 32 位 ARM 指令集和 16 位 Thumb 指令集，由 8 级流水线组成，可以比 5 级流水线的 ARM9 达到更高的运行频率，同时支持 Nor Flash 和 Nand Flash 两种启动方式[5]。

使用的开发板是天嵌公司生产的 TQ2440 开发板，是目前国内较流行的基于 S3C6410 的 ARM11 开发平台，也是国内公认的性价比较高的，其微处理器采用 Samsung S3C6410XH-66，ARM1176JZF-S 内核，板载 128MBs Mobile DDR、256 MB Nand Flash 、2 MB Nor Flash，板载 5 线异步串行口（UART0）、100Mb/s DM9000 网卡、USB HOST 接口、USB Device 接口和一个 SD 卡接口，集成了 4 线电阻式触摸屏接口和 JTAG 接口等[6]。

本文测试环境使用了直接在 Linux 系统（RHEL 6.1，内核版本为 2.6.32）下，以 arm-linux-gcc-4.3.2 作编译工具链，其中包括了编译器、链接器、汇编器等开发工具。使用的 u-boot 是当前最新版的 u-boot-2012.04。

3 移植步骤

1）建立相关目录与文件

在 board\samsung 建立 S3C6410 的目录，并且把目录 smdk6400 中的所有文件复制过去，重命名文件 smdk6400.c 为 s3c6410.c，并修改同一目录下 Makefile：将 smdk6400.o 改为 s3c6410.o[7]。

同时，在 include\configs\ 目录中创建一个新文件 s3c6410.h，将 smdk6400.h 内容拷贝过去。

2）为相应 CPU 体系修改相应文件

a）添加外设存储的操作

arch\arm\cpu\arm1176\s3c64xx\cpu_init.S 后面加入如下的代码[8]：

```
    bne    check_dmc1_ready
    nop
#if defined(CONFIG_TQ6410)
#define SROM_BC1_REG_Tacs     (0x0)
#define SROM_BC1_REG_Tcos     (0x4)
#define SROM_BC1_REG_Tacc     (0xE)
#define SROM_BC1_REG_Tcoh     (0x1)
#define SROM_BC1_REG_Tah      (0x4)
#define SROM_BC1_REG_Tacp     (0x6)
#define SROM_BC1_REG_PMC      (0x0)
#define SROM_BW_REG_DATA  \
  ((1<<7)|(1<<6)|(1<<4))
#define SROM_BW_REG_BC1   (0xf << 4)
#define SROM_BC1_REG_DATA  \
    ((SROM_BC1_REG_Tacs << 28)|\
    (SROM_BC1_REG_Tcos << 24)|\
    (SROM_BC1_REG_Tacc << 16)|\
    (SROM_BC1_REG_Tcoh << 12)|\
    (SROM_BC1_REG_Tah << 8)|\
    (SROM_BC1_REG_Tacp << 4)|\
    (SROM_BC1_REG_PMC))

  ldr    r0, =ELFIN_SROM_BASE
  ldr    r1, [r0, #SROM_BW_REG_DATA]
  mov    r2, #(~SROM_BW_REG_BC1)
  and    r1, r1, r2
  mov    r2, #SROM_BW_REG_DATA
  orr    r1, r1, r2
  str    r1, [r0, #INDEX_SROM_BW_REG]
  ldr    r1, =SROM_BC1_REG_DATA
  str    r1, [r0, #INDEX_SROM_BC1_REG]
#endif
  mov    pc, lr
```

b）存储控制相关修改

首先在 arch\arm\include\asm\arch- s3c64xx\s3c6400.h 中宏 ELFIN_SROM_BASE 的定义（其定义值为 0x70000000 是 SROM 总线宽度和等待控制寄存器的地址）后加入以下代码，其作用是为 SROM 控制器的操作提供基础：

```
#define INDEX_SROM_BW_REG     0x0
#define INDEX_SROM_BC0_REG    0x4
#define INDEX_SROM_BC1_REG    0x8
#define INDEX_SROM_BC2_REG    0xC
#define INDEX_SROM_BC3_REG    0x10
#define INDEX_SROM_BC4_REG    0x14
#define INDEX_SROM_BC5_REG    0x18
```

c）时钟频率的修改

因 6410 和 6400 的资源差不多，而主频和外设有点差别。具体而言，S3C6410 里包含 3 个 PLL（锁相环），APLL，MPLL，EPLL，通过设置它们将输入时钟同步输出达到操作 CPU 的工作频率的目的。

根据 S3C6410 硬件手册，ARMCLK 和 HCLK 的时钟比必须是整数，用来同步 ARM 内核和 AXI 总线接口二者的时钟。S3C6410 没有对 CPU 工作在 533 MHz 做限制，不过，对于工作在 533 MHz 以上时，例如 667 MHz 时，只能支持 1:2.5:5

的时钟比（ARMCLK=667 MHz,HCLKX2=266 MHz,HCLK=133 MHz)。

幸运的是,在修改 u-boot 的源码时,我们无需进行过多的计算，而只要修改一些宏的有效性即可，可对 include\configs\s3c6410.h 修改如下：

①换“#define CONFIG_CLK_533_133_66”为“#define CONFIG_CLK_667_133_66”。

②在 include\common.h 中的下列条件编译语句加入条件 defined(CONFIG_S3C6410)：

```
#if defined(CONFIG_S3C24X0) || \
    defined(CONFIG_LH7A40X) || \
    defined(CONFIG_S3C6400) || \
    defined(CONFIG_EP93XX)
```

3)为目标板修改相应文件

a)加入新的宏定义

在 board\samsung\S3C6410\lowlevel_init.S 中,为原条件编译项“#ifndef CONFIG_S3C6400”加入新条件 CONFIG_S3C6410,即将其改为“#if ! defined(CONFIG_S3C6400) && ! define(CONFIG_S3C6410)”

同样,将同一文件中其 else 部分“elif ! defined(CONFIG_S3C6400)&&! defined(CONFIG_S3C6410)”改为

#elif ! defined(CONFIG_S3C6400)&&! defined(CONFIG_S3C6410)。

从代码的注释我们可以知道,这是因为这些条件编译分支所作用的代码对以往的非 S3C64XX 的开发板是无条件的,而且是有作用的,然而这些代码对 S3C64XX 的开发板没有必要,因为加入上述条件。

b)网卡的移植

由于目标板 S3C6410 用的网卡是 dm9000 而不是 CS8900，因此把有关 CS8900 的宏定义及函数删除，并加入 dm9000 相应的内容,具体内容如下：

在 board\samsung\s3c6410\s3c6410.c 进行如下修改：

①屏蔽无关函数定义:cs8900_pre_init、board_flash_get_legacy,并在 board_init 中注释掉它们的调用。

②将函数 virt_to_phy_smdk6400 更名为 virt_to_phy_tq6410。

③将 board_eth_init 中关于网卡初始化的函数:

```
#ifdef CONFIG_CS8900
    rc = cs8900_initialize(0, \
    CONFIG_CS8900_BASE);
#endif
```

替换为：

```
#if defined CONFIG_DRIVER_DM9000
    rc = dm9000_initialize(bis);
#endif
```

④此外,还应在 drivers\net\dm9000x.c 中,代码：

```
DM9000_ior(DM9000_MRCMDX);
  rxbyte = DM9000_inb(DM9000_DATA) & 0x03;
```

后添加如下永真分支操作：

```
#if 1
u8   temp; temp=DM9000_ior (DM9000_MRRH); temp=DM9000_ior (DM9000_MRRL);
#endif
```

⑤将其中的关于 cs8900 网卡的 3 个“#define CONFIG_CS8900_XXX”替换成：

```
#define CONFIG_NET_MULTI          1
#define CONFIG_DRIVER_DM9000      1
#define CONFIG_DM9000_NO_SROM     1
#define CONFIG_DM9000_USE_16BIT   1
#define CONFIG_DM9000_BASE  0x18000300
#define DM9000_IO(CONFIG_DM9000_BASE)
#define DM9000_DATA  \
            (CONFIG_DM9000_BASE+4)
```

再根据实际环境定义以下宏（基本可知名识义）：CONFIG_ETHADDR、CONFIG_NETMASK、CONFIG_IPADDR、CONFIG_SERVERIP、CONFIG_GATEWAYIP。

c)添加 NAND 启动功能

u-boot 一般从 ROM、NOR Flash 等设备启动，现已可从 NAND Flash 启动,但是支持的 CPU 种类还不多。所谓 SPL,即 Second Program Loader。为使目标板支持 NAND 启动功能,我们作如下修改：

①nand_spl\board\samsung\s3c6410\config.mk 中未被注释(非以“#”开头的行)内容修改为：

```
include $(TOPDIR)/board/$(BOARDDIR)/config.mk
PAD_TO := $(shell expr $(TEXT_BASE) + 4096)
ifeq ($(debug),1)
    PLATFORM_CPPFLAGS += -DDEBUG
endif
```

②nand_spl\board\samsung\s3c6410\Makefile 中：

```
# from board directory
$(obj)lowlevel_init.S:
   @rm -f $@
   @ln -s      \
   $(TOPDIR)/board/samsung/smdk6400/lowlevel_init.S $@
```

改为：

```
# from board directory
$(obj)lowlevel_init.S:
   @rm -f $@
    @ln -s    \
    $(TOPDIR)/board/samsung/s3c6410/lowlevel_init.S $@
```

4)为得出正确编译结果修改相应文件

a)高层选项设置

对 include\configs\s3c6410.h 进行修改，主要是注释

CONFIG_S3C6400 的宏定义，并于同处将 CONFIG_S3C6410 宏 定义为 1，即：

```
//#define CONFIG_S3C6400        1
#define CONFIG_S3C6410          1
```

b）为使 u-boot 支持在目标板上使用 FLASH，应该作出如下修改：

将“#define CONFIG_SYS_FLASH_CFI 1”替换为“#define CONFIG_SYS_NO_FLASH 1”

然后，将以下宏的定义注释，使其失效：

CONFIG_FLASH_CFI_DRIVER、
CONFIG_SYS_FLASH_CFI_WIDTH、
CONFIG_FLASH_CFI_LEGACY、
CONFIG_SYS_FLASH_LEGACY_512Kx16、
ONFIG_SYS_FLASH_ERASE_TOUT、
CONFIG_SYS_FLASH_WRITE_TOUT。

c）修改编译选项生成必要的目标文件

这个操作通常是在相关的 Makefile 或 config 文件中加入编译规则 COBJS-XXX 来生成必要的目标文件，如下所示：

①在 board\samsung\s3c6410\Makefile 中加入编译选项：COBJS-y := s3c6410.o。

② arch\arm\cpu\arm1176\s3c64xx\Makefile 中位于“COBJS-$(CONFIG_S3C6400)+= cpu_init.o speed.o”后加入：COBJS-$(CONFIG_S3C6410)+= \cpu_init.o speed.o

d）修改主根录下 Makefile 和 MAKEALL 文件，加入对板子的申明

①因为我们使用的是 TQ6410 目标板，所以在顶层的 MAKEALL 中的 LIST_ARM11 中加入“s3c6410”

②在根目录下的 Makefile 中为目标板添加新的编译规则：

```
s3c6410_config:unconfig
@$(MKCONFIG) $(@: _config = ) arm \
 arm1176 s3c6410 samsung s3c64xx
```

其中 arm 指 cpu 体系结构，arm1176 指 cpu 体系，S3c6410 是目标主板对应的目录，S3c64x0 指片上系统。编译后会自动在/include/下产生 config.mk 文件，内容如下：

```
ARCH = arm              BOARD = S3c6410
CPU = arm1176           SOC = S3c64x0
```

5）编　译

```
#make s3c6410_config
#make
```

编译成功生成 u-boot.bin。

6）烧　写

把编译生成 u-boot.bin 烧写到目标板 flash 中。

至此，移植过程结束。

4　结束语

Bootloader 是连接硬件和操作系统的桥梁，为后续开发提供坚实的基础。功能强大的 Bootloader 通常凝聚了操作系统的必不可少功能。文中在分析 u-boot 的基本启动流程后，通过详细的步骤分析了基于 S3C6410 的 u-boot 移植实例。目前笔者移植的 u-boot 已在目标板上结合 Linux 系统稳定运行，从实践上已经证明本文所介绍方法的正确性，也希望本文对尚在探索的开发人员能带来一点帮助。

参考文献：

[1] 章健，袁义江. u-boot在ARM平台上的移植及应用[J]. 微计算机信息，2007(8):137-138.
ZHANG Jian，YUAN Yi-jiang. Transplant and application of u-boot based on ARM system[J]. Microcomputer Information，2007(8):137-138.

[2] 黄荐渠，秦东兴，赵曦，等. u-boot的启动及移植分析[J]. 微计算机信息，2008(24):76-78.
HUANG Jian-qu，QIN Dong-xing，ZHAO Xi，et al. The booting and porting analyze of u-boot[J]. Microcomputer Information，2008(24):76-78.

[3] 李向阳，戴学丰，邵林. u-boot在S3C44B0上的移植与分析[J]. 现代电子技术，2009(6)：31-33，37.
LI Xiang-yang，DAI Xue-feng，SHAO Lin. U-Boot transplant and analysis in S3C44B0X[J]. Modern Electronic Technique，2009(6)：31-33，37.

[4] 申爽. 基于S3C2440的u-boot分析与移植[J]. 计算机系统应用.2012，21(5):222-225.
SHEN Shuang. U-boot analysis and transplantation based on S3C2440[J]. Computer Systems and Applications，2012，21(5):222-225.

[5] Sumsuang Electronics. S3C6410X user's manual (Rev 1.20)[R]. Korea:Sumsuang Electronics，2009:1-1371.

[6] 广州天嵌计算机科技有限公司. 基于TQ6410的Linux使用教程(V0.1)[R]. 广州天嵌计算机科技有限公司，2010.

[7] 李夏，王化深，易海旺. u-boot在S3C2410上的移植分析[J]. 微计算机应用，2006，27(5)：596-600.
LI Xia，WANG Hua-sen，YI Hai-wang. Analyzing the transplantation of u-boot on S3C2410[J]. Microcomputer Information，2006，27(5)：596-600.

[8] Neddy11.S3C6410移植u-boot[EB/OL]. (2011-07-16)[2012-06-23].http:/z/www.cnblogs.com/Neddy/archive/2011/07/16/2108286.htm.

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
44. 基于 PCI 总线的 MPEG2 码流播放卡驱动程序开发
45. 基于磁盘异或引擎的 RAID5 小写性能优化
46. 基于 IEEE1588 的时钟同步技术研究
47. 基于 Davinci 平台的 SD 卡读写优化
48. 基于 PCI 总线的图像处理及传输系统的设计
49. 串口和以太网通信技术在油液在线监测系统中的应用
50. USB30 数据传输协议分析及实现
51. IEEE 1588 协议在工业以太网中的实现
52. 基于 USB30 的设备自定义请求实现方法
53. IEEE1588 协议在网络测控系统中的应用
54. USB30 物理层中弹性缓冲的设计与实现
55. USB30 的高速信息传输瓶颈研究
56. 基于 IPv6 的 UDP 通信的实现
57. 一种基于 IPv6 的流媒体传送方案研究与实现
58. 基于 IPv4-IPv6 双栈的 MODBUS-TCP 协议实现
59. RS485CAN 网关设计与实现
60. MVB 周期信息的实时调度
61. RS485 和 PROFINET 网关设计
62. 基于 IPv6 的 Socket 通信的实现
63. MVB 网络重复器的设计

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
25. WindML 中 Mesa 的应用
26. VxWorks 下图形用户界面开发中双缓冲技术应用
27. VxWorks 上的一种 GUI 系统的设计与实现
28. VxWorks 环境下 socket 的实现
29. VxWorks 的 WindML 图形界面程序的框架分析
30. VxWorks 实时操作系统及其在 PC104 下以太网编程的应用
31. 实时操作系统任务调度策略的研究与设计
32. 军事指挥系统中 VxWorks 下汉字显示技术
33. 基于 VxWorks 实时控制系统中文交互界面开发平台
34. 基于 VxWorks 操作系统的 WindML 图形操控界面实现方法
35. 基于 GPU FPGA 芯片原型的 VxWorks 下驱动软件开发
36. VxWorks 下的多串口卡设计
37. VxWorks 内存管理机制的研究
38. T9 输入法在 Tilcon 下的实现
39. 基于 VxWorks 的 WindML 图形界面开发方法
40. 基于 Tilcon 的 IO 控制板可视化测试软件的设计和实现
41. 基于 VxWorks 的通信服务器实时多任务软件设计

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
33. LINUX ARM 下的 USB 驱动开发
34. Linux 下基于 I2C 协议的 RTC 驱动开发
35. 嵌入式下 Linux 系统设备驱动程序的开发
36. 基于嵌入式 Linux 的 SD 卡驱动程序的设计与实现
37. Linux 系统中进程调度策略
38. 嵌入式 Linux 实时性方法
39. 基于实时 Linux 计算机联锁系统实时性分析与改进
40. 基于嵌入式 Linux 下的 USB30 驱动程序开发方法研究
41. Android 手机应用开发之音乐资源播放器
42. Linux 下以太网的 IPv6 隧道技术的实现
43. Research and design of mobile learning platform based on Android
44. 基于 linux 和 Qt 的串口通信调试器调的设计及应用

45. 在 Linux 平台上基于 QT 的动态图像采集系统的设计
46. 基于 Android 平台的医护查房系统的研究与设计
47. 基于 Android 平台的软件自动化监控工具的设计开发
48. 基于 Android 的视频软硬解码及渲染的对比研究与实现
49. 基于 Android 移动设备的加速度传感器技术研究
50. 基于 Android 系统振动测试仪研究

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
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
19. 基于 WinCE 的嵌入式图像采集系统设计
20. 基于 ARM 与 WinCE 的掌纹鉴别系统
21. DCOM 协议在网络冗余环境下的应用
22. Windows XP Embedded 在变电站通信管理机中的应用
23. XPE 在多功能显控台上的开发与应用
24. 基于 Windows XP Embedded 的 LKJ2000 仿真系统设计与实现
25. 虚拟仪器的 Windows XP Embedded 操作系统开发
26. 基于 EVC 的嵌入式导航电子地图设计

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
15. 基于 PowerPC 嵌入式内核的多串口通信扩展设计
16. 基于 PowerPC 的多网口系统抗干扰设计
17. 基于 MPC860T 与 VxWorks 的图形界面设计
18. 基于 MPC8260 处理器的 PPMC 系统
19. 基于 PowerPC 的控制器研究与设计
20. 基于 PowerPC 的模拟量输入接口扩展
21. 基于 PowerPC 的车载通信系统设计
22. 基于 PowerPC 的嵌入式系统中通用 IO 口的扩展方法
23. 基于 PowerPC440GP 型微控制器的嵌入式系统设计与研究
24. 基于双 PowerPC 7447A 处理器的嵌入式系统硬件设计

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
22. ARM CortexM3 处理器故障的分析与处理
23. ARM 微处理器启动和调试浅析
24. 基于 ARM 系统下映像文件的执行与中断运行机制的实现
25. 中断调用方式的 ARM 二次开发接口设计
26. ARM11 嵌入式系统 Linux 下 LCD 的驱动设计
27. Uboot 在 S3C2440 上的移植
28. 基于 ARM11 的嵌入式无线视频终端的设计
29. 基于 S3C6410 的 Uboot 分析与移植
30. 基于 ARM 嵌入式系统的高保真无损音乐播放器设计
31. UBoot 在 Mini6410 上的移植
32. 基于 ARM11 的嵌入式 Linux NAND FLASH 模拟 U 盘挂载分析与实现
33. 基于 ARM11 的电源完整性分析

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
17. 基于 UEFI 的可信 BIOS 研究与实现
18. 基于 UEFI 的国产计算机平台 BIOS 研究
19. 基于 UEFI 的安全模块设计分析
20. 基于 FPGA NiosⅡ的等精度频率计设计
21. 基于 FPGA 的 SOPC 设计
22. 基于 SOPC 基本信号产生器的设计与实现
23. 基于龙芯平台的 PMON 研究与开发
24. 基于 X86 平台的嵌入式 BIOS 可配置设计
25. 基于龙芯 2F 架构的 PMON 分析与优化
26. CPU 与 GPU 之间接口电路的设计与实现
27. 基于龙芯 1A 平台的 PMON 源码编译和启动分析
28. 基于 PC104 工控机的嵌入式直流监控装置的设计
29. GPGPU 技术研究与发展
30. GPU 实现的高速 FIR 数字滤波算法
31. 一种基于 CPUGPU 异构计算的混合编程模型
32. 面向 OpenCL 模型的 GPU 性能优化

Programming:

1. 计算机软件基础数据结构 - 算法
2. 高级数据结构对算法的优化
3. 零基础学算法
4. Linux 环境下基于 TCP 的 Socket 编程浅析
5. Linux 环境下基于 UDP 的 socket 编程浅析
6. 基于 Socket 的网络编程技术及其实现
7. 数据结构考题 - 第 1 章 绪论
8. 数据结构考题 - 第 2 章 线性表
9. 数据结构考题 - 第 2 章 线性表 - 答案