PaperPass检测报告简明打印版

比对结果（相似度）：
总　体：20 % *（总体相似度是指本地库和网络库的综合比对结果）*
本地库：19 % *（本地库相似度是指论文与学术期刊、学位论文数据库的比对结果）*
网络库：2 % *（网络库相似度是指论文与互联网资源的比对结果）*

编　号：VIP9579F27CD2AB8CD63
标　题：嵌入式系统设计
作　者：何伟强
长　度：25868 字符(不计空格)
时　间：2014-5-30 10:34:40
比对库：学术期刊（1990-2013）、学位论文（硕博库1990-2013）、互联网资源
查真伪：http://www.paperpass.com/check.aspx

相似资源列表（学术期刊、学位论文）：

1. 相似度：1％ 篇名:《基于移动自组织网骨干路由器的嵌入式实时操作系统的开发——任务管理和通信》
   来源：学位论文 《重庆大学》 2005 作者: 李铁军
2. 相似度：1％ 篇名:《一种μC/OS-II中任务调度机制的改进方法》
   来源：学术期刊 《现代计算机（专业版）》 2013年10期 作者: 赵国富等
3. 相似度：1％ 篇名:《μCOS-Ⅱ优先级调度算法改进与微内核研究》
   来源：学位论文 《长春工业大学》 2007 作者: 苑野
4. 相似度：1％ 篇名:《PPP协议在μCOS-Ⅱ操作系统中的设计与实现》
   来源：学位论文 《东北大学》 2005 作者: 刘国满
5. 相似度：1％ 篇名:《柴油机电子控制器嵌入式实时软件系统研究》
   来源：学位论文 《上海交通大学》 2006 作者: 于世涛
6. 相似度：1％ 篇名:《嵌入式RTOS中任务优先级反转问题研究》
   来源：学术期刊 《信息技术》 2006年9期 作者: 李屏等
7. 相似度：1％ 篇名:《嵌入式RTOS中任务调度问题研究》
   来源：学术期刊 《辽宁工程技术大学学报（自然科学版）》 2004年 作者: 冀常鹏等
8. 相似度：1％ 篇名:《基于嵌入式实时操作系统Web服务器的研究与实现》
   来源：学位论文 《北京交通大学》 2004 作者: 蒋智勇
9. 相似度：1％ 篇名:《使用UML对μC/OS-Ⅱ建模与分析》
   来源：学术期刊 《计算机工程与设计》 2008年11期 作者: 胡琰华等
10. 相似度：1％ 篇名:《基于PC的控制系统的可靠性及实时性的研究和实现》
    来源：学位论文 《天津大学》 2003 作者: 周鹏鹏
11. 相似度：1％ 篇名:《基于Windows CE的汽车性能测试系统研制》
    来源：学位论文 《西华大学》 2012 作者: 孙泽海
12. 相似度：1％ 篇名:《关于RTOS抢占式调度及优先级反转的几点探讨》
    来源：学术期刊 《计算机工程与设计》 2007年19期 作者: 宋丰未等

13. 相似度：1％ 篇名:《嵌入式操作系统μC／OS-Ⅱ的一种内存管理算法》
来源：学术期刊 《微电子学与计算机》 2011年11期 作者: 李平勇等
14. 相似度：1％ 篇名:《嵌入式RTOS中就绪任务查找算法和优先级反转的解决方案》
来源：学术期刊 《计算机应用》 2003年6期 作者: 万柳等
15. 相似度：1％ 篇名:《μC/OS-Ⅱ实时系统任务调度优化》
来源：学术期刊 《计算机工程》 2007年19期 作者: 徐亮等
16. 相似度：1％ 篇名:《μ C\OS-Ⅱ操作系统的任务切换》
来源：学术期刊 《数字技术与应用》 2010年6期 作者: 司新生
17. 相似度：1％ 篇名:《实时内核μCOS-Ⅱ在MSP430中的实现与应用研究》
来源：学位论文 《山东理工大学》 2006 作者: 赵士伟
18. 相似度：1％ 篇名:《μCOS-Ⅱ实时操作系统在电动机在线监测系统中的应用与研究》
来源：学位论文 《浙江大学电气工程学院》 2006 作者: 阳跃
19. 相似度：1％ 篇名:《基于ARM的嵌入式系统设计》
来源：学位论文 《电子科技大学》 2007 作者: 郭朗
20. 相似度：1％ 篇名:《嵌入式操作系统》
来源：学术期刊 《科技创新导报》 2007年33期 作者: 刘秋平
21. 相似度：1％ 篇名:《嵌入式实时操作系统μCOS-Ⅱ的研究与应用研究与应用》
来源：学位论文 《南京航空航天大学》 2008 作者: 黄飞飞
22. 相似度：1％ 篇名:《基于μC/OS-Ⅱ内存管理改进方法的研究与实现》
来源：学术期刊 《微型电脑应用》 2008年11期 作者: 刘晋等
23. 相似度：1％ 篇名:《基于FPGA技术的嵌入式微处理器设计研究》
来源：学位论文 《湖北大学》 2009 作者: 何丽平
24. 相似度：1％ 篇名:《优先级反转和死锁的资源管理模式研究与实现》
来源：学术期刊 《计算机工程与设计》 2011年8期 作者: 王溪波等
25. 相似度：1％ 篇名:《具有SDMMC接口的智能卡的实时操作方法的研究和应用》
来源：学位论文 《北京邮电大学》 2007 作者: 沈雅娜
……

相似资源列表（互联网）：

1. 相似度：1％ 标题：《[Docin]嵌入式视频传输系统的应用研究》
http://www.docin.com/p-471709841.html
2. 相似度：1％ 标题：《无标题网页》
http://scholar.google.com/schhp?hl=zh-CN&as_sdt=8b76c84d29de56ee77f71f2d76011e58

全文简明报告：

摘 要

近年来，嵌入式应用越来越广泛，物联网、智能家居、可穿戴式设备等也慢慢走入人们的生活中，嵌入式应用在我们的生活中扮演着越来越重要的角色。

{56％： 嵌入式操作系统是嵌入式应用的核心，编写嵌入式操作系统，能加深对操作系统基本原理的理解和实现。} {44％： 本课题基于 ARM11微处理器的开发板，按照软件工程的基本流程，实现了一个简单可用的小型嵌

入式操作系统，｝｛49％：该小型操作系统具基本的多任务调度、中断服务、时间管理和内存管理功能。｝采用按优先级抢占式的调度机制来进行任务调度，具有较高的实时性，同时使用固定分区法来进行内存分区管理。｛42％：此外，该系统还有任务延时、挂起、删除等基本操作。｝｛43％：最后对该系统进行测试，验证系统设计的正确性，并提出系统的不足和改进的地方。｝

关键词 嵌入式； 操作系统； 多任务调度； 内存管理； 优先级抢占式调度

绪论

1.1 课题来源及研究目的与意义

｛44％：嵌入式开发是当下热门的开发，嵌入式技术普遍的应用于各个领域。｝学习嵌入式开发，大多选择51单片机或者ARM芯片来进行学习开发，由于ARM芯片是大多数嵌入式设备的主要开发平台，｛42％：而且ARM芯片具有性能高、体积小、功耗低等特点，因此学习ARM嵌入式开发具有一定的优势。｝

｛75％：学习嵌入式开发，可大概分为：｝｛67％：嵌入式应用开发、嵌入式操作系统系统开发和硬件开发。｝｛49％：嵌入式应用和嵌入式系统操作系统开发一般都紧密结合，而当下嵌入式操作系统虽多，｝但即使是像μC/OS或者QNX这些小巧的嵌入式操作系统内核，其代码量也是非常多的，｛54％：而像嵌入式Linux的，其代码就更加庞大了。｝对学习操作系统的学生来说，由于缺少对操作系统设计和实现的经验，在没有一个比较清晰的思路和逻辑的情况下，去阅读内核代码是非常徒劳无用的。因此，我们需要自己动手，实际去编写一个嵌入式操作系统，哪怕这个操作系统只能进行简单的任务调度，这也能加深我们对操作系统原理的理解，激发我们的热情和兴趣，继续去为这个内核添加更多的功能。

｛49％：选择设计嵌入式操作系统这个题目，具有重大而深远的意义。｝｛46％：计算机系统里面最重要的部分就是操作系统了，在学习和设计操作系统的过程中，｝｛42％：能加深我们对操作系统基本原理的理解，对计算机系统的工作过程也会更加清晰。｝｛46％：虽然嵌入式操作系统和通用的计算机系统具有一定的差别，但其基本的工作原理都是相同的，｝｛46％：而嵌入式操作系统相对于通用计算机系统，其内核更加小巧，更加适合学习和开发。｝因此本论文所写的小型嵌入式操作系统，其内核功能也是相对简单，但主要还是以加强对操作系统的理解，并运用基本的原理来设计开发一个能正常运行和工作的操作系统。

｛76％：1.2 嵌入式操作系统的现状与分析｝

｛42％：嵌入式系统是从20世纪70年代微处理器出现后发展起来的，当时的系统结构和功能相对单一，主要用于工业控制[1]。｝｛41％：如今，通信、工业控制、消费电子、航空、航天等都存在着嵌入式设备的身影，嵌入式已经普遍的应用于各个领域中。｝近年来，嵌入式技术的发展越来越突飞猛进，移动设备、平板电脑等已成为人们生活中不可或缺的电子产品，现在物联网、智能家居、可穿戴式设备等嵌入式技术也正大力发展中。

｛52％：嵌入式操作系统（Embedded Operating System，EOS）是嵌入式系统的核心部分，与通用计算机操作系统相比，具有专用性、可裁剪、易移植、高可靠等特点。｝如今，已经拥有相当多的主流嵌入式操作系统，比如VxWorks、Windows CE、FreeRTOS、μC/OS-Ⅱ/Ⅲ、嵌入式Linux、eCOS、QNX等等，｛44％：还有应用于移动设备上的操作系统也属于嵌入式操作系统，像Android、iOS、Windows Phone、Firefox OS等等。｝｛50％：这些嵌入式操作系统既有商业付费的，也有开源免费的，都是当前普遍应用于各种设备上的嵌入式操作系统。｝

｛44％：虽然低端的嵌入式设备不一定具备嵌入式操作系统，但大多数嵌入式设备都由嵌入式操作系统来管理

其相应的软件和硬件资源。} 不同的嵌入式操作系统包含不同的基本功能组件，但都具备一个基本的内核，这个内核主要的功能就是提供多任务的管理、时间的管理、事件的管理、内存的管理和设备的管理等， 实现既有简单也有复杂，像μC/OS-Ⅱ的内核就相对简单，而嵌入式Linux就和桌面版的Linux差不多了， 因此功能也相对更强大。

{40%：目前，大多数嵌入式操作系统的技术大都以国外为主，国内嵌入式的发展技术与美国等发达国家相比还存在着一定的差距，} 但总体的发展趋势还是相当好的，像智能家居、可穿戴式设备等国内也搞得很火热。而自主的嵌入式操作系统并没有得到广泛的应用，因此，我国不论是嵌入式系统还是相关的嵌入式技术发展，还是具有比较大的发展空间，国内的嵌入式开发前景还是非常不错的。

1.3 本论文的主要工作

{43%：本论文所设计的小型嵌入式操作系统完成的主要工作和解决的难点有以下：}

查阅国内国外有关嵌入式的文献资料，比如嵌入式发展的历史与现状、应用领域、嵌入式系统的基本设计和实现等。

学习ARM的体系结构和基本汇编，掌握S3C6410的硬件结构和ARM嵌入式的开发，并编写开发板上相应的引导程序和硬件驱动。 其中的主要难点是向开发板的NAND Flash写入操作系统的引导程序，中断服务程序的编写，系统更新程序的编写等。

利用所学操作系统的基本原理，在开发板上实现一个简单的操作系统，该系统采用了按优先级抢占式的调度机制来进行任务调度，并拥有简单的内存管理功能。 {47%：其中设计难点有任务控制块的设计，任务的切换、调度、中断的实现等。}

本系统还实现了基本的时间管理功能，如让任务进行延时等操作，同时还有任务的挂起、恢复、删除等操作。

在本嵌入式操作系统的基础上设计测试应用来验证该系统的功能，并在最后对该系统提出一些能继续进行改进的地方。

ARM技术及开发板介绍

2.1 ARM体系结构

2.1.1 ARM处理器的简介

{48%：嵌入式系统的核心硬件就是嵌入式处理器，嵌入式处理器的体系结构如果按不同的内核系列可以分为51、AVR、MIPS、PowerPC、ARM等，} 像51单片机采用的就是51系列的内核芯片，Arduino采用的就是AVR的芯片，而ARM就是目前最广泛使用的嵌入式处理器芯片。 从1991年到2014年，ARM处理器的历史出货量已经超过了500亿颗，广泛的应用于移动领域、嵌入式领域、企业和家用等市场[2]。

{45%：ARM既是一间公司的名字，也是一系列处理器的名称。} ARM的全称是Advanced RISC Machine，由此可看出ARM处理器的指令集是精简指令集（Reduced Instruction Set Computer，RISC）。 {46%：ARM处理器的

主要特点是功耗低、省电、性能强大、成本低，拥有16位 Thumb、32位 ARM和 Thumb-2双指令集，} 最重要的是ARM拥有许多领域方面的合作伙伴，像苹果、三星、高通等，因此使得 ARM处理器能全面广泛的应用于多个领域。

ARM处理器多数为哈佛结构，拥有16/32位指令集，多处理器状态模式等设计技术。 ARM处理器不同的内核采用不同的体系结构，像经典处理器的 ARM7采用的是 ARMV4 T的结构，Cortex- A采用的是 ARMV7- A的结构，具体可到 ARM公司的官网查看 ARM内核采用的体系结构版本。 {44 %：目前市场上主要使用的内核有ARM7、ARM9的老架构，还有近年来比较流行的Cortex-A和Cortex-M系列的架构芯片。}

{63 %： 2.1.2 ARM处理器的工作状态和工作模式}

{47 %： ARM处理器一般拥有2种工作状态和7种工作模式，因为 ARM处理器拥有16/32位指令集，} {52 %：因此 ARM处理器可以在 ARM和 Thumb及 Thumb-2这两种工作状态间切换。} ARM状态就是ARM处理器完全工作在32位指令下的状态，在此状态下的指令长度均为32位。 Thumb状态就是工作在16位指令下的状态，这时的指令代码只有16位，占用的内存空间小，代码密度大，能提供比32位程序代码更好的性能。 {45 %：Thumb-2状态是ARM处理器新的状态，首次出现在ARM11系列的芯片。} Thumb-2同时具有16位和32位的指令，能提供更高性能、功耗更小和占用内存更小的优点。 {51 %：ARM处理器复位后只处于ARM状态，可通过相应的指令集在ARM状态和Thumb状态下进行切换。}

{72 %：ARM有7种工作模式，分别是用户 模式、快速 中断 模式、外部 中断 模式、管理 模式、中止 模式、未定义 指令 模式和系统 模式。} 当工作模式为用户模式时，处理器将不能改变当前的工作模式，除非发生异常，其它的工作模式能进行互相切换。 通过相应的指令可以向程序状态寄存器CPSR的第0到第4位，即[M4: M0]写入相应的值即可进入对应的工作模式。 具体的工作模式如下图所示。

图2-1 ARM处理器的工作模式

2.1.3 ARM处理器的寄存器

{51 %：ARM处理器有31个通用寄存器，6个状态寄存器，总共是37个32位的寄存器，ARM状态下不同工作模式的寄存器如下图所示。}

图2-2 ARM状态下的寄存器

{48 %：从图2-2可看出，ARM处理器在不同的工作模式下所使用的寄存器是不同的。} {60 %：通用寄存器R0- R7在所有模式下是共用的，快速中断模式下有自己专用的寄存器 R8- R12，} {49 %：系统模式和用户模式下使用相同的堆栈指针 R13（Stack Pointer， SP）和程序链接寄存器 R14（Link Register，} LR），其它模式都有自己特定的寄存器。 {59 %：最后一个通用寄存器是程序计数器R15，总共有31个通用寄存器。}

{45 %：状态寄存器有1个当前程序状态寄存器 CPSR，5个备份程序状态寄存器 SPSR，分别用于5种工作模式，} {42 %：用户模式和系统模式没有备份程序状态寄存器，程序状态寄存器的格式如下图所示。}

{72 %：图2-3 程 序状 态寄 存器 格式}

{49 %：M4-M0为模式选择位，决定处理器工作于哪种模式。} {51 %：T位为ARM与Thumb指令切换，T为

1时执行Thumb指令，为0时执行ARM指令。} {52%：F位为快速中断控制位，F为1时禁止FIQ中断，为0时允许快速中断。} {51%：I位为中断控制位，I为1时允许外部IRQ中断，为0时禁止IRQ中断。} 第27位到31位为条件码标志，详细可参考ARM手册。

由于本论文所写的嵌入式操作系统是工作在ARM状态下的，因此只简单介绍ARM状态下的寄存器情况，对于Thumb及Thumb-2状态下的寄存器情况并没有做相关的介绍。

2.1.4 ARM处理器的异常处理

{69%： ARM处理器拥有7种不同类型的异常，分别是复位、未定义指令、软件中断、指令预取中止、数据访问中止、外部中断请求、快速中断请求，} {47%：它们的优先级及对应的异常向量地址如下图所示。}

{51%：图2-4 ARM异常类型、优先级及向量地址}

7种异常可分为6级，其中复位优先级最高，未定义指令和软件中断最低，而且这两个异常是互斥的，不可能同时发生，所以它们的优先级是相同的。 当复位的引脚有效时，系统便无条件的进入管理模式，并把PC指向0x00000000处开始执行指令。

{48%：当ARM处理器发生异常后，如果是复位异常，会立即中止当前运行的指令，如果是其它的异常，处理器会执行完当前指令后，再去处理异常。} ARM处理器异常的响应过程如下：

{42%：将当前工作模式下状态寄存器的值保存到对应异常模式下的备份状态寄存器中，以便保护当前任务的状态信息。}

{47%：设置当前状态寄存器为相应的异常工作模式，而且禁止IRQ外部中断，如果进入的是复位模式或快速中断模式，还要禁止FIQ快速中断。}

{49%：把异常指令的下一条地址保存到异常模式下的寄存器R14中，当异常处理完后，程序才能返回原来的指令处继续向下执行。}

{46%：给程序计数器PC强行赋值，跳到对应的异常向量地址处执行相应的处理函数。}

{45%：每种异常模式下都有自己对应的SP和LR两个寄存器，分别用来存放堆栈指针和断点地址。} 以上的4步工作都是由ARM处理器的硬件自动完成的，我们只能做的是设计异常处理程序，并从异常处理程序中返回到原来的程序处。

{46%：由于 ARM处理器采用了多级流水线的技术，因此在实际编程时，第3步将引起异常指令的下一条地址保存到异常工作模式下的 R14中，} 该地址往往不是正确的返回地址。 {41%：因此我们通常在进入异常处理程序后，修改LR的值，以保证返回时是正确的地址。}

{49%：ARM处理器从异常程序返回到原来的程序处继续向下执行的过程如下：}

{48%：将异常模式下的备份状态寄存器的值复制到原理的状态寄存器中，还原到被中断前的工作状态。}

{42%：将LR的值装入到程序计数器PC中，使得程序能返回原来的程序处，这里LR的值为返回地址值。}

{50%：清除CPSR中的中断屏蔽位，打开IRQ外部中断和FIQ快速中断。}

这里的异常中断的返回顺序不能混乱，只能是先恢复CPSR的值，再从LR中恢复断点地址，返回原来的程序。{42%：如果顺序搞乱了，系统就会发生错误。}

2.2 Ok6410开发板的介绍

{52%：本论文所写的嵌入式操作系统是在飞凌公司的OK6410开发板下所完成的，OK6410开发板是基于三星公司的ARM11处理器S3C6410。}

2.2.1 S3C6410芯片简介

由 S3 C6410芯片手册[3][4]可知， S3 C6410是一个16/32位的 RISC处理器， 用来提供一种有经济效益的、低功耗的、高性能，能适用于移动电话和一般应用的处理器解决方案。 S3 C6410采用64/32位的内部总线架构，为2.5 G和3 G通信服务提供了优化的 H/ W性能， {89%：该64/32位的内部总线架构是由 AXI、 AHB和 APB总线组成的。} {55%：它还有很多强大的硬件加速器，比如2D图像处理、图形显示和缩放处理、运动视频处理、音频处理等。} 一个集成的多格式编解码器（Multi Format Codec， MFC）和MPEG4/H.263/H.264编解码器和VC1解码器。{53%：这种H/W编解码器能支持实时视频会议和电视输出的NTSC和PAL两种模式。} 此外，该处理器含有一个3D加速器，支持OpenGL ES 1.1/2.0的图形渲染。 {76%：这种3D引擎还有两个可编程着色器：} {83%：一个像素着色器和一个顶点着色器。}

{47%：S3C6410具有一个优化的接口连接到外部存储器。} 这种优化的接口，外部存储器是能在高速通信服务上维持高内存带宽。 {81%：内存系统具有双重外部存储器端口，DRAM和Flash/ROM。} 在DRAM端口可以配置为支持mobile DDR、DDR、mobile SDRAM和SDRAM内存。 在Flash/ROM端口可以支持NOR Flash、NAND Flash、OneNand、CF和ROM等类型的外部存储器。

{64%：为了降低系统的总体成本和提高整体功能， S3 C6410包含了许多硬件外设，} {73%：例如相机接口、 TFT24位真彩色 LCD控制器、系统管理器（电源管理等）、4个的 UART、32个} {62%： DMA、5个32位定时器（其中有2个是 PWM输出）、通用 I/ O接口（ GPIO），} {77%： I2 S总线接口、 I2 C总线接口、 USB主设备、 USB OTG设备高速传输（} {78%： 480 Mbps）、3通道 SD/ MMC主机控制器和 PLL时钟发生器。}

S3C6410的ARM子系统是基于ARM1176JZF-S内核。 {83%：它包括独立的16KB指令和16KB的数据缓存、16KB指令和16KB的数据TCM。} {84%：它还包括一个完整的MMU来处理虚拟内存管理。} ARM1176JZF-S是一款单芯片的微控制器，其中包括JAVA加速器。 ARM1176JZF-S包括一个专用的矢量浮点协处理器，允许高效的实现各种加密方案和3D图形应用程序。 S3C6410采用实际标准的AMBA总线架构。 {51%：这种强大的、行业标准的特点让S3C6410能够支持许多工业标准的操作系统。} 下图2-5就是S3C6410的芯片架构图。

图2-5 S3C6410芯片架构

2.2.2 Ok6410开发板简介

{72%：OK6410开发板采用S3C6410芯片，拥有强大的内部资源和视频处理能力，可以稳定运行在667MHz主频

上，支持Mobile DDR和多种NAND Flash。｝｛61％：OK6410开发板上集成了多种高端接口，如液晶屏、以太网、复合视频信号、摄像头、SD卡、USB等，并配备红外接收头、温度传感器等[5]。｝

该开发板拥有256 MB Mobile DDR内存和 MLC4 GB NAND Flash外存，主频为533 MHz/667 MHz，｛91％：总共有4个串口，包括1个五线 RS-232电平串口（DB9母座）和3个三线 TTL电平串口，｝ 1个 RTC实时时钟，4个 LED，6个按键，1个蜂鸣器，还有其它的硬件资源，详细可参考开发板硬件手册[5]。 下图2-6就是OK6410开发板的实物图。

图2-6 OK6410开发板

2.3 本章小结

本章主要简单介绍了ARM的体系结构，比如ARM的工作状态、工作模式、寄存器和异常处理。 然后简单介绍了所使用的OK6410的开发板的简单情况，包括S3C6410的芯片，最后详细介绍了如何在Linux下搭建基本的交叉开发环境。

系统分析与设计

3.1 系统的设计目标

｛42％：本论文所设计的小型嵌入式操作系统，是在 ARM开发板上实现的，该系统具备基本的多任务调度，｝也就是说应用开发者能创建多个任务来进行工作，每个任务间通过本系统的调度算法来进行分配任务的运行， 同时本系统具有简单的内存管理功能，开发者可以在任务运行时，根据需要进行申请内存和释放内存。 ｛49％：经分析，本嵌入式操作系统需要实现以下的基本功能：｝

任务的基本操作功能。 ｛50％：这里包括任务的创建、任务的挂起和恢复、任务的删除等，这里的难点主要是多任务调度的设计。｝

中断服务功能。 ｛43％：中断服务的功能模块是本操作系统的核心，也是难点之一。｝ 中断服务涉及开发板相关的底层操作，大部分采用ARM汇编来编写中断服务的进入和退出。 中断服务子程序（Interrupt Service Routines，ISR）一般都是采用简单的C语言来编写，主要完成在中断时进行的操作，比如系统的时钟服务。

时间管理功能。 这里的时间管理功能主要用于系统的时间节拍，可以用来对任务进行延时，也可用来获取系统运行的总节拍数。

内存管理功能。 这里是比较简单的内存管理，主要采用固定分区法，在预先分配好的二维数组里进行内存分配。

3.2 系统的总体结构

系统的总体架构包括：｛50％：系统的引导和启动、硬件的初始化、系统的运行等。｝｛44％：本论文主要把整个系统分成4个模块：｝｛51％：引导启动模块、系统更新模块、串口通信模块、操作系统模块。｝｛49％：其中操作系统模块根据系统的基本功能又细分为任务管理模块、时间管理模块、中断服务模块和内存管理模块。｝总体的结构模块如下图所示。

引导启动模块。 这个模块的主要功能是引导开发板的启动，主要是以ARM汇编为主。 主要包括设置ARM异常向量的地址、设置外设接口的地址、关闭看门狗、设置CPU的主频、初始化SDRAM和NAND Flash、把代码复制到SDRAM中等。

系统更新模块。 由于操作系统不可能一次编写，就能成功运行所有功能，因此需要不停的往开发板重复烧写新的操作系统程序。 {40％：由于每次烧写目标程序时都需要使用u-boot在Windows下才能把程序写进NAND Flash上，} 比较麻烦，而且在Linux下没有相关的Linux驱动，因此，本课题里设计了一段能自行更新NAND Flash上目标程序的代码， 并集合到系统上。 这个系统更新模块主要采用串口通信来获取新的目标程序，系统引导启动后就能直接选择更新系统或者运行系统，即方便又简单。

串口通信模块。 这里的串口通信模块主要方便调试，开发板通过串口与主机进行通信，可以让开发板输出相关的信息，然后在主机上显示，而且这个串口通信模块还包括主机上的串口通信软件。 这个通信软件是本课题里自行编写的软件，其中串口通信方面采用了第三方写的插件，而且这个软件结合系统更新模块， 主要是通过串口来传输二进制文件来更新开发板上的系统程序代码。

操作系统模块。 {59％：本模块可以分为任务管理模块、中断服务模块、时间管理模块和内存管理模块。}

{63％：图3-1嵌入式操作系统的总体模块图}

3.3 引导启动模块的设计

引导启动模块是整个嵌入式系统的首要模块，主要功能就是引导并启动操作系统，跟bootloader类似。 整个引导启动模块所做的工作首先就是初始化相关的硬件，然后跳到系统的启动界面。

引导启动模块往往与芯片和开发板紧密结合，因此不同的开发板其实现过程往往不同，但其思路基本是一样的，下面基于OK6410开发板，介绍其引导模块的设计过程。

{81％：设置异常处理函数的入口地址。} {45％：根据ARM的体系结构可知，地址0x0000000到0x0000001F是ARM异常向量的入口地址，} {41％：当ARM芯片启动或复位后，系统会自动从0x00000000的地址处开始执行程序，因此我们需要在0x00000000处设置一条跳转指令，} {41％：跳转到程序的初始化处，其余的异常向量入口地址需要填写相应的异常处理函数的入口地址，} {60％：以便异常发生时，系统跳转到相应的处理函数处。}

初始化相关的硬件。 {47％：当系统进入初始化处程序后，就进行相关的硬件的初始化了。} 这里首先要禁止所有的IRQ和FIQ中断，然后根据S3C6410的芯片手册[4]，还需设置外设寄存器的地址后才能设置相关寄存器的值。 接着是关闭看门狗，大多数嵌入式芯片都有看门狗的功能，因此需要关闭看门狗，不然系统会每隔一段时间就会自动重启。 然后初始化其它的硬件，比如CPU的频率，SDRAM和NAND Flash，这些寄存器的设置只需根据相关的芯片手册来进行初始化即可。

（3）系统硬件初始化完后，还需要进行重定位代码。 由于S3C6410芯片启动后会自动从NAND Flash中复制8KB的数据到芯片的SRAM中运行， 但往往一个操作系统的大小会超过8 KB，因此，我们需要把程序复制到内存SDRAM上， 然后从SRAM跳到SDRAM中继续运行，这一步骤就是重定位。 {47％：重定位只需将NAND

Flash中的代码段TEXT和数据段DATA复制到SDRAM中，最后清零BSS段即可。｝

｛55％：重定位后，就准备跳到SDRAM中继续执行代码了。｝ 这里一般是跳到main函数开始执行主程序，但在本系统中，由于存在系统更新模块，因此是首先跳入到更新函数，然后再执行main函数。

最后是异常处理程序的编写。 ｛43％：由于发生异常后，系统会跳到异常向量地址处，执行异常向量地址处的函数。｝ ｛42％：在进入异常处理程序前，首先需要保存当前任务数据，即将CPU相应的寄存器数据入栈。｝ 由于ARM存在流水线的问题，因此寄存器LR的返回地址不一定是正确的返回地址，需要根据哪种异常来进行修改。 ｛49％：然后进入异常处理程序，当异常处理完毕后，还原被中断任务数据。｝

ARM有7种异常，由于复位异常不需编写异常处理程序，所以实际只需编写6个异常处理的函数。 但因为本系统中只用到外部中断IRQ，所以只编写了IRQ的处理函数，其余5个异常处理程序，开发者可根据实际需要进行编写。

3.4 串口通信模块的设计

串口通信模块对于嵌入式系统的调试是非常重要的，本模块可分为主机和开发板两部分，开发板和主机通过串口可以互相发送和接收数据。

绝大多数的开发板都有一个串口插座，OK6410开发板有一个DB9的串口母座，可直接与计算机的串口进行连接通信。 要进行串口的通信，首先需要设置开发板的串口，比如串口波特率，数据位，停止位，校验位等，可根据芯片手册来设置相应的寄存器。 要通过开发板的串口发送数据，只需将该数据逐位写到相应的寄存器便可发送到PC的串口上，接收数据也是类似，从相应的寄存器去读取数据。

开发板上实现基本的串口通信比较简单，而在主机上，比如 Windows系统，往往有较多的串口通信软件， 但由于本系统存在系统更新模块，而系统的更新，是通过串口来获取新的程序代码来实现的， 因此，本系统还实现了一个能够发送二进制文件的串口通信软件。 该软件基于Qt来开发，串口通信的实现主要通过第三方的插件来实现，除了基本的串口通信外，还能配合系统的更新模块来发送更新系统的程序，简单方便。

3.5 系统更新模块的设计

｛45％：当程序进入系统更新模块后，就可以选择更新系统程序或者运行系统。｝ 系统更新模块主要用到串口通信，利用串口通信来接收主机的系统更新代码，把代码复制到NAND Flash上，然后重启系统，就能运行新的系统程序了。

系统更新模块的难点主要在于接收新的代码和把代码写入到NAND Flash上。 本系统从内存地址0x5FC00000（离内存尾端4MB）处开始存放系统的更新代码，当接收完代码数据后，然后在将数据写入到NAND Flash上。 不同的NAND Flash其读写的数据方式也不同，因此需要根据NAND Flash的芯片手册来进行编写。 这里还要注意S3C6410上的NAND Flash陷阱，OK6410开发板的NAND FLash是以页来读取数据的，每页有4KB是用来存储数据，218个字节来存放校验和等信息。 按照原理，S3C6410处理器是从前两页读取8KB的数据到SRAM中，但其实并不是这样的。 S3 C6410会从前4页，每页读取2 KB，总共8 KB的数据到 SRAM中，所以，当把系统代码更新时，｛47％：并不能直接把数据从 NAND Flash的第一页开始写下去，｝ 而是需要把8 KB数据写到4页里，｛46％：第2到第5页的前2 KB数据与前一页的后2 KB数据相同，｝ 如下图所示。

图3-2 NAND Flash前4页的数据写法

至此，关于开发板从启动到准备运行操作系统，本设计把这一阶段分为了三个模块来讲解，其中串口通信模块和系统更新模块并不是必须的，但对于系统的调试和更新是非常方便的，如果到了应用正式部署的时候，可以选择性的删除。这三个模块与开发板的硬件紧密结合，不同的开发板其实现过程也是不同的，因此这里只是简单的介绍了在OK6410下的设计过程，其它的需要根据实际的开发板进行相应的移植和修改。当系统的环境准备好后，就可以开始载入操作系统并开始运行了。

3.6 任务管理模块的设计

3.6.1 任务的定义及其结构

{50％：当人们在生活中遇到一个大而困难的问题时，往往会把这个大问题分解成多个简单和容易解决的小问题。} 在实际编程中，当面对大的应用程序时，我们也会把它编写成一个个小程序来完成目的，这种方法即能提高CPU的利用率，同时又缩短程序的执行时间。由于嵌入式系统都是与具体的应用紧密结合的，都是针对解决某一类问题，因此可以把这一类问题分解成许多小问题，再交给操作系统来进行处理。利用这种思路，在嵌入式操作系统里，就可以把每个大问题转化为多个任务，每个任务都有自己相应的工作函数，由操作系统来进行任务的调度和管理，这样就能实现一个多任务的操作系统了。

每个任务可以看做是一个线程，都是一段简单的程序，典型的任务都是一个无限循环的函数，由任务控制块来进行管理，{51％：一般都包括任务的栈地址，任务的优先级，任务的函数地址等信息[7]。} {63％：每个任务都属于整个应用的某一部分，都被赋予一定的优先级，有自己独立的栈空间，彼此独立运行[6]。}

本论文设计的嵌入式操作系统由任务控制块链表来管理各个任务，任务的基本模型如下所示。

图3-3 任务链表

{46％：任务可以分为普通任务和系统任务，系统任务为空闲任务和统计任务，空闲任务是必须存在的，当系统没有其它任务可以运行时，} 空闲任务就会运行，这样 CPU就不会没事可做了，统计任务用于统计 CPU的利用率。{48％：普通任务即为开发者创建的任务。}

{71％：每个任务都有5种状态，分别是就绪态、运行态、等待态、休眠态、中断态。} 当任务控制块创建完后，任务就进入就绪态了，当任务得到 CPU而运行后，处于运行态，当任务进入休眠后，{47％：就处于休眠态，当任务被挂起后就进入等待态，被中断的任务处于中断态。} 各种状态的转换图如下所示。

图3-4 任务状态转换图

3.6.2 任务的调度机制

在嵌入式操作系统里，采用哪种调度策略，对操作系统的表现是有很大影响的，如果调度机制不完善，会影响系统的实时性。{44％：嵌入式操作系统的调度算法可分为优先级法和时间片轮转法，优先级法又可分为非抢占式优先级法和抢占式优先级法，} 时间片轮转法也可分为时间固定与可变两种方式[8]。{44％：对于大多数的嵌入式实时操作系统，为了使系统能够快速响应外部突发事件，一般都采用基于优先级的算法[6]。} {54％：至于抢占式调度和非抢占式调度，由于在抢占式调度中，只要最高优先级的任务一旦就绪，就能得到 CPU的使用权，

} {42%：而非抢占式调度中，当前任务会一直占用 CPU，直到其运行完成才会让出 CPU。} {49%：因此，为了确保系统的实时性，本课题所设计的嵌入式操作系统采用了按优先级的抢占式调度机制。}

{58%：采用按优先级的抢占式调度策略，系统总能优先运行最高优先级的就绪任务。} {58%：当一个任务在运行期间，使得另外一个更高优先级的任务进入了就绪状态，那么当前正在运行的任务的 CPU使用权就会被更高优先级的就绪任务所占用，} {62%：如果是中断服务使一个更高优先级的任务进入了就绪状态，那么当中断完成后，并不会继续运行被中断的任务，} {65%：而是去运行更高优先级的就绪任务。}

{46%：如下图所示，两个不同优先级任务和一个中断服务的执行情况，当低优先级任务被中断后，} {71%：系统进入中断服务子程序 ISR，中断服务子程序 ISR同时使一个更高优先级的任务进入就绪状态，} {46%：当中断服务程序执行完后，调度器会选择更高优先级的就绪任务来运行，而不是恢复被中断了的任务继续运行[9][10]。}

{68%：图3-5 优先级抢占式调度}

{54%：基于优先级的调度策略，每个任务都拥有一个由设计者按照任务的重要性来编排的优先级号。} {48%：任务的优先级设计是十分重要的，可以分为支持同优先级和不同优先级两种。} {49%：支持同优先级任务即多个任务可以拥有相同的优先级，而不同优先级是每个任务必须分配各不相同的优先级。} {51%：采用支持同优先级的方式比较适合现实情况，每个任务的优先级高低并不是绝对的，每个任务的地位基本相同，} 并有一定的周期性[8]，但采用这种方式，任务的调度和任务的控制块就会变得相对复杂， 对于初做系统，可以采用相对简单的方式，更好的改进方式可以留到下一阶段来继续完成。 因此，本设计采用不同优先级的方式，虽然这种设计方法比较简单，但也是有比较多的问题， 其中就有优先级反转的问题，本设计里没有设计出针对此问题的解决方案，但已经有许多参考资料提出来了解决的方法， 这一部分也是留到下一阶段继续改进的地方。

{46%：采用不同优先级的方式，每个任务都拥有一个不同的优先级，因此，可以把任务的ID号等同于优先级号。} {51%：这里优先级号采用整数来表示，0为最高优先级，数字越小，优先级越高，高优先级的任务先运行，低优先级的任务后运行。} {48%：当系统进行调度时，可以在任务控制块链表中查找最高优先级就绪的任务来运行，} 由于创建任务时，并不一定是按照任务的优先级来分别创建的，因此，如果在任务控制块链表中来查询， {50%：其效率比较低，对系统的实时性有一定的影响。} 所以，这里可以采用一个有序表，即一个一维数组，数组0表示0号任务，数组1表示1号任务， {42%：以此类推，每个数组指向对应的任务控制块地址，即数组0是执行0号任务的任务控制块。} 当进行查询时，就可以从数组0开始往下查找，如果数组N为空，表明不存在N号任务； {49%：如果数组N存在，那么数组N所指向的任务控制块就是最高优先级就绪任务了。} 模型图如下所示。

图3-6 优先级有序表

{50%：这里采用顺序查找的算法虽然比较简单，但并不是最好的。} {50%：我们可以在创建任务时，根据任务的优先级顺序插入到任务控制块链表中相应的位置，} {46%：这样每次查找时就能更加快捷，但创建任务所花费的时间也会相对增加，} 特别是当任务比较多的时候，这些改进也可留到下阶段改进。

3.6.3 任务的创建

{43%：每个任务都由任务控制块链表中的任务块进行管理，当系统运行后，会把每个空的任务控制块连接成一个链表，} {53%：称为空任务块控制链表，当需要创建任务时，便可从这个链表中拿出一个空的任务控制块

，｝ 来设置任务的相关信息。 ｛41％：所有创建好的任务控制块会连接到另外一个任务控制块链表中，当系统进行调度时，便会从这一链表中选出任务块来运行任务。｝

任务的创建最关键就是设置任务的栈，本系统中每个任务都拥有一个固定的栈，栈空间的大小由开发者定义，毕竟不同开发板其内存资源也是不同的。 栈空间主要用于切换任务时，保存当前任务在CPU上寄存器的数据，以便将来恢复当前任务继续运行。 当任务首次创建时，PC寄存器会指向当前任务的函数入口地址，当函数被调度运行时，便可跳到对应的函数，开始执行程序。

当设置好任务的栈后，还需要把任务块连接到任务控制块链表中，这样任务的状态就可以设置为就绪，等待系统调度运行了。

｛49％：任务链表和任务块的基本模型如下所示。｝

图3-7 任务链表图

3.6.4 任务的挂起和恢复

每个普通任务都可以挂起任务自身或者其它任务，当任务被挂起时，就会处于挂起态，这时，不论任务的优先级是什么，都不会得到系统的调度运行。

被挂起的任务只能通过其它任务来恢复，只要把被挂起任务的状态清掉，那么该任务就可继续参与到系统的调度中。

3.6.5 任务删除

任务删除操作可以删除其它任务或者删除任务自身，为了让被删除的任务能够释放自身所占用的资源， 我们可以先发送一个删除任务的消息给对方，当对方收到这个消息后，就可以进行适当的步骤后再删除自身， 这种方法要比直接删除任务更安全、灵活。

3.7 中断服务模块的设计

中断服务模块主要为操作系统提供中断响应机制，当系统接收到外部中断请求后，会停止当前正在运行的任务，从而转到中断服务子程序进行中断处理。 本系统暂时只需要外部中断请求IRQ，因此只讲解外部中断的请求过程，对于其它中断比如快速中断FIQ等可以参考S3C6410芯片手册进行编写。

每个任务需要打开中断后才能响应中断，在任务堆栈的初始化时，应当把每个任务的状态寄存器即 CPSR设为0x0000001 F， ｛44％：即处于系统工作模式并打开外部中断，这样当系统产生中断时，就会跳转到相应的中断处理函数。｝ ｛45％：本系统的主要中断处理函数有时钟定时器中断处理和按键中断处理。｝

时钟定时器主要是完成系统在每个时钟节拍需要做的工作，我们首先需要设置系统时钟的节拍，比如系统时钟节拍为10ms，即每秒中断10次。 这里需要根据S3C6410芯片手册，设置相应的时钟寄存器，把设置的值写入到相应的位即可。 由于ARM11采用的中断向量的方式来设置中断，所以中断设置比较简单，只需把中断的处理函数地址写到对应的向量位， ｛43％：那么当中断产生时，就会自动进入中断处理函数，进行中断处理。｝

{55%：时钟定时器的中断处理函数主要完成的工作为：} 给系统的时钟计数器加1，然后遍历任务控制链表，把所有延时任务的延时器减1，如果延时任务的延时器变为0，还要将该任务的延时状态清零。

处理中断的函数基本都是使用ARM汇编来完成，主要是进入中断前，需要保存CPU的寄存器数据到当前任务的栈空间中，然后在进入中断处理函数。 {44%：退出中断后，需要把任务栈中的寄存器数据恢复到处理器中，从而使得任务能继续运行。}

按键中断也是首先设置好相应的中断位，然后设置中断处理函数的地址，当有按键按下时，系统就会跳到对应的按键的中断处理函数。

3.8 时间管理模块的设计

{43%：时间管理模块可以提供任务的延时和系统时钟节拍的功能。} 任务的延时主要根据时钟节拍来计算出任务所需延时的时钟节拍数，本系统的时钟节拍为10，即1秒会进行10次中断，时间片为100ms。 如果任务需要延时1秒，那么延时的节拍数即为10。 任务进入延时后，系统应当重新引发一次调度，运行下一个就绪任务。

每次时钟中断，系统都会检查任务块，对于有延时的任务块，系统会减去1个延时节拍数，当延时节拍数变为0，那么任务块的延时状态就会清零。

此外，还需记录系统从开始运行到至今已经经过多少个时钟节拍，因此，系统还有一个时钟计数器，每次时钟节拍中断时都会加1。

3.9 内存管理模块的设计

{46%：内存管理功能是操作系统内核中一个重要的功能，本小型嵌入式操作系统提供一个比较简单的内存管理功能，} {42%：任务能在运行时申请内存空间，当任务不需要这些空间时，可以释放且还给操作系统。} 内存管理的技术比较多，从操作系统教程就可知道有地址空间与重定位、分区管理、分页技术、虚拟存储管理等等。{42%：其中分区的管理算法简单，易于实现，但碎片问题严重，内存利用率低[11]。}

本系统采用比较简单的固定分区法来实现内存的管理。 {89%：固定分区法就是内存中分区的个数固定不变，各个分区的大小固定不变，根据大小的不同还可分为等分方式和差分方式，} {40%：所谓等分，就是各个分区大小相同，所谓差分，就是分区具有不同大小[12]。} 利用等分方式，实现方法简单，需要处理的开销也很小，但缺点也很明显，任务申请的内存空间一定要在分区的大小之内，否则将会引起灾难性的结果。 对于更好的分区方法，将留到下阶段的改进。

采用固定分区法，我们可以通过定义一个二维数组就能实现内存分区，比如一个8位的数组 MxN， {42%：内存分区的大小为8* M* N个字节，总共可以分为 M个内存块，每个内存块有 N个字节，} {43%：每次任务申请的内存都需要小于 N个字节。} 如下图所示。

图3-8 固定分区

这种分区方法虽然比较粗糙，内存的利用率低，但实现简单，开销小，实时性也较好。

{45%：有了内存分区后，我们还需要一个内存控制块来管理内存分区里的内存块，实现真正的内存分配。}

{65%：内存控制块需要设置内存分区的起始地址、内存块的大小、内存块的数量，已经分配出去的内存块数量等信息。} 对于各个内存块，我们需要把它们连接成一个链表，那么分配内存时，就可直接在链表中取出该内存块给任务，{52%：任务释放内存时，只需将该内存块连接回链表中即可。} 如下图所示。

{72%：图3-9 内存控制块和内存分区的关系}

3.10 本章小结

{43%：本章首先介绍了整个嵌入式操作系统的设计目标和一个总体的结构，把整个系统功能划分成4个模块，} {43%：引导模块、系统更新模块、串口通信模块和操作系统模块，然后阐述了各个模块的设计思路。}

系统实现

4.1 系统开发环境的搭建

嵌入式应用的开发一般都是采用主机与开发板结合的交叉开发模式，因为嵌入式设备大多没有操作系统，即使是配有操作系统，一般情况下也是没有相应的编译和开发工具，不足以用来开发软件，也就是说开发板上不能自行编译软件程序后运行。因此，需要在主机（PC机）上先为开发板搭建对应的开发环境，然后编译出对应的应用程序，最后烧写到开发板上，才能正常运行。所以一般用到的交叉开发模式为：在主机上编辑程序代码、编译成对应开发板上的应用程序，然后烧写到开发板上，最后在开发板上运行和验证应用程序。

在本设计里，主机是指PC机，开发板也就是OK6410开发板，所开发的嵌入式操作系统首先也是在主机上进行编写程序，然后通过交叉编译后再把目标程序烧写到OK6410开发板上，最后是验证该系统的正确性。

嵌入式的开发环境既能在Windows下搭建，又能在Mac OS或Linux下搭建，由于本设计使用Linux操作系统作为开发的平台，因此这里只讲解在Linux下如何搭建相应的开发环境，{41%：对于其它操作系统也是类似，这里就不讲解了。}

本设计里搭建的嵌入式开发环境只要配置相应的交叉开发工具链即可，因为平常在PC上所使用的编译工具一般都是gcc、ld等，它们编译出来的程序一般都是运行在x86平台上的，对于ARM平台上的嵌入式设备来讲，是不能运行x86的应用程序，{48%：因此需要使用交叉编译工具来生成ARM平台上的目标程序，这里的交叉编译工具就是arm-linux-gcc、arm-linux-ld等。}

本设计里所使用的主机操作系统为Fedora 20 Xfce，所使用的交叉编译工具链是Sourcery CodeBench Lite 2013.11-24（GCC版本号为4.8.1）。可以到http://www.mentor.com/embedded-software/sourcery-tools/sourcery-codebench/editions/lite-edition/处下载最新的ARM交叉编译工具链。{44%：注意，这里ARM的交叉工具链有两个版本：} EABI和GNU/Linux，这里的GNU/Linux版本是用来编译嵌入式Linux内核或Linux应用的，{44%：EABI版本适用于编译自己的嵌入式操作系统或应用，不能用于编译Linux应用，由于本文设计的嵌入式操作系统不涉及Linux，} 因此本设计选择EABI版本来作为工具链。此处下载到的安装包格式为.tar.bz2。

{47%：在Fedora 20里安装交叉编译工具链的推荐步骤为：}

首先在/usr/local/目录里创建一个名为CodeSourcery的文件夹（非必须，只是推荐）。命令为：

mkdir /usr/local/CodeSourcery

解压下载的安装包。 命令分为2步，第一条命令将把.tar.bz2解压为tar包，然后再使用第二条命令解压tar包得到安装的文件夹：

bunzip2 arm-2013.11-24-arm-none-eabi-i686-pc-linux-gnu.tar.bz2

tar -xf arm-2013.11-24-arm-none-eabi-i686-pc-linux-gnu.tar

解压后得到arm-2013.11的文件夹，本课题里把它重命名为arm-none-eabi，并移到第一步所创建的文件夹CodeSourcery下，命令为：

mv arm-2013.11 /usr/local/CodeSourcery/arm-none-eabi

{43%：把交叉工具链的目录添加到系统环境变量中，具体为编辑用户目录下的.bashrc文件，命令为：}

vim ~/.bashrc

然后添加bin目录的路径和lib目录的路径，如下：

PATH=$PATH: /usr/local/CodeSourcery/arm-none-eabi/bin

LD_LIBRARY_PATH=$LD_LIBRARY_PATH: /usr/local/CodeSourcery/arm-none-eabi/lib

最后是输出环境变量，保存修改的.bashrc文件即可：

export PATH

{44%：至此，交叉编译链就配置好了，可以使用命令让修改的环境变量立即生效：}

source ~/.bashrc

{43%：最后可以使用下面命令测试一下交叉编译工具链是否安装正确：}

arm-none-eabi-gcc -v

如果出现如下图2-7的GCC版本号，那么基本的开发环境就搭建好了，可以开始编写和编译ARM应用程序了。

{64%：图4-1 ARM交叉编译工具链版本号}

4.2 任务管理模块的实现

{55%：4.2.1 任务控制块的定义及初始化}

{42%：根据第3章任务控制块的设计，具体的代码实现如下图所示。}

图4-2 任务控制块的定义

{40%：其中tcb_stk_ptr就是每个任务的栈指针，该栈指针指向每个任务的栈地址；} {44%： tcb_prio是任务的优先级号，即任务号， tcb_next和tcb_prev分别指向下一个和前一个任务控制块，} task_status是任务的状态，tcb_delay为任务的延时节拍， tcb_del_req是任务的删除标志。 其中定义了3种任务状态，就绪、延时和挂起，定义如下图所示。

图4-3 任务状态的定义

{43%：任务控制定义好后，还需要定义任务控制块的链表指针、任务块的变量，当前运行任务块的指针、最高优先级就绪任务的指针，} {64%：优先级有序表等，如下图所示。}

{59%：图4-4 任务控制块的相关变量}

{42%：当定义好任务控制块的所有变量后，需要把空任务块连接成空任务块控制链表，主要代码如下图所示。}

{53%：图4-5 空任务块链表}

4.3.2 任务的创建

{41%：任务控制块初始化后，系统会创建系统任务，然后才会创建开发者定义的普通任务。} {53%：系统和普通任务都是通过create_task()函数来创建。} {42%：create_task()需要四个参数，分别是任务函数的入口地址，任务函数的参数，栈顶地址和任务的优先级号。} {40%：首先函数会判断任务的优先级号是否在正常范围内，即0到最小优先级号之间，接着判断该任务号是否已经被其它任务申请了，} 这里是通过os_tcb_prio_table[prio]来判断，如果为NULL，则说明此任务号还没被使用， {45%：接着会初始化该任务的栈和任务块，代码如下图所示。}

图4-6 create_task()函数部分代码

其中init_task_stack()和init_tcb()是创建任务的重点函数， init_task_stack()用来初始化任务的栈空间，即从栈顶地址分配寄存器空间， {42%：用于保存任务调度时的任务现场数据，代码如下图所示。}

图4-7 init_task_stack()函数代码

{43%： init_tcb()函数主要是从os_tcb_free_list里取出一个空任务块，把它连接到任务控制块链表中，} {41%：接着设置任务相应的值后，该任务就处于就绪运行的状态了，代码如下。}

图4-8 init_tcb()函数部分代码

当创建的任务进入就绪状态后，会返回到create_task()函数中继续运行，此时create_task()函数会判断系统是否

已经在运行， {41%：如果系统已经运行，则会进入任务调度函数进行任务调度。} 如果系统还没有开始运行，则会返回，直到start_os()函数运行后才开始调度多任务。

4.3.3 多任务的调度

系统开始运行前，会执行start_os()函数，才会开始执行多任务的调度。 {44%：start_os()函数首先从优先级有序表找出就绪的最高优先级任务，然后开始运行就绪任务，代码如下。}

图4-9 start_os()函数

{50%：其中，sched_new()函数负责找出最高优先级就绪的任务，代码如下。}

图4-10 sched_new()函数

{44%：查找最高优先级就绪任务的算法比较简单，直接从 os_ tcb_ prio_ table开始从0往下找，} {42%：因为优先级越高的任务，其任务号越低，所以，只要 os_ tcb_ prio_ table[prio]不为 NULL，} 就能知道该任务号是存在的，所以我们就可以通过该任务的 task_ status来判断是否为就绪， 如果就绪，则把 os_ prio_ high_ ready设为该任务号。

start_task()函数运行就绪任务，将任务从栈寄存器中恢复数据，就可以开始运行了，本函数通过ARM汇编来编写，代码如下图4-11和4-12所示。

图4-11 start_task()函数

图4-12 restore_task()函数

系统开始运行后，就由sched()函数来专门负责任务的调度了，比如正在运行的任务进入延时后，就会调用sched()函数来重新进行任务的调度运行，代码如下所示。

图4-13 sched()函数

{45%：该函数首先判断当前运行的任务与最高优先级的就绪任务是否相同，如果不相同，则进行任务的调度，选择最高优先级就绪的任务来运行。} 切换任务由 switch_ task()函数来完成，该函数使用 ARM汇编来编写，主要先保存当前任务的数据， {47%：然后恢复最高优先级就绪任务的栈寄存器数据即可恢复现场来运行，主要代码如下。}

图4-14 switch_task()函数主要代码

任务的调度主要由选择最高优先级就绪任务和切换任务这两部分来完成，本设计只是用比较简单的算法来实现基本任务调度原理， 因此没有复杂的调度算法，而切换任务只需首先把当前任务的数据保存起来，然后把最高优先级就绪任务的地址保存到当前栈指针 SP处， 就能由 ldr指令恢复现场数据了。

4.3.4 任务的挂起和恢复

{57%：挂起任务，就能让任务停止运行。} {45%：挂起任务的函数为suspend_task()，调用该函数，可以挂起任务自身，也可以挂起其它有效任务。} {46%：如果需要挂起任务自身，传递的参数为OS_PRIO_SELF，其它任务只需传递相应的任务号即可。} 该函数会改变任务的状态为TASK_SUSPEND，从而使它不能就绪而停止运行。 {40%：如果挂起的任务是当前任务自身，改变任务状态后，还会调用sched()来进行重新的任务调度，选择最高优先级就绪任务来进行运行。} 基本代码如下所示。

图4-15 任务挂起函数

{51%：任务的恢复只能通过其它任务来恢复被挂起的任务，函数为resume_task()。} 该函数将任务状态的TASK_SUSPEND清零，如果该任务没有设置延时状态，该位清零就会变成就绪状态，也就是TASK_READY，同时会进行新的任务调度，代码如下所示。

图4-16 任务恢复函数

任务的挂起和恢复相对简单，只要设置任务状态的相应位为对应的状态即可，设置后只要调用sched()函数就能进行新的任务调度了。

4.3.5 任务的删除

任务删除可以删除其它任务或者删除任务自身，首先要调用delete_task_request()来发送删除任务的消息，被删除任务接受到删除消息后，调用delete_task()来删除任务自身。 删除任务首先把任务挂起，并把延时时间设为0，避免被调度。 {43%：接着把该任务块从os_tcb_list中归还到os_tcb_free_list中，再引发新的任务调度。} 代码如下所示。

图4-17 delete_task_request()部分代码

图4-18 delete_task()部分代码

4.4 中断服务模块的实现

{46%：由于系统启动时，本设计就已经设置外部中断服务程序的入口地址，所以当发生外部中断IRQ后，系统会跳转到irq_isr函数处执行中断处理。} {42%：执行中断处理前，首先需要保存任务现场，然后跳转到中断处理程序处理中断，处理完毕后，退出中断，恢复任务。} 核心代码如下所示。

图4-19 irq_isr函数

{41%：其中handle_irq用来跳到开发者定义的中断处理函数，代码如下。}

图4-20 handle_irq函数

退出中断，由exit_interrupt函数来完成。 退出中断前，该函数会进行一次任务调度，如果没有更高优先级就绪的任务，那么会恢复被中断的任务接着运行，否则会去运行更高优先级就绪的任务，代码如下所示。

图4-21 exit_interrupt函数

{44%：如果没有更高优先级就绪任务，那么函数就会返回到irq_isr函数处，} {53%：接着恢复被中断的任务去运行，如果此时有了更高优先级就绪的任务，} {40%：那么就会执行interrupt_switch_task()函数来运行更高优先级就绪任务。} interrupt_switch_task()首先得出更高优先级就绪任务的任务块指针，接着恢复寄存器数据就能开始运行了，代码比较简单，如下所示。

图4-22 interrupt_switch_task()函数

中断服务模块还有一个重要的功能，就是能提供开中断或关中断功能，系统默认是打开中断的，但有时一些代码不能受到中断的影响，因此需要关闭中断，那些不希望被中断的代码段叫做临界段。打开和关闭中断涉及到ARM指令，因此这部分代码是通过ARM汇编来编写的。{51%：本系统通过enter_critical()和exit_critical()两个函数来退出和打开中断，这两个函数其实是用宏定义来实现，代码如下。}

4-23 退出和打开中断宏定义

退出中断首先需要保存CPSR状态寄存器，由save_cpsr()来完成，然后关闭中断即可。恢复中断由restore_cpsr()来完成，只需要恢复原先保存的CPSR寄存器即可。代码如下所示。

{51%：图4-24 保存和恢复CPSR寄存器代码}

4.5 时间管理模块的实现

{41%：时钟中断服务程序为time_tick()，该函数的主要功能是给时间计数器计数，} {43%：接着遍历所有任务控制块，如果该任务有延时，就把延时计数器减1，并判断计数器是否减为0，} 如果是，还需要把该任务的延时状态给清零。代码如下所示。

图4-25 time_tick()函数代码

本系统提供多个函数接口给开发者调用，比如delay()函数，能够让当前任务休眠指定个时钟节拍，sleep()函数，让当前任务休眠指定的秒数，msleep()让任务休眠指定的毫秒数，sleep_hmsm()能用时、分、秒、毫秒来作为传递参数进行休眠延时。其中sleep()、msleep()、sleep_hmsm()函数都是先把时间转化为毫秒数，然后计算出休眠的时间节拍后，再调用delay()函数来进入休眠，所以delay()函数是所有休眠函数的最终调用函数，delay()函数的核心代码如下所示。

图4-26 delay()函数代码

休眠函数msleep()的代码如下所示。

图4-27 msleep()函数代码

剩下的sleep()和sleep_hmsm()函数代码都是比较简单，都是把休眠的时间转化为毫秒，再调用msleep()就可以了。{50%：任务进行休眠后，会进行一次新的任务调度，从而运行最高优先级就绪任务，} {40%：延时任务会在时间中断服务程序里减去延时计数器，当延时计数器减为0时，} 如果任务没有被挂起，那么该任务就会再次成为就绪任务，等待调度。

任务也可以取消延时，由其它任务进行调用cancel_delay()函数从而取消被延时的任务，并进行一次新的调度。取消延时的函数如下所示。

图4-28 cancel_delay()函数

4.6 内存管理模块的实现

{51％：内存控制块主要用来记录内存分区的状态信息，其定义如下所示。}

图4-29 内存控制块的定义

{49％：内存控制块中 mem_ addr指向内存分区的起始地址， mem_ free_ list指向下一个内存块地址， block_ len表示内存块的长度，} {46％：即大小， num_ blocks表示内存块的个数， num_ free表示空闲的内存块的数目。} 当系统初始化时，首先会把内存分区内的各个内存块连成一个链表，当任务需要内存块时，就从链表中取出内存块，返回给任务。

图4-30 create_mem()函数核心代码

核心代码如上图所示，以定义的 u8 array[10][50]为例，内存分区就是 array这个数组， 其中有10个内存块，每个内存块大小为50个字节，所以每次分配内存，就是把 array[0]、 array[1]、 array[2]的地址返回给调用者。 因此，还需要把全部内存块连接在一起，这里可以把 array[0]指向 array[1]， {43％： array[1]指向 array[2]，以此类推，然后内存控制块的 mem_ addr指向内存分区的首地址，} {43％：也就是 array， mem_ free_ list指向内存块，每申请一次内存， mem_ free_ list就指向下一个内存块，} 从而达到分配内存的目的。

{54％：当内存分区创建好后，就可以在任务里申请和释放内存空间了。} 这里需要要注意，申请内存空间，并不像一般程序里申请多少个字节就能获取多少个字节的空间，在本嵌入式操作系统里， 申请的空间必须小于或等于内存块的大小，如果定义的内存分区为 u8 array[10][50]， 那么每次申请的内存空间绝对不能超过50个字节，而且只能申请10次内存空间。 {44％：如果申请的空间超过内存块的大小，那么只会返回NULL指针。} 申请内存块使用get_mem()函数，该函数首先判断申请内存的大小，然后判断该内存分区是否还有空余的内存块，如果有，则把该内存块的地址返回给调用者。 具体代码如下所示。

图4-31 get_mem()函数代码

{40％：释放内存的函数为free_mem()，该函数首先把释放的内存块指向内存分区里的空余内存块，然后把mem_free_list指向该内存块，从而达到归还到内存分区中。} 释放内存空间代码如下。

图4-32 free_mem()函数代码

4.7 本章小结

{44％：本章主要详细讲解了操作系统模块的实现代码，也就是整个小型嵌入式操作系统的内核代码，} 包括系统变量的定义，任务的创建和调度等功能的实现，中断服务功能的实现，时间管理和内存管理的功能实现，{46％：这四个模块组合起来，就是一个简单操作系统的基本实现。}

系统测试

5.1 测试环境

（1）主机配置

处理器： Intel Core i3-3217U，1.8GHz

内存： 4GB

硬盘： 320GB

操作系统： Windows 7旗舰版

（2）开发板配置

芯片： 三星S3C6410，ARM 11，533 MHz

内存： DDR 256MB

NAND Flash： 4GB

5.2 测试结果及分析

{51％：下面针对操作系统的各个功能模块进行测试。}

（1）任务管理模块

{64％：表5-1 任务管理模块的测试用例和结果}

{54％：测试用例操作描述预期结果实际结果测试状态}

{42％：1-1使用任务创建函数，分别创建2个任务，并显示出任务的栈地址和函数的入口地址。} {48％：创建相应优先级的任务，并打印出该任务的栈地址和函数入口地址。} 如图5-1所示通过

{45％：1-2进行2个任务的调度，每个任务打印出各自的优先级号后休眠1秒。} {41％：2个任务各自打印自己的优先级号，然后延时1秒。} 如图5-2所示通过

1-3在用例1-2的基础上，当系统运行100个时钟节拍后，创建一个更高优先级的任务，观察3个任务的调度次序。 {50％：当优先级更高的任务创建后，更高优先级的任务会比2个优先级低的任务更先运行。} 如图5-3所示通过

1-4两个任务，当系统时钟节拍为10时，挂起一个任务； 当时钟节拍为50时，恢复任务。 一开始两个任务分

别打印自己的优先级号，当一个任务被挂起后，只有一个任务在运行，当被挂起的任务恢复后，才有两个任务运行。 如图5-4所示通过

1-5两个任务，当系统时钟节拍为20时，删除一个任务。 删除任务后只有一个任务在运行。 如图5-5所示通过

用例 1-1

{ 43 %： 创建任务函数代码如下所示，分别创建两个优先级为5和6的任务： }

```
create_task(task_5, NULL, &task_stk[5][STK_SIZE - 1], 5);

create_task(task_6, NULL, &task_stk[6][STK_SIZE - 1], 6);
```

运行结果如下图所示：

图5-1 用例 1-1 运行结果

用例1-2

创建2个任务分别打印自己的优先级号，并休眠1秒，代码如下：

```
prio_t prio;

while (1) {

prio = os_tcb_current_ptr->tcb_prio;

uart_print("[app] I am task ");

uart_print_int(prio);

uart_print("\n");

sleep(1);
```

运行结果如下图所示：

图5-2 用例1-2 运行结果

用例1-3

{ 50 %： 当系统运行节拍超过100后，创建一个优先级为1的任务： }

```
if (is_created == 0 && get_os_time() )= 100) {

create_task(task_1， NULL， &task_stk[1][STK_SIZE - 1]， 1);

is_created = 1;
```

运行结果如下图所示：

图5-3 用例1-3运行结果

用例1-4

优先级为5和6的两个任务，当时钟节拍为10时，任务5挂起任务6，当时钟节拍为50时，任务5恢复任务6，代码如下：

```
if (get_os_time() == 10)

suspend_task(6);

else if (get_os_time() == 50)

resume_task(6);
```

运行结果如下所示：

图5-4 用例1-4运行结果

用例1-5

优先级为5和6的两个任务，当系统时钟节拍为20时，删除优先级为6的任务，代码如下：

```
if (delete_task_request(OS_PRIO_SELF) == OS_TASK_DEL_REQ) {

uart_print(    "I am going to delete myself\n"    );

delete_task(OS_PRIO_SELF);
```

运行结果如下所示：

图5-5 用例1-5运行结果

（2）时间管理模块

{ 54 %： 表5-2 时间管理模块的测试用例和结果 }

{54%：测试用例操作描述预期结果实际结果测试状态}

{45%：2-1创建一个任务，打印自己的优先级号后延时2秒。} {57%：任务每隔2秒打印一次信息。} 如图5-6所示通过

{41%：2-2创建一个任务，打印自己的优先级号后延时500毫秒。} {63%：任务每隔500毫秒打印一次信息。} 如图5-7所示通过

{49%：2-3创建2个任务，当高优先级任务打印自己的优先级后延时50秒，当时钟节拍为30时，低优先级任务取消高优先级任务的延时。} {44%：高优先级任务运行一次后就进入延时，当时钟节拍为30时，高优先级任务取消延时，继续运行。} 如图5-8所示通过

用例2-1

{45%：创建一个优先级为5的任务，每次打印自己的信息后延时2秒，代码如下：}

```
while (1) {
prio = os_tcb_current_ptr->tcb_prio;
uart_print("[app] I am task ");
uart_print_int(prio);
uart_print("\n");
sleep(2);
```

运行结果如下所示：

图5-6 用例2-1运行结果

用例2-2

{40%：创建一个优先级为5的任务，每次打印自己的信息后延时500毫秒，代码如下：}

```
while (1) {
prio = os_tcb_current_ptr->tcb_prio;
uart_print("[app] I am task ");
uart_print_int(prio);
```

uart_print("\n");

msleep(500);

运行结果如下所示：

图5-7 用例2-2运行结果

用例2-3

优先级为5和6的任务，优先级为5的任务运行一次后延时50秒，优先级为6的任务当时钟节拍为30时，取消优先级为5的任务的延时，代码如下：

while (1) {

prio = os_tcb_current_ptr->tcb_prio;

uart_print("[app] I am task ");

uart_print_int(prio);

uart_print("\n");

if (get_os_time() == 30)

cancel_delay(5);

sleep(1);

运行结果如下所示：

图5-8 用例2-3运行结果

（3）内存管理模块

{49％：表5-3 内存管理模块的测试用例和结果}

{54％：测试用例操作描述预期结果实际结果测试状态}

3-1创建2个任务，1个任务在时钟节拍为10时申请一个内存块，另一个任务在时钟节拍为20时申请一个内存块，申请的内存块写进当前的系统时钟节拍，并打印出当前内存块的信息。 {45％：2个任务各申请一块内存，并显示内存块的信息。} 如图5-9所示通过

3-2在用例3-1的基础上，当时钟节拍为50时，2个任务都释放自己申请的内存块。 2个任务释放自己占用的内存块。 如图5-10所示通过

用例3-1

创建2个优先级分别为5和6的任务，在时钟节拍为10和20的时候申请一个内存块，代码如下：

```
if (get_os_time() == 10) {
ptr = get_mem(BLOCK_LEN);
if (ptr !  = NULL) {
uart_print("[app] Task ");
uart_print_int(prio);
uart_print(  " used one memory,  content is:  "  );
*ptr = get_os_time();
uart_print_int(*ptr);
uart_print("\n");
```

运行结果如下：

图5-9 用例3-1运行结果

用例3-2

当时钟节拍为50时，两个任务释放各自申请的内存，代码如下：

```
if (get_os_time() == 50) {
uart_print("[app] Task ");
uart_print_int(prio);
uart_print(" release one memory\n");
free_mem(ptr);
```

运行结果如下：

图5-10 用例3-2运行结果

5.3 本章小结

本章简单列出了系统测试的基本环境，接着把系统的各个模块和功能做了测试，并把相应的结果显示出来，系统测试的结果表明系统的各个功能模块是能按照预期结果正确运行的。

结 论

{46％：通过在ARM平台上设计一个小型的嵌入式操作系统，能够加深对操作系统基本原理的理解，又能避免涉及过多的传统计算机的硬件知识。} 在本设计过程中，虽然遇到过很多底层硬件代码编写和系统设计的问题，但最终还是把一个简单的操作系统的基本雏形做出来了。

{48％：本系统具有基本的多任务调度和简单的内存管理功能，采用按优先级抢占式的调度算法，} {42％：优先级最高的任务总能第一时间运行，多任务的调度主要靠中断服务和时间管理来共同完成。} 每个系统时钟节拍的中断，系统都会对每个任务控制块进行检查，退出中断时会查找最高优先级的就绪任务，从而保证最高优先级的任务的运行。 任务的切换是本系统的重点，需要保存当前任务的寄存器数据到栈中，然后再把需要切换的任务的栈中的数据恢复到CPU的寄存器中，就可以完成任务的切换了。 {50％：任务的管理功能除了任务的创建和调度切换外，还能进行挂起任务、恢复任务、删除任务的操作。} 时间的管理功能包括任务的延时、取消延时、获取系统运行时间和设置系统运行时间。 内存管理功能采用了比较简单的方法来完成，主要把二维数组划分为不同的内存块来进行分配和管理，任务能在运行时申请需要的内存和释放不需要的内存。

{44％：虽然本系统实现了操作系统的最基本的功能，但是还有很多不足和需要改进的地方，} 像文中里面谈到的任务优先级的设计、任务的调度算法、内存的管理和分配等， 这些都是能够优化改善的，希望能在下一阶段进行更进一步的改进。 此外这个内核延还有许多能伸出来的功能，比如任务的同步和通信、信号量、事件的管理等。

通过本次的毕业设计，我进一步的了解操作系统基本的实现过程，虽然过程比较繁琐和复杂，但看到自己设计的系统能运行起来，真的感到无比的欣悦。 {41％：今后还要学习更多的操作系统的设计思路和实现过程，做一个即完善又实用的系统。}