摘　要

随着微处理器技术与信息技术的不断发展，嵌入式系统的应用也进入到国防、工业、能源、交通以及日常生活中的各个领域。嵌入式系统的软件核心是嵌入式操作系统。然而，国内在嵌入式系统软件开发上有很多困难，主要有：国外成熟的 RTOS 大都价格昂贵并且不公开源代码，用好这些操作系统需对计算机体系结构有深刻理解。针对以上问题，免费公开源代码的嵌入式操作系统就倍受瞩目了，μC/OS-II 就是其中之一。μC/OS-II 是面向中小型应用的、基于优先级的可剥夺嵌入式实时内核，其特点是小巧、性能稳定、可免费获得源代码。

本文在深入研究 μC/OS-II 内核基础上，将其运用于实际课题，完成了基于 ARM 架构的 μC/OS-II 移植及实时同步交流采样的误差补偿研究。本文主要工作内容和研究成果如下：

1. 剖析了 μC/OS-II 操作系统内核，重点研究了 μC/OS-II 内核的任务管理与调度算法机理，得出了 μC/OS-II 内核优点：任务调度算法简洁、高效、实时性较好(与 Linux 相比)。

2. 介绍了 ARM9 体系架构，重点讲叙了 MMU(存储管理单元)功能。为了提高交流采样系统的取指令和读数据速度，成功将 MMU 功能应用于本嵌入式系统中。

3. 完成了 μC/OS-II 操作系统在目标板上的移植，主要用汇编语言编写了启动代码、开关中断、任务切换和首次任务切换等函数。

4. 针对国内外提出的同步交流采样误差补偿算法的局限性，本文从理论上对同步交流采样的准确误差进行了研究，并尝试根据被测信号周期的首尾过零点的三角形相似法，求出误差参数并对误差进行补偿。此外，考虑到采样周期 ΔT 不均匀，经多次采样后会产生累积误差，本文也给出了采样周期 ΔT 的优化算法。

5. 完成了系统硬件设计，并根据补偿算法和 ΔT 优化法则，编写了相应采样驱动和串口驱动。最后对实验数据进行了分析和比较，得出重要结论：该补偿算法实现简单，计算机工作量小，精度较高。

关键词：嵌入式系统，RTOS，μC/OS-II，ARM，同步交流采样

Abstract

With the development of Microprocessor Technology and Information Technology, embedded system has being applied to national defense, industry, energy, transportation, as well as all fields of daily life. Embedded system software is the core of embedded operating system. However, there is still much difficulty in developing embedded system so far, and it mainly includes: most overseas stable and highly reliable embedded RTOS are very expensive and not open source, as well as applying RTOS well needs mastering computer system structure. In view of the above problems, free open source embedded operating systems can be considered first, and μC/OS-II is one of them. As an embedded real-time kernel based on task priority, μC/OS-II has being mainly used in the minitype embedded application. It has lots of strongpoint such as small size, without paying, high stability and reliability.

Based on deep study of μC/OS-II kernel, this paper applies it to actual project, and completes transplanting it to ARM Microprocessor as well as research on real time synchronous AC sampling error compensation. This paper mainly contains aspects of work and research as follows:

1. μC/OS-II operating system is analyzed in detail, and especially μC/OS-II task management and scheduling algorithm mechanism is focused on. Then, it comes to a conclusion that μC/OS-II core has better advantages such as simple, efficient task scheduling algorithm, and high real-time quality compared with Linux operating system.

2. ARM9 architecture is well introduced, and among it MMU function (Memory Management Unit) is presented particularly and specifically. In order to improve the speed of instructions fetch and data read, MMU is applied to this embedded system successfully.

3. Porting μC/OS-II operating system to target board is successfully completed. The main work is to design boot code, close/open interruption, task switching and first task switching function etc by using assembly Language.

4. Aiming at the limitations of these synchronous AC sampling error compensation algorithms which have been raised at home and abroad, this paper carries out deep research on accurate synchronous AC sampling error in theory, tries to work out error parameters and then compensates error precisely based on triangle similar law which can be used in the triangles standing at zero-crossing point nearby in a measured signal cycle. In addition, taking into account the uneven sampling period ΔT, after many times repeated sampling it will result in a cumulative error, and this paper also gives an optimization algorithm to sampling period ΔT.

5. This paper has completed the system hardware structure design, and has finished corresponding sample-driven and serial port driven program in accordance with the error compensation algorithm and ΔT optimization rule. Finally with experimental data

analyzed and compared, an important conclusion is drawn: this kind of error compensation algorithm is easy to be realized, needs lower computer workload, and has high precision.

Keywords: Embedded operating system, RTOS, μC/OS-II, ARM, Synchronous AC sampling

湖北工業大學

学位论文原创性声明和使用授权说明

原创性声明

本人郑重声明：所呈交的学位论文，是本人在导师指导下，独立进行研究工作所取得的研究成果。除文中已经标明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的研究成果。对本文的研究做出贡献的个人和集体，均已在文中以明确方式标明。本声明的法律结果由本人承担。

学位论文作者签名：杨辉　　日期：2009年6月25日

学位论文版权使用授权书

本学位论文作者完全了解学校有关保留、使用学位论文的规定，即：学校有权保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权湖北工业大学可以将本学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。

学位论文作者签名：杨辉　　指导教师签名：

日期：2009年6月25日　　日期：09年6月25日

第1章 绪论

1.1 课题意义和目标

随着嵌入式微处理器的发展，嵌入式操作系统日新月异，而且他们被广泛地应用于工业的各个领域。嵌入式实时操作系统的引入，使产品的实时性和可靠性得到显著提升。作为一名嵌入式开发者，很有必要掌握一种嵌入式实时操作系统。

本课题的意义在于，以源代码开放的实时操作系统 μC/OS-II 为内核[1]，以基于 S3C2440 微处理器进行嵌入式系统的开发和研究，这对于利用开源代码[2-4]开发嵌入式实时操作系统具有极好参考价值。另外，本论文对同步交流采样误差进行了研究，这对提高交流采样精度有着重要意义。

对于本课题，主要有两个目标：一是掌握实时操作系统原理，以著名的嵌入式实时操作系统 μC/OS-II 为研究对象，彻底搞清其实现细节和运行方式，并实现在微处理器上移植和调通系统。二是搭建一个 ARM9 嵌入式同步交流采样系统(包括系统的硬件设计，μC/OS-II 的移植，采样和串口驱动的编写等)，从理论和实践上对本文提出的同步交流采样误差补偿进行研究和验证。

1.2 嵌入式系统概述

本文是基于嵌入式系统的开发和研究，所以对嵌入式系统、嵌入式操作系统、实时操作系统、嵌入式实时操作系统要有清楚地认识，以下是它们的定义和介绍。

1.2.1 嵌入式系统和嵌入式操作系统

嵌入式系统[5]定义为：以应用为主体，软件和硬件均可裁减，适应对功能、可靠性、成本、以及功耗等要求较严格的专用计算机系统。嵌入式系统由硬件和软件两大部分组成，其中软件核心是嵌入式操作系统。

嵌入式操作系统[6](Embedded Operating System)是一种支持嵌入式应用的专用软件。嵌入式操作系统与通用操作系统相比，具有更强的实时性和硬件依赖性。

1.2.2 实时操作系统

实时操作系统是指能支持实时控制的专用操作系统软件。实时的含义并非指速度快，而是指有响应时间的限定，超过一定时间都认为是非法的，所以，实时

操作系统具有可预测性和高可靠性。实时操作系统与普通操作系统的主要区别有：任务处理的确定性、响应灵敏度、用户参与控制、可靠性以及故障保护措施[7][8]。所以在实时操作系统中，任务处理时间是确定的、可预测的，这一点非常关键[9][10]。

实时操作系统分两种，一种是“硬实时”，另一种是“软实时”。“硬实时”指对任务处理和时间要求上，只能作出承诺或拒绝。“软实时”先对系统完成该任务的能力进行分析和估计，然后由计算出的概率决定任务的执行与否。目前对软实时操作系统的调度算法研究颇多，主要停留在采用经典反馈控制技术的常规控制算法上[11-14]。对“硬实时”方面的调度算法研究，包括速率单调调度法[15](RM)、时限调度法(DM) [16]和最早时限优先调度算法(EDF)[17]。EDF 调度总是比固定优先级调度取得更高的利用率。相对于速率单调调度法(RM)和时限调度法(DM)，EDF 的时限是任意的，更适用于实际的实时系统。当前，使用较多的实时操作系统都属于“软实时”。

1.2.3 嵌入式实时操作系统 μC/OS-II 优点

μC/OS-II 嵌入式实时操作系统，已通过了美国航空航天管理局 FAA 的安全认证，它的稳定性和可靠性均已通过严格验证。μC/OS-II 具有以下优点：

(1) 源代码公开且免费。

(2) 代码量小，其核心代码只有 8K 字节左右，另外用户还可以根据自己的需要对它进行裁剪和修改，所以用户实际系统的代码量还可以减少。

(3) 内核对微处理器以及程序存储器，数据存储器的要求不高，能够适应各种不同的硬件系统。

1.2.4 嵌入式实时操作系统开发的发展动向

目前，嵌入式实时操作系统及嵌入式开发环境的发展方向分别是：

(1) 嵌入式实时操作系统正向实时超微内核(Nanokernel)方向发展。

上世纪80年代末期，国外就提出了微内核[18-20] (Microkernel)思想，就是把传统操作系统的共性抽取出来，作为构成各种操作系统微内核的基础，而把具体功能放在微内核之外。本文的μC/OS-II就是一种微内核。

近几年，国外出现了超微内核[21] (Nanokernel)。实际上，超微内核是一种更基本的内核代码层，在它基础之上可以构造各种嵌入式实时操作系统。这种实时内核具有更好的可重用性，同时可伸缩性[22](Scalability)也较强，所谓可伸缩性，就是指能支持和满足多种实时应用的需求。

(2) 嵌入式开发环境将更开放和更集成化。

在嵌入式开发中，应用软件的使用，需要开发者具有一定深度的操作系统知识。所以，要开发和设计高性能的操作系统应用软件，功能强大的交叉开发工具必不可少。目前，国外的操作系统集成开发环境正向高度集成、编译优化、仿真和验证等方向发展。

1.3 本文的主要工作和安排

本文主要的工作内容是：

(1) 研究和分析了 μC/OS-II 源代码，为 μC/OS-II 在目标板上的移植打下基础。

(2) 完成了本采样系统硬件电路的设计。

(3) μC/OS-II 基于 ARM920T(S3C2440)系统的启动代码的设计实现。

(4) 编写 μC/OS-II 到 ARM 的移植代码，总结 μC/OS-II 移植的关键技术等。

(5) 编写串口驱动和采样驱动代码，完成应用程序的开发。

(6) 在 ADS 开发环境下，采用 ARMulator 和 H-JTAG 仿真器调试和测试内核及验证移植的正确性(包括驱动程序)。

(7) 本文从理论上推导出同步交流采样中误差的主要来源，提出了同步交流采样的优化算法，在移植了 μC/OS-II 的 S3C2440 硬件平台上加以分析和验证。

论文安排：

第一章绪论，主要介绍嵌入式实时系统相关的概念。

第二章详细剖析 μC/OS-II 的代码及工作机理。

第三章介绍 ARM9 的体系结构，包括编程模型和内存管理单元(MMU)的工作原理。

第四章分析移植 μC/OS-II 的方法和步骤，特别是针对不同的处理器必须要修改的几个函数，并把 μC/OS-II 移植到 S3C2440 目标板上。

第五章从理论上推导出软件同步交流采样[23]误差的主要来源，提出同步交流采样的优化算法。

第六章进行系统硬件电路设计，同时由误差补偿算法和采样周期优化法则，编写实时采样驱动程序和串口驱动程序，并从实验数据中分析和验证这种算法的可行性。

第七章对本论文进行了总结和展望，说明了本文取得的成果以及不足之处。

第 2 章 嵌入式实时操作系统 μC/OS-II 的剖析

2.1 引言

μC/OS-II 操作系统的多任务内核，主要包括任务管理、时钟管理、任务间通信与同步、中断管理等功能。这些功能是通过内核函数的形式交给任务调用的，且功能函数全部用 ANSI C 语言编写。

本章对 μC/OS-II 操作系统内核进行详细分析，是为后面 μC/OS-II 在 S3C2440 目标板上的移植打基础。文中首先简单介绍了任务的概念，而后围绕任务状态转换图说明相关函数的功能，详细讲叙了任务链表的原理和作用。任务调度分任务级调度和中断级调度两种，任务级调度函数为 OS_Sched( )，中断级调度函数为 OSIntExt( )。任务的就绪组 OSRdyGrp 和就绪表数组 OSRdyTbl[]非常关键，它们是判断任务是否处于就绪态的重要依据。信号量、消息邮箱、消息队列是任务间的通讯机制，其中事件控制块 OS_EVENT 是一种数据结构，它是实现任务间通讯的基础。时钟节拍函数 OSTimeTick( )是俗称操作系统心跳函数，是用来判断任务的延时节拍是否结束，如果结束就将该任务设置成就绪态，为任务调度作准备。

2.2 μC/OS-II 的任务管理与调度

2.2.1 任务、任务优先级和任务切换

在 μC/OS-II 中，任务是一个无限的循环，其程序结构如下：

```
void task(void *pdata )
{
  for( ;;)
  { /*用户代码*/ }
}
```

每个任务都赋予一定的优先级，数值越小优先级越高，每个任务都有它自己的一套 CPU 寄存器和堆栈空间。每个任务都处于以下 5 种状态中的一种：睡眠态、就绪态、运行态、等待态和中断态。

图 2-1 是 μC/OS-II 控制下的任务状态转换图，详细描叙了在调用相关内核函数后，任务从一种状态转换到另一种状态的过程。

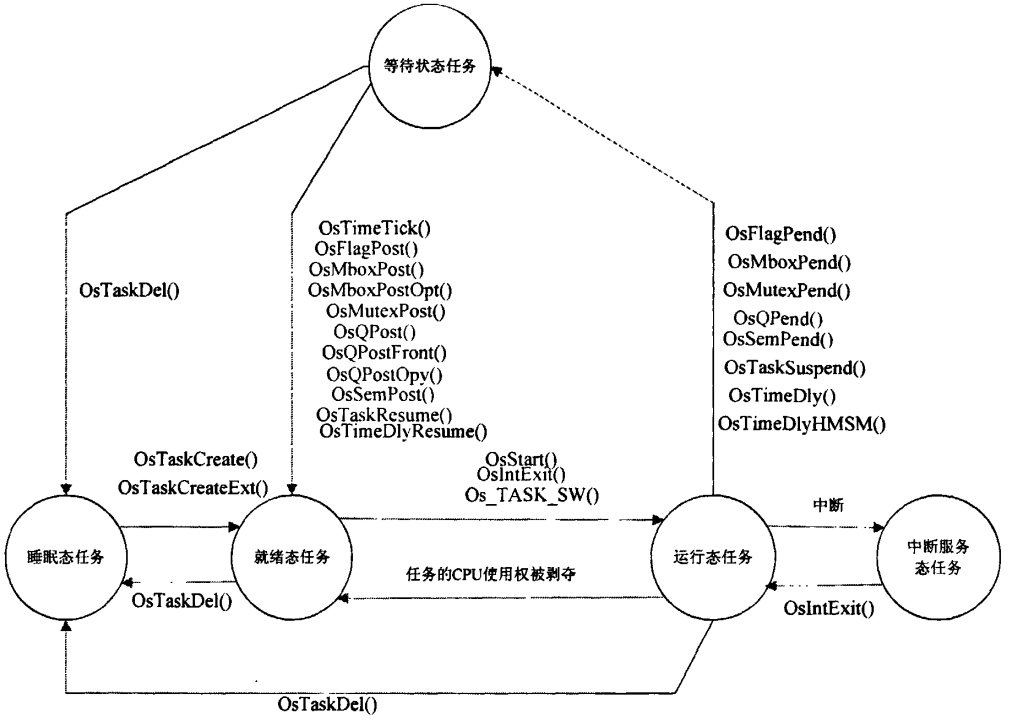

图 2-1　任务状态转换图

在初始化 μC/OS-II 时，所有的任务控制块 OS_TCB 被链接成单向空任务链表，任务的总数可以根据实际需求选取，最多为 64 个。事先建立的任务控制块指针数组(数组的每个元素是对应任务控制块的地址)的大小就是任务的总数，各个任务控制块(一种结构体)是单向空任务链表的一个节点，图 2-2 是初始化 μC/OS-II 时的空任务链表。

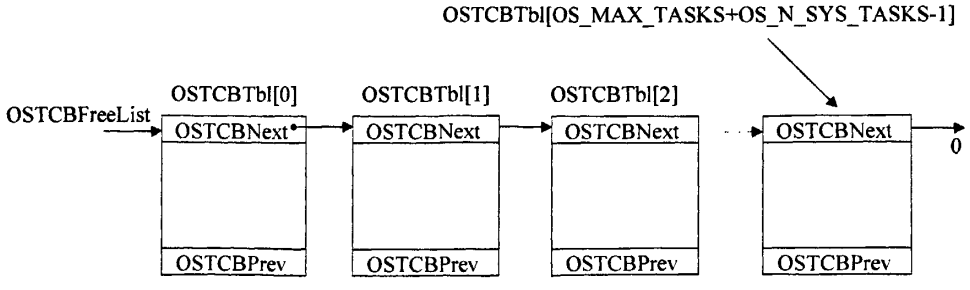

图 2-2　空任务链表

下面分析建立任务后的情况，假设现在建立了两个任务，分别是空闲任务和统计任务，且先建立空闲任务，显然，OSTCBTbl[0]空任务控制块赋给空闲任务，OSTCBTbl[1]空任务控制块赋给统计任务，OSTCBFreeList 指向链表中的第三个空任务控制块(也就是 OSTCBTbl[2]任务控制块)。同时，关键的是已建立任务的前向

任务控制块指针 OSTCBPrev 发生了作用，指向上一个任务控制块，如图 2-3 所示。

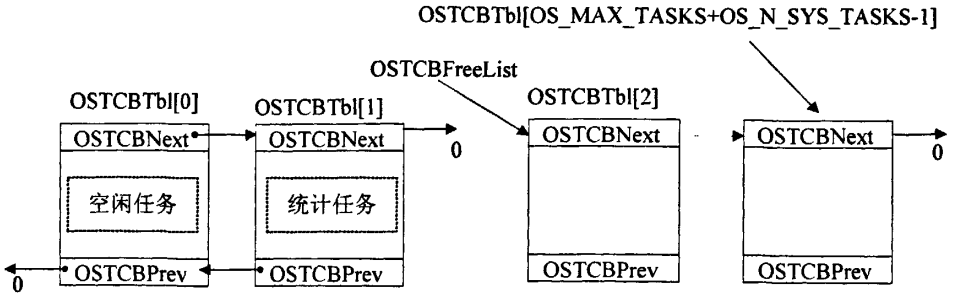

图 2-3 任务建立后的链表图

任务级的任务切换一般是通过宏调用 OS_TASK_SW( )实现，这个宏调用通常含有软中断指令或指令陷阱(TRAP)[24-25]，因为 μC/OS-II 假定任务切换是靠中断级的服务程序实现的。任务切换的过程是：先将被挂起任务的 CPU 寄存器(保护现场)压入栈中，然后将要切换的任务寄存器值从堆栈中恢复(恢复现场)。

2.2.2 任务管理与调度

μC/OS-II 是可剥夺型内核，优先级最高的任务一旦被时钟函数或信号量等置于就绪态，就会立即拥有 CPU 所有权并开始执行。μC/OS-II 与 Linux 调度法则不同，μC/OS-II 不支持时间片轮转调度[26]，另外每个任务的优先级不相同并且唯一。

μC/OS-II 的任务调度包括任务级和中断级两种，分别由函数 OS_Sched( )和 OSIntExt( )完成。下面仅分析任务级调度函数 OS_Sched( )。

如图 2-4 所示，任务调度都发生在临界段代码区中(中断关闭)，首先是判断是否在中断服务程序 ISR 或调度禁止时调用 OS_Sched( )，如果不是，就从就绪表中找到最高优先级的就绪任务，另外还要判断这个就绪任务是否为当前任务，如果不是当前任务就需要调用函数 OS_TASK_SW( )进行任务切换，如果这个就绪任务就是当前任务，就直接退出调度函数 OS_Sched( )。

μC/OS-II 的任务调度是严格按照优先级 prio 进行的，μC/OS-II 总是运行优先级最高的就绪任务，因此怎样确定优先级最高的就绪任务就非常关键了。在 Linux 操作系统中，这一调度算法比较复杂，但在 μC/OS-II 中却相当简单，这主要取决于 μC/OS-II 的独特查表法。

如图 2-5 所示，就绪表的每位对应了各个任务的标志，置 1 表示处于就绪态，清 0 表示该任务处于等待或者还没有建立。就绪表有 2 个变量：OSRdyGrp 和 OSRdyTbl[]，其中 OSRdyGrp 是任务组状态变量，每一比特位代表任务组中是否有任务进入就绪态。如果有任务进入就绪态，OSRdyGrp 和 OSRdyTbl[]的相应位

就置 1。μC/OS-II 有 64 个优先级，分别取值 0~63，可以用图 2-5 左下角的 6 位二进制表示，高 3 位“YYY”决定了在 OSRdyTbl[]数组中的第几个元素，同时也对

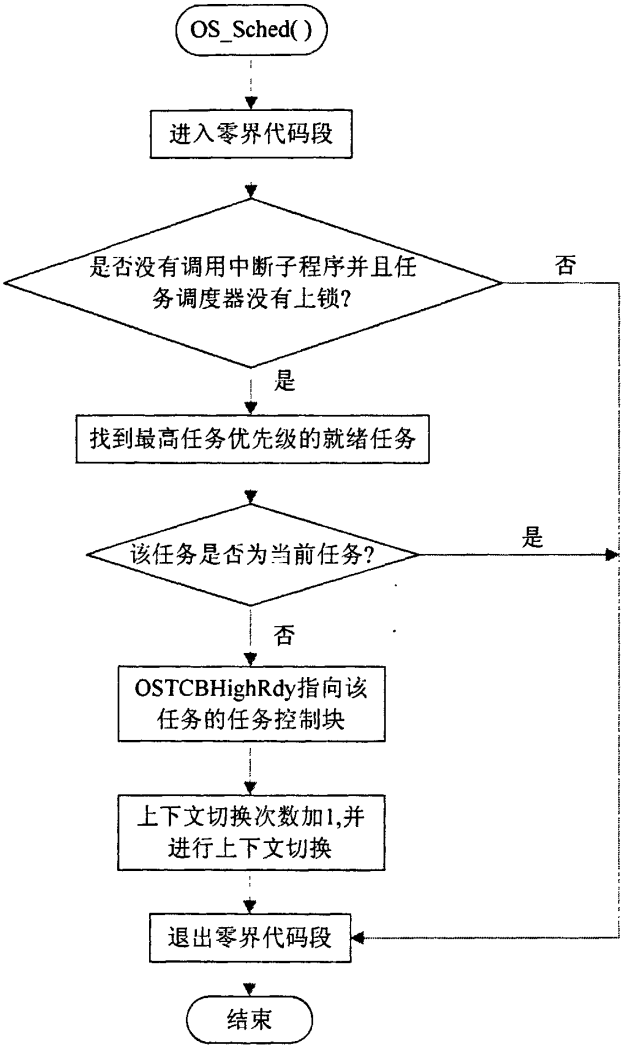

图 2-4　任务调度函数流程图

应了在 OSRdyGrp 的第几位，低 3 位“XXX”确定了 OSRdyTbl[]数组中对应元素的第几位。为了说明原理，以优先级 40 为例，优先级 40 用二进制表示为 101000，高 3 位 101 决定了 OSRdyTbl[5]，低 3 位 000 决定了 OSRdyTbl[5]内 0bit 的位置。现在要将 OSRdyGrp 和 OSRdyTbl[]的相应位置位，用数学式表示如下：

$OSRdyGrp \mid= 0x20$

$OSRdyTbl[5] \mid= 0x01$

由以上的数学式可以看出：要想将第 m 位置 1，就需与 2^m 相或。为了方便，μC/OS-II 把 2^m (m=0~7)的 8 个值预先装在数组 OSMapTbl[]中，如表 2-1 所示。

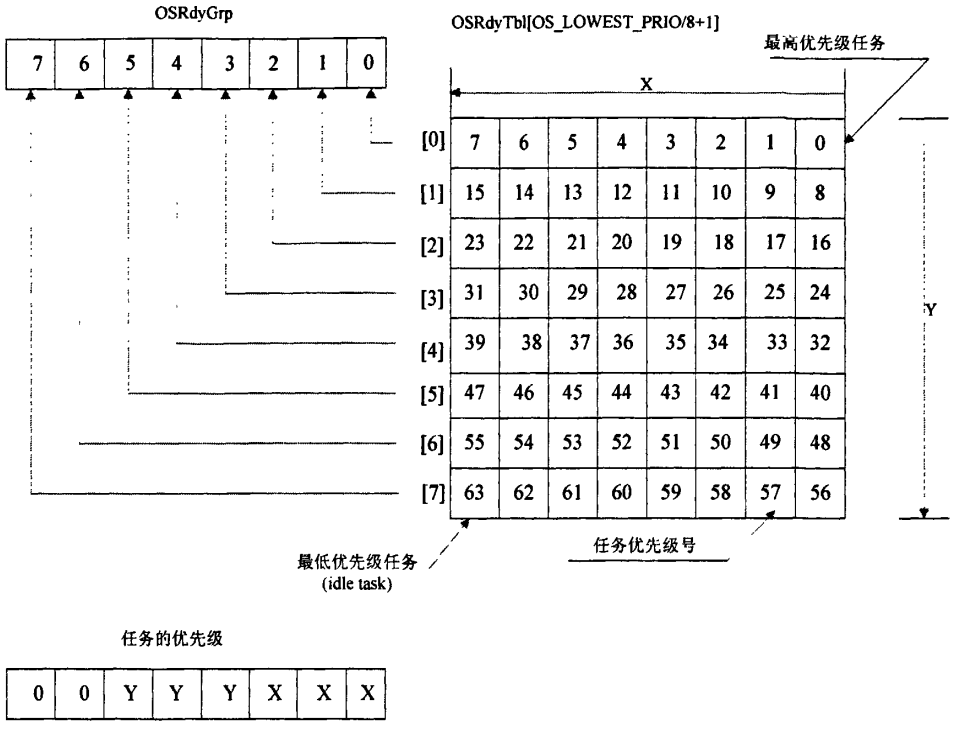

图 2-5　任务就绪表

利用 OSMapTbl[]，通过优先级 prio 来设置 OSRdyGrp 和 OSRdyTbl[]的数学式如下：

$OSRdyGrp \mid= OSMapTbl[prio >> 3]$

$OSRdyTbl[prio >> 3] \mid= OSMapTbl[prio \& 0x07]$

为了尽快找到优先级最高的就绪任务，μC/OS-II 用查表法实现，这个表就是数组 OSUnMapTbl[]，详见参考文献[1]。

现在要获知最高优先级的就绪任务，可以用如下简短程序实现：

```
y=OSUnMapTbl[OSRdyGrp]
x=OSUnMapTbl[OSRdyTbl[y]]
OSPrioHighRdy=(INT8U)((y<<3)+x)
```

至此，μC/OS-II 的调度的确非常简洁和高效，主要原因有以下两点：

(1) μC/OS-II 的调度只考虑优先级，不考虑其它因素。

(2) μC/OS-II 中每个任务只能拥有唯一的且各自不同的优先级。

2.2.3 μC/OS-II 的实时性分析

操作系统的实时性主要由以下几个参数决定：系统响应时间、任务切换时间以及中断延迟时间。其中，系统响应时间除了与后两者有关外，还和调度延迟时

间有关，所以归根结底，操作系统的实时性衡量标准主要由任务切换时间、中断延迟时间以及调度延迟时间三者决定。和操作系统 Linux 相比，μC/OS-II 具有更好的实时性，下面从影响实时性的三点要素，对 μC/OS-II 和 Linux 的实时性进行分析和比较。

表 2-1 OSMapTbl[]的值

下标	位掩码(二进制)
0	00000001
1	00000010
2	00000100
3	00001000
4	00010000
5	00100000
6	01000000
7	10000000

1. 中断延迟时间

中断延迟是指当外设发出中断信号时，CPU 可能因为正在处理别的中断，导致不能及时处理该中断请求，使得 CPU 经过一段时间后才能去响应，这段时间就叫做中断延迟。从定义可以知道，中断延迟时间的多少取决于关闭中断的时间(排除中断嵌套，发生中断嵌套时中断肯定是允许的)。在 Linux 操作系统中，有些内核线程 thread 是不允许被中断的，对于具体的实时应用来说，这种内核线程长时间的中断关闭是无法接受的。而在 μC/OS-II 操作系统中，由于允许中断嵌套，μC/OS-II 在中断服务程序里不需要关闭中断，除非有原子操作或存在代码临界区，这时中断才必须关闭，并且这个关闭时间也是非常短的，因此 μC/OS-II 的中断延迟时间比 Linux 操作系统要短得多。

2. 调度延迟

调度延迟的概念：从中断服务的结束到目标任务被切换所需要的时间。在中断服务结束后的任务调度中，如果目标任务能被切换并开始运行，那么调度延迟时间的长短就只取决于调度算法。Linux 操作系统的“非可剥夺”内核使得调度延

迟时间比较长，并且是难以预测的。μC/OS-II 是“可剥夺”型内核，其调度算法也非常简单和高效，从这点看，μC/OS-II 的调度延迟时间比 Linux 操作系统短并且可以预测。

3. 上下文切换时间

上下文切换又叫任务切换，就是将当前任务的寄存器状态压栈保存起来(保护现场)，同时把将要运行的任务的寄存器状态从存储区中出栈 (恢复现场)。因此，CPU 的寄存器数量和操作系统的实现决定了任务切换时间的长短。在 μC/OS-II 操作系统中，每个任务都有自己的堆栈区，任务切换也只需要二十多条汇编指令就可完成，因此 μC/OS-II 的上下文切换时间很短。

2.3 任务间的通讯与同步

2.3.1 事件控制块

在 μC/OS-II 中，事件控制块是为了实现共享资源的访问和任务间的通信，事件控制块 0S_EVENT 的构造如下：

```
typedef struct
{
  void *OSEventPtr; //指向消息或者消息队列的指针
  INT8U OSEventTbl[OS_EVENT_TBL_SIZE]; //等待任务列表
  INT16U OSEventCnt; //计数器(当事件是信号量时)
  INT8U OSEventType; //事件类型，可以是信号量、消息邮箱、消息队列等
  INT8U OSEventGrp; //等待任务所在的组
} OS_EVENT;
```

结构中有两个成份. OSEventGrp 与. OSEventTbl[]，它们构成了一个“等待(任务)位图”，用于维护任务的等待和唤醒状态。与前面的就绪任务组 OSRdyGrp 和就绪任务表 OSRdyTbl[]类似，实现方法也相同。

. OSEventPtr 用于消息邮箱和消息队列中保存消息，. OSEventCnt 用于信号量时的计数器，. OSEventType 是事件类型标志，可以是信号量、消息邮箱、消息队列等。

2.3.2 信号量

共享资源的访问、事件标志以及多任务间进行通讯和同步，都要用到信号量。对信号量的操作有 3 种：初始化(initialize)，等信号(wait)，发信号(post)。

在使用 μC/OS-II 的信号量时，以下两部分必不可少：一部分是信号量计数值，为 16 位；另一部分是等待信号量的等待任务所在的组和等待任务列表。μC/OS-II 的等待信号量函数 OSSemPend( )和发送信号量函数 OSSemPost( )的程序流程图如图 2-6 和图 2-7 所示。

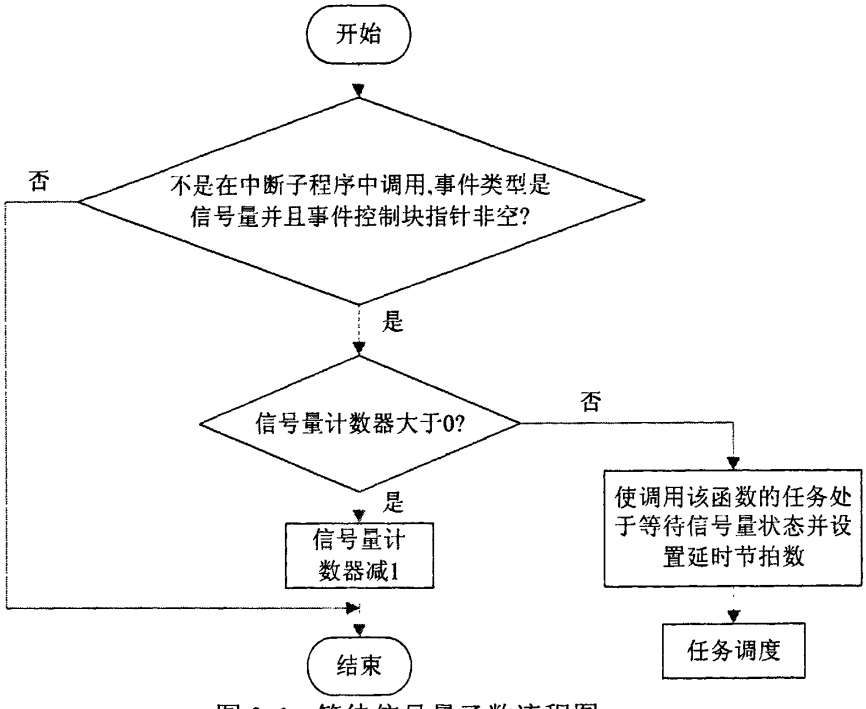

图 2-6　等待信号量函数流程图

μC/OS-II 信号量最大的特点是可以设置等待时钟节拍数，所以实际上任务只要不是一直等待，并且只要等待节拍的时间到，系统就会放弃等待信号量而进入就绪态，所以不会发生死锁[27](死锁就是两个任务无限期地互相等待对方控制着的资源)。

2.3.3 消息邮箱和消息队列

消息邮箱是一种提供指针变量传递的通讯机制。发送消息由函数 OSMboxPost(OS_EVENT *pevent，void *msg)完成，这里的 pevent 指向消息邮箱，msg 为发送的消息地址。接收消息则由函数*OSMboxPend(OS_EVENT *pevent，INT16U timeout，INT8U *err)完成，timeout 为等待消息邮箱的极限时间节拍数。

消息队列是 μC/OS-II 用于发送邮箱阵列的通讯机制。任务或中断服务子程序可以将一则消息放入消息队列，任务也可以从消息队列中得到消息。消息队列的具体操作过程和消息邮箱很相似，主要函数有 OSQPost( )和 OSQPend( )等。

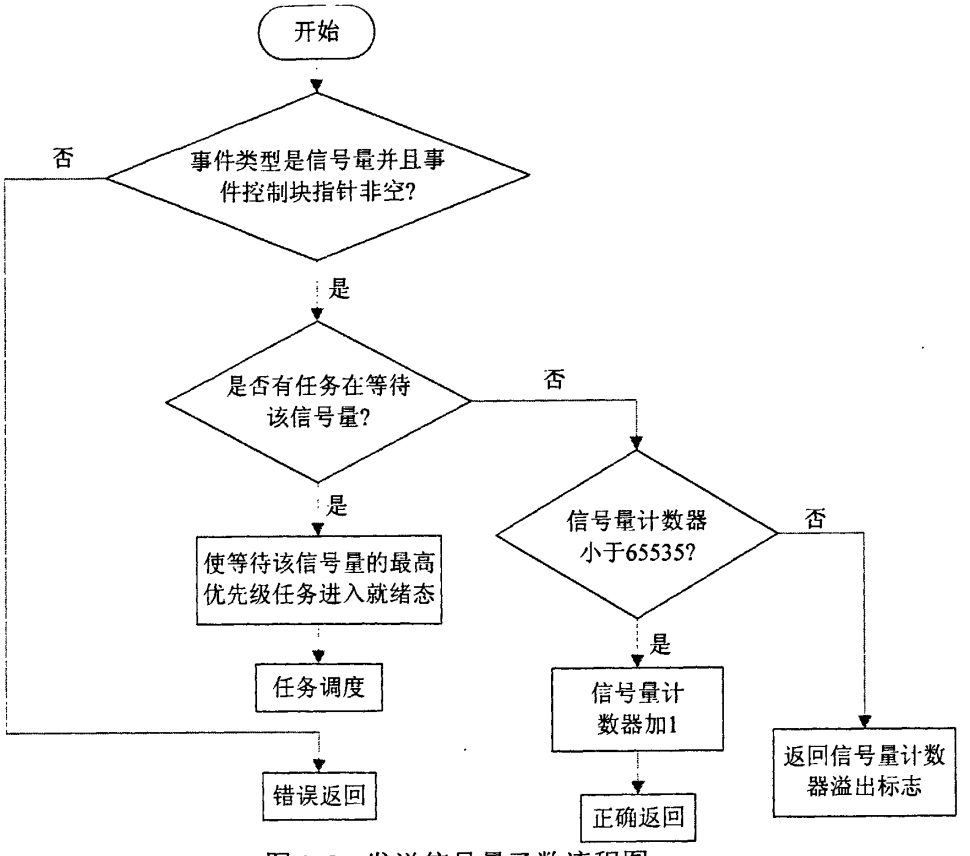

图 2-7　发送信号量函数流程图

2.4 中断与时钟节拍

在 μC/OS-II 操作系统中，中断服务子程序 ISR 要用汇编语言编写，当发生中断时，将 CPU 寄存器入栈，再执行中断服务程序，中断服务子程序处理完成后，对于可剥夺型内核而言，系统进入优先级最高的就绪任务处执行。

嵌入 μC/OS-II 内核后，中断服务程序与一般中断处理有所不同。中断发生时，寄存器入栈，同时中断嵌套变量 OSIntNesting 加 1 以通知内核进入了中断服务程序(OSIntNesting 的初始值为 0)。执行完中断程序后，在弹出寄存器前，要调用 OSIntExit( )函数，将 OSIntNesting 减 1，并根据 OSIntNesting 的值来判断是否进行中断级的任务调度。如果 OSIntNesting 大于 1 表示有中断嵌套，如果 OSIntNesting 为 0，就得进行中断级的任务调度，将 CPU 切换到更高优先级的任务。OSIntExit( ) 的程序流程图如图 2-8 所示。

时钟节拍是周期性发生的时钟中断，在 μC/OS-II 中，一般将时钟频率设置在

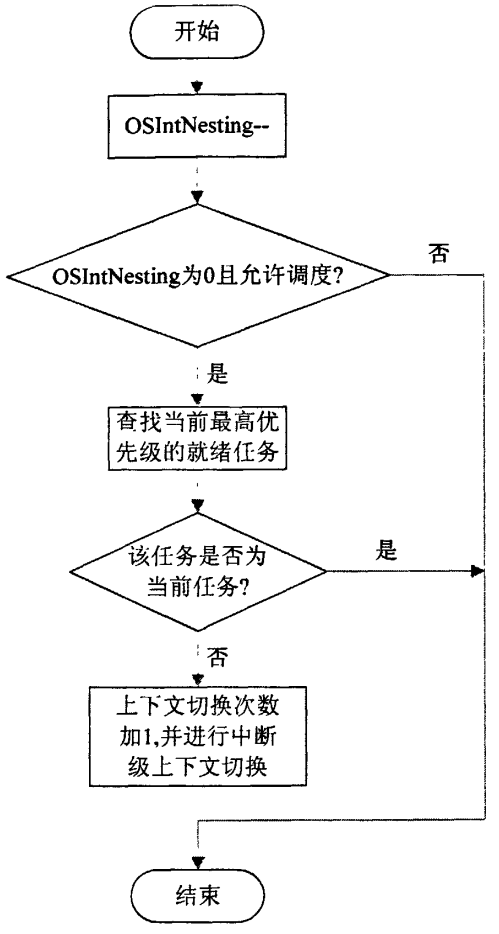

图 2-8 OSIntExit( )流程图

10~100Hz 之间。当时钟中断发生时，OSTime 就加 1，所以 OSTime 的值是操作系统启动后的时钟节拍数目。μC/OS-II 的时钟中断核心函数是 OSTimeTick( )，其主要功能是判断任务的延时节拍是否结束从而将该任务置于就绪态。OSTimeTick( ) 的程序结构如图 2-9 所示。

2.5 μC/OS-II 的启动

μC/OS-II 的操作系统初始化函数为 OSInit ( )，其主要功能是初始化操作系统所有的变量和数据结构以及建立空闲任务 OS_TaskIdle( )或者统计任务 OS_TaskStat( )，其中空闲任务总是处于就绪态并且优先级为最低值 63，这样可以确保多任务系统启动前至少有一个任务可以运行。多任务启动是通过调用 OSStart( ) 函数实现，其主要功能是从任务就绪表中找出优先级最高就绪任务的任务控制块，

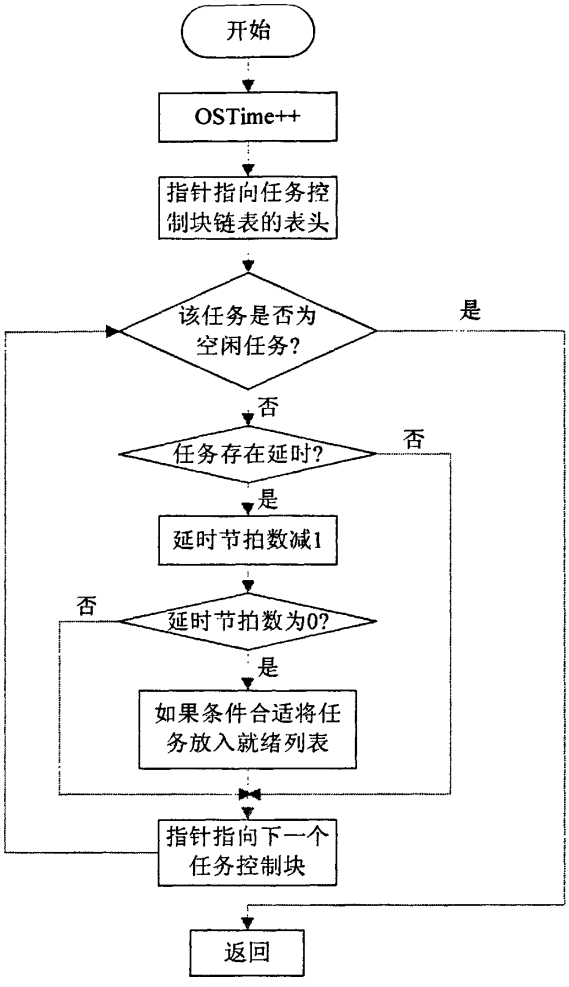

图 2-9　OSTimeTick( )流程图

从而得知初始寄存器的栈顶地址。然后，OSStart( )调用函数 OSStartHighRdy( )将任务栈中保存的现场(上次保存的寄存器状态)弹回到 CPU 寄存器中，也就是恢复现场，执行中断返回，使系统进入用户应用程序运行，就可进行多任务的调度。

2.6 小结

操作系统的核心是内核，本章主要对 μC/OS-II 操作系统内核进行了剖析，这为后文 μC/OS-II 在交流采样系统中的移植作铺垫。本章重点研究和分析了 μC/OS-II 内核的任务管理与调度算法，通过与 Linux 内核相比，μC/OS-II 内核具有任务调度算法简洁、高效、实时性好等优点。另外，也介绍了 μC/OS-II 任务间的通讯与同步机制，包括信号量、消息邮箱和消息队列，其中信号量机制被用于本课题交流采样系统的数据处理任务和串口通讯任务中。

第3章 ARM9体系结构

3.1 引言

介绍了ARM的异常模式、寄存器组、异常处理以及MMU(存储管理单元)功能。MMU功能，实际上是引入了虚存的概念，另外MMU的激活需要执行协处理器相关指令。MMU初始化也是本嵌入式系统软件编程的重要部分，如果不使用MMU功能，H-JTAG在线调试无法进行。MMU的初始化函数是MMU_Init( )。

本嵌入式系统的程序入口点在地址0x00000000处，但0x00000000~0x001fffff是Norflash区，程序运行速度较慢，需要把代码从0x00000000~0x001fffff拷贝到0x30000000~0x301fffff处，然后执行MMU功能，实现虚地址重映射，具体地址变换信息则在地址转换页表中。

开通MMU功能后，当异常中断到来时，例如复位信号，本应跳到物理地址0x00000000处，但实际上跳到物理地址0x30000000处执行程序。因为一旦复位，CPU的PC值就会指向虚地址0x00000000，但在重映射的作用下使虚地址0x00000000对应于物理地址0x30000000，从而导致CPU真正访问的物理地址是0x30000000，尽管PC值为0x00000000。

3.2 ARM9的编程模型

3.2.1 处理器模式

ARM是目前应用比较广泛的RISC嵌入式微处理器之一，其突出特点是低成本、低功耗和高性能。表3-1是ARM的7种工作模式及描述，除User模式外的其它6种模式都叫做特权模式，除User模式和System模式外的其它5种模式都叫做异常模式[28]。这里着重强调两种模式：SVC模式和IRQ模式。这两种模式之间的转换可以通过硬件或软件来实现。μC/OS-II内核在运行过程中，大部分时间处于SVC模式，如果时钟中断产生，系统会自动从SVC模式切换到IRQ模式，在中断服务的末尾，则需要用户通过编程使CPU从IRQ模式恢复到SVC模式。

表 3-1 处理器模式

处理器状态或模式	描述
用户模式(User)	正常用户模式,程序正常执行模式
中断模式(IRQ)	处理普通中断
快中断模式(FIQ)	处理快速中断,支持高速数据传送或通道处理
管理模式(SVC)	操作系统保护模式,处理软件中断(SWI)
中止模式(Abort)	处理存储器故障,实现虚拟存储器和存储器保护
未定义模式(Undefined)	处理未定义的指令陷阱,支持硬件协处理器的软件防真
系统模式(System)	运行特权操作系统任务

3.2.2 ARM 寄存器

如表 3-2 所示，ARM 寄存器分两种：通用寄存器和状态寄存器。通用寄存器 31 个，包括程序计数器 PC，状态寄存器 6 个，包括 CPSR 和各种模式下的 SPSR，这里需要强调一点，在用户模式和系统模式下没有 SPSR[28]。

表 3-2 ARM 状态下的寄存器组织

	模式						
	特权模式						
	异常模式						
	用户	系统	管理	中止	未定义	普通中断	快速中断
通用寄存器和程序计数器	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						
	无		SPSR_svc	SPSR_abt	SPSR_und	SPSR_irq	SPSR_fiq

通用寄存器有 R0~R7、R8~R14 和 R15。其中 R0~R7 是与处理器模式无关的寄存器，R8~R14 是与处理器模式有关的寄存器，R13 是堆栈指针 SP，R14 是链接

寄存器 LR，R15 是程序计数器 PC，注意所有 7 种模式都共用同一个 PC。

状态寄存器有 CPSR 和 SPSR，CPSR 又叫当前程序状态寄存器，SPSR 又叫保存的程序状态寄存器。注意所有模式都共用一个当前程序状态寄存器 CPSR，每种异常模式都有自己的 SPSR，但用户模式和系统模式没有 SPSR。

3.2.3 异常处理

ARM 处理器对异常中断的响应过程是：首先将 CPSR 的内容保存到将要执行的异常中断对应的 SPSR 中，然后设置当前 CPSR 中相应的位，并将返回地址装进 lr_mode 中，最后将 PC 设置成异常中断向量地址，从而跳转到相应的异常中断处执行程序。

ARM 处理器对异常中断返回的过程是：首先将 LR 的值减去偏移量送到 PC 中，同时将 SPSR 值复制到 CPSR，另外需要注意的是：如果在进入异常时设置了中断禁止位，这里还要清除。

应用程序总是从复位开始的，因此复位异常程序无需返回，详见参考文献[29]。

3.3 存储管理单元 MMU

3.3.1 虚存的工作原理

MMU 的功能是，能使各个任务在自己的私有存储空间中运行。在带 MMU 的操作系统下，运行的任务不需要知道其它任务的存储空间需求情况。

MMU 简化了任务编程，因为它允许使用虚拟存储器。MMU 作为转换器，将程序和数据的虚拟地址转换为实际的物理地址。这就允许多个程序使用相同的虚拟地址，而各自存储在物理存储器的不同位置。虚拟地址是由编译器和连接器在定位程序时分配；物理地址用来访问实际的主存硬件模块(物理上程序放在这里)[30]。

为了使虚拟存储器能够工作，MMU 采用了地址重定位，就是在 CPU 访问存储空间之前，转换处理器核输出的存储地址。当处理器核输出一个虚拟地址时，MMU 就将虚拟地址的高位用重定位寄存器的值替换它，这就生成一个物理地址，如图 3-1 所示。

虚拟地址的低位部分是一个偏移量，它转换成物理存储器的一个特定地址。图 3-1 表示的是将虚拟存储器 0x04000000 为起始的任务地址，通过重定位寄存器转换成以 0x08000000 开始的物理地址。

一个重定位寄存器只能转换一块存储空间，这块存储空间的大小由虚拟地址

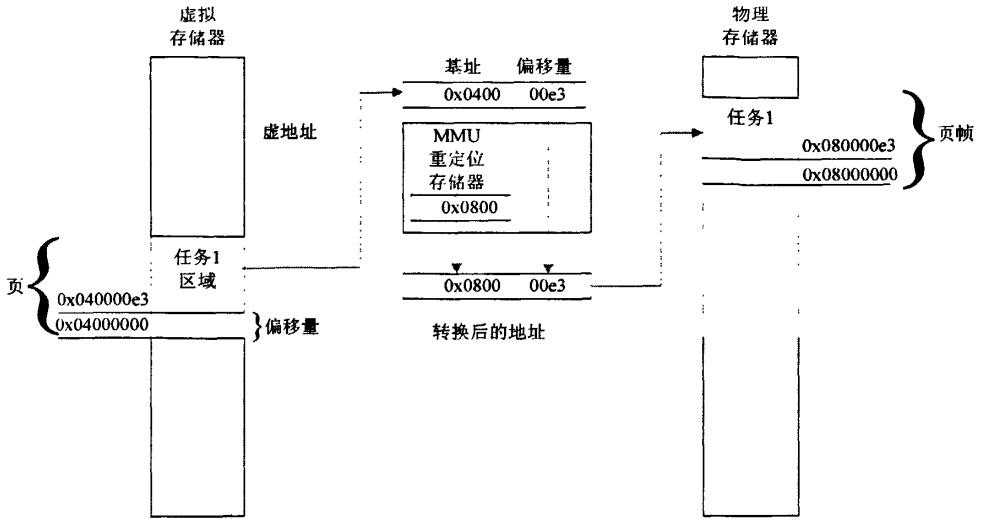

图 3-1 虚拟存储器到物理存储器的映射

的偏移量决定。这样的一块虚拟存储空间称为一页(page)，而转换所对应的那块物理存储空间称为一个页帧(page frame)。ARM 的 MMU 硬件有多个重定位寄存器，以便将多个页转换成多个页帧[30]。

3.3.2 页表和转换旁路缓冲器 TLB

实际上，MMU 中临时存放的是由 64 个重定位寄存器组成的全相联缓存(cache)，这个缓存称为转换旁路缓冲器(TLB)。除了重定位寄存器外，MMU 还使用页表来存放虚拟存储器映射的数据。页表中的每个页表项代表了将虚拟存储器的一个页转换到物理存储器的一个页帧所需的所有信息。

ARM 的 MMU 采用 2 级页表结构：一级页表(L1)和二级页表(L2)。一级页表只有一个 L1 主页表，这个 L1 主页表又称段页表。L1 主页表包含 2 种页表项：保存二级页表起始地址的页表项和保存转换 1MB 页的普通页表项。当 L1 页表作为页目录时，其页表项代表的是 1MB 虚拟空间的 L2 粗页表或 L2 细页表的地址；当 L1 页表作为 1MB 页的普通页表时，其页表项包含的是物理存储器中 1MB 页帧的首地址。

一个 L2 粗页表有 256 个页表项，占用 1KB 的主存空间，每个页表项将一个 4KB 的虚拟存储块转换成一个 4KB 的物理存储块。粗页表支持 4KB 或 64KB 的页，页表项包含的是 4KB 或 64KB 的页帧首地址。一个 L2 细页表有 1024 个页表项，占用 4KB 的主存空间，每个页表项转换一个 1KB 的存储块。细页表支持 1KB，4KB 或 64KB 的虚存页，每个页表项包含 1KB，4KB 或 64KB 物理页帧的首地址。

图 3-2 是基于段的虚地址到物理地址的变换过程。

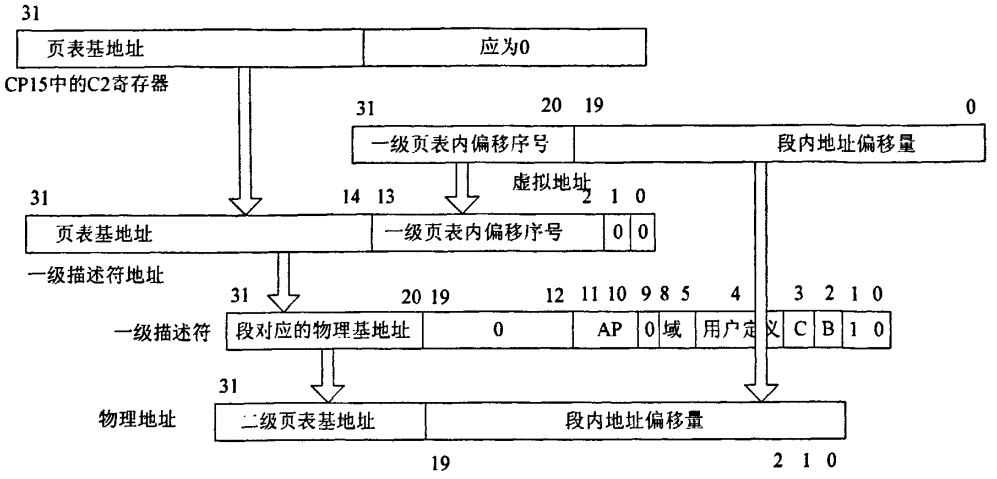

图 3-2 基于段的地址变换

在本系统中，用的是基于段的虚存重映射，将虚地址 0x00000000~0x00100000 的 1M 空间重映射到物理地址 0x30000000~0x30100000，而从 0x30000000 开始的虚地址重映射到自身(地址不变)。

图 3-3 和图 3-4 分别为执行内存管理 MMU 前后的存储情况：执行前从 0x00000000 开始没有中断向量表及后续代码，执行后，显然中断向量表及后续代码出现在 0x00000000 开始处，表明虚存实现成功，重映射机制已开启。

实现虚存的代码如下：

```
void MMU_Init(void)
{
int i, j;
MMU_DisableDCache( );
MMU_DisableICache( );
 for(i=0; i<64; i++)
    for(j=0; j<8; j++)
        MMU_CleanInvalidateDCacheIndex((i<<26)|(j<<5));
        MMU_InvalidateICache();
MMU_DisableMMU( );
MMU_InvalidateTLB( );
MMU_SetMTT(0x00000000, 0x00100000, (int)__ENTRY,RW_CB);
MMU_SetMTT(0x00200000, 0x2ff00000, 0x00200000, RW_FAULT);
MMU_SetMTT(0x30000000, 0x31f00000, 0x30000000, RW_CB);
MMU_SetMTT(0x32000000, 0x33f00000, 0x32000000, RW_CB);
MMU_SetMTT(0x34000000, 0x3ff00000, 0x34000000, RW_FAULT);
MMU_SetMTT(0x40000000, 0x47f00000, 0x40000000, RW_NCNB);
```

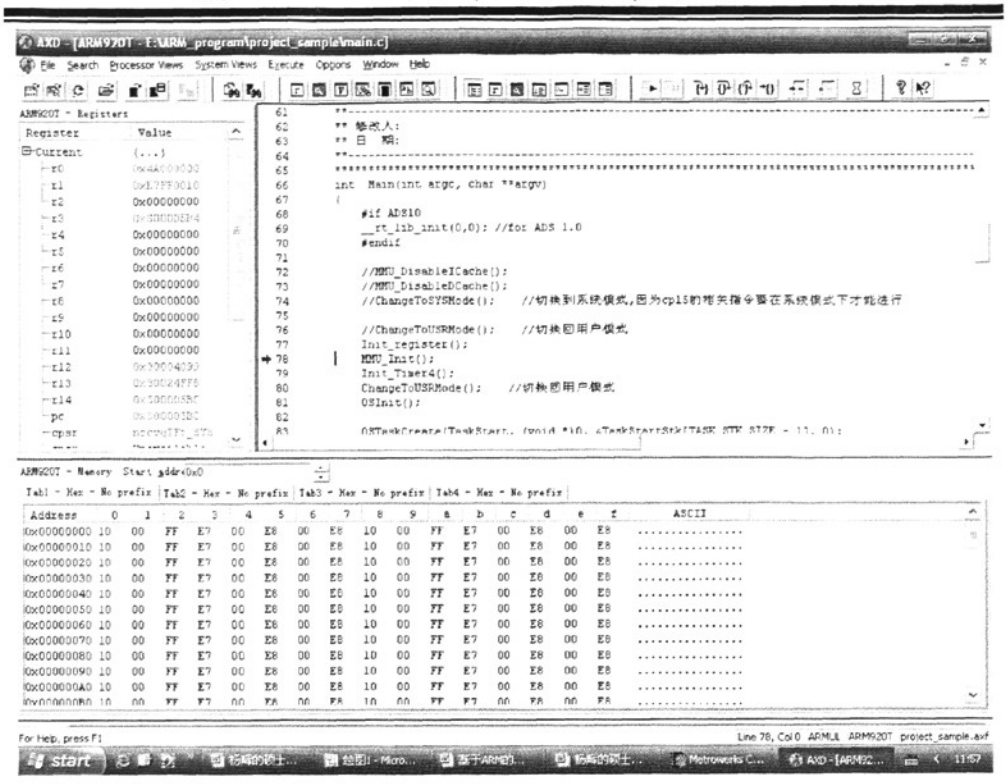

图 3-3 虚存实现前存储器图

图 3-4 虚存实现后存储器图

```
    MMU_SetMTT(0x48000000, 0x5af00000, 0x48000000, RW_NCNB);
    MMU_SetMTT(0x5b000000, 0x5b000000, 0x5b000000, RW_NCNB);
    MMU_SetMTT(0x5b100000, 0xfff00000, 0x5b100000, RW_FAULT);
    MMU_SetTTBase((int)TABLE);
    MMU_SetDomain(0x55555550|DOMAIN1_ATTR|DOMAIN0_ATTR);
    MMU_SetProcessId(0x0);  //设置 ID 号
    MMU_EnableAlignFault( );  //使能地址对齐检查功能
    MMU_EnableMMU( );
    MMU_EnableICache( );
    MMU_EnableDCache( ); // DCache should be turned on after MMU is turned on.
}
void MMU_SetMTT(int vaddrStart, int vaddrEnd, int paddrStart, int attr)
{
    volatile U32 *pTT;
    volatile int i, nSec;
    pTT=(U32*)_MMUTT_STARTADDRESS+(vaddrStart>>20);
    nSec=(vaddrEnd>>20)-(vaddrStart>>20);
    for(i=0; i<=nSec; i++) *pTT++=attr |(((paddrStart>>20)+i)<<20);
}
```

其中，函数 MMU_SetMTT(int vaddrStart, int vaddrEnd, int paddrStart, int attr)是生成基于段的转换页表。vaddrStart 为虚存的开始地址，vaddrEnd 为虚存的结束地址，paddrStart 为对应的开始物理地址，attr 是基于这个段转换的所有存储单元的属性，包括读写、缓存、写缓冲等。_MMUTT_STARTADDRESS 为转换基于段的转换页表的首地址。每 1M 字节的虚地址到实地址的转换属性只需 4 字节，所以对于 32 位 ARM 处理器来说，基于段的转换最多只需要 4K 字，即 16K 字节容量。

3.3.3 高速缓存 Cache 和写缓冲器

高速缓存是全部用硬件来实现的，Cache 与主存之间以块(cache line)为单位进行数据交换。当 CPU 读取数据或指令时，它同时会将数据或指令保存到缓存块中。这样，当 CPU 再次读取相同的数据或指令时，不会从主存中读取，而是从缓存块中得到数据。因为缓存的速度远远大于主存的速度，所以开通缓存后，系统的整体性能提高很多。写缓冲区由一些高速的存储器构成，它主要用来优化向主存储器中的写入操作。当 CPU 向主存中写入时，它先将数据写入到写缓冲区中，写缓冲区会在适当的时候将数据写入到主存。

在 Cache 中，地址映像是指把主存地址空间映像到 Cache 地址空间，从而建

立主存地址到 Cache 地址之间的对应关系。常用的地址映像方式和变换方式包括全相联映像和变换方式、直接映像和变换方式以及组相联映像和变换方式三种[31]。

在 ARM 中，缓存的替换算法有两种：伪随机替换算法和轮转法[32]。伪随机替换算法通过一个伪随机数发生器产生一个伪随机数，当替换旧块时，将编号为该伪随机数的缓存块替换掉，这种算法简单，易于实现，但效果较差。轮转法通过一个计数器，依次选择将要被替换出去的缓存块，这种算法容易预测最坏的情况但可能造成缓存平均性能急剧地变化。

3.3.4 协处理器 CP15 和 MMU 配置

控制 MMU 操作的控制寄存器值 CP15 :C1 的位描述如下所示：

0 位：M 表示，使能 MMU 功能，0 表示禁用，1 表示使能。

2 位：C 表示，数据缓存功能，0 表示禁用，1 表示使能。

3 位：W 表示，写缓冲器功能，0 表示禁用，1 表示使能。

8 位：S 表示，系统功能，0 表示禁用，1 表示使能。

9 位：R 表示，ROM 功能，0 表示禁用，1 表示使能。

12 位：I 表示，指令缓存功能，0 表示禁用，1 表示使能。

13 位：V 表示，设置高地址向量表功能，0 表示向量表在 0x00000000，1 表示向量表在 0xffff0000。

为了设置相关位，必须对协处理器 CP15 的 C1 进行操作，包括读取和置位。下面是设置控制寄存器位的 C 程序，用函数实现，第 1 个参数 value 包含要改变的控制值，第 2 个参数 mask 选择要改变的位。

```
void controlSet(unsigned int value, unsigned int mask)
{
  unsigned int clformat;
  __asm{ MRC   p15, 0, clformat, c1, c0, 0} //读取控制寄存器
  clformat&=~mask;  //清除要改变的位
  clformat|=value;     //设置要改变的位
  __asm{ MCR   p15, 0, clformat, c1, c0, 0} //写控制寄存器
}
```

3.4 小结

本章主要介绍了 ARM 的异常模式、寄存器组、异常处理和 MMU(存储管理单元)功能等。在本同步交流采样系统中，目标板的存储空间很特殊(从 0x00000000 地址开始是 ROM 区)，为了使 H-JTAG 能在线调试和提高采样系统的取指令和读数据速度，需把代码从 0x00000000~0x001fffff 拷贝到 0x30000000~0x301fffff 处，然后执行 MMU 功能，实现虚地址重映射。这样一切代码和数据都是在 SDRAM 里存取，系统的整体性能得到了很大改善。

第4章 μC/OS-II在目标板上的移植

4.1 引言

移植，就是编写与处理器相关的代码，让一个实时操作系统内核能在某种CPU上正常运行。在移植过程中，大部分μC/OS-II代码都用标准C编写，但与处理器相关的代码仍然要用汇编语言编写，因为对CPU寄存器的操作只能用汇编实现，当然用汇编语言编写移植代码是移植的重点和难点。μC/OS-II 内核的移植工作，主要包括编译的设置、数据类型(比如整形、字符型等)的定义、宏定义的设置，以及相关内核调度函数的编写等，这些编程都是用汇编实现。另外，通过一些条件编译的宏定义，对μC/OS-II内核进行相关裁剪，以减少移植μC/OS-II内核所需要的存储容量。本章重点编写了启动代码、开关中断、任务切换和首次任务切换等函数，均用汇编语言编写。

4.2 编写S3C2440的启动代码

移植 μC/OS-II 和编写驱动程序均在 ADS1.2 平台下进行。ADS 是 ARM Developer Suite 的简称，是ARM公司推出的关于ARM处理器的编译、链接和调试集成开发环境系统。它与一般传统调试方法的调试系统不同。ADS 集成开发环境把编译、链接和仿真调试分别集成在两个环境中：把编译和链接集成在一个环境中，全名叫做 CodeWarrior for ARM Developer Suite，一般简称 CodeWarrior for ADS；而把仿真和调试环境集成在 ARM eXtended Debugger 下，简称 AXD。图 4-1 和图 4-2 是 CodeWarrior for ADS 和 AXD 的环境窗口，具体使用方法详见参考文献[33]。

4.2.1 中断向量表

中断向量表，实际上是由8条(32字节大小)程序跳转指令组成。如下所示：

```
b   ResetHandler
b   HandlerUndef    ;handler for Undefined mode
b   HandlerSWI      ;handler for SWI interrupt
b   HandlerPabort   ;handler for PAbort
b   HandlerDabort   ;handler for DAbort
b   .               ;reserved
```

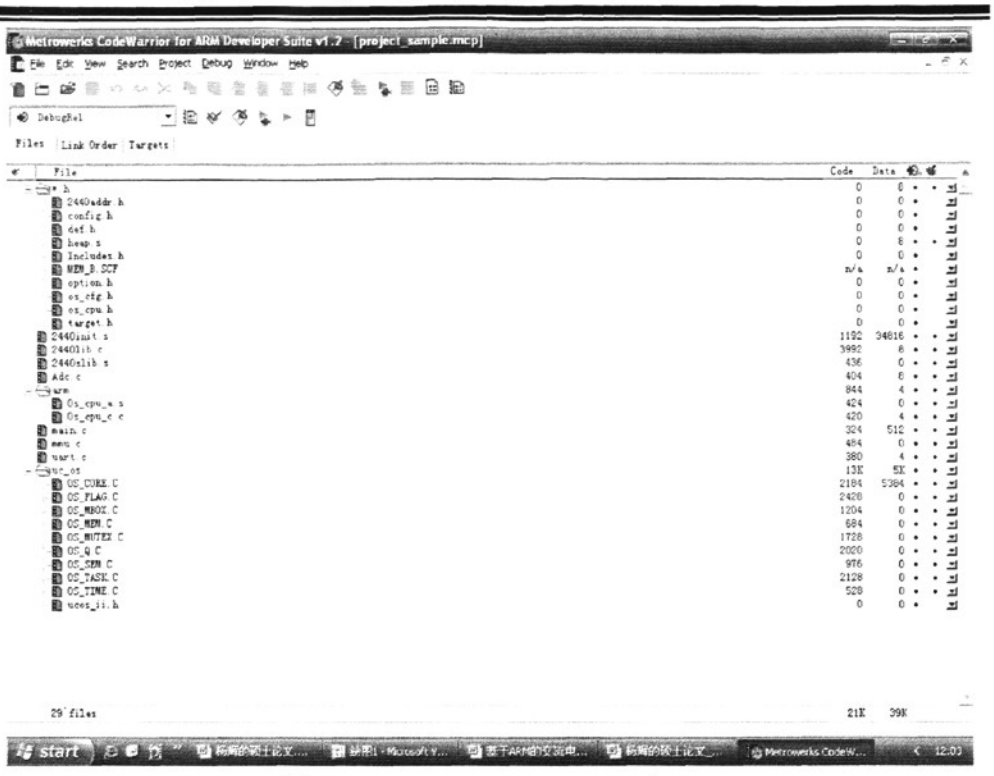

图 4-1　CodeWarrior for ADS 窗口

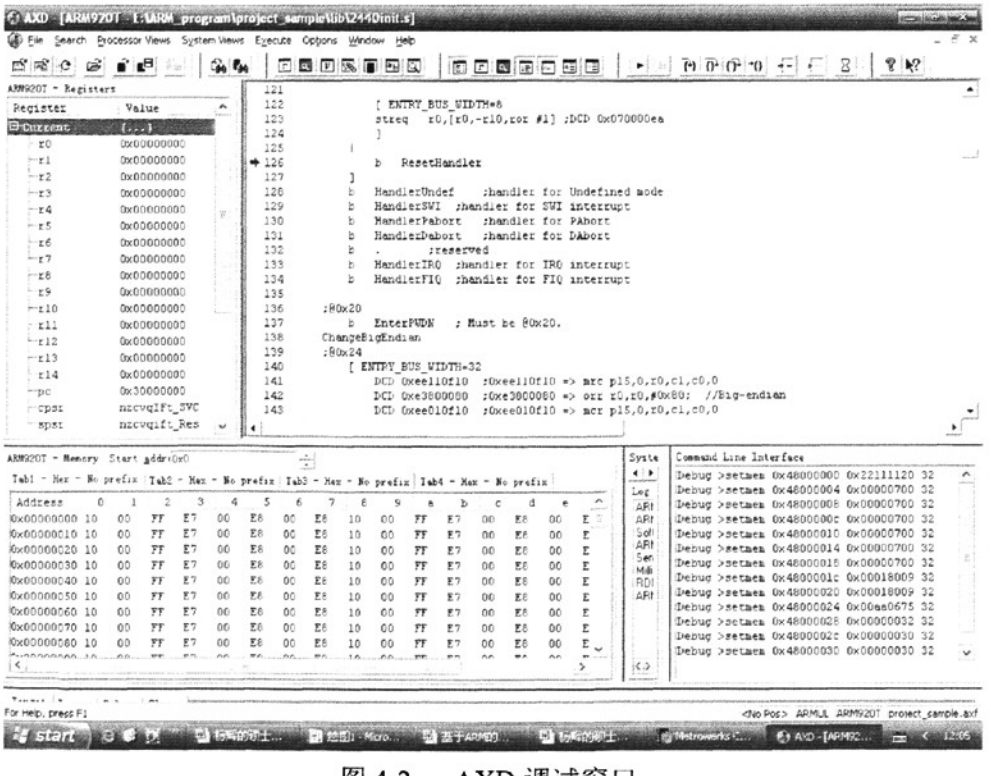

图 4-2　AXD 调试窗口

```
b  HandlerIRQ      ;handler for IRQ interrupt
b  HandlerFIQ      ;handler for FIQ interrup
```

中断向量表通常放在物理地址 0x00000000 开始处(实际上如果用到了重映射或虚存，中断向量表往往放在 SDRAM 或内部 SRAM 中，同时被重映射到 0x00000000，这样当发生异常时可以显著提高程序的执行速度)。当发生异常时，程序跳转到异常向量表的相应位置，因为要执行的是 b 跳转指令，所以 CPU 就跳到各个异常程序处。这里重点讲述当发生 IRQ 异常时，程序的执行过程，因为 IRQ 中断源较多，为了执行各种中断源引发的子程序，必须要用到二级中断向量表。

当有 IRQ 异常发生时，程序跳转到中断向量表的地址 0x00000018 处，执行跳转指令 b HandlerIRQ，在启动程序里，已经有宏语句 HandlerIRQ HANDLER HandleIRQ 和宏定义等，所以程序就转到 isrIRQ 标号处执行，因为二级中断向量表的存在，根据寄存器 INTOFFSET 的值就可跳转到二级中断向量表装的中断子程序标号处，注意：二级中断向量表存放的是各种 IRQ 子程序的地址，所以从二级中断向量表提供了中断子程序的入口点。

4.2.2 目标板初始化代码

所谓目标板初始化代码就是根据系统设计的硬件资源，对其进行初始化，主要是指 SDRAM、Flash(包括 Norflash 和 Nandflash 等)、时钟设置、中断、看门狗等资源的配置，以便系统能正常启动。在本系统中，微处理器是三星公司的 ARM9 内核 S3C2440，SDRAM 是 Hynix 公司的 HY57V561620(64M 大小，地址范围：0x30000000~0x33ffffff)，Norflash 是 AMD 公司的 Am29LV160DB(2M 大小，系统板从 Norflash 启动时，地址范围：0x00000000~0x001fffff)。目标板初始化程序如下：

```
ldr r0, =WTCON          ;watch dog disable
ldr r1, =0x0
str r1, [r0]
ldr r0, =INTMSK
ldr r1, =0xffffffff     ;timer interrupt not masked
str r1, [r0]
ldr r0, =INTSUBMSK      ;all sub interrupt disable
mov r1, #0x000000ff
orr r1, r1, #0x00007f00
str r1, [r0]
ldr r0, =CLKDIVN
```

```
ldr r1, =CLKDIV_VAL        ; 为 5=1:4:8
str r1, [r0]
;Configure UPLL
ldr r0, =UPLLCON
ldr r1, =((U_MDIV<<12)+(U_PDIV<<4)+U_SDIV)
str r1, [r0]
nop      ; Caution: After UPLL setting, at least 7-clocks delay must be inserted for
nop      ; setting hardware be completed.
nop
nop
nop
nop
nop
;Configure MPLL
ldr r0, =MPLLCON
ldr r1, =((M_MDIV<<12)+(M_PDIV<<4)+M_SDIV)
str r1, [r0]
 ldrr0, =SMRDATA
 ldrr1, =BWSCON          ; BWSCON Address
 add     r2, r0, #52     ; End address of SMRDATA
0
 ldrr3, [r0], #4
 strr3, [r1], #4
 cmp   r2, r0
 bne    %B0
```

4.2.3 初始化堆和栈

对于 ARM，堆是向上生长的，栈是向下生长的。局部变量占用栈(stack)空间(但其初始化值为数据，占用 RO 空间)，程序中动态申请的如 malloc( )和 new( )函数申请的内存占用堆(heap)空间。在转入 C 应用程序前，必须要为 C 程序准备堆和栈空间。所以，必须根据具体目标平台的存储器资源，对堆栈的初始化函数__user_initial_stackheap( )进行移植，这主要工作是正确设置堆(heap)和栈(stack)的地址。可以使用 C 或 ARM 汇编语言来编写，并至少返回堆基址(保存在 R0 中)，栈基址(保存在 R1)是可选的。以下是一个简单的用汇编语言编写的堆栈的初始化函数__user_initial_stackheap( )，代码如下所示。r0，r1，r2，r3 分别被赋值堆的基地址 heap base，栈的基地址 stack base，堆的顶地址 heap limit，栈的顶地址 stack

limit。注意，如果在工程中没有自定义这个函数，那么在缺省情况下，编译器/链接器会把|Image$$ZI$$Limit|作为堆(heap)的基址(即把 heap 和 stack 区放置在 ZI 区域的上方，这也被认为是标准的实现)。但是，如果使用 scatter 文件实现分散加载机制，链接器并不生成符号|Image$$ZI$$Limit|，这时就必须自己重新实现__user_initial_stackheap( )函数并且设置好堆基址和栈顶，否则链接时会报错。关于重定义__user_initial_stackheap( )函数时有几点需要注意：一是不要使用超过 96 字节的 stack，二是不要影响到 R12(IP，用作进程间调用的暂存寄存器)，三是按规则返回参数值(R0：heap base；R1：stack base；R2：heap limit；R3：stack limit)，四是让堆区保持 8 字节对齐。详见参考文献[34]。

```
__user_initial_stackheap
  LDR    r0, =bottom_of_heap
  LDR    r1, FIQStack
  LDR    r2, =UsrStackSpace
  LDR    r3, =bottom_of_Stacks
```

另外，在 ARM 中，7 种异常模式要分别分配堆栈(栈)，其程序如下：

```
InitStacks
  mrs  r0, cpsr
  bic  r0, r0, #MODEMASK
  orr  r1, r0, #UNDEFMODE|NOINT
  msr  cpsr_cxsf, r1      ;UndefMode
  ldr  sp, UndefStack
  orr  r1, r0, #ABORTMODE|NOINT
  msr  cpsr_cxsf, r1      ;AbortMode
  ldr  sp, AbortStack
  orr  r1, r0, #IRQMODE|NOINT
  msr  cpsr_cxsf, r1      ;IRQMode
  ldr  sp, IRQStack
  orr  r1, r0, #FIQMODE|NOINT
  msr  cpsr_cxsf, r1      ;FIQMode
  ldr  sp, FIQStack
  orr  r1, r0, #SVCMODE|NOINT
  msr  cpsr_cxsf, r1      ;SVCMode
  ldr  sp, SVCStack
  mov  r2, lr
  orr  r1, r0, #MODEMASK|NOINT
  msr  cpsr_cxsf, r1
```

```
ldr    sp, UserStack
```

4.3 移植 μC/OS-II

4.3.1 不依赖编译器的数据结构

微处理器不同，字长也不同，为了确保可移植性，代码没有使用与编译器相关的数据类型，所以必须参考所使用的 C 编译器文档，修改 OS_CPU.H 中所使用的数据类型的定义。本人在开发中使用的是 ADS 编译器。另外 ARM 的堆栈入口宽度为 32 位。定义的数据类型如下：

```
typedef  unsigned  char   BOOLEAN;
typedef  unsigned  char   INT8U;
typedef  signed    char   INT8S;
typedef  unsigned  short  INT16U;
typedef  signed    short  INT16S;
typedef  unsigned  int    INT32U;
typedef  signed    int    INT32S;
typedef  float            FP32;
typedef  double           FP64;
typedef  INT32U           OS_STK;
```

4.3.2 函数 OS_ENTER_CRITICAL 和 OS_EXIT_CRITICAL 的编写

除了不依赖编译器的数据类型外，OS_CPU.H 文件中还包括一些宏定义，这些宏定义与硬件系统结构有关，如开关中断和进行任务切换的宏定义等。下面是开关中断的宏定义，并都用软中断实现：

```
__swi(0x02) void OS_ENTER_CRITICAL(void);    //关中断
__swi(0x03) void OS_EXIT_CRITICAL(void);     //开中断
```

与其它的实时内核一样，μC/OS-II 内核需要访问代码的临界段时，首先必须禁止中断才能访问，在访问完毕后中断重新打开。μC/OS-II 内核采用了两个宏定义来禁止和允许中断：OS_ENTER_CRITICAL( )和 OS_EXIT_CRITICAL( )。为了提高代码的效率，我们使用汇编语言来实现禁止中断和允许中断这两个宏定义。核心代码如下：

```
void SWI_Exception(int SWI_Num, int *Regs)
{
  OS_TCB    *ptcb;
```

```
switch(SWI_Num)
 {
   case 0x02: //关中断函数 OS_ENTER_CRITICAL()，参考 os_cpu.h 文件
     __asm
      {
        MRS       R0, SPSR
        ORR       R0, R0, #NoInt
        MSR       SPSR_c, R0
      }
      OsEnterSum++;   break;
    case  0x03:  //开中断函数 OS_EXIT_CRITICAL()，参考 os_cpu.h 文件
      if (--OsEnterSum = = 0)
       {
          __asm
           {
             MRS       R0, SPSR
             BIC       R0, R0, #NoInt
             MSR       SPSR_c, R0
           }
       }    break;
      ......
}
```

4.3.3 函数 OS_TASK_SW 的编写

在本系统移植中，OS_TASK_SW( )以软中断的宏定义实现，功能是实现任务级的上下文切换，宏定义如下：

```
__swi(0x00)   void   OS_TASK_SW(void);    //任务级任务切换函数
```

任务级任务切换和中断级的任务切换相比较，任务级的任务切换都是在非异常模式下完成的。OS_TASK_SW( )函数的功能：使寄存器的值入栈以保存任务现场，将当前堆栈指针 SP 存入任务控制块 TCB 中，然后恢复最高优先级任务的寄存器现场(恢复现场)，执行软中断返回指令，这样任务级的任务切换就完成了。任务级的任务切换函数的核心实现代码如下：

```
OSIntCtxSw   ;下面为保存任务环境
  LDR       R2, [SP, #28]                ;获取 PC
  LDR       R12, [SP, #24]               ;获取 R12
```

```
    MRS     R0, CPSR
    LDR     R1, =OSTCBCur
    LDR     R1, [R1]
    LDR     R1, [R1]
    ADD     R1, R1, #68
    MSR     CPSR_c, #(NoInt | SYS32Mode)
    MOV     R4, SP
    MOV     SP, R1
    MOV     R1, LR
    STMFD   SP!, {R1-R2}                    ;保存 LR,PC
    STMFD   SP!, {R6-R12}                   ;保存 R4-R12
    MSR     CPSR_c, R0
    LDMFD   SP!, {R6-R11}                   ;获取 R0-R3
    ADD     SP, SP, #8                      ;出栈 R12,PC
    MSR     CPSR_c, #(NoInt | SYS32Mode)
    STMFD   SP!, {R6-R11}                   ;保存 R0-R3
    LDR     R1, =OsEnterSum                 ;获取 OsEnterSum
    LDR     R2, [R1]
    STMFD   SP!, {R2, R3}       ;保存 CPSR,OsEnterSum
    BL      OSTaskSwHook        ;调用钩子函数
    LDR     R5, =OSPrioCur      ;OSPrioCur <= OSPrioHighRdy
    LDR     R6, =OSPrioHighRdy
    LDRB    R7, [R6]
    STRB    R7, [R5]
    LDR     R6, =OSTCBHighRdy   ;OSTCBCur <= OSTCBHighRdy
    LDR     R6, [R6]
    LDR     R5, =OSTCBCur
    STR     R6, [R5]
OSIntCtxSw_1    ;获取新任务堆栈指针
    LDR     R5, [R6]
    ADD     SP, R5, #68   ;17 寄存器 CPSR,OsEnterSum,R0-R12,LR,SP
    LDR     LR, [SP, #-8]
    MOV     SP, R4
    MSR     CPSR_c, #(NoInt | SVC32Mode)    ;进入管理模式
    MOV     SP, R5                          ;设置堆栈指针
    LDMFD SP!, {R5, R6}             ;CPSR, OsEnterSum
```

```
;恢复新任务的 OsEnterSum
LDR       R3, =OsEnterSum
STR       R5, [R3]
MSR       SPSR_cxsf, R6          ;恢复 CPSR
LDMFD     SP!, {R0-R12, LR, PC }^        ;运行新任务
```

4.3.4 函数 OS_STartHighRdy 的编写

OS_STartHighRdy( )函数在系统启动时只执行一次，它在启动函数 OSStart( ) 中调用。其主要功能是：将就绪表中最高优先级任务的栈指针加载到堆栈指针 SP 中，中断返回后强制执行最高优先级的就绪任务，并设置系统运行标志 OSRunning 为 TRUE。注意的是：该函数只在多任务启动时调用一次，以后的多任务调度和任务切换就由 OS_TASK_SW( )和 OSIntCtxSw ( )两个函数来实现。

OS_STartHighRdy( )功能由软中断实现，主要代码如下：

```
__OSStartHighRdy
  MSR       CPSR_c, #(NoInt | SYS32Mode)
  ;告诉 μC/OS-II 自身已经运行
  LDR       R4, =OSRunning
  MOV       R5, #1
  STRB      R5, [R4]
  BL        OSTaskSwHook     ;调用钩子函数
  LDR       R6, =OSTCBHighRdy
  LDR       R6, [R6]
  MOV       R4, SP
  B         OSIntCtxSw_1       ;见 OS_TASK_SW( )中的标号 OSIntCtxSw_1
```

4.4 小结

本章主要用汇编语言编写了启动代码、开关中断、任务切换和首次任务切换等函数。这几个函数和启动代码的编写是移植 μC/OS-II 的难点和关键，其中开关中断、任务切换和首次任务切换函数均用软中断机制实现。

第5章 同步交流采样误差分析和补偿

5.1 引言

随着电力系统的迅速发展，电网容量的扩大使电网系统日趋复杂，如何实现优越的电网监控、调度至关重要，而在电力系统自动化中的关键是准确采集各种电力参数。采样分直流采样和交流采样。直流采样，就是对经过整流后的直流信号进行采样[35]，而交流采样是将二次测得的电压、电流经高精度CT(电流互感器)、PT(电压互感器)变成交流小信号，再由微机处理[36]。交流采样是对被测信号的瞬时值进行采样，因而与实际值比较接近，近年来，有针对交流采样系统植入实时性较好的μC/OS-II研究，并取得不少成果[37][38]。目前对电力参数如电流、电压和功率等的采样都是基于同步交流采样，然而同步交流采样很难实现理想的同步，都存在或多或少的同步误差。针对测量中出现的不同步问题，目前提出了两种解决方法。第一，在同步误差保持不变的条件下，通过对采样得到的数据进行处理或对测量结果进行修正来减小最终测量结果误差。目前这一方面的研究在国内外比较多，提出了基于各种平台或理论的算法，如准同步算法[39-41]，特殊窗法[42]等新算法。但这些算法都有较突出的缺陷，就是处理过程相当复杂，测量时间和数据处理时间花费较多。第二，通过减小同步误差来减小测量误差。目前这方面的研究成果主要有双速率采样法[43-46]、优化选择采样点数[47]两种方法。但双速率采样法、优化选择采样点数的应用范围都很有局限，双速率采样法需要与特定的测量算法相配合才能使用，而优化选择采样点数的同步精度不稳定，波动性较大。另外，上述所有研究成果都忽视了或没有考虑采样时间间隔不均匀的影响。

本章从理论上推导了交流采样(正余弦信号)的误差公式，得出误差c由初相位角α，采样周期ΔT，信号周期T所决定。初相位角α和信号周期T都可通过采样的点数和采样周期ΔT算出，然后就可利用理论误差对被测信号进行补偿。另外，ΔT也有一个优化算法，可以减小一个被测信号周期内的同步误差。

5.2 交流采样原理

对于周期电压信号，其电压有效值的理论公式为：

$$U=\sqrt{\frac{1}{T}\int_0^T u^2(t)dt} \tag{5-1}$$

现将式(5-1)离散化，设相邻两采样点的时间间隔为ΔT_n，采样点的瞬时值为$U(n)$，一个周期内的采样点数为N，则

$$U=\sqrt{\frac{1}{T}\sum_{n=1}^{N}u^2(n)\Delta T_n} \tag{5-2}$$

若相邻两次采样的时间间隔相等均为ΔT，且$N=\frac{T}{\Delta T}$，有：

$$U=\sqrt{\frac{1}{T}\sum_{n=1}^{N}u^2(n)\Delta T_n}=\sqrt{\frac{1}{N}\sum_{n=1}^{N}u^2(n)} \tag{5-3}$$

式(5-3)就是交流采样的电压有效值计算方法。

5.3 误差分析

设采样间隔为τ弧度，则$\tau=2\pi\frac{\Delta T}{T}$，另设一个周期内的采样点数为$N$，则一个周期内的周期误差可定义为：$\Delta\varphi=2\pi-N\tau$。如果信号周期是采样周期的整数倍，那么$\Delta\varphi=0$。但是软件同步采样总是不能实现理想的同步，总会存在或多或少的同步误差。下面讨论$\Delta\varphi\neq 0$时的有效值误差[48]。

现有被测电压信号$u(t)=U_m\sin(\omega t)$，则$U_m/\sqrt{2}$为有效值，其平方值为$U_m^2/2$，假设一个周期内以采样周期τ弧度采样N个点，并设第一个点的位置在α弧度处，则第i个点的相位为：

$$\omega t_i=\alpha+(i-1)\tau \quad (i=1,\ 2,\ \cdots\cdots N)$$

各点的瞬时采样值为：$u_i=U_m\sin[\alpha+(i-1)\tau]$，由$N$个采样值求得有效值$U'$的平方为：

$$\begin{aligned}U'^2&=\frac{1}{N}\sum_{i=1}^{N}U_m^2\sin^2[\alpha+(i-1)\tau]=\frac{U_m^2}{N}\sum_{i=1}^{N}\sin^2[\alpha+(i-1)\tau]\\&=\frac{U_m^2}{N}\sum_{i=1}^{N}\frac{1-\cos 2[\alpha+(i-1)\tau]}{2}=\frac{U_m^2}{2}\left\{1-\frac{1}{N}\sum_{i=1}^{N}\mathrm{Re}\left(e^{j(2[\alpha+(i-1)\tau])}\right)\right\}\end{aligned} \tag{5-4}$$

其中，Re{ }为取复数的实部，$\sum_{i=1}^{N}\mathrm{Re}\left(e^{j(2[\alpha+(i-1)\tau])}\right)$为一等比数列，上述等比级数求和，则有：

$$U'^2=\frac{U_m^2}{2}\left\{1-\frac{1}{N}\mathrm{Re}\left(\sum_{i=1}^{N}e^{j(2[\alpha+(i-1)\tau])}\right)\right\}=\frac{U_m^2}{2}\left\{1-\frac{1}{N}\mathrm{Re}\left(\frac{e^{j2\alpha}[1-e^{j2\pi N}]}{1-e^{j2\tau}}\right)\right\}$$

$$=\frac{U_m^{\ 2}}{2}\left\{1-\frac{1}{N}\mathrm{Re}\left(\frac{e^{j(2\alpha-\tau+N\tau)}\dfrac{e^{-jN\tau}-e^{jN\tau}}{2}}{\dfrac{e^{-j\tau}-e^{j\tau}}{2}}\right)\right\}$$

$$=\frac{U_m^{\ 2}}{2}\left(1+\frac{\sin(2\pi-N\tau)\cos[2\alpha-\tau-(2\pi-N\tau)]}{N\sin\tau}\right)$$

$$=\frac{U_m^{\ 2}}{2}\left[1+\frac{\sin(\Delta\varphi)\cos(2\alpha-\tau-\Delta\varphi)}{N\sin\tau}\right]=\frac{U_m^{\ 2}}{2}(1+c) \tag{5-5}$$

其中，c为电压有效值平方的相对误差，$c=\dfrac{\sin(\Delta\varphi)\cos(2\alpha-\tau-\Delta\varphi)}{N\sin\tau}$，$\tau$的范围为$0<\tau<2\pi$且$\tau\neq\pi$。又$\Delta\varphi=2\pi-N\tau$，$\tau=2\pi\dfrac{\Delta T}{T}$，代入得：

$$c=\frac{\sin\left(2\pi-N2\pi\dfrac{\Delta T}{T}\right)\cos\left[2\alpha-2\pi\dfrac{\Delta T}{T}-\left(2\pi-N2\pi\dfrac{\Delta T}{T}\right)\right]}{N\sin\left(2\pi\dfrac{\Delta T}{T}\right)}$$

$$=\frac{\sin\left[2\pi\left(1-N\dfrac{\Delta T}{T}\right)\right]\cos\left[2\alpha+2\pi\dfrac{\Delta T}{T}(N-1)\right]}{N\sin\left(2\pi\dfrac{\Delta T}{T}\right)} \tag{5-6}$$

5.4 误差补偿

由5.3节的式(5-6)可以看出，误差c与初相位角α，采样点数N，采样周期ΔT，以及信号周期T有关。而$N=\mathrm{int}\left(\dfrac{T-t_\alpha}{\Delta T}\right)+1$，int( )为取整，所以实际上误差$c$理论上由初相位角$\alpha$，采样周期$\Delta T$，信号周期$T$所决定。

(1) 初相位角α和信号周期T的测定

这里用过零点的方法估算α、T，通过采样点的正负可判断一个周期内的初相位的位置，如图5-1所示。在首尾零点附近，根据三角形相似[49]，有：

$t_\alpha=\dfrac{u(1)}{u(1)-u'(N)}\Delta T$，$t_\beta=\dfrac{u'(1)}{u'(1)-u(N)}\Delta T$，信号周期$T=N\Delta T-t_\beta+t_\alpha$，求出$t_\alpha$和$t_\beta$，就可求得信号周期$T$。至于$\Delta T$可由微处理器的定时器设定。

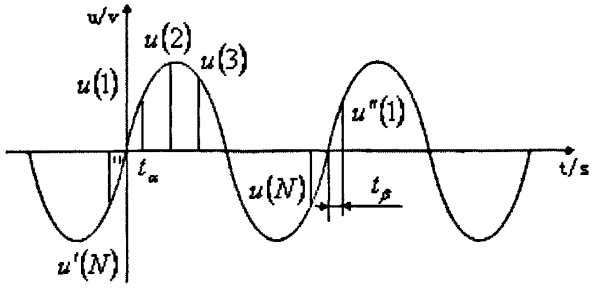

图 5-1 信号采样

(2) ΔT的优化

设定时器的计数周期为ϕ，则采样周期ΔT对应的定时器值为$\Delta T/\phi$，它一般不为整数，注意如果为整数则不存在ΔT的优化问题，在这里讨论的是不为整的情况，截取整数为Z，小数为L，即：

$$Z=\mathrm{int}\left(\frac{\Delta T}{\phi}\right),\quad L=\frac{\Delta T}{\phi}-\mathrm{int}\left(\frac{\Delta T}{\phi}\right)$$

显然$Z<\Delta T/\phi<Z+1$，所以在实际应用中总存在误差，这一误差累积则引起一个信号周期内的周期误差的改变。在理论上，周期误差$\Delta\varphi=2\pi-N\tau=2\pi-N(Z+L)\phi\frac{2\pi}{T}$。在一个被测信号周期内，如果取$Z$为定时器值时，则周期误差的实际值$\Delta\varphi'=2\pi-NZ\phi\frac{2\pi}{T}$，则正偏差$\Delta\varphi'-\Delta\varphi=NL\phi\frac{2\pi}{T}$。以$Z+1$为计数值时，则周期误差的实际值$\Delta\varphi'=2\pi-N(Z+1)\phi\frac{2\pi}{T}$，则负偏差为$N(1-L)\phi\frac{2\pi}{T}$。因此，要减小实际周期误差和理论周期误差的偏差，须对目前在采样过程中定时器计数值取常数的常规作法进行改进[50][51]。

设置一累加单元S_i对偏差进行累加，每隔一个采样周期累加一次，且S_i是作为第i次采样前，第$i-1$次采样后对定时器重新装值的依据，S_i用算式表示如下：

$$S_i=\begin{cases}0 & i=0\\ S_{i-1}+L & 1\le i\le N\end{cases}$$，直到S_i结果超过0.5，那么下次定时器的值就应在原值上加1，否则设置不变，另外S_i也要减1，直到一个信号周期结束。$S_i>0$表示采样时间超前，$S_i<0$表示采样时间滞后。显然，$-0.5<S_i\le 0.5$，注意左边不能取等号。因为S_i的取值，只有两种情况：$S_i=S_{i-1}+L$或$S_i=S_{i-1}+L-1$，前者的条件

是$S_{i-1}+L\le 0.5$，后者是$S_{i-1}+L>0.5$。对于$S_i=-0.5$，很显然，后一种情况不成立。所以只可能$S_i=S_{i-1}+L$且$S_{i-1}+L\le 0.5$，则$S_{i-1}=S_i-L=-0.5-L$，又因为$0<L<1$，所以$S_{i-1}<-0.5$，S_{i-1}又可分为两种情况，如此类推，则$S_{i-2}<-0.5$，$S_{i-3}<-0.5$，.....$S_1<-0.5$，而事实上，$S_1<-0.5$根本不可能。如此分析，$S_i=-0.5$也就不可能了。下面举个例子说明ΔT优化的用法，假设$L=0.68$，则$S_1=0.68>0.5$，根据四舍五入法，第一次采样时间的计数值改为$Z+1$较合适，另取$S_1=0.68-1=-0.32$，$S_2=0.68+S_1=0.68-0.32=0.36<0.5$，第二次采样时间的计数值为$Z$，$S_3=0.68+S_2=0.68+0.36=1.04>0.5$，第三次采样时间的计数值为$Z+1$，且$S_3=1.04-1=0.04$，再求$S_4$，如此类推，直到$S_N$为止。

第i次采样的理想时间与实际时间的差值，我们定义为同步误差$\Delta\varepsilon_i$，则$\Delta\varepsilon_i=S_i\phi$，当然，这里选择前一个信号周期的最后采样点为参考点，这样被测信号周期的第一个采样点的同步误差正好为$S_1\phi$。显然，对于第i次采样的同步误差总有$|S_i\phi|\le 0.5\phi$。当$i=N$时，$\Delta\varepsilon_N=S_N\phi$，事实上这就是整个被测信号的同步误差，且$|\Delta\varepsilon_N|\le 0.5\phi$。如果不采用上面$S_i$的计算方法，则$\Delta\varepsilon_i=iL\phi$或$i(1-L)\phi$，其中$\Delta\varepsilon_i=iL\phi$为正同步误差，$i(1-L)\phi$为负同步误差。当$i=N$时，正好是$NL\phi$或$N(1-L)\phi$，而$|NL\phi|$和$|N(1-L)\phi|$均远远大于$0.5\phi$，可见，$\Delta T$优化算法显著减少了整个信号周期的同步误差。

ΔT优化后，采样周期就变得不是很均匀，也就是说每个ΔT间可能有一个ϕ的差距，但是采样的理想时间与实际时间就更接近了，特别在ϕ很小就几乎是同一点。在ΔT优化条件下：

$\Delta\varphi'=2\pi-\sum_{i=1}^{N}\left\{[Z+L-(S_i-S_{i-1})]\phi\frac{2\pi}{T}\right\}=2\pi-N(Z+L)\phi\frac{2\pi}{T}+S_N\phi\frac{2\pi}{T}$，而理论周期误差$\Delta\varphi=2\pi-N\tau=2\pi-N(Z+L)\phi\frac{2\pi}{T}$，所以实际周期误差和理论周期误差的偏差$\Delta\varphi'-\Delta\varphi=S_N\phi\frac{2\pi}{T}$，且$|S_N\phi\frac{2\pi}{T}|<0.5\phi\frac{2\pi}{T}$，可见，大大减小了信号周期误差。

5.5 小结

本章从理论上推导出了交流采样(正余弦信号)的误差公式，得出误差c由初相位角α，采样周期ΔT，信号周期T所决定。另外尝试根据被测信号周期的首尾过零点三角形相似，求出误差参数T和初相位角α，然后利用理论误差对被测信号有

效值进行补偿。

此外，考虑到采样周期ΔT不均匀，经多次采样后会产生累积误差，本文也给出了采样周期ΔT的优化算法。

第6章 系统设计和结果分析

6.1 引言

本章是系统硬件和系统软件的设计，以及最后测得结果的数据分析。在系统硬件设计中，主要是电源电路、复位电路、异步串口电路和A/D采样电路的设计。在系统软件设计中，有采样驱动和串口驱动程序。最后，结合不同的采样间隔，运用补偿算法与否，得出不同的运算结果。这个结果值，制成表格形式，可直观地分析和对比，得出重要结论，为以后同步交流采样的实际工程给出补偿算法和实验依据。

6.2 系统硬件设计

Samsung公司推出的32位RISC处理器S3C2440，是一款高性价比和高性能的嵌入式微处理器。S3C2440提供了丰富的硬件资源，包括：16KB的数据Cache、16KB的指令Cache、内部SRAM、LCD控制器、3通道UART、外部存储器控制器、具有PWM功能的5通道定时器、8通道10位ADC、60个中断源、4通道DMA和PLL倍频器等，这样可降低系统成本和减少外围器件。图6-1是S3C2440的功能引脚。

本系统硬件设计包括电源电路、复位电路、UART异步串行接口电路、A/D采样电路。

6.2.1 电源电路设计

图6-2为电源电路图。在整个系统中，使用3.3V直流稳压电源给S3C2440供电，外接交流电经过变压器输出5V，而后通过LM1117-33三端稳压集成芯片，在靠近器件的电源引脚处以及各器件的电源和地之间加上了大量滤波电容，以滤除噪声和提高系统电源的质量。旁边的二极管是电源指示灯，开关按下接通外界电源时就发光。

6.2.2 复位电路设计

S3C2440和H-JTAG的复位连在一起，都为nRESET端，这也为了调试方便。当开关按下时，MAX811 输出低电平，系统复位。另外这种芯片也可上电复位，

且复位脉冲宽度大于 140ms，是一种可靠性很高的专用复位芯片。如图 6-3 为复位电路。

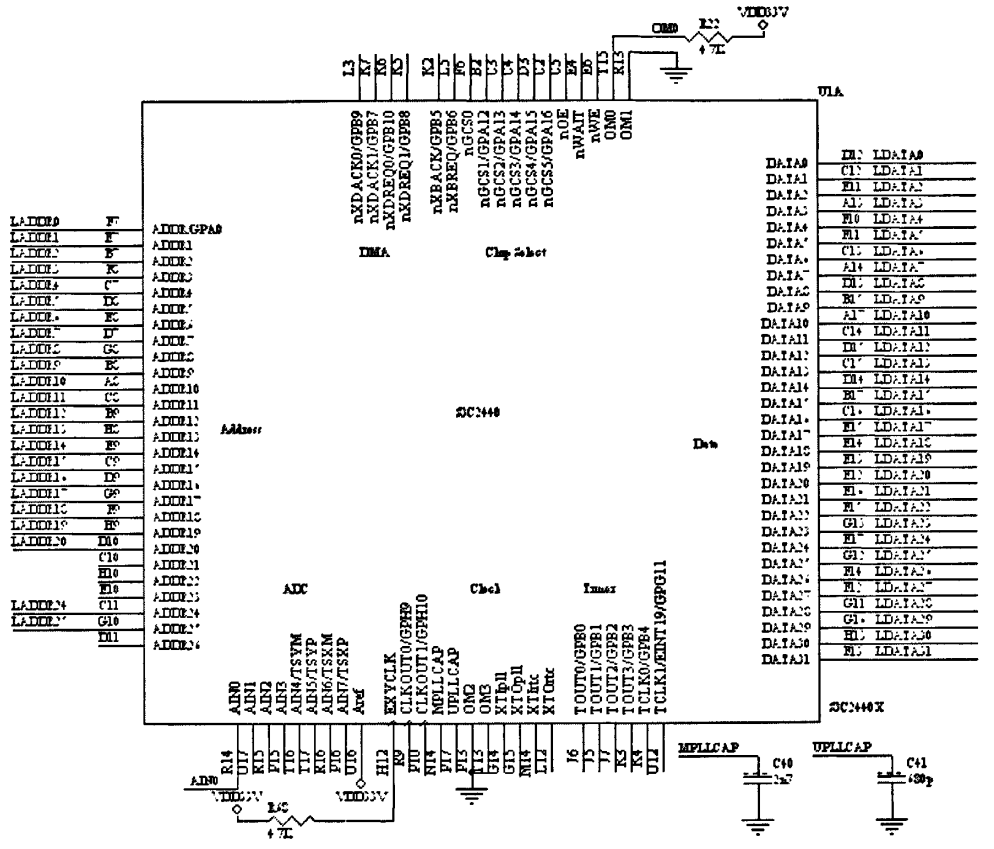

图 6-1　S3C2440 的功能引脚

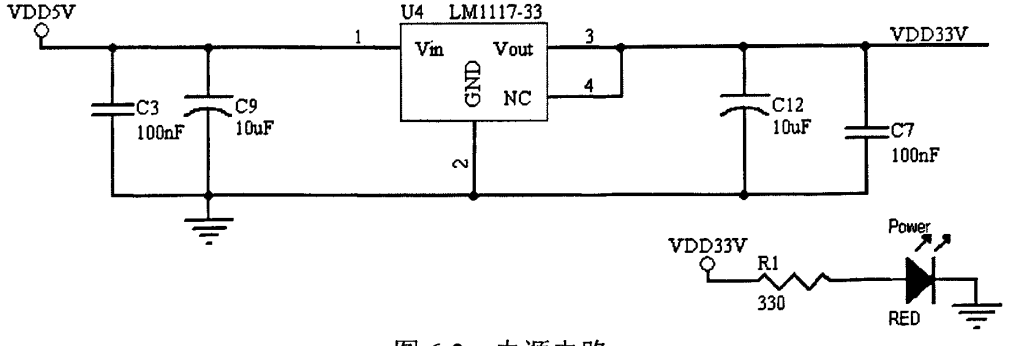

图 6-2　电源电路

6.2.3 异步串口电路设计

RS232 是嵌入式应用中较为常见的一种串行通讯协议。S3C2440 内部集成了三个独立的异步串行端口，并且都可以用于中断模式或 DMA(直接存储器存储)模

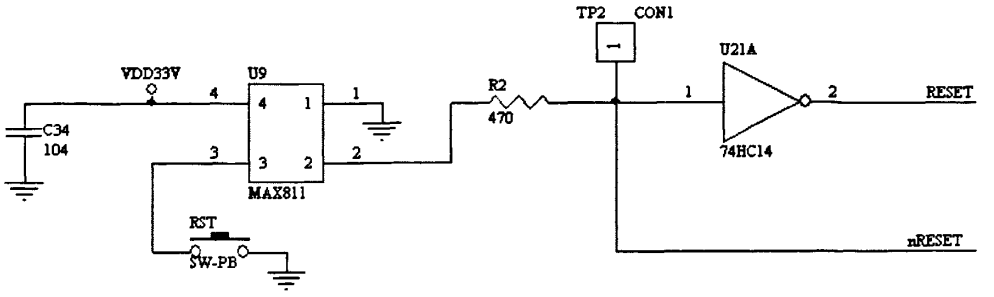

图 6-3　复位电路

式。串口最高传输速率为 115.2kbps(kilo bits per second)，每个通用异步收发串口通道包含 2 个 64 位先进先出 FIFO 队列。S3C2440 的串口还支持波特率可编程，可红外收/发，1~2 个停止位，5 位~8 位的数据宽度以及奇偶校验位。

RS232 串口通常采用 9 芯标准接口。RS-232C 标准所定义的高、低电平信号分别是：逻辑“1”指-5V~ -15V 电平，而逻辑“0”指+5V~+15V 电平。S3C2440 的 LVCMOS 电路所定义的高、低电平信号分别是：逻辑“1”指 2V~3.3V 电平，而逻辑“0”指 0V~0.4V 电平。显然，RS-232C 和 LVCMOS 的 S3C2440 间要顺利进行通信，电平转换必不可少。常用的这两种电平转换芯片有 MAX232、MAX3232 等。本系统采用 MAX3232 芯片完成 UART 串行接口和 S3C2440 的电平转换。MAX3232 的应用电路如图 6-4 所示。

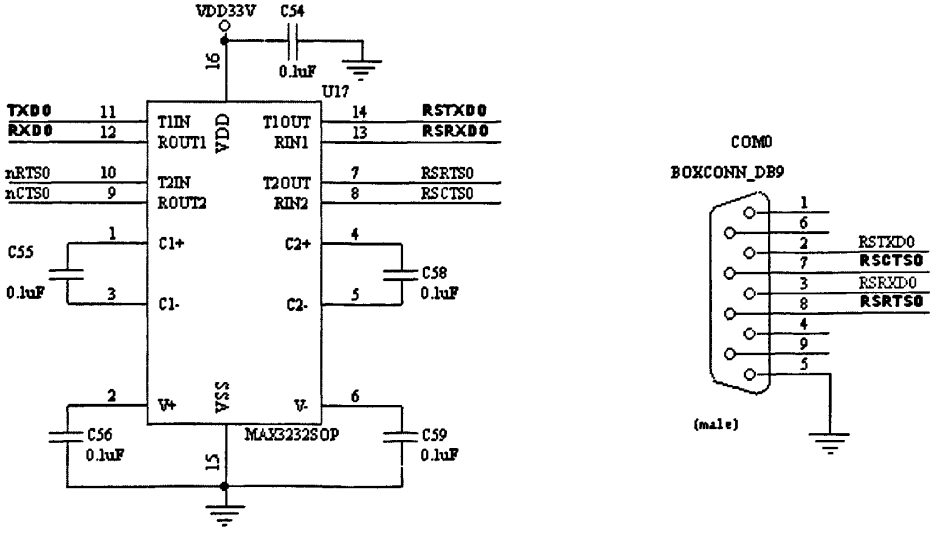

图 6-4　异步串行接口电路

6.2.4 采样电路设计

本系统采样电路如图 6-5 所示，Vi 与函数发生器的输出端相连，为交流采样的被测信号。DC 是外加直流电压，起抬高 Vi 作用，因为 S3C2440 的 A/D 采样范

围为 0~3.3V，Vo 输出接 S3C2440 的 AIN0 端，下面具体分析电路原理：

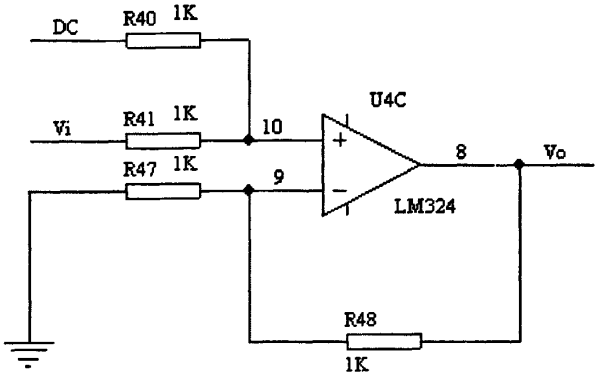

图 6-5 采样电路

LM324 为集成运算放大器，设 10 节点的电压为 V10，根据“虚断”和“虚短”概念，有：

$$V_i - V_{10} = \frac{V_i - V_{DC}}{R_{41} + R_{40}} \bullet R_{41} \tag{6-1}$$

$$\frac{V_{10} - 0}{R_{47}} = \frac{V_o - 0}{R_{47} + R_{48}} \tag{6-2}$$

将 $R_{40} = R_{41} = R_{47} = R_{48} = 1K\Omega$ 代入，解得：

$$V_o = V_i + V_{DC} \tag{6-3}$$

在本设计中，DC 端直流电压为 2V，所以最后输入到 S3C2440 的采样电压信号为：

$$V_o = V_i + 2 \tag{6-4}$$

6.3 系统软件设计

6.3.1 采样驱动设计

S3C2440 内部集成了 8 通道 A/D 转换器，转换精度为 10 位，最大采样频率非常高，达 500ksps(kilo samples per second)。本驱动程序的思想是:定时器定时时间到(即采样间隔时间)，就设置相关寄存器启动采样，在采样中断中，将采样数据存入缓冲区，并判断被测信号的一个周期的采样是否完成，如果完成了一个周期的采样就发送信号量通知任务已有采样结果，这时会发生中断级的任务切换，进入数据处理任务中。数据处理任务负责将采样的数据进行处理，将测得的最终结果存入新的缓冲区，以便串口发送。图 6-6 为采样的程序流程图。

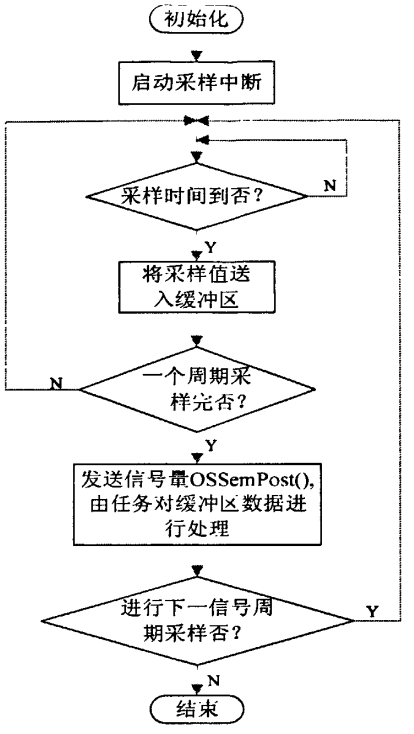

图 6-6　采样流程图

另外，图 6-7 为采样时间的优化流程图，其中 $Z=\text{int}\left(\frac{\Delta T}{\phi}\right)$，$L=\frac{\Delta T}{\phi}-\text{int}\left(\frac{\Delta T}{\phi}\right)$，$\phi$ 为定时器的计数周期，ΔT 为采样周期(采样间隔)。 N' 是设置的最大采样点数，超过该值就结束采样。

同时，当一个被测信号周期的采样点结束后，就发送信号量，这时会进行中断级的任务切换。切换到数据处理任务 process_task( )中，数据处理任务的流程图见图 6-8 所示。其中需要注意的是：在信号量未到时，任务不是静止死循环的等待，而是去执行其它就绪态的任务，等待是形象的说法。另外，这里的 N 是单个被测信号周期的采样总点数。

从以上结构流程图看，完成一次采样并没有立即发送信号量，而是等到一个被测信号周期全部采完后才发送信号量，转交给采样任务对所采数据进行处理。可以看出，采样驱动和采样时间优化实现较简单，计算机工作量不大。

6.3.2 串口驱动设计

在串行通讯中，需要一个缓冲区来解决外部设备的处理速度和 CPU 速度不匹配问题。发送数据时，先把数据写到缓冲区中，然后程序会自动地把数据逐个取出并往外串口发送。接收数据时，先将这些预收的数据存于缓冲区，直到收到若干个满

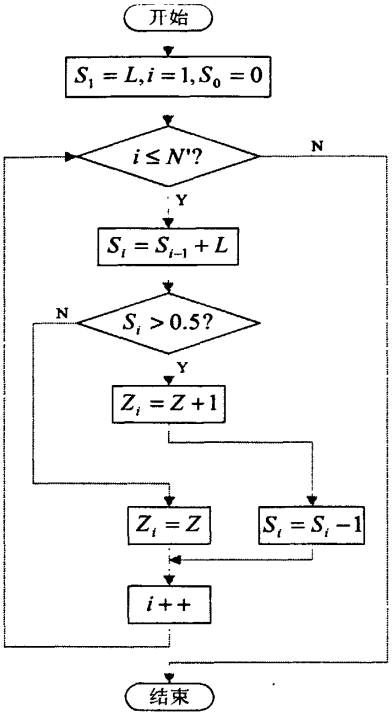

图 6-7　采样时间的优化流程图

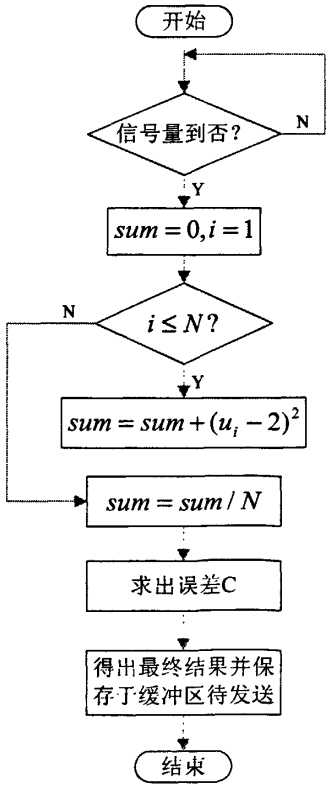

图 6-8　数据处理任务流程图

足一定数量的字节后才交给 CPU 处理。在实际工程中，需要两个缓冲区：发送缓冲区和接收缓冲区。当然，这里的缓冲区一般定义为数组。

在 μC/OS-II 内核中，信号量是常用的通信与同步的机制，在串口通讯中，可引入信号量对缓冲区发送和缓冲区接收的操作进行同步。当然，这里用到了数据发送信号量和数据接收信号量。用信号量的串口操作过程如下：用户任务想在缓冲区中写数据，但如果缓冲区满，任务就在等待信号量，但同时 CPU 可以不空等待而去运行别的任务，直到信号量到来，这样可以显著提高 CPU 的工作效率。待中断服务子程序 ISR 从缓冲区读走数据后就立即唤醒此睡眠的任务，然后操作系统就会进行任务切换，去执行刚才等待信号量的任务；用户任务读数据时，如果缓冲区为空，也可以等待信号量(也叫在信号量上睡眠)，待外部设备有数据存进缓冲区就唤醒信号量，同样操作系统也会进行任务切换，去执行等待信号量的任务，进行读操作。

接收中断发生时，中断服务子程序 ISR 从串口 UART 的接收缓冲器中读出收到的字节，并存入接收缓冲区，待到接收的字节数满足要求，就发送信号量进行任务切换，执行用户读操作任务程序。在这个过程中，同时查询缓冲区是否已满，如果缓冲区已满，就放弃接收到的字符，即不进行接收。这里需要注意的是，无论是接收缓冲区还是发送缓冲区，其大小应该合理设置，一方面可以降低数据丢失的几率，另一方面又可以避免存储空间的不必要浪费。发送数据时，发送缓冲区的大小和发送信号量的初始值设成一样，其含义是表示发送缓冲区是空的，如果在发送过程中，发送缓冲区已满，用户任务就在信号量上睡眠(等待信号量)。如果发送缓冲区没有满，有发送空间，用户任务就可以向发送缓冲区中写入数据。如果写入的字节是发送缓冲区的首字节，那么就允许发送中断。这样，发送中断服务子程序 ISR 从发送缓冲区中取出最早写入的字节，并发送至 UART，这个发送串口的操作将会触发下一次发送中断，如此循环下去，一直到发送缓冲区的最后一个字节被取走并成功发送，此时发送中断需要重新关闭。在中断服务子程序 ISR 向串口 UART 输出的同时，给任务发信号量，发送任务就会根据信号量计数值大小判断发送缓冲区是否可用。

以下是相关的串口通讯函数有：

Uart_Init(int baud, int number_serial)；用于初始化指定的串行口，包括对指定的串行口设定串口通信的波特率及相关寄存器的设置。

Uart_SendByte(unsigned char data)；向已选定的串行口发送字节单位的数据。

Uart_GetByte( )；接收已选串行口的数据，并设定超时时间，如果等待超时，返回 False。另外，使用前需先建立串行口的两个信号量，分别用于发送和接收。

图 6-9 为串口的发送程序流程图，同样在等待信号量的时间里去执行其它就绪态的任务。

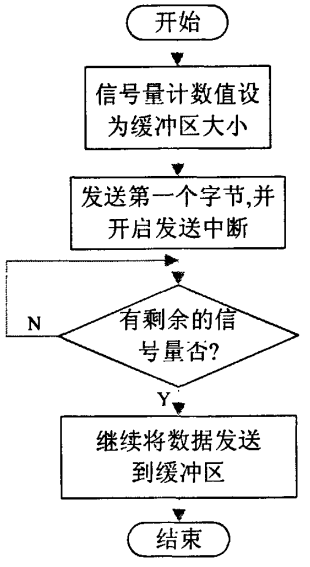

图 6-9 串口发送流程图

6.4 实验数据

在本实验中，我们用函数发生器产生标准的正弦电压 $u(t)=\sin(2\pi\cdot 50t)$ 作为被测对象，显然信号频率 f 为 $50Hz$，幅度 U_m 为 $\pm 1V$，用移植了 μC/OS-II 的微处理器 S3C2440 对标准信号进行实时采样，通过测得结果就可算出信号的有效值。然后把这个有效值与理想有效值进行比较，就可知道测量误差。表 6-1 是采样时间间隔(采样周期) ΔT 为 0.05ms，采用和没有采用以上减小误差的改进算法时，测出的有效值。

表6-1 ΔT=0.05ms、优化和不优化下的测量结果(mv)

第n周期	传统方法(不计误差)	改进方法(误差补偿)
1	687	710
2	687	711
3	687	709
4	688	709
5	688	711
6	687	710
7	688	711
8	687	711
9	687	710
10	687	709

表6-2是采样间隔时间ΔT为0.5ms，采用和没有采用以上减小误差的改进算法时，测出的有效值。

表6-2 ΔT=0.5ms、优化和不优化下的测量结果(mv)

第n周期	传统方法(不计误差)	改进方法(误差补偿)
1	667	686
2	667	687
3	667	687
4	668	687
5	668	686
6	667	686
7	668	687
8	667	686
9	667	687
10	667	687

表6-3是采样间隔时间ΔT为3ms，采用和没有采用以上减小误差的改进算法时，测出的有效值。

表6-3 ΔT=3ms、优化和不优化下的测量结果(mv)

第n周期	传统方法(不计误差)	改进方法(误差补偿)
1	750	743
2	749	742
3	749	742
4	750	743
5	748	743
6	751	743
7	750	742
8	750	742
9	749	743
10	748	743

6.5 结果分析

从以上测量结果可以看出，通过优化算法，以及减小采样时间间隔，能提高测量精度。标准信号的有效值为：

$$U=\frac{1}{\sqrt{2}}\approx 0.707V=707mV$$

ΔT为0.05ms时采用优化算法的测量误差为：

$$\Delta c = \frac{(709 \sim 711) - 707}{707} \times 100\% = 0.2829\% \sim 0.5658\%$$

ΔT 为0.05ms时没有采用优化算法的测量误差为：

$$\Delta c = \frac{(687 \sim 688) - 707}{707} \times 100\% = -2.8289\% \sim -2.6874\%$$

ΔT 为0.5ms时采用优化算法的测量误差为：

$$\Delta c = \frac{(686 \sim 687) - 707}{707} \times 100\% = -2.8289\% \sim -2.9703\%$$

ΔT 为0.5ms时没有采用优化算法的测量误差为：

$$\Delta c = \frac{(667 \sim 668) - 707}{707} \times 100\% = -5.5163\% \sim -5.6577\%$$

ΔT 为3ms时采用优化算法的测量误差为：

$$\Delta c = \frac{(742 \sim 743) - 707}{707} \times 100\% = 4.9505\% \sim 5.0919\%$$

ΔT 为3ms时没有采用优化算法的测量误差为：

$$\Delta c = \frac{(748 \sim 751) - 707}{707} \times 100\% = 5.7992\% \sim 6.2235\%$$

至此，可以得出如下重要结论：

(1) 补偿算法能有效减小测量误差，提高测量精度。但是影响测量误差的主要因素是采样的时间间隔，补偿算法只能在合适的采样时间间隔条件下，更进一步地提高测量精度。

(2) 在采样时间间隔选取上，要尽可能地与被测信号周期成倍数关系，这样可以提高测量精度。

6.6 小结

本章主要对系统硬件/软件进行了设计，并对实验测得数据进行了分析和总结。在系统硬件设计中，A/D 采样电路是关键，2V 直流电压要求非常稳定且纹波较小，否则很影响测量精度。在系统软件设计中，补偿算法和 ΔT 优化是编写采样驱动的依据。

从实验结果看，在一定的采样周期下，补偿算法能有效地减小测量误差，提高测量精度。但采样周期过大，补偿算法效果不明显。另外，要尽可能地选取采样时间间隔与被测信号周期成倍数关系，这样可以提高测量精度。

第7章 总结与展望

μC/OS-II 虽然比较适合于嵌入式操作系统的应用开发，但 μC/OS-II 也有一些缺点和不足：它仅仅是一个内核，只提供了操作系统的基本功能，没有用户接口、网络功能等，特别在互联网发达的今天没有网络功能是不能接受的，因此 μC/OS-II 内核离广泛应用还有一定距离；同时 μC/OS-II 不是针对特定的硬件，要实现在特定的硬件上运行还必须实现相应的移植程序，这也带来移植的难度。

本文就基于 μC/OS-II 的交流采样研究方面所作的工作有：

(1) 对实时内核 μC/OS-II 进行了详细的分析，特别对内核的关键函数如关中断函数 OS_ENTER_CRITICAL( )、开中断函数 OS_EXIT_CRITICAL( )、任务切换函数 OS_TASK_SW( )和运行优先级最高的就绪态任务的函数 OS_StartHighRdy( ) 等作了详细的分析，并结合目标板的资源对这些函数进行了编写，这是移植 μC/OS-II 最重要的工作，也是最容易出问题的地方。

(2) 介绍了 ARM9 的体系架构，特别对缓存 Cache 和 MMU 的功能和原理进行了分析，编写了实现虚存和缓存的代码，提高了整个嵌入式系统快速和高效的性能。

(3) 本文从理论上探讨了电压同步交流采样有效值的误差公式，提出了一种误差补偿方法。另外，对采样周期误差也进行了分析，并给出了采样周期误差的优化算法。

(4) 在 ADS1.2 的开发环境下，用 ARMulator 和 H-JTAG 仿真调试和测试内核，编写了串口驱动和实时采样驱动程序。通过串口超级终端将最终测得结果显示在 PC 机上，分析数据并验证了误差补偿算法的正确性和可行性。

由于初次接触嵌入式操作系统，操作系统的移植花了很多时间，最后在有经验的同学的指导下才得以调试通过。虽然本课题取得了较好结果，但仍有一些不足需要继续研究：

(1) 在本实验中，我们用的是函数发生器产生的标准正弦信号作为被测信号，被测信号的频率都选用 50Hz，对于不同频率的信号本文没有进行实验。

(2) 本文只对正余弦信号进行了测试和验证，对于不规则信号的有效值测量，没有作进一步探讨。

参考文献

[1] Jean J.Labrosse 著,邵贝贝译.嵌入式实时操作系统 μC/OS-II (第二版)[M],北京航空航天大学出版社,2003,156-158.

[2] 郑巍.大型自由和开源软件进化研究[J],计算机工程与设计,2008,29(11):1.

[3] Warren Sack, Françoise Détienne, Nicolas Ducheneaut, Jean-Marie Burkhardt, Dilan Mahendran and Flore Barcellini. A Methodological Framework for Socio-Cognitive Analyses of Collaborative Design of Open Source Software[J]. Computer Supported Cooperative Work (CSCW), Springer Netherlands, 2006, 15:229-250.

[4] E. Haruvy, A. Prasad and S. P. Sethi. Harvesting Altruism in Open-Source Software Development[J], Journal of Optimization Theory and Applications, Springer Netherlands, 2003, 118:381-416.

[5] DANIEL P. BOVET and MARCO CESATI. 深入理解 LINUX 内核[M],中国电力出版社,2001.

[6] W. Richard Stevens 著,尤晋元译. UNIX 环境高级编程[M],机械工业出版社,2002.

[7] 魏忠,蔡勇,雷红卫.嵌入式开发详解[M].北京:电子工业出版社,2003.

[8] 沈绪榜,何立民主编.《2001 嵌入式系统及单片机国际学术交流会论文集》,北京航空航天大学出版社,2001 年 10 月.

[9] 张腾.嵌入式实时操作系统内核的研究与实现[M],北京邮电大学,2001.

[10] 李仕勇等.多任务操作系统在嵌入式系统开发中的应用[J],北方交通大学报,2002,26:79-82.

[11] Lu C, Stankovic J A, Tao G, Son S H, Marley M. Performance Specifications and Merics for Adaptive Real-Time Systems. IEEE Real-Time Systems Symposium, Orlando, FL, December 2000, 27-30.

[12] Chenyang Lu, John A. Stankovic, Sang H. Son and Gang Tao. Feedback Control Real-Time Scheduling: Framework, Modeling, and Algorithms[J], Springe Netherlands 2002, 23:85-126.

[13] Abdelzaher T F, Lu C. Modeling and Performance Control of Internet Servers, 39th IEEE Conference on Decision and Control, Sydney, Australia, December 2000, 2234--2239.

[14] 魏立峰,于海斌.一种基于自适应控制的软实时调度算法研究[J],系统仿真学报,2004,16(4):760-764.

[15] 罗玎玎,赵海,孙佩刚等.RM 算法的运行时开销研究与算法改进[J],通信学报,2008,29(2):79-86.

[16] 涂刚,阳富民,卢炎生.基于至少满足弱硬实时限制的调度算法[J],华中科技大学学报(自然科学版),2006,32(2):67-69.

[17] 洪艳伟,赖娟.一种硬实时调度算法的可行性判定及实现[J],计算机与信息技术,2006,6:21.

[18] Mark R. Heckman, Cui Zhang, Brian R.Becker. Towards applying the composition Principle to verify a Microkernel operating system,Lecture Notes in Computer Science,Springer Berlin or Heidelberg,1996,1125:235-250.

[19] Tian zhou Chen,Wei Hu and Yi Lian.Power-Efficient Microkernel of Embedded Operating System on Chip,Lecture Notes in Computer Science,Springer Berlin or Heidelberg,2006,4186:473-479.

[20] Kun-Yuan Hsieh, Yung-Chia Lin, Chien-Chin Huang and Jenq-Kuen Lee. Enhancing Microkernel Performance on VLIW DSP Processors via Multiset Context Switch[J],Journal of Signal Processing Systems, Springer New York,2008,51(3):257-268.

[21] Eric Verhulst.The Rationale for Distributed Semantics as a Topology Independent Embedded Systems Design Methodology and its Implementation in the Virtuoso RTOS[J],Design Automation for Embedded Systems,Springer Netherlands,2002,6(3):277-294.

[22] John C.Reynolds.An Overview of Separation Logic[J],Lecture Notes in Computer Science,Springer Berlin or Heidelberg,2008,4171:460-469.

[23] 颜小君,刘觉民,陈明照等.一种改进的软件同步交流采样法[J],仪表技术,2007,4:10.

[24] 王海燕.单片机系统中软件陷阱抗干扰技术的研究[J],四川文理学院学报(自然科学),2008,18(5):1-2.

[25] 李雪莉,张兆莉,史晓龙.面向嵌入式Linux系统的软中断设计与实现[J],微计算机信息,2007,23(5-2):60.

[26] 李小平,王海波,王守峰.时间片随机到达的轮转调度算法分析[J],哈尔滨理工大学学报,2001,6(5):14-16.

[27] 赵宁,井海明,马增强等.操作系统中多进程并行时的死锁问题[J],铁路计算机应用,2007,16(12):48.

[28] 田泽.嵌入式系统开发与应用[M],北京:北京航空航天大学出版社,2005,73-74.

[29] 杜春雷.ARM体系结构与编程[M],北京:清华大学出版社,2003,265,268-270.

[30] Andrew N.Sloss,Dominic Symes,Chris Wright 编著,沈建华译.ARM 嵌入式系统开发—软件设计与优化[M],北京:北京航空航天大学出版社,2005,473,474-476.

[31] 聂俊岚,王宝珠.影响高速缓冲存储器提高命中率的几种因素[J],中国仪器仪表,1998,2:11.

[32] Jim Handy.The Cache Memory Book[M],Second edition,Academic Press,1998.

[33] 赵星寒,周春来,刘涛.ARM开发工具ADS原理及应用[M],北京:北京航空航天大学出版社,2006.

[34] ARM公司.ARM Developer Suite__Compilers and Libraries Guide.

[35] 冯兴田,刘静.基于C8051F单片机进行功率测量时的交直流采样比较[J],电子设计应用,

2008,3:98-99.

[36] 于月平.交流采样量测量误差来源及解决方法[J],电力自动化设备,2008,28(11):118.

[37] 周秀丽,张辉宜,陈文星.μC/OS-II 在交流同步采样中的应用[J],计算机技术与发展,2007,17(6):200-202.

[38] 沈安文,刘琳霞.基于 μC/OS-II 的电能校验仪的研究[J],电测与仪表,2005,42(474):13-15.

[39] 张盎然,陈明华,杨扬.基于准同步算法的谐波分析方法[J],电测与仪表,2002,39(1):10-12.

[40] 张超,韩富春.基于 DSP 和准同步算法的谐波监视控制器的研制[J],计测技术,2008,28(3):9-10.

[41] 李加升,柴世杰,戴瑜兴.基于插值理论的准同步算法在谐波检测中的应用研究[J],电测与仪表,2008,45(510):2-3.

[42] Andria G,Savino M,Trotta A.Windows and interpolation algorithms to improve electrical measurement accuracy[J],IEEE Tans on IM,1989,38(4):856-863.

[43] 蔡菲娜,左伍衡.改进的双速率同步采样法及其傅里叶变换[J],浙江大学学报(工学版),2005,39(3):415-417.

[44] 周军,李孝文,盛艳.双速率同步采样法在电力系统谐波测量中的应用[J],计量学报,1999,20(2):151-154.

[45] 方伟林,王立功.双速率同步采样法在交流测量中的应用[J],电测与仪表,1997,34(376):21-23.

[46] 裘云,王健,秦霆镐.两种改进的软件同步采样实现方法的分析[J],电测与仪表,2000,37(9):5-7.

[47] 盛新富,戚庆成(Sheng Xinfu,Qi Qingcheng).常用电工参数数字化测量中的非同步采样误差(Non-synchronous sampling errors in the digital measurement of common electrical parameter)[J],电工电能新技术(Advanced Technology Electrical Engineering and Energy),1998,17(1):10-14.

[48] 史旺旺,陈虹.交流采样的误差分析与补偿[J],电测与仪表,1999,36(406):4.

[49] 徐垦,程时杰.有功功率测量的异步采样方法[J],电子测量技术,2007,30(9):65.

[50] 黄纯,彭建春,刘光晔等.周期电气信号测量中软件同步采样方法的研究[J],电工技术学报,2004,19(1):76-77.

[51] 陈飞,尹斌,姜锋.软件同步采样实现方法的分析与比较[J],仪表技术,2005,6:49-50.

致谢

光阴似箭，就快毕业了，三年的求学生涯即将结束。说心里话，有机会能在湖北工业大学读完本科继续攻读硕士学位，的确非常之难得。本论文是在潘健副教授严格的指导下完成的。潘老师严谨的学术作风，正直的做人原则，以及认真的做事态度和不断创新的追求精神，永远都是我工作和学习的榜样。在此，向潘老师表示最诚挚的感谢和敬意！

此外，还要感谢身边的同学在这三年中给我的帮助和支持。还要感谢陪我一起渡过了七年美好时光的同寝室室友。

深深感谢我的父母，感谢他们在经济上和精神上给予我的莫大支持，没有他们在我背后的支撑，我不可能顺利毕业！

感谢所有帮助过和支持过我的同学和朋友们！