《嵌入式系统设计》检测报告

PaperRater™
PR（PaperRater）论文检测系统

总相似率：29.09%

基本信息

文档名称	嵌入式系统设计	报告二维码防伪
报告编号	293fb0f1-b7d5-4343-9655-03cb8fc53274	
文档字数	29151	
提交人姓名	何伟强	
提交方式	粘贴文本检测	
检测范围	中国学术期刊数据库、中国学位论文全文数据库、中国专利全文数据库、中国重要会议论文全文数据库、英文论文全文数据库、港澳台学术文献库、PaperRater云论文库（涵盖互联网资源、互联网文档资源、未分类论文资源）、自建比对库	
提交时间	2014-05-28 17:10:58	

检测报告指标详情

原创率	抄袭率	引用率	字数统计	参考文献字数
70.91%	27.44%	1.65%	29151	845

相似片段位置图

注：红色部分为论文相似部分，绿色部分为论文引用部分

相似片段详情（仅显示前10条）

序号	篇名	来源	相似率
1	小卫星星务管理硬件平台软件仿真的研究	中国学位论文全文数据库	0.38%
2	Android模拟器中运行纯C++程序(一) - 我的事情真的有那么多吗? - ...	PaperRater云论文库	0.3%
3	嵌入式环境下的USB设备研究—基于μC/OS-Ⅱ的U盘文件系统设计	中国学位论文全文数据库	0.2%
4	μC/OS-Ⅲ中的高效时钟节拍管理机制_EEWorld电子工程世界搜索中心	PaperRater云论文库	0.21%
5	嵌入式非对称多处理器操作系统的构建	中国学位论文全文数据库	0.22%
6	基于ARM和μC/OS-Ⅱ的嵌入式系统的研究与开发	中国学位论文全文数据库	0.21%
7	基于ARM和μClinux的嵌入式系统研究与应用	中国学位论文全文数据库	0.2%
8	1.4 ARM处理器工作模式-vaqeteart-ChinaUnix博客	PaperRater云论文库	0.15%
9	32位RISC处理器研究及实现	中国学位论文全文数据库	0.16%
10	51c_interrupt 51单片机C语言中断指令和 服务子程序 SCM 开发 182...	PaperRater云论文库	0.15%

文档原文标注

嵌入式操作系统是嵌入式应用的核心，制作一个嵌入式操作系统，能加深对操作系统基本原理的理解和基本的实现过程。ARM11该小型操作系统具有基本的多任务管理、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多任务调度，提高系统的实时性，还有使用固定分区的内存管理来分配内存。此外，该系统还有任务延时、挂起、删除等基本操作。最后对该系统进行了测试，验证系统设计的正确性，并提出系统的不足和改进。

关键词 嵌入式操作系统；

1.1 课题来源及研究目的与意义

嵌入式开发无疑是当下热门的开发领域之一，嵌入式所涉及的领域非常广泛。51ARMARMARM ARM

学习嵌入式开发，一般可分为三个方向：嵌入式应用开发、嵌入式操作系统系统开发和嵌入式硬件开发。C/OS-QNXLinux对于学习操作系统的学生来说，由于缺少对操作系统的设计和实现的经验，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。

选择制作嵌入式操作系统这个题目，意义重大而深远。操作系统是计算机系统里面最核心的部分，通过学习和实现操作系统的过程，能加深我们对操作系统的基本原理的理解，对计算机系统的工作过程也会更加清晰。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一样的，而嵌入式操作系统相对通用的计算机系统，其内核更加小巧，非常适合学习和开发。

1.2 嵌入式操作系统的现状与分析

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]。如今，嵌入式系统已经广泛的应用于各个领域中，如通信设备、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。

Embedded Operating SystemEOSVxWorksWindows CEFreeRTOSC/OS-/LinuxeCOSQNX AndroidiOSWindows PhoneFirefox OS这些嵌入式操作系统既有商业付费的，也有开源免费的，都是当前广泛应用于各种设备上的嵌入式操作系统。

虽然低端的嵌入式设备一般都不需要嵌入式操作系统，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。C/OS-LinuxLinux

而自主的嵌入式操作系统也是有的，只不过没有用得那么广泛，因此，我国不论是嵌入式系统还是相关的嵌入式技术发展，还是有比较大的发展空间，国内的嵌入式开发前景还是非常有前景的。

1.3

本论文所写的小型嵌入式操作系统主要完成的工作和解决的难点有以下：

查阅国内外有关嵌入式系统的文献资料，包括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。

ARMS3C6410ARMNAND Flash

利用学习的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，该系统是采用了按优先级抢占式的调度方法来调度任务，并拥有简单的内存管理功能。其中设计的难点主要是任务控制块的设计，任务调度的设计等。

本系统还实现了基本的时间管理功能，能让任务进行休眠和恢复等操作，同时任务还有挂起、恢复、删除等操作。

ARM技术及开发板介绍

2.1 ARM

2.1.1 ARM处理器的简介

51AVRMIPSPowerPCARM5151ArduinoAVRARM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]。

ARM既可以认为是一家公司的名字，也可以认为是一系列处理器的名称。ARMAdvanced RISC MachineARMReduced Instruction Set ComputerRISCARM16Thumb32ARMThumb-2ARMARM

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。ARMARM7 ARMV4TCortex-AARMV7-AARMARMARM7ARM9Cortex-ACortex-M

2.1.2 ARM处理器的工作状态和工作模式

ARM处理器一般由2种工作状态和7种工作模式。ARM16/32ARMARMThumbThumb-2ARM状态就是ARM处理器完全工作在32位指令下的状态，因此在此状态下的指令均为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，占用内存空间减小，代码密度变大，能提供比32位程序代码更好的性能。Thumb-2ARMARM11Thumb-21632ARMARMARMThumb

ARM体系结构可以工作在7种工作模式下，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。当处理器工作在用户模式时，除非发生异常，否则处理器将不能改变当前的工作模式，其它的工作模式之间能进行互相切换。CPSR04[M4:M0]具体的工作模式如下表2-1所示。

表2-1 ARM处理器的工作模式

CPSR[M4:M0]

用户模式 USR ARM处理器正常的工作模式 10000

快速中断模式 FIQ 处理高速中断，用于高速数据传输或通道处理 10001

外部中断模式 IRQ 用于普通的中断处理 10010

管理模式 SVC 操作系统使用的保护模式，处理软中断SWI 10011

中止模式 ABT 当数据或指令预取中止时进入该模式，用于虚拟存储和存储保护 10111

未定义指令模式 UND 处理未定义的指令陷阱，可用于支持硬件协处理器的软件仿真 11011

系统模式 SYS 运行具有特权级的操作系统任务 11111

2.1.3 ARM处理器的寄存器

ARM处理器拥有37个32位的寄存器，其中有31个为通用寄存器，剩下的6个寄存器为状态寄存器。ARM状态下不同工作模式下的寄存器如下表2-2所示。

表2-2 ARM状态下的寄存器

工作模式 用户模式 系统模式 管理模式 中止模式 未定义模式 外部中断模式 快速中断模式

R0

R1

R2

R3

R4

R5

R6

R7

R8 R8_FIQ

R9 R9_FIQ

R10 R10_FIQ

R11 R11_FIQ

R12 R12_FIQ

R13(SP) R13_SVC R13_ABT R13_UND R13_IRQ R13_FIQ

R14(LR) R14_SVC R14_ABT R14_UND R14_IRQ R14_FIQ

R15(PC)

状态寄存器 CPSR

SPSR_SVC SPSR_ABT SPSR_UND SPSR_IRQ SPSR_FIQ

从表2-2可看出，ARM处理器工作在不同工作模式下所使用的寄存器是不同的。通用寄存器R0到R7是7种模式下共用的；快速中断模式下有自己专用的R8到R12寄存器；R13和R14除了用户模式和系统模式使用相同的堆栈指针（Stack Pointer，SP）和程序链接寄存器（Link Register，LR）外，其它模式都有自己特定的寄存器。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。

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，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表2-3所示。

表2-3 程序状态寄存器格式

31 30 29 28 27 26 -- 8 7 6 5 4 3 2 1 0

N Z C V Q 保留 I F T M4 M3 M2 M1 M0

M4-M0为模式选择位，决定处理器工作于哪种模式。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。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。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。2731ARM

ARMARMThumbThumb-2

2.1.4 ARM处理器的异常处理

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，它们的优先级及对应的异常向量地址如下表2-4所示。

表2-4 ARM异常类型、优先级及向量地址

异常类型 优先级 工作模式 异常向量地址

复位 RESET 1 管理模式 0x00000000

未定义指令 UND 6 未定义模式 0x00000004

软件中断 SWI 6 管理模式 0x00000008

指令预取中止 PABT 5 中止模式 0x0000000C

数据访问中止 DABT 2 中止模式 0x00000010

外部中断请求 IRQ 4 外部中断模式 0x00000018

快速中断请求 FIQ 3 快速中断模式 0x0000001C

7种类型的异常可分为6级，其中复位的级别最高，未定义指令和软件中断的级别最低，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。PC0x00000000

当ARM处理器发生异常后，除了是复位异常立即中止当前运行指令外，其余的处理器都是尽量完成当前指令后，再去处理异常。ARM处理器对异常的响应过程如下：

CPSRSPSR

设置当前状态寄存器的工作模式位来进入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。

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（LR）中，这样能使异常处理程序执行完后能返回原来的程序处继续向下执行指令。

给程序计数器PC强行赋值，跳转到相应的异常向量地址处执行相应的处理程序。

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。4ARM

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。LR

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：

将异常模式下的SPSR值复制到CPSR中，使得原来的CPSR的状态从相应的SPSR中恢复，回到被中断前的工作状态。

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。

清除CPSR中的中断屏蔽位，开放IRQ外部中断和FIQ快速中断。

CPSRLR如果顺序错了，程序就会发生错误了。

2.2 Ok6410

OK6410OK6410ARM11S3C6410

2.2.1 S3C6410

S3C6410[3][4]S3C641016/32RISCS3C641061/322.5G和3G通信服务提供了优化的H/W性能。该64/32位的内部总线架构是由AXI、AHB和APB总线组成的。它还包括许多强大的硬件加速器，比如运动视频处理、音频处理、2D图像处理，图形显示和缩放处理等。Multi Format CodecMFC MPEG4/H.263/H.264编解码器和VC1解码器。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。3DOpenGL ES 1.1/2.0这种3D引擎包括两个可编程着色器：一个顶点着色器和一个像素着色器。

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。DRAMFlash/ROMDRAMmobile DDRDDRmobile SDRAMSDRAM Flash/ROMNOR FlashNAND FlashOneNandCFROM

S3C6410TFT 24LCD4UART32DMA5322PWMI/OGPIOI2SI2CUSBUSB OTG480Mbps3 SD/MMCPLL

S3C6410ARMARM1176JZF-S16KB16KB16KB16KBTCM它还包括一个完整的MMU来处理虚拟内存管理。ARM1176JZF-SJAVAARM1176JZF-S3DS3C6410AMBAS3C64102-5S3C6410

2-5 S3C6410

2.2.2 Ok6410

OK6410S3C6410667MHzMobile DDRNAND FlashOK6410开发板上集成了多种高端接口，如复合视频信号、摄像头、USB、SD卡、液晶屏、以太网，并配备温度传感器和红外接收头等。这些接口可作为应用参考帮助用户实现高端的产品设计。

256MB Mobile DDRMLC 4GB NAND Flash533MHz/667MHz41RS-232DB93TTL1个RTC实时时钟，4个LED，6个按键，1个蜂鸣器。[5]2-6OK6410

2-6 OK6410

2.3

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。OK6410S3C6410Linux

3.1

ARM经过分析，本嵌入式操作系统具有以下的基本功能：

这里包括用户任务和系统任务的创建、任务的挂起和恢复、任务的删除等。开发者可以使用相应的任务函数进行操作管理。

中断服务的功能模块是本操作系统的重点，也是难点之一。ARM而中断服务子程序（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。同时任务的调度也是由中断服务来完成，包括任务数据的保存和恢复。

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。

开发者可以在任务里随时申请和释放内存。

3.2

系统的总体结构主要包括整个嵌入式操作系统的引导启动、硬件的初始化、操作系统的运行等。本论文主要把整个系统分割成4个模块：引导启动模块、系统更新模块、串口通信模块、操作系统模块。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。总体的结构模块如图3-1所示。

ARM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。

u-bootWindowsNAND FlashLinuxLinuxNAND Flash

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。

这个就是本嵌入式操作系统的实现，具体的就是任务管理模块、中断服务模块、时间管理模块和内存管理模块。

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，跟平常所见的bootloader并无区别。

OK6410

设置异常处理函数的入口地址。ARM0x00000000x0000001FARMARM0x000000000x00000000

当系统跳转到初始化处后，就可以开始设置相关的硬件了。IRQFIQS3C6410[4]CPUSDRAM NAND Flash

（3）初始化CPU频率、SDRAM和NAND Flash后，还需要重定位代码。S3C6410NAND Flash 8KBSRAM8KBSDRAMSRAMSDRAMNAND FlashTEXTDATASDRAMBSS

SDRAMmainmain

最后是进入异常处理程序的代码编写。由于发生异常后，会跳到异常向量地址处，执行异常向量地址处的函数。CPUARMLR接着跳入到异常处理程序。异常处理完毕后，恢复用户现场。

ARM76IRQIRQ5

3.4 串口通信模块的设计

同时主机可向开发板发送数据，开发板也可接收主机发来的数据。

OK6410DB9要进行串口的通信，首先需要设置开发板的串口信息，比如串口波特率，数据位，停止位，校验位等。相关的设置可根据芯片手册来设置相应的寄存器。PC WindowsQt

3.5 系统更新模块的设计

当程序进入系统更新模块后，就可以选择更新系统程序代码或者运行系统。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。

NAND Flash0x5FC000004MBNAND FlashNAND Flash上的数据读写需要根据NAND Flash的芯片手册来操作。S3C6410NAND FlashOK6410NAND FLash4KB218S3C64108KBSRAM S3C641042KB8KBSRAMNAND Flash8KB4252KB2KB

图3-2 NAND Flash前4页的数据写法

OK6410当系统的环境准备好后，就可以开始载入操作系统内核并开始运行了。

3.6 任务管理模块的设计

3.6.1 任务的定义及其结构

人们在实际生活中解决一个大而复杂的问题时，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。这种方法即能提高CPU的利用率，同时又加快了程序的执行速度。

一个任务，也称作一个线程，是一段简单的程序，该程序可以任务CPU完全只属于该程序自己。当某个问题分解成多个任务后，每个任务都是属于整个应用的某一部分，每个任务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]。典型的任务都是一个无限循环的函数[7]，由任务控制块来进行管理，一般都包括任务的栈地址，任务的优先级，任务的状态等信息。

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。

3-3

CPUCPU普通任务即为开发者创建的任务。

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。当任务控制块设置完后，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，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。

3-4

3.6.2

在嵌入式操作系统里，任务的调度是非常关键的，操作系统的实时性和多任务的能力主要取决于所采用的调度策略。从操作系统的调度策略角度来看，可分为基于优先级的调度策略和基于时间片的轮转调度策略；从调度方式上来讲，可分为抢占式调度、非抢占式调度以及可选择抢占式调度等；从时间片上来看，可分为固定与时间片可变的调度[8]。在一般的嵌入式操作系统里，为了能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调度算法[6]。CPUCPU因此，为了保证任务的实时性，本设计的嵌入式操作系统是采用基于优先级的抢占式调度机制。

采用按优先级的抢占式调度策略，任何时刻都运行着最高优先级的就绪任务。当一个运行的任务使一个更高优先级的任务进入了就绪状态，当前运行任务的CPU使用权就会被更高优先级就绪任务所抢占，如果是中断服务使一个更高优先级的任务进入就绪状态，那么当中断完成后，被中断的任务将会被挂起，优先级更高的任务就会开始运行。

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，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，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]。

图3-5 优先级抢占式调度

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。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。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。[8]

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。这里优先级号采用整数来表示，0为最高优先级，数字越大，优先级越低。优先级高的任务先运行，优先级底的任务后运行。0011000NN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。

3-6

这里采用顺序查找的算法虽然比较简单，但并不是最优。

3.6.3

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，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。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。

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。

任务链表和任务块的基本模型如下所示。

3-7

3.6.4

3.6.5

3.7 中断服务模块的设计

IRQFIQS3C6410

CPSR0x0000001FIRQ这样当有中断时，系统就会自动跳转到中断处理函数。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。S3C6410ARM11

110

当退出中断处理函数后，系统会进行一次任务调度，从而运行最高优先级就绪的任务，因此并不一定继续运行刚才被中断的任务。

ARMCPUCPU

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

时间管理模块可以提供任务的延时和系统的运行时间。10110100ms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。

10如果任务没有被挂起，那么该任务就会进入就绪状态。

1

3.9 内存管理模块的设计

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动态内存管理功能。

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。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。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。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]。

8MxN8*M*NMNN

3-8

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。

图3-9 内存控制块和内存分区的关系

3.10

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，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。

4.1 系统开发环境的搭建

PC

PCOK6410OK6410

WindowsMac OSLinuxLinuxLinux

PCgccldx86ARMx86ARMarm-linux-gccarm-linux-ld

Fedora 20 XfceSourcery CodeBench Lite 2013.11-24GCC4.8.1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。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，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。EABILinux此处下载到的安装包格式为.tar.bz2

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）。

mkdir /usr/local/CodeSourcery

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

arm-2013.11arm-none-eabiCodeSourcery

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

把交叉工具链的目录添加到环境变量中，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

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

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

2-7GCCARM

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

4.2.1 任务控制块的定义及初始化

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

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；tcb_priotcb_next tcb_prevtask_statustcb_delaytcb_del_req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。

4-3

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，优先级有序表等，如下图所示。

图4-4 任务控制块的相关变量

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。

4-5

4.3.2

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。系统和普通任务都是通过create_task()函数来创建。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。0os_tcb_prio_table[prio]NULL

init_task_stack()init_tcb()init_task_stack()

init_tcb()os_tcb_free_list

4-8 init_tcb()

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，如果系统已经运行，则会进入任务调度函数进行任务调度。start_os()

4.3.3

start_os()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。
4-9 start_os()
sched_new()
4-10 sched_new()
os_tcb_prio_table0os_tcb_prio_table[prio]NULLtask_statusos_prio_high_ready
start_task()ARM4-114-12
图4-11 start_task()函数
4-12 restore_task()
sched()sched()
4-13 sched()
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。switch_task()ARM
图4-14 switch_task()函数主要代码
SPldr
4.3.4
挂起任务，就能让任务停止运行。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。OS_PRIO_SELFTASK_SUSPENDsched()
4-15
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()。TASK_SUSPEND TASK_READY

4-16
sched()
4.3.5
delete_task_request()delete_task()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。os_tcb_listos_tcb_free_list
4-17 delete_task_request()
图4-18 delete_task()部分代码
4.4 中断服务模块的实现
IRQirq_isr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。
4-19 irq_isr
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
exit_interrupt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。
4-21 exit_interrupt
irq_isrinterrupt_switch_task()interrupt_switch_task()
4-22 interrupt_switch_task()
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。enter_critical() exit_critical()
4-23 退出和打开中断宏定义
CPSRsave_cpsr()restore_cpsr()CPSR
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
time_tick()10
图4-25 time_tick()函数代码
delay()sleep()msleep()sleep_hmsm()sleep()msleep()sleep_hmsm()delay()delay()delay()
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
sleep()sleep_hmsm()msleep()0

cancel_delay()取消延时的函数如下所示。
4-28 cancel_delay()
4.6 内存管理模块的实现
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
mem_addrmem_free_listblock_lennum_blocksnum_free
4-30 create_mem()
u8 array[10][50]array1050array[0]array[1]array[2]array[0]array[1]array[1]array[2]mem_addr arraymem_free_listmem_free_list
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。u8 array[10][50]5010NULL get_mem()
4-31 get_mem()
free_mem()mem_free_list释放内存空间代码如下。
图4-32 free_mem()函数代码
4.7

5.1
1
Intel Core i3-3217U1.8GHz
4GB
320GB
操作系统：Windows 7旗舰版
2
芯片：三星S3C6410，ARM 11，533 MHz
DDR 256MB
NAND Flash：4GB
5.2
下面针对操作系统的各个功能模块进行测试。
1
表5-1 任务管理模块的测试用例和结果
测试用例 操作描述 预期结果 实际结果 测试状态
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。 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。 5-1
1-2 21 21 5-2
1-3 1-21003 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。 5-3
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。 5-4
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。 5-5
1-1
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, &taskstk[5][STK_SIZE - 1], 5);
create_task(task_6, NULL, &taskstk[6][STK_SIZE - 1], 6);
运行结果如下图所示：
5-1 1-1
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：
5-2 1-2
1-3
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, &taskstk[1][STK_SIZE - 1], 1);
is_created = 1;
}
运行结果如下图所示：
5-3 1-3
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);

5-4 1-4
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);
}

5-5 1-5
2
表5-2 时间管理模块的测试用例和结果
测试用例 操作描述 预期结果 实际结果 测试状态
2-1 创建一个任务，打印自己的优先级号后延时2秒。 任务每隔2秒打印一次信息。 5-6
2-2 创建一个任务，打印自己的优先级号后延时500毫秒。 任务每隔500毫秒打印一次信息。 5-7

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。 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。 5-8
2-1
52
while (1) {
prio = os_tcb_current_ptr-tcb_prio;
uart_print("[app] I am task ");
uart_print_int(prio);
uart_print("\n");
sleep(2);
}

5-6 2-1
2-2
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);
}

5-7 2-2
2-3
565506305
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);
}

3
表5-3 内存管理模块的测试用例和结果
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。 2个任务各申请一块内存，并显示内存块的信息。 5-9
3-2 3-1502 2个任务释放自己占用的内存块。 5-10
3-1
2561020
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");
}
}

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);
}

5.3

ARM

本系统具有基本的多任务调度和简单的内存管理功能。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。CPU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。

参考文献

1 舒展.嵌入式系统综述.现代计算机.2011年，05期：44-46

2 Anand Lal Shimpi.ARM Partners Ship 50 Billion Chips Since 1991-Where Did They Go?.2014.http://www.anandtech.com/show/7909/arm-partners-ship-50-billion-chips-since-1991-where-did-they-go

3 Samsung Semiconductor, Inc.Samsung S3C6410 Mobile Processor.2008

4 Samsung Electronics, Inc.S3C6410X RISC Microprocessor USER’ S MANUAL.2009

5 飞凌嵌入式.OK6410开发板硬件手册

6 孙传群.嵌入式操作系统（EOS）的研究、实现及应用.扬州大学硕士学位论文.2004:18-20,42

7 史毓达.嵌入式操作系统μC/OS-Ⅱ调度机制与算法研究.湖北教育学院学报.2007,24(2)：60-63

8 曹营.嵌入式系统任务调度机制的研究与实现.大连理工大学硕士学位论文.2010：10-14

9 Jean Labrosse，Michael Barr，周东，何小庆.嵌入式操作系统中的抢占式调度策略.单片机与嵌入式系统应用.2003年，09期：5-6

10 万柳.μC/OS-Ⅱ嵌入式操作系统中抢占式调度策略分析.微计算机应用.2005年，01期：116-118

11 吴文峰.嵌入式实时系统动态内存分配管理器的设计与实现.重庆大学硕士学位论文.2013：9-17

12 赵奎，张帆.一种嵌入式系统存储管理方案.企业技术开发.2005年，01期：23-24