东北大学

硕士学位论文

# 基于RTEMS操作系统的嵌入式开发平台

姓名：薛海鹏

申请学位级别：硕士

专业：计算机应用技术

指导教师：于戈

20031201

# 基于 RTEMS 操作系统的嵌入式开发平台

## 摘要

随着嵌入式技术的迅速发展，全世界嵌入式系统带来的工业年产值已猛增到 1 万亿美元以上[1]。无疑，嵌入式产品的市场是巨大的，面对激烈的市场竞争，人们越来越希望可以在更短的时间内开发出所需产品。满足这一愿望的方法就是引入易于掌握和使用的开发平台，从而缩短开发时间、提高产品性能。在目前的四种主流开发平台中，实时操作系统(Real Time Operating System)开发平台因其自身的实时性，可移植性和通用性，在面向工业的嵌入式系统开发中有着不可替代的优势。

本课题设计了以实时操作系统为软件基础，以高性能 32 位 CPU 为硬件基础的面向工业的嵌入式开发平台。通过性能分析和测试，最终在 RTEMS(Real-Time Executive for Multiprocessor Systems)实时操作系统的基础上，设计并开发了基于 Motorola ColdFire MCF5307CPU 的嵌入式开发平台。

RTEMS 操作系统是由美国军方使用的实时导弹系统发展而来的可供嵌入式应用的多任务实时操作系统,可以为多种实时应用提供高性能的软件开发环境。RTEMS 的主要特征有[2]：基于优先级的多任务实时系统；支持多任务的通信和同步；支持中断管理；支持动态内存管理和分配；支持多处理器；支持速率单调调度算法；支持文件系统；支持多种网络协议，有强大的网络功能；RTEMS 为上层应用的开发提供了符合 POSIX1003.1b 标准，ITRON 规范和 RTIED 的 API 支持。实验显示，在衡量实时性的两个主要指标—中断响应时间和任务切换时间的比较中，RTEMS 的实时性能在任务过载的情况下可以和业界著名的嵌入式开发平台 VxWORKS 相媲美，而 RTlinux 却比它们慢了一个数量级。

CPU 选型为 Motorola 公司的 MCF5307 嵌入式处理器（Coldfire 系列）。ColdFire 是 Motorola 公司继承 68K 系列 CPU 的嵌入式微处理器，主要面向工业控制和低成本的通讯市场。ColdFire 最大的优点是和它所相关的资源特别丰富，包括：编译器、调试器、硬件资源（原理图、PCB 图）、源代码、操作系统等。

本文最后详细阐述了该嵌入式开发平台的设计思想、实现方法及调试的全过程。在自主设计的开发板上，本课题成功地实现了 RTEMS 的移植，为嵌入式软件的开发提供了强大的开发平台。

关键词 RTEMS 嵌入式 RTOS 开发平台 BSP 实时操作系统

**An Embedded Development Platform Based on RTEMS Operating System**

# Abstract

Nowadays, with the break out of embedded systems for industrial applications, a huge market of embedded systems is forming and thriving. However, to develop the embedded software in a short time is very difficult. The best way to solve this problem is using embedded development platforms, which will help developers do thier work smoothly and quickly. By using Real Time Operating System, not only can developers functionalize but also standardize the embedded software.

For this we developed an embedded development platform, which based on a Real Time Operating System and a high performance 32 bits CPU. Through comparing the performence of different operating systems and different CPUs, we choosed RTEMS(Real-Time Executive for Multiprocessor Systems) and the Motorola Coldfire MCF5307 to develop our platform.

Initially, RTEMS stood for the Real-Time Executive for Missile Systems, but today the "M" means Multiprocessor. RTEMS is a real-time executive which provides a high performance environment for embedded applications including many features: priority-based multitasking, preemptability controlled on a per-task basis, intertask communication and synchronization, responsive interrupt management, dynamic memory allocation, homogeneous and heterogeneous multiprocessor support, rate monotonic scheduling, file system support, and networking support. By calculating the time of interrupt latency and task switch context, which are key properties of any hard-real time system, we choose RTEMS because RTEMS has the similar performence with VxWorks, while RTLinux's latencies are substantially higher on the loaded system.

We choosed Motorola Coldfire MCF5307, because ColdFire is the very famous CPU like M68K and it is low cost for control and communication. The advantages of MCF5307 are its rich free development resources, including compiler, debuger, hardware deisgn (PCB file), and open sources operating systems.

In the end, this thesis discusses the design, implementation and debugging of this platform. Moreover, we had successfully ported the RTEMS to our development board.

In a word, this platform will give a firm support to the embedded software development.

**Key words** Embedded, RTEMS, BSP, Platform, Real Time Operating System

# 声　明

本人声明所呈交的学位论文是在导师的指导下完成的。论文中所取得的研究成果除加以标注和致谢的地方外，不包含其他人已经发表或撰写过的研究成果，也不包括本人为获得其他学位而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示谢意。

本人签名：薛海鹏

日　　期：2003.12

# 第一章 引言

## 1.1 背景

计算机，作为二十世纪人类最伟大的发明之一，伴随着新世纪的曙光也进入了一个充满机遇和挑战的阶段——后 PC 时代。

传统的计算机在人们的心目中就是一台主机加上一个显示器，通过装配不同的应用软件，以不同面目出现并应用在各个行业，其典型产品为 PC。而嵌入式计算机则是以嵌入式系统的形式隐藏在各种装置、产品和系统中。例如，VCD 机、机顶盒、手机、PDA（如商务通等）、车载 GPS 系统、数控机床、网络冰箱等。在后 PC 时代里，嵌入式系统已经逐渐成为舞台的主角，在产业的发展中起着举足轻重的作用。

嵌入式系统的定义[1]是(Embedded Systems)：以（1）应用为中心、以（2）计算机技术为基础、（3）软件硬件可裁剪、（4）适应应用系统对功能、可靠性、成本、体积、功耗严格要求的专用计算机系统。

嵌入式系统的重要的特点有 4 个：专用性，功能强大，性价比高，开发周期短。

今天，世界嵌入式系统硬件和软件开发工具市场规模约 2000 亿美元，嵌入式系统带来的工业年产值已达 1 万亿美元以上[6]。嵌入式系统已经在很多应用领域得以应用。

随着信息技术的发展，数字化产品空前繁荣。嵌入式软件已经成为数字化产品设计创新和软件增值的关键因素，是未来市场竞争力的重要体现。由于数字化产品具备硬件平台多样性和应用个性化的特点，因此嵌入式软件呈现出一种高度细分的市场格局，国外产品进入也很难垄断整个市场，这为我国的软件产业提供了一个难得的发展机遇。嵌入式支撑软件是嵌入式系统的基础，而与嵌入式操作系统紧密联系的开发调试工具是嵌入式支撑软件的核心，它的集成度和可用性将直接关系到嵌入式系统的开发效率。

国际上看，嵌入式软件市场约占整个软件市场的 10%左右[1]。虽然我国目前的嵌入式软件市场刚刚起步，但巨大的国内软件产品需求和极具发展潜力的国际软件市场，将给我国软件产业的跨越式发展带来重大机遇。考虑到未来两年数字化产品的快速普及促使嵌入式软件需求进一步增大，我们有理由相信，嵌入式软件市场将超过软件市场的平均增长率，年需求达 50 亿人民币以上。

## 1.2 任务来源

随着嵌入式技术的不断成熟与发展,面向工业设备的嵌入式产品开发已经形成了一个很大的产业，包括：工业控制、工业仿真、汽车车载设备、仪器仪表监测、实时监控、医疗仪器等。面对巨大的市场需求，为了提高嵌入式产品的开发效率，满足不同工业应用开发的需要，我们迫切需要一套针对工业嵌入式产品开发的嵌入式开发平台。本课题主要解决嵌入式开发平台的设计和实现。该课题已上报国家 863 计划项目，并于 2002 年正式启动。

## 1.3 解决方法

嵌入式产品的市场是巨大的，面对激烈的市场竞争，如何能够在更短的时间内开发出所需的功能呢？解决这一问题的方法就是引入易于掌握和使用的开发平台，从而缩短开发时间提高产品性能。

目前嵌入式系统的开发平台主要包括下面 4 类[1]:

1. 实时在线仿真系统 ICE(In-Circuit Emulator)
2. 高级语言编译器(Compiler Tools)
3. 源程序模拟器(Simulator)
4. 实时操作系统(RTOS)

与前 3 种开发平台相比较，实时操作系统(RTOS)有着不可替代的优势：

RTOS 摆脱了以往开发者必须直接在处理器上直接编写的弊端，将系统软件和应用软件的开发分开，避免了每次开发都需要进行系统开发的重复劳动。

RTOS 是一段嵌入在目标硬件和应用软件之间的代码，将用户的应用程序和硬件很好的连接起来，允许应用开发者不必过多的关心硬件平台。

RTOS 为用户提供一个可靠的实时内核，提供给用户标准的 API，并根据各个任务的优先级，合理地在不同任务之间分配 CPU 时间。

基于 RTOS 开发出的程序，具有较高的可移植性，一些成熟的通用程序可以作为专家库产品推向社会。

通过采用实时操作系统(RTOS)开发平台，促进了嵌入式软件的函数化、产品化和标准化，从而减少了大量重复劳动，提高了知识创新率。

# 第二章 嵌入式开发平台

## 2.1 嵌入式开发平台的重要性

嵌入式技术是一个新兴的高技术领域，在短时间内开发出所需功能是不容易的，但是激烈的市场竞争则要求产品能够快速上市，这一矛盾要求嵌入式处理器能够有容易掌握和使用的开发平台，从而提高产品的性能并缩短开发周期。本章将介绍目前主流的四类嵌入式开发平台，并给出选用实时操作系统(RTOS)作为开发平台的理由。

从事嵌入式开发的往往是非计算机专业人士，面对市场上成百上千种处理器，选择是一个问题，学习掌握处理器结构及其应用更需要时间，因此开发平台的引入将能很好的满足开发者的需求。好的开发平台除能够开发出处理器的全部功能外，还应当是用户友好的。目前嵌入式系统的开发平台主要包括下面四类[1]：

**1. 实时在线仿真系统 ICE(In-Circuit Emulator)**

今天，计算机辅助设计已经非常发达，但是实时在线仿真系统(ICE)仍是进行嵌入式应用系统调试最有效的工具。ICE 有三个主要特点：一、ICE 可以通过实际执行，对应用程序进行原理性检验，排除以人的思维难以发现的设计逻辑错误；二、ICE 可以在应用系统中仿真微控制器的实时执行，发现和排除由于硬件干扰等引起的异常执行行为；三、高级的 ICE 带有完善的跟踪功能，可以将应用系统的实际状态变化、微控制器对状态变化的反应、以及应用系统对控制的响应等以一种录像的方式连续记录下来，以供分析。

嵌入式应用的特点是和现实世界中的硬件系统有关，存在各种异变和事先未知的变化，这就给微控制器的指令执行带来了各种不确定性，这种不确定性可以通过 ICE 的实时在线仿真才能发现，特别是在分析可靠性时要在同样条件下多次仿真，以发现偶然出现的错误。

ICE 不仅是软件硬件排错工具，而且也是提高和优化系统性能指标的工具。高档 ICE 工具(如美国 NOHAU 公司的产品)是可根据用户需求进行裁剪的系统，亦可根据需要选择配置各种档次的实时逻辑跟踪器(Trace)、实时映象存储器(Shadow RAM)及程序效率实时分析功能(PPA)。

**2. 高级语言编译器(Compiler Tools)**

C 语言作为一种通用的高级语言，主要特点有：一、大幅度提高了嵌入式系统工程师的工作效率，使之能够充分发挥出嵌入式处理器日益提高的性能，缩短产品进入市场时间；二、C 语言便于移植和修改，使产品的升级和继承更迅速；三、

C 语言编写的程序易于在不同的开发者之间进行交流，从而促进了嵌入式系统开发的产业化。

和通用计算机中的 C 语言编译器不同的是，嵌入式系统中的 C 语言编译器要进行专门优化，以提高编译效率。优秀的嵌入式系统 C 编译器代码长度和执行时间仅比以汇编语言编写的同样功能程序长 5~20%。区别嵌入式 C 编译器工具的重要指标就是编译的质量。

新型的微控制器指令及 SOC 速度不断提高，存储器空间也相应加大，已经达到甚至超过了目前的通用计算机中的微处理器，为嵌入式系统工程师采用过去一直不敢问津的 C++语言创造了条件。C++语言强大的类、继承等功能更便于实现复杂的程序功能。特别是 1995 年初在日本成立的 Embedded C++技术委员会经过几年的研究，针对嵌入式应用制订了减小代码尺寸的 EC++标准。(更多的信息，可查询 http://www.tasking.com)。

C/C++/EC++引入嵌入式系统，使得嵌入式开发和个人计算机、小型机等之间在开发上的差别正在逐渐消除，软件工程中的很多经验、方法乃至库函数可以移植到嵌入式系统。在嵌入式开发中采用高级语言，还使得硬件开发和软件开发可以分工，从事嵌入式软件开发不再必须精通系统硬件和相应用汇编语言指令集。

**3. 源程序模拟器(Simulator)**

源程序模拟器是在人机接口完备的一些工作平台上，例如小型机和 PC，通过软件手段来模拟执行为某种嵌入式处理器内核编写的源程序测试工具。简单的模拟器可以通过指令解释方式逐条执行源程序，分配虚拟存储空间和外设，供程序员检查；高级的模拟器甚至可以利用计算机的外部接口模拟出处理器的 I/O 电气信号。

模拟器软件独立于处理器硬件，一般与编译器集成在同一个环境中，是一种有效的源程序检验和测试工具。但值得注意的是，由于模拟器毕竟是用一种处理器来模拟另一种处理器的运行，在指令执行时间、中断响应、定时器等方面很可能与实际处理器有相当的差别。另外它无法和 ICE 一样，仿真嵌入式系统在应用系统中的实际执行情况。

**4. 实时操作系统 (Real Time Operation System, RTOS)**

实时操作系统(RTOS)是目前新兴的嵌入式应用软件的基础和开发平台。RTOS 是一段嵌入在目标代码中的软件，用户的应用程序都建立在 RTOS 之上。不但如此，RTOS 还是一个可靠性和可信性很高的实时内核，将 CPU 时间、中断、I/O、定时器等资源都包装起来，留给用户一个标准的 API，并根据各个任务的优先级，合理地在不同任务之间分配 CPU 时间。

RTOS 是针对不同处理器优化设计的高效率实时多任务内核，商品化的 RTOS

可以面对几十个系列的嵌入式处理器 MPU、MCU、DSP、SOC 等提供类同的 API 接口，这是 RTOS 基于设备独立的应用程序开发基础。因此基于 RTOS 上的 C 语言程序具有极大的可移植性。据专家测算，优秀 RTOS 上跨处理器平台的程序移植只需要修改 1~5%的内容。在 RTOS 基础上可以编写出各种硬件驱动程序、专家库函数、行业库函数、产品库函数，和通用性的应用程序--起，可以作为产品销售，促进行业内的知识产权交流。

## 2.2 RTOS 技术对嵌入式软件发展的重要意义

与前 3 种开发平台相比较，实时操作系统(RTOS)有着不可替代的优势：

RTOS 摆脱了以往开发者必须直接在处理器上直接编写的弊端，将系统软件和应用软件的开发分开，避免了每次开发都需要进行系统开发的重复劳动。

RTOS 是一段嵌入在目标硬件和应用软件之间的代码，将用户的应用程序和硬件很好的连接起来，允许应用开发者不必过多的关心硬件平台。

RTOS 为用户提供一个可靠的实时内核，提供给用户标准的 API，并根据各个任务的优先级，合理地在不同任务之间分配 CPU 时间。

基于 RTOS 开发出的程序，具有较高的可移植性，一些成熟的通用程序可以作为专家库产品推向社会。

嵌入式工业的基础是以应用为中心的芯片设计和面向应用的软件开发。“实时操作系统(RTOS)进入嵌入式工业的意义不亚于历史上机械工业采用三视图的贡献，对嵌入式软件的标准化和加速知识创新是一个里程碑”[1]。

# 第三章 实时操作系统

上一章已经讨论过：以实时操作系统为基础的开发平台在面向工业的嵌入式应用产品中有这不可替代的优势。因此本课题首先对实时操作系统作了全面的研究。通过对现有的支持嵌入式应用的实时操作系统进行调查、分析和比较，从而为下面选择合适的操作系统奠定了基础。本章将探讨本课题对实时操作系统的调研分析结果，并给出衡量一个可供嵌入式应用的 RTOS 性能的重要因素。

## 3.1 实时操作系统介绍

实时操作系统(RTOS)是嵌入式应用软件的基础和开发平台，主要是服务于嵌入式应用产品的开发人员的。RTOS 一般都通过一个实时多任务内核(Kernel)来完成如下的功能：任务管理、任务间同步和通信、存储器优化管理和实时时钟服务等。除了实时内核之外，具体到某个特定环境下的 RTOS 还需要有一些外围模块来满足实际系统的需要。

一、实时系统（**Real-Time Systems**）

多数嵌入式系统有一个共同的特点[7]：对系统的实时性和可靠性有严格的要求，这些系统也被称为实时系统。实时系统是指能在确定的时间内执行其功能并对外部的异步事件做出响应的计算机系统。一个系统具有实时性并不是说该系统的响应和处理速度非常快；而是说该系统的响应和处理速度对于特定的应用来说足够快。一个高速系统也未必是实时系统。每个应用对时限(或速度)的要求是千差万别的，所以设计实时系统的时候，应该明确目标系统的实时要求是什么，不可盲目追求高速度，以免浪费资源。

实时系统又分两种[11]：软实时系统和硬实时系统。软实时系统要求任务运行的越快越好，并不要求某一任务必须在限定的时间内完成。硬实时系统则要求任务的运行不仅要正确无误而且要准时，如果任务在时限到来之前未能完成，就会发生无法预测的灾难性后果。大多数实时系统是这两者的结合。而且大多数的实时系统都是嵌入式的。

二、前后台系统

传统的嵌入式系统设计方法都不使用 RTOS，被称为前后台（Foreground/Background）系统或超循环（Super-Loop）系统[12]。嵌入式应用程序一般是一个无限的循环，循环中调用相应的函数完成相应的操作，这个循环体可以看成后台行为。中断服务程序处理异步事件，这部分可以看成前台行为。后台

也可以叫做任务级，前台叫做中断级。后台程序逐个检查每个任务是否具备了运行的条件，并逐个完成相应的操作。时间相关性很强的关键操作一定是靠中断服务来保证的。中断服务程序会给一个事件的发生置一个标志，然后退出中断，等后台程序运行到检查这个标志的时候才能完成事件的处理，再把事件发生标志复位。如果把事件处理也放在中断服务中进行，中断服务程序就会变的特别长，影响后面的中断。

在衡量嵌入式系统处理信息的及时性上有一个指标是任务级响应时间。前后台系统在最坏的情况下任务级响应时间取决于整个循环的执行时间。由于循环的执行时间不是常数，程序经过某一特定部分的准确也就不能确定。如果程序修改了，整个循环的时序也会受到影响。

前后台系统结构简单，额外开销少，适合于相对简单的单片机系统，在微波炉、电话机、玩具等产品中有广泛的应用。但是对于复杂一些的系统，它就显得力不从心了。

三、任务与多任务

在嵌入式系统中，一个任务也称作一个线程，是指一个程序分段[13]。这个程序分段被操作系统当作一个基本单元来调度。它大致相当于一般计算机操作系统中进程的概念。每个任务都是整个应用的一部分，都被赋予了一个优先级，有它自己的一套 CPU 寄存器和自己的堆栈空间。

一般情况下，每个任务都是一个无限的循环。不同的 RTOS 可能会给任务定义了不同种类的状态。典型的，每个任务会处于一下 5 种状态之一：就绪、休眠、运行、挂起和中断态。

多任务的实现实际上是靠 CPU 在许多任务之间转换、调度。CPU 按照某种调度算法，轮番服务于一系列任务中某一个。每个任务都象一个独立的前后台系统。多任务的运行使 CPU 的利用率得到最大的发挥，并使应用程序模块化。多任务的最大特点是，开发人员可以将很复杂的应用程序层次化，应用程序更容易设计和维护。

四、内核(Kernel)

在多任务系统中，内核负责管理各个任务，或者说为每个任务分配 CPU 时间，并负责任务之间的通信。内核提供的基本服务是任务调度和任务切换。因为实时内核允许将应用分解为若干个任务，由内核来管理它们，所以使用实时内核可以大大简化应用程序的设计。内核本身增加应用程序的额外负荷，代码空间增加 ROM 的用量，内核本身的数据结构增加了 RAM 的用量，这些代价和使用 RTOS 所带来的好处和便利相比较，还是值得的。

五、任务切换与调度

任务切换(Context Switch)是指内核停止当前执行的任务，转而去执行另一个任务的过程。一般的步骤是：先把当前任务的当前状态，即 CPU 寄存器中的内容，保存到当前任务堆栈；然后，把下一个要运行任务的环境从该任务的栈中取出并装入 CPU 寄存器中。这时，下一个任务就可以被 CPU 运行了。

调度(Scheduler 或 Dispatcher)是实时内核的另一个重要功能，它决定任务运行的时机和先后顺序。多数实时内核是基于优先级调度算法的。每个任务根据其重要程度的不同被赋予一定的优先级。CPU 总是让处于就绪状态的优先级最高的任务先运行。但是，究竟何时让高优先级掌握 CPU 的使用权，这就要看是是什么类型是内核。内核按其工作方式可分为两种不同的类型：非抢占式和抢占式。

六、非抢占式内核和抢占式内核

非抢占式（non-preemptive）内核要求每个任务自动放弃 CPU 的所有权[8]。它使用某种调度算法决定下一个要运行的是哪一个任务，等到当前的任务主动让出 CPU 的使用权之后才调度下一个要执行的任务。系统的异步事件还是由中断服务来处理。中断服务可以使一个高优先级的任务由挂起态变为就绪态。但中断服务结束后，CPU 控制权要回到原先被中断了的那个任务。使用非抢占式内核时，任务级响应时间比前后台系统要快的多，此时间取决于最长任务的执行时间。然而，使用非抢占式内核时，如果最长任务的执行时间不能确定，系统的实时性也就不能保证。

抢占式（preemptive）内核的实时性最好[8]。当系统响应时间很重要时，要使用抢占式内核。这种内核时，如果当前运行的任务使一个比它优先级更高的任务进入了就绪态，当前的任务就会马上被剥夺 CPU 的控制权，而那个更高优先级的任务会得到执行。如果是中断服务程序使一个更高优先级的任务进入就绪态，等中断完成时，中断了的任务会被挂起，而优先级高的那个任务开始运行。抢占式内核使得最高优先级任务的执行时间成为确定的，所以使得任务级响应时间得以最优化，而且是可知的。

目前大多数商业上的 RTOS 的实时内核都是抢占式的。本文将要提出的 RTEMS，其内核可以自由定制为抢占式或非抢占式。

七、任务优先级和优先级反转

实时系统中任务的优先级分为两种[14]。程序执行的过程中，如果任务的优先级保持不变，就称为静态优先级。在静态优先级系统中，所有任务的优先级在编译时就是已知的。如果程序执行时，任务的优先级可以动态改变，则称为动态优先级。

实时操作系统必须处理的一个问题是优先级反转。假设有三个任务 A、B、C，其中任务 A 的优先级最高，任务 B 次之，任务 C 的优先级最低。如果当前任务 A

和任务 B 由于某事件的发生而挂起，任务 C 正在执行。在执行过程中，任务 C 使用到某一共享资源，如打印机。这时任务 A 等待的事件到来了，则任务 A 会抢占到 CPU 而执行。但是，任务 A 运行过程中也要使用那个已被任务 C 占用的共享资源，它只能再次进入挂起状态等待该资源。当任务 C 继续运行的时候，任务 B 等待事件的发生会使任务 B 抢到 CPU。等任务 B 处理完毕以后，仍然是处于就绪态的任务 C 接着执行，直到任务 C 释放掉那个共享资源，实时内核才会让任务 A 获得该资源继续执行。这种情况下，任务 A 的优先级实际上降到了任务 C 的优先级水平。由于任务 B 剥夺了任务 C 的 CPU 使用权，使任务 A 的状况更加恶化。很明显，任务 A 和任务 B 的优先级发生了反转。纠正这个问题的一种方法是，在任务 C 使用共享资源时，提升它的优先级，等它释放共享资源后予以恢复。任务 C 的优先级应提高到高于允许使用该资源的所有任务的优先级。多任务内核这种为防止发生优先级反转而动态变换优先级的机制叫做优先级继承。

八、信号量（**semaphore**）

信号量是一种在多任务内核中普遍使用的约定机制。它主要用于：

- 任务互斥
- 任务同步
- 标志事件的发生

信号量就是象一个计数式的标志，它表明资源的使用情况和事件的发生情况。任务要想访问共享资源，必须先取得标志该资源的信号量。对信号量的操作有三种：建立（create）、等待（wait）和释放(post).

希望得到信号量的任务先要执行 wait 操作，如果该信号量有效（值大于 0），则信号量的值减 1，任务继续执行；如果信号量的值小于 0，等待信号量的任务就被列入等待该信号量的任务的列表（waiting list）。任务以 post 操作释放信号量。如果没有任务在等待信号量，信号量的值只是简单的加 1；如果有任务在等待该信号量，那么就会有一个任务进入就绪队列。

九、消息邮箱和消息队列

除了信号量之外，消息邮箱（message mail box）和消息队列（message queue）也常用于任务间的同步与通信。

消息邮箱常被设计成一个指针型变量，该指针指向的内容就是消息。通过内核服务，一个任务或一个中断服务程序可以把一则消息放到邮箱里去。同样，一个任务也可以通过调用内核服务接收这则消息。每个邮箱都有等待其中消息的任务列表，要得到消息的任务会因为邮箱为空而被挂起。内核提供的邮箱操作一般有：初始化邮箱、把消息放入邮箱(Post)和等待消息进入邮箱(Pend)等。

消息队列实际上就是邮箱的队列，队列的每个元素都是指向消息的指针。通

过内核服务，一个任务或一个中断服务程序可以把一则消息（该消息的指针）放到消息队列里去。同样，一个任务也可以通过调用内核服务从队列中取出消息。每个消息队列也都有一个等待其中消息的任务的等待列表。如果队列中没有消息，等待消息的任务就被挂起并放入该队列的等待列表中。内核提供的对消息队列操作与对邮箱操作类似。

十、时钟节拍

时钟节拍是是实时系统中周期性的时钟中断。中断之间的时间间隔取决于不同的应用。时钟的节拍式中断使得内核可以将任务延迟若干个时钟节拍，以及当任务等待事件发生时，提供等待超时的依据。时钟节拍的频率越高，系统的额外开销就越大。

各种实时内核都有将任务延时若干个时钟节拍的功能。然而，并不是每一次延时都刚好是时钟节拍的整数倍，延时的抖动总是会存在的。

## 3.2 支持嵌入式应用的 RTOS 介绍

从八十年代起，国际上就有人开始进行一些商用嵌入式系统的开发。据嵌入式系统杂志的最新统计报告，世界各国有 40 多家公司已成功地推出了 200 余种可供嵌入式应用的实时操作系统，其中著名的操作系统有[20]：

**Windows CE**：Microsoft Windows CE 是一个简洁的，高效率的多平台操作系统。它不是简化的 Windows95 版本，而是从整体上为有限资源的平台设计的多线程，完整优先权，多任务的操作系统。它的模块化设计允许它对于从掌上电脑到专用的工业控制器的用户电子设备进行定制。操作系统的基本内核需要至少 200K 的 ROM。从 SEGA 的 DreamCast 游戏机到现在大部分的高价掌上电脑，都采用了 Windows CE，但是无奈价格太高，使得整个产品的成本急剧上升。

**VxWorks**：去年 VxWorks 所在的公司 WindRiver 兼并了 pSOS 所在的 ISI 公司，使得该公司现在有两大 RTOS 系统。VxWorks 是目前嵌入式系统领域中使用最广泛，市场占有率最高的系统。它支持多种微处理器，如 x86，i960，Sun Sparc，Motorola MC68xxx，MIPS RX000，POWER PC 等等。 VxWorks 使用的是和 UNIX 不兼容的环境，大多数的 API 是专有的。它采用 GNU 的编译和调试器。

**pSOS**：ISI 公司已经被 WindRiver 公司兼并，pSOS 现在也属于 WindRiver 公司的产品。这个系统是一个模块化，高性能的实时操作系统，专为嵌入式微处理器设计，提供一个完全多任务环境，在定制的或是商业化的硬件上提供高性能和搞可靠性。可以让开发者将操作系统的功能和内存需求定制成每一个应用所需的系统。开发者可以利用它来实现从简单的单个独立设备到复杂的、网络化的多处理器系统。pSOS 的主要缺点在于其上下文切换时间长，实时性不强。

**QNX**：QNX 是一个实时的，可扩充的操作系统，它遵循 POSIX.1 (程序接口)和 POSIX.2 (Shell 和工具)、部分遵循 POSIX.1b(实时扩展)。它提供了一个很小的微内核以及一些可选的配合进程。其内核仅提供 4 种服务：进程调度、进程间通信、底层网络通信和中断处理，其进程在独立的地址空间运行。所有其它 OS 服务，都实现为协作的用户进程，因此 QNX 内核非常小巧(QNX4.x 大约为 12Kb)而且运行速度极快。这个灵活的结构可以使用户根据实际的需求将系统配置成微小的嵌入式操作系统或是包括几百个处理器的超级虚拟机操作系统。

**Palm OS**：3Com 公司的 Palm OS 在 PDA 市场上占有很大的市场份额，它有开放的操作系统应用程序接口（API），开发商可以根据需要自行开发所需要的应用程序。目前已经有总共 3500 多个应用程序可以运行在 Palm Pilot 上，其中大部分应用程序均为其他厂商和个人所开发，使得 Palm Pilot 的功能得以不断增多。这些软件包括计算器、各种游戏、电子宠物、地理信息等等。在开发环境方面，可以在 Windows 95/98，Windows NT 以及 Macintosh 下安装 Palm Pilot Desktop；PlamPilot 可以与流行的 PC 平台上的应用程序如 Word，Excel 等进行数据交换。

**OS-9**：Microwave 的 OS-9 是为微处理器的关键实时任务而设计的操作系统，广泛应用于高科技产品中，包括消费电子产品，工业自动化，无线通讯产品，医疗仪器，数字电视/多媒体设备中。它提供了很好的安全性和容错性。与其他的嵌入式系统相比，它的灵活性和可升级性非常突出。

**LynxOS**：Lynx Real-time Systems 的 LynxOS 是一个分布式、嵌入式、可规模扩展的实时操作系统，它遵循 POSIX.1a、POSIX.1b 和 POSIX.1c 标准。LynxOS 支持线程概念，提供 256 个全局用户线程优先级；提供一些传统的，非实时系统的服务特征；包括基于调用需求的虚拟内存，一个基于 Motif 的用户图形界面，与工业标准兼容的网络系统以及应用开发工具。

**EEOS**：EEOS 是中科院计算所组织开发的开放源码的嵌入式操作系统。该嵌入式操作系统重点支持 p-Java，要求一方面小型化，一方面能复用 Linux 的驱动和其他模块。计算所将在 2-3 年内持续投入，以建立扩展能力强，功能完善，且稳定、可靠的嵌入式操作系统平台。包含 E2 实时操作系统，E2 工具链及 E2 仿真开发环境的完整环境。E2 实时操作系统包含 RTOS 内核系统和一些支持组件，是一个扩展能力强，功能完善的操作系统，支持 POSIX 标准。E2 实时操作系统强调模块化（其 API 接口标准，底层支持清晰），其各模块解决方案可移植、可互换。此外 E2 实时操作系统提供开放源码。E2 工具链基于 GNU 系列的交叉编译环境，开放多平台，能够支持大多数硬件平台。E2 实时操作系统同时支持 MSDEV 等桌面环境的工具链，能够方便 WIN32 程序员使用。E2 仿真开发环境使用 Win32 系统仿真目标机或做远程调试的主机。E2 仿真开发环境能够仿真扩展的目标机设备或借

用 NT 系统的设备，并提供扩展工具进行性能，行为可视化，评估系统的性能。其仿真速度快，能够方便地容易建立开发环境。EEOS 采用先进的内核仿真调试技术，能够使普通 C 程序员能快速熟悉实时系统的结构和开发过程，大幅度降低学习，研究，开发实时系统的代价。

**HOPEN OS**：HOPEN OS 是北京凯思集团自主研制开发的嵌入式操作系统，由一个体积很小的内核及一些可以根据需要进行定制的系统模块组成。其内核 Hopen Kernel 一般为 10KB 左右大小，占用空间小，并具有实时、多任务、多线程的系统特征。使用者可以很容易地对这一操作系统进行定制或作适当开发，该系统可以广泛应用于：移动计算平台(PDA)、 家庭信息环境（机顶盒，数字电视）、通讯计算平台（媒体手机）、 车载计算平台（导航器）、 工业、商业控制（智能工控设备，POS/ATM 机）、 电子商务平台（智能卡应用，安全管理）。该系统目前已能在：X86、PA-RISC、Power PC、ARM、Strong ARM、MIPS、68XXX 等多种芯片上运行。

从国内外的嵌入式实时操作系统的发展状况来说，嵌入式时系统的应用正在从传统的单一应用范围、狭窄应用对象、简单实现功能向现在和未来社会需要的应用进行转变[18]。但是在众多已有的 RTOS 中，能够在任务管理、网络通信、文件管理等方面向嵌入式 Internet 应用提供全面支持的 RTOS 并不多。而且商业化的 RTOS 大多价格昂贵，在很大程度上也限制了它们在一些研究领域的使用。作为一个基于 32 位高性能处理器的新型嵌入式开发平台，不仅要求具有良好的实时内核，而且要提供网络协议栈和一定的文件管理能力。本文后面提出的嵌入式实时操作系统 RTEMS 就是为满足这样的需要而开发的。

## 3.3 实时操作系统的关键技术指标

评价一个实时操作系统一般可以从任务调度、内存管理、任务通讯、内存开销、任务切换时间、最大中断禁止时间等几个方面来衡量。

**1. 任务调度机制**

RTOS 的实时性和多任务能力在很大程度上取决于它的任务调度机制[15]。从调度策略上来讲,分优先级调度策略和时间片轮转调度策略；从调度方式上来讲,分可抢占、不可抢占、选择可抢占调度方式；从时间片来看,分固定与可变时间片轮转。

**2. 上下文切换时间**

在多任务系统中，上下文切换指的是当处理器的控制权由运行任务转移到另外一个就绪任务是所发生的事件序列。当运行的任务转为就绪、挂起、或删除时，另外一个被选定的就绪任务就成为当前运行任务，上下文切换包括保存当前任务的状态，决定哪个任务运行，恢复将要运行的哪个任务的状态，保护和恢复上下

文是依赖相关处理器的。因此，上下文切换事件是影响 RTOS 性能的一个重要指标。

**3.** 内存管理

内存管理分实模式与保护模式[7]。从表面上看，保护模式和实模式并没有太大的区别，二者都使用了内存段、中断和设备驱动来处理硬件，但二者有很多不同之处。我们知道，在实模式中内存被划分成段，每个段的大小为 64KB，而这样的段地址可以用 16 位来表示。内存段的处理是通过和段寄存器相关联的内部机制来处理的，这些段寄存器（CS、DS、SS 和 ES）的内容形成了物理地址的一部分。具体来说，最终的物理地址是由 16 位的段地址和 16 位的段内偏移地址组成的。用公式表示为： 物理地址=左移 4 位的段地址+偏移地址。

在保护模式下，段是通过一系列被称之为“描述符表”的表所定义的，段寄存器存储的是指向这些表的指针。在实模式中，段长是固定的(为 64KB)，而在保护模式中，段长是可变的，其最大可达 4GB。

**4.** 最小内存开销

RTOS 的设计过程中,最小内存开销是一个较重要的指标,这是因为在工业控制领域中的某些工控机（如上下位机控制系统中的下位机），由于基于降低成本的考虑，其内存的配置一般都不大，例如康拓 5000 系列 5185 板[19]，其基本内存配置仅为 256K SRAM+128K EEPROM，而在这有限的空间内不仅要装载实时操作系统，还要装载用户程序。因此，在 RTOS 的设计中，其占用内存大小是一个很重要的指标，这是 RTOS 设计与其它操作系统设计的明显区别之一。

**5.** 最大中断延迟时间

当 RTOS 运行在内核态或执行某些系统调用的时候，是不会因为外部中断的到来而中断执行的。只有当 RTOS 重新回到用户态时才响应外部中断请求，这一过程所需的最大时间就是最大中断延迟时间。

**6.** 任务切换时间

当由于某种原因使一个任务退出运行时，RTOS 保存它的运行现场信息、插入相应队列、并依据一定的调度算法重新选择一个任务使之投入运行，这一过程所需时间称为任务切换时间。

上述几项中,最大中断延迟时间和任务切换时间是评价一个 RTOS 实时性最重要的两个技术指标。

# 第四章 RTEMS 操作系统

在初步决定选用 RTEMS 后，本课题对 RTEMS 又进行了进一步详细的分析和研究，包括对 RTEMS 内核的 17 个主要管理器的详细分析。本章将探讨本课题对 RTEMS 做的研究和分析，重点是 RTEMS 的内核管理器。

## 4.1 选择 RTEMS 的原因及其历史，特点

**选择 RTEMS 的原因**

现代控制系统通常建立在 RTOS 上，因为 RTOS 可以对底层的硬件提供足够的抽象，例如调度，网络和其它的一些服务。那些开放源代码的 RTOS 对开发者是十分有吸引力的[17]，因为从经济角度开可以节省许可费这样的开销，而从技术方面看，开放源代码就意味着你可以对它进行你想要的控制。本课题选用的 RTEMS 正是这样一个开放开放源代码的嵌入式 RTOS。

上一章本文已经分析过最大中断延迟时间和任务切换时间是评价一个 RTOS 实时性最重要的两个技术指标。实验显示[5]，在衡量实时性的两个主要指标中断响应时间和任务切换时间的比较中，RTEMS 的实时性能在任务过载的情况下可以和业界著名的嵌入式开发平台 VxWORKS 相媲美，而 RTlinux 却比它们慢了一个数量级。这是本课题选用 RTEMS 的第二个重要原因。

**RTEMS 的历史**

RTEMS（Real-Time Executive for Multiprocessor Systems）最初是美国军方使用的实时导弹系统。最初的 M 代表 missiles。但是随着 RTEMS 的发展，它已不仅仅在导弹中应用，M 就代表了 Military[3]。后来他是支持多处理器的 C 语言的应用就把 M 代表了 Multiprocessor, Ada 语言应用仍然保持着 Military 的意思。RTEMS 也是开放源代码的 RTOS, 可以在 http://www.OARcorp.com 下载到源代码及相关的技术信息。它具有十分强大的功能，现简单介绍。

**RTEMS 建立标准**

RTEMS 支持多种 API,包括 POSIX1003.1b, ITRON 和 RTEID 用户可以根据自己的需要进行选择。同时支持 C,C++,Java,Ada95 多种语言。

**RTEMS 支持的硬件系列**

RTEMS 支持 13 个 CPU 系列(如 Intel i80386, Motorola MC68xxx, PowerPC 等)。同时支持 60 多种的 BSP,并且数量还在不断增加。

**RTEMS 主要的特征**

- 以优先级为基础的多任务实时系统。
- 任务调度可以优先级抢占控制。
- 支持多任务的通信和同步。
- 支持中断管理。
- 支持动态内存分配。
- 支持多处理器。
- 支持速率单调调度算法(RMS)，可以保证任务的硬实时性。
- 支持标准 Unix 文件系统
- 支持多种网络协议：IP, TCP, UDP, TFTP, FTP, Dialer, NTP, ICMP, ARP, DNS, dient, HTTP, PPP, SNMP。具有强大的网络功能。

## 4.2 RTEMS 简介

**RTEMS 的体系结构**

RTEMS 一个主要的设计目标就是为实时系统的两个层提供一个连接的桥梁。如图 4.1 所示，RTEMS 就是应用程序和目标硬件之间的一个缓冲区。上层是标准应用部分，下层是设备驱动及跟硬件系统对应的板极支持包（BSP）。

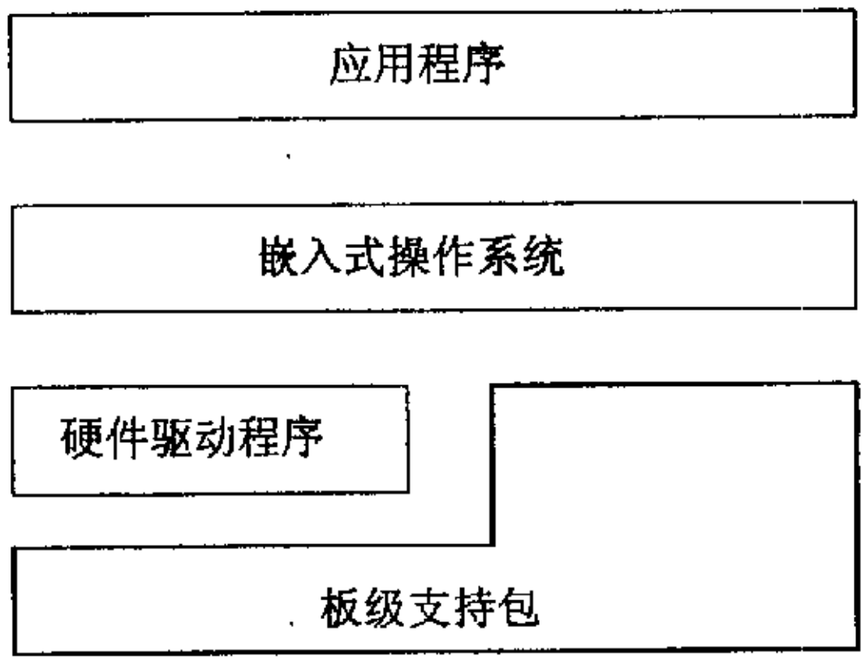


图 4.1 RTEMS 体系结构

Figure 4.1 Architecture of RTEMS

## 4.3 RTEMS 的内部体系结构及各模块功能

RTEMS 可以看成是一系列不同层面的元素，它们互相和谐的工作，为实时的应用系统提供各种服务。各种代表应用的执行接口把 RTEMS 的命令分成了一些组，每个组完成一些特定的服务，我们统称其为管理器。由多个管理器所使用的函数被包含在执行内核中。执行内核依赖一个很小的 CPU 依赖子程序。所有的这些元素组合起来就为开发一个高效的实时应用系统提供了一个强有力的实时环

境。具体结构如图 4.2 所示。

RTEMS 共有 17 个管理器，分别是初始化（initialization）,任务（task），时钟（clock）,计时器（timer）,信号量（semaphore）,消息（message）,事件（event），信号（signal）,分区（partition）,区域（region）,双端口内存（dual ported memory）,I/O,致命错误（fatal error），速率单调（rate monotonic）,用户扩展（user extensions），多处理器（multiprocessing）[2]。其中，初始化，任务是必须的，其他的可以根据用户的需要及实际情况可以适当剪裁。下面将会对这些管理器做逐一介绍。

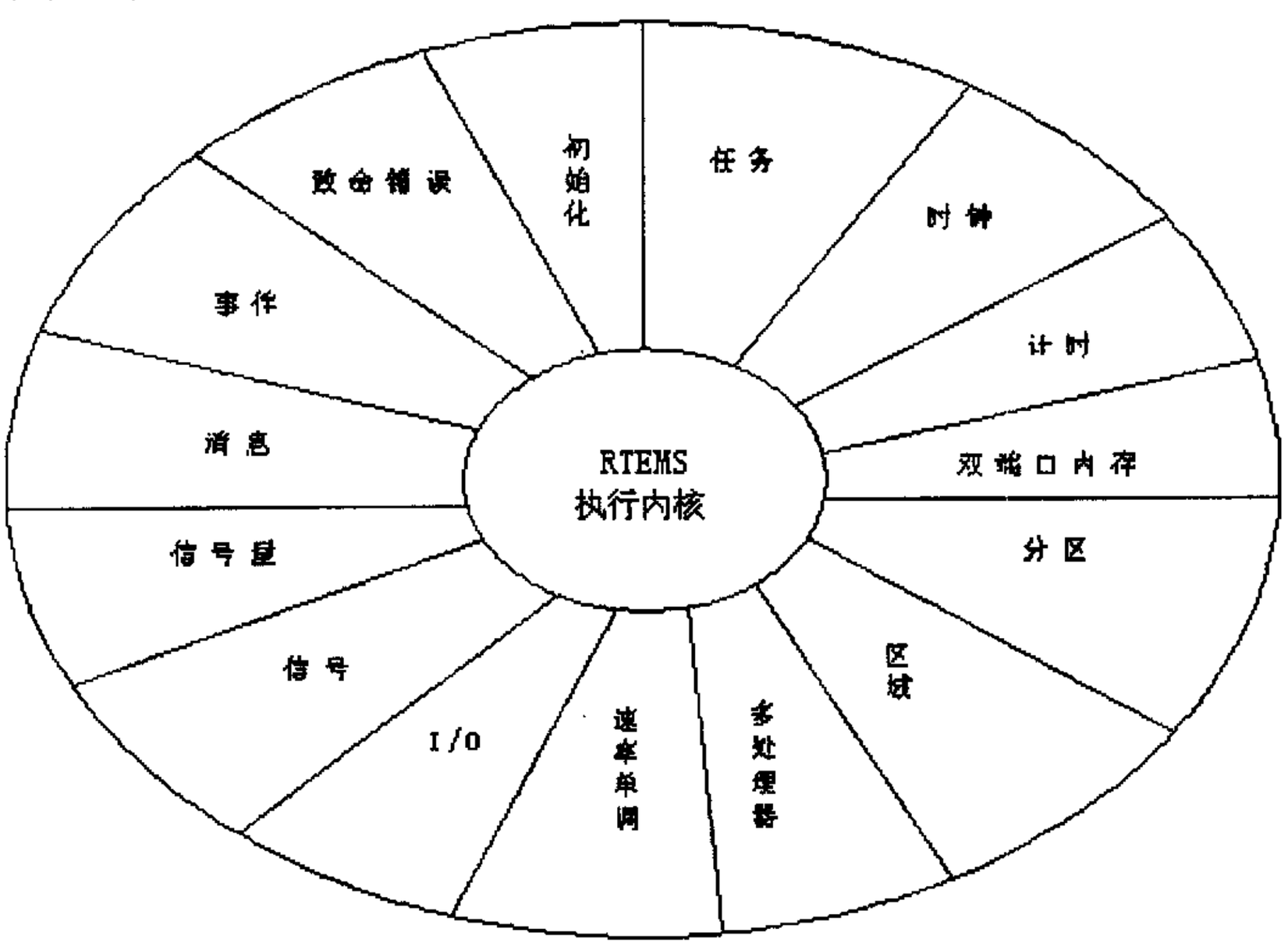


图 4.2 RTEMS 内部体系结构

Figure 4.2 Interior Architecture of RTEMS

### 4.3.1 RTEMS 中的一些重要概念

1) 对象

RTEMS 是面向对象的，这样有利于它的模块化应用。可以动态的创建，删除，和操作一些预先定义的对象类型。包括：任务，消息队列，信号量，内存区域，内存分块，计时器，端口和速率单调周期等。每一个对象，都有一个名字和 ID,名字是用户任意取的，通常是能表示这个对象意义的符号。ID 是 RTEMS 给分配的，唯一的，不可改变的，在对这个对象的操作中使用。ID 是 32 位无符号数。通常有三部分组成：如图 4.3 所示

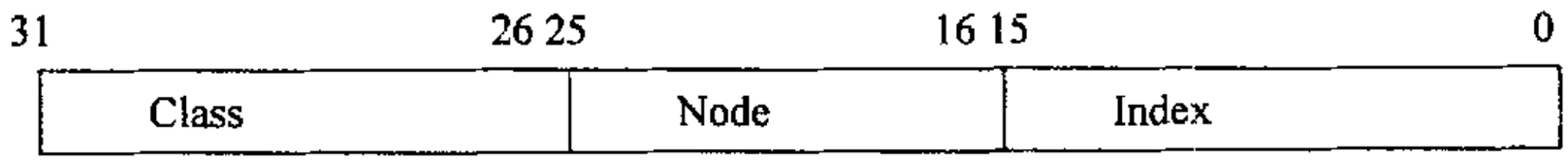


图 4.3 ID 结构

Figure 4.3 Sturcture of ID

其中，高 6 位为 Class 项，表示该对象的类型。中间 10 位为 Node 项，指的是该对象所在节点的号。低 16 位为 Index 项，代表这个对象在他所在的对象类中的索引。这三部分组合起来，使得在一个复杂的多处理器的系统中也可以很快的找到指定的对象。

**2) 通信和同步**

在实时多任务应用中，执行线程中通信和同步是必须的。对一个实时的应用应该具有以下能力：

- 任务之间传递数据
- 任务和中断服务程序之间传递数据
- 任务之间的同步
- 任务和中断服务程序之间同步

RTEMS 中有几种管理机制可以提供通信和同步。但是不同的机制可以提供的通信和同步的级别不同，可以根据用户和应用的需要，选择适当的机制。这几种机制包括：

- 信号量（Semaphore）
- 消息队列（Message Queue）
- 事件（Event）
- 信号（Signal）

其中，信号量支持对共享资源的互斥和同步存取。消息队列既支持通信又支持同步。而事件机制主要提供很好的同步机制。信号管理支持异步通信，通常在意外处理中使用。

**3) 内存管理**

RTEMS 的内存管理可以分为两类：动态内存分配和地址转换。动态内存分配应用于那些在执行过程中内存需求有变化的应用中。内存转换应用于那些于其他的 CPU 或 Input/Output 处理器共享内存的应用中。

RTEMS 中管理内存的主要有 3 个管理器：

- 区域（Region）
- 分区（Partition）
- 双端口内存（Dual Ported Memory）

其中，分区管理可以管理和维持固定大小的缓冲区。区域可以管理不同大小

的内存块，并且这些内存块可以被应用动态的获取和释放。双端口内存管理为内部和外部的端口之间 RAM 地址空间的地址转换提供支持。

### 4.3.2 RTEMS 中各管理模块功能及实现

了解了 RTEMS 的总体体系结构及一些重要概念后，下面介绍一下 RTEMS 各管理模块的功能，主要命令及操作。

#### 4.3.2.1 初始化管理（Initialization Manager）

初始化管理主要是用来初始化和关闭 RTEMS 用的。初始化包括创建和开始所有的被配置的初始化任务。激发每一个用户提供的设备驱动的初始化程序。在多处理器的配置下，这个管理还要初始化处理器之间的通信层。提供的主要命令有：

- rtems_initialize_executive – 初始化 RTEMS
- rtems_initialize_executive_early – 初始化 RTEMS 但不开始多任务
- rtems_initialize_executive_late – 完成初始化并且开始多任务
- rtems_shutdown_executive – 关闭 RTEMS

初始化任务与其它的一般用户任务的不同就是他们是在用户初始化任务表中定义并在 RTEMS 初始化的过程中创建并开始的。但是它跟其它的任务都是采取同样的调度机制，所以它必须也要配置一个优先级及执行模式（系统初始化任务通常配置的优先级最高），保证它在其它的用户任务执行前执行。初始化任务如果只有在初始化中使用时，它可以在完成自己的任务时把自己删除，为其它的任务释放更多的空间。初始化任务也可以转化成一般任务，这种转化通常要包括改变优先级和执行模式。RTEMS 不能自动删除初始化任务。在多处理器的配置中，系统初始化任务就不会删除自己，而是，转换成多处理器服务任务来初始化多处理器通信接口层，来保证系统的一致性，处理各个远节点的请求。

初始化管理的主要操作：

■ 初始化 RTEMS

当板级支持包(BSP)完成了自身的初始化后就调用 rtems_initialize_executive 命令进行系统初始化，过程如图 4.4。在初始化过程中所进行的活动，是根据系统的配置表（configuration table）和 CPU 信息表（CPU Dependent Information Table）的内容来进行的。当进行到最后一步初始化多任务，并且使得多任务变成 enabled 后，控制就不会回到 BSP 中，直到 rtems_shutdown_executive 被调用 rtems_initialize_executive 可以用 rtems_initialize_executive_early 和 rtems_initialize_executive_late 这两个命令的组合来代替。

■ 关闭 RTEMS

调用 rtems_shutdown_executive 命令就可以把 RTEMS 关闭，并且将控制权交

给了 BSP。

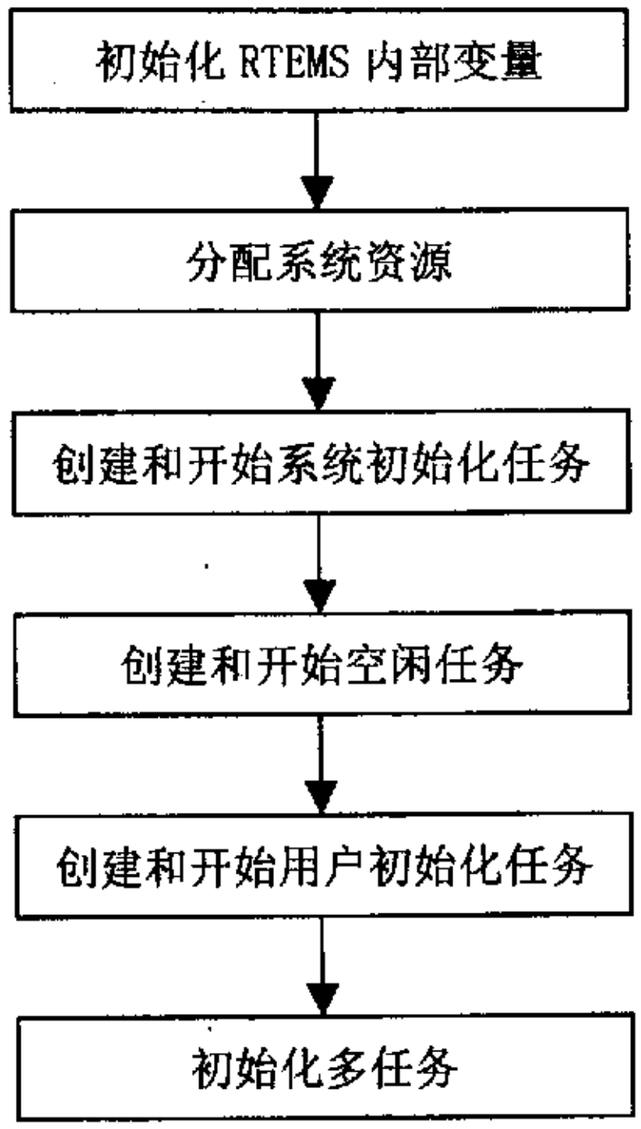


图 4.4 RTEMS 初始化流程

Figure 4.4 Initialization Flow of RTEMS

#### 4.3.2.2 任务管理（Task Manager）

任务管理提供任务的创建，删除和管理命令。这些命令包括：

- rtems_task_create – 创建一个任务
- rtems_task_ident – 获得任务的 ID
- rtems_task_start – 开始一个任务
- rtems_task_restart – 重新开始一个任务
- rtems_task_delete – 删除任务
- rtems_task_suspend – 挂起一个任务
- rtems_task_resume – 恢复一个任务
- rtems_task_is_suspend – 判断一个任务是否是挂起的
- rtems_task_set_priority – 设定任务的优先级
- rtems_task_mode – 改变当前任务的模式
- rtems_task_get_note – 获取任务的记事本入口
- rtems_task_set_note – 设置任务的记事本入口
- rtems_task_wake_after – 一段时间后唤醒任务
- rtems_task_wake_when – 在特定的时间唤醒任务
- rtems_task_variable_add – 加入一个任务变量

- rtems_task_variable_get – 获取一个任务变量的值
- rtems_task_variable_delete – 删除一个任务变量

从 RTEMS 的观点来看，任务就是获取系统资源的最小执行线程。每个任务都已有一个任务控制模块（TCB），它包含了一个任务的所有执行的信息.在系统初始化的过程中,RTEMS 就为每一个已经配置的任务保留一个 TCB。当创建一个新任务时,TCB 被分配。删除一个任务时，就释放 TCB,返回到 TCB 空闲表中。TCB 中记录了任务的名字，ID,当前优先级，当前和开始状态，执行模式，记事本地址集，TCB 用户扩展指针，调度控制结构，和一个阻塞任务需要的数据等。

任务有 5 种状态：执行，就绪，阻塞，睡眠，和不存在。其中前四种为活动状态[21]。

任务的优先级：RTEMS 支持 255 个优先级，从 1 到 255，每个任务都有一个优先级。数字越小，优先级越高。同一个优先级可以分给多个任务。一个任务的优先级是可以改变的。优先级是用来帮助调度算法来调度任务的，优先级高的任务，更有可能得到处理器的执行时间

任务的执行模式：一个任务的执行模式由四个元素组成，分别是

- 优先级可以抢占（preemption）
- 异步信号处理（ASR processing）
- 时间片（timeslicing）
- 中断级别（interrupt level）

执行模式可以用来改变 RTEMS 的调度进程,改变任务的执行环境。

preemption 元素决定任务何时放弃对处理器的控制权.如果 preemption 被设为 disabled(RTEMS_NO_PREEMPT), 只要任务处于执行状态，他就不会释放对处理器的控制权，直到执行结束，即使是有一个更高优先级的任务处于就绪状态。如果 preemption 设为 enabled(RTEMS_PREEMPT), 当有更高优先级的任务就绪时，处理器收回当前任务的使用权，并且交给高优先级的任务.

timeslicing 元素来决定当任务是同优先级时是如何分配处理器的。如果 timeslicing 设为 enalbed(RTEMS_TIMESLICE) , RTEMS 就会限制每个任务的执行时间, 超过了执行时间就分给下一个任务。时间片的长度是由应用决定的,在配置表中定义.如果 timeslicing 设为 disabled(RTEMS_NO_TIMERSLICE), 这个任务就一直执行直到有更高的优先级的任务处于就绪状态。如果，RTEMS_NO_PREEMPT 被选中，timeslicing 元素就被忽略。

ASR processing 用来决定接收到的信号何时被任务处理。如果信号处理设为 enabled(RTEMS_ASR), 信号将会在这个任务的下一次执行时被处理。如果信号处理设为 disabled(RTEMS_NO_ASR),任务接收到的所有信号都保持挂起状态，直到

信号处理被使能。这个元素只对那些建立了处理异步信号程序的任务起作用。

Interrupt level 元素用来决定当任务执行时，哪级中断可以使能。RTEMS_INTERRUPT_LEVEL(n)定义了任务执行在中断级别 n 上。

任务管理的主要操作：

■ 创建任务

通过 rtems_task_create 命令来创建任务。系统给任务分配任务控制模块（TCB）和一个唯一的 ID。新创建的任务处于睡眠状态。

■ 获取任务的 ID

通过 rtems_task_ident 命令来获取任务的 ID。这个 ID 可以在调用其他的命令时使用。

■ 开始和重新开始任务

调用 rtems_task_start 命令可以使睡眠状态下的任务处于就绪状态。调用 rtems_task_restart 命令可以 restart 一个已经 started 的任务。虽然两次的任务的起始地址，最初优先级和执行模式相同，但是两次的参数不同，以区别两次不同的启动。所有被 restart 的任务，都处于就绪状态。

■ 挂起和恢复一个任务

rtems_task_suspend 和 rtems_task_resume 是一对命令。前者可以使任务挂起，通过后者，已经挂起的任务可以解除挂起状态。

■ 推迟正在执行的任务

通过 rtems_task_wake_after 命令，使得一个任务睡眠一段时间，这段时间过后，任务重新被唤醒。rtems_task_wake_when 命令，使得一个任务睡眠到一个指定的时间，到指定的时间时，就被唤醒了。

■ 改变任务的优先级

通过 rtems_task_set_priority 命令可以改变任务的优先级。但是当 rtems_task_restart 命令对任务重新开始时，任务的优先级复位到初始值，原来的改变全部清空。

■ 改变任务的执行模式

rtems_task_mode 可以用来获得和改变任务的执行模式。同样，通过 rtems_task_restart 命令重新启动任务后，可以复位任务的初始模式。

■ 记事本地址

RTEMS 为每一个任务提供了 16 个记事本区域。每个记事本区域包含了 4 字节的笔记信息。rtems_task_set_note 可以使使用者设置任务的一个特定笔记的记事本的入口地址。rtems_task_get_note 可以使使用者获得特定任务的 16 个笔记本中的任何一个笔记。

■ 任务删除

调用 rtems_task_delete 可以把任务删除，同时跟这个任务相关的资源也被释放。但是，一些动态分配给这个任务的资源不能自动返回给 RTEMS。因此，在任务被删除前，使用者先把那些动态分配的资源释放掉，再删除任务。

#### 4.3.2.3 中断管理（Interrupt Manager）

在任何的实时系统中，为了满足应用的实时性要求，对外部产生的中断的快速应答机制是必不可少的。中断管理就是 RTEMS 提供这一机制的模块。它主要的命令有：

- rtems_interrupt_catch – 建立一个中断服务子程序（ISR）
- rtems_interrupt_disable – 关中断
- rtems_interrupt_enable – 开中断
- rtems_interrupt_flash – 闪中断
- rtems_interrupt_is_in_progress – 中断是否在进行

RTEMS 的中断级别为 256 级,从 0 到 255。级别越低,优先级越高。高优先级的中断可以可以中断低优先级的中断。因此，中断是可以嵌套的。当执行关键区的代码时,要屏蔽所有的可屏蔽中断。然后,在执行完了之后,再恢复中断。如果，在执行关键区的代码时，产生了不可屏蔽中断（NMI）,因为系统不能保护关键区，就会产生不可预知的结果。

中断管理的主要操作：

■ 建立一个 ISR

通过 rtems_interrupt_catch 可以为系统建立一个 ISR。并把用户的 ISR 的地址写到 RTEMS 的向量表中。

■ ISR 可以调用的系统指令

ISR 要调用系统指令来同步中断与应用任务。这个同步包括 ISR 传递给目标任务的消息，事件，信号等。ISR 可以调用的 RTEMS 系统指令包括：

- 任务管理
- task_get_note, task_set_note, task_suspend, task_resume
- 时钟管理
- clock_get, clock_tick
- 消息，事件，信号管理
- message_queue_send, message_queue_urgent
- event_send
- signal_send
- 信号量管理

- semaphore_release
- 双端口管理
- port_external_to_interal, port_internal_to_external
- I/O 管理
- io_initialize, io_open, io_close, io_read, io_write, io_control
- 致命错误管理
- fatal_error_occurred
- 多处理器
- multiprocessing_announce

#### 4.3.2.4 时钟管理（Timer Manager）

时钟管理提供了跟时间相关的一些系统能力的支持。包括 3 条指令：

- rtems_clock_set – 设置系统的日期和时间
- rtems_clock_get – 获取当前系统日期和时间信息
- rtems_clock_tick – 声明一个系统时钟滴答(tick)

RTEMS 系统的时间的基本单位是一个滴答（tick）。它的长度由用户在系统配置表中定义。时钟管理的主要操作是：

■ 声明一个滴答（tick）

用户的实时时钟 ISR 调用命令 rtems_clock_tick 通知 RTEMS 已经逝去一个 tick。RTEMS 通过一秒钟发生的 tick 数来决定一秒钟调用 rtems_clock_tick 命令的数量。该命令调用的频率决定了系统实时应用的刷新率。

■ 设置系统时间

通过 rtems_clock_set 命令可以设置系统的时间。它通常跟计时器（timer）项关联。

■ 获取系统时间

通过 rtems_clock_get 命令可以获取当前系统的时间。但是要注意，必须先通过 rtems_clock_set 设置，然后再用 rtems_clock_get 获取,否则，将会返回 RTEMS_NOT_DEFINED 状态。

#### 4.3.2.5 计时器管理（Timer Manager）

计时器管理用来支持计时设备的。它包括以下命令：

- rtems_timer_create – 创建一个计时器
- rtems_timer_ident – 获取一个计时器的 ID
- rtems_timer_cancel – 取消一个计时器
- rtems_timer_delete – 删除一个计时器
- rtems_timer_fire_after – 在一段时间后点燃计时器

- rtems_timer_fire_when – 在特定的时间点燃计时器
- rtems_timer_reset – 计时器复位

计时器是RTEMS的一个对象。用来调度一段特定的程序(timer service routine)在一段特定的时间后(这段时间由计时器决定)执行。看门狗就是一种特殊的计时器。

主要的操作：

■ 创建一个计时器

通过 rtems_timer_create 命令可以创建一个计时器。系统还会为这个计时器分配一个计时器控制块（TMCB）来管理这个新创建的计时器。新创建的计时器没有相关联的计时器服务程序，是不活跃的。

■ 获取计时器 ID

当一个计时器被创建了之后，RTEMS 就会给他分配一个唯一的 ID。通过 rtems_timer_ident 命令可以获得这个计时器的 ID,以便在调用其他的计时器命令中使用。

■ 初识化一个间隔计时器

rtems_timer_fire_after 命令可以初始化一个计时器，使得它在一段时间过后点燃一个计时器服务子程序。当间隔逝去后，计时器服务子程序就被 rtems_clock_tick 命令激发。

■ 初始化一个时间计时器

rtems_timer_fire_when 命令可以初始化一个计时器，使得它在一个特定的时间点燃一个计时器服务子程序。当间隔逝去后，计时器服务子程序就被 rtems_clock_tick 命令激发。

■ 取消计时器

rtems_timer_cancel 指令可以停止一个特定的计时器。一旦计时器被取消了，计时器服务子程序就不会被点燃，直到计时器被重新初始化。命令 rtems_timer_reset, rtems_timer_fire_after, rtems_timer_fire_when 都可以重新初始化计时器。

■ 计时器复位

rtems_timer_reset 命令把一个被 rtems_timer_fire_after 初始化的计时器恢复到原来的时间间隔。

■ 删除计时器

rtems_timer_delete 命令用来删除一个计时器。如果这个计时器正在运行，并且没有过期，这个计时器就会自动取消。TMCB 也会被回收，返回到 TMCB 空闲表中。

#### 4.3.2.6 信号量管理（Semaphore Manager）

信号量管理应用标准的 Dijkstra 计数信号量来进行同步和互斥操作。主要的命令有：

- rtems_semaphore_create – 创建一个信号量
- rtems_semaphore_ident – 获取信号量的 ID
- rtems_semaphore_delete – 闪出一个信号量
- rtems_semaphore_obtain – 获得信号量
- rtems_semaphore_release – 释放信号量
- rtems_semaphore_flush – 解除所有等待该信号量的任务的阻塞

RTEMS 支持二元信号量和多元信号量。二元信号量可以取 0，1 两个值，多元信号量可以是任何非负的整数。二元信号量是控制单个资源的互斥访问。多元信号量是控制资源集的访问。

同时也可以通过信号量来实现任务之间的同步。首先设信号量的初值为 0，任务 A 到达了同步点后，需要等待任务 B,于是就在同步点发出 rtems_semaphore_obtain 指令。任务 B 到达同步点后，发出了一个相应的 rtems_semaphore_release 指令，任务 A 收到了该指令,继续执行，达到了同步。

下面介绍几个竞争共享资源的几个相关概念:

■ 优先权倒置

高优先级的任务要对共享资源读取时,资源被低优先级任务占用,直到低优先级的任务完成后,高优先级才能读取共享资源的现象。( 比如: 低优先级占有了共享资源,但是他却由于中间优先级的任务占用 CPU 而不能执行,由于不能执行,也就不能释放资源,高优先级的任务得不到资源,也只能处于挂起状态。) 这种现象是要避免的。

■ 优先级继承

它是一种避免优先级倒置的算法。把占用资源的低优先级的任务的优先级增加到当前等待这个资源的任务中的最高的优先级。当执行完成后,释放资源,并把优先级恢复成原来。RTEMS 支持这种算法。

■ 优先级置顶

把占用资源的低优先级任务提高到等待和将要等待这个资源的所有任务中的最高优先级。当执行完成后,释放资源,并把优先级恢复成原来。虽然这个算法可以避免多次改变优先级，但是由于计算所有任务的最高级很难实现，所以通常采用优先级继承算法。

信号量管理器的主要操作：

■ 创建一个信号量

rtems_semaphore_create 命令就可以创建一个二元或是多元信号量。如果一个

二元信号量创建的初值为 0，则表示资源已经被占用了，创建这个信号量的任务被看成是这个信号量的拥有者。并且在创建时，等待这个信号量的任务队列的排列顺序也确定了（是按 FIFO 还是按任务优先级）。同时，优先级继承还是优先级置顶算法也被选择。RTEMS 也会给每个新创建的信号量分配一个信号量控制块（SMCB）。

■ 获取信号量的 ID

当一个信号量被创建了之后，RTEMS 就会给他分配一个唯一的 ID。通过 rtems_semaphore_ident 命令可以获得这个信号量的 ID,以便在调用其他的信号量命令中使用。

■ 获得信号量

rtems_semaphore_obtain 命令用来获得指定的信号量。算法如下：

if semaphore's count is greater than zero

then decrement semaphore's count

else wait for release of semaphore

return SUCCESSFUL

如果不能立即获得信号量,通常有三种处理情况:

缺省情况下，任务将会一直等待直到得到信号量。

如果属性为 RTEMS_NO_WAIT，则立即错误 error 码，不进行任何等待。

设置一个等待时间极限，任务等待的时间超过了这个极限，就返回 error 码，不再等待。

■ 释放信号量

rtems_semaphore_release 命令来释放指定的信号量。算法如下：

if no tasks are waiting on this semaphore

then increment semaphore's count

else assign semaphore to a waiting task

return SUCCESSFUL

■ 删除一个信号量

rtems_semaphore_delete 命令从系统中删除一个信号量。SMCB 将会被回收。所有该信号量等待队列中的任务都被返回一个码表示该信号量被删除了。

#### 4.3.2.7 消息管理器（Message Manager）

消息管理器通过 RTEMS 消息队列来提供通信和同步机制。主要的命令有：

- rtems_message_queue_create – 创建一个队列
- rtems_message_queue_ident – 获得队列的 ID
- rtems_message_queue_delete – 删除一个队列

- rtems_message_queue_send – 把消息放在队列的尾部
- rtems_message_queue_urgent – 把消息放在队列的首部
- rtems_message_queue_broadcast – 广播消息
- rtems_message_queue_receive – 从队列中接收消息
- rtems_message_queue_get_number_pending – 获取消息队列消息数量
- rtems_message_queue_flush – 除去消息队列中所有的消息

消息是用来储存信息的变长缓冲区，用来支持任务间通信的。消息的长度及储存在消息中的信息由用户定义，可以是实际的数据，指针，或者为空。

消息队列是有多条消息组成，通常消息队列的发送和接收是按 FIFO 的原则进行的。

主要的操作：

■ 创建一个消息队列

rtems_message_queue_create 命令创建一个消息队列。用户自定义名字，队列的最大长度和最大消息数。用户可以选择 FIFO,还是任务优先级方式作为任务等待队列的管理机制。RTEMS 为消息队列分配一个队列控制块（QCB）来管理消息队列。

■ 获取消息队列的 ID

一个消息队列创建后，ETEMS 就会给每个消息队列分配一个唯一的 ID,通过 rtems_message_queue_ident 来获取。以便在调用其他的消息队列命令中使用。

■ 接收消息

用 rtems_message_queue_receive 来接收特定消息队列里的消息。如果消息队列里存在消息，就取出消息，复制到调用者的消息缓冲区里，并返回消息的长度。如果消息队列里没有消息,下列 3 种处理情况：

缺省情况下，调用者任务一直等待直到消息队列有新消息。

如果属性为 RTEMS_NO_WAIT, 则不等待，立即返回一个 error 码。

设置一个等待时间极限，如果在这个极限时间内还没有得到消息，就不等待了，返回 error 码。

如果一个任务要等待一个消息，它就进入消息队列的任务等待队列。在队列中的位置按 FIFO 或优先权方式决定。

■ 发送一个消息

通过 rtems_message_queue_send 和 rtems_message_queue_urgent 来发送一个消息。如果没有任务等待时，这两个命令就会把消息发送到消息队列中。两者的不同就在于，前者把消息放在消息队列的最后方。后者把消息放在消息队列的最前方。

■ 广播一个消息

rtems_message_queue_broadcast 命令发送同一个消息给每一个在任务等待队列里的任务。每个任务的消息缓冲区里都得到了消息，每个任务就被激活。被激活任务的数量返回给调用者。

■ 删除一个队列

rtems_message_queue_delete 命令从系统中删除一个消息队列。它对应的 QCB 及跟它有关的内存缓冲区都被释放了。这个队列中的消息会返回到系统消息缓冲区。

#### 4.3.2.8 事件管理（Event Manager）

事件管理为任务间的通信和同步提供了一个很好的方法。主要有两个命令：

- rtems_event_send – 发送一个事件
- rtems_event_receive – 接收事件

任务通过事件标志（Event Flag）来通知另一个事件一个重要状况的发生。一个事件与 32 个事件标志相联系，一个或几个事件标志组合起来就构成了一个事件集（event set）。要注意事件有如下的特征:

事件提供了同步的机制

事件的目标是任务

一个任务可以同时等待多个事件

事件之间是独立的

事件不保持和传递数据

事件不能够排队（ 也就是说同一个事件在没被接收时发送给同一个任务多次，除了第一个事件，后面的都没有作用。）

一个事件集（event set）发送给了一个任务，这个 event set 就是 posted,但是如果它没被任务接收，它就是 pending 的。事件条件是否被满足有两种决定算法:

RTEMS_EVENT_ANY,只要一个要求的事件 posted,就满足了。

RTEMS_EVENT_ALL, 所有的要求的事件都 posted 才能满足。

主要的操作：

■ 发送一个事件

rtems_event_send 指令使得一个任务发送一个事件给一个目标任务。根据目标任务的状态有两种情况: a 目标任务正在等待一个事件的发生而处于阻塞状态。如果这个发送的事件满足了等待的任务的事件条件，等待任务就变为就绪状态。如果这个等待的任务的事件条件没被满足，这个事件是 pending 的，等待的任务仍处于阻塞状态。b 目标任务没有等待事件 这个事件就处于 pending。

■ 接收一个事件

rtems_event_receive 命令用来使得一个任务接收一个特定的事件。如果这个任务被接收的事件所满足，一个 successful 返回码立即返回。如果任务的事件条件没有被满足，会出现 3 种情况：

缺省条件下，任务将会一直等待直到事件条件满足。

如果属性选为 RTEMS_NO_WAIT, 任务不等待，立即返回 error 码。

设定一个等待时间极限，如果等待的时间超过了极限，就不等待了，返回一个 error 码。

#### 4.3.2.9 信号管理（Signal Manager）

信号管理是用来提供异步通信机制的。主要的命令有：

- rtems_signal_catch – 建立一个 ASR
- rtems_signal_send – 给一个任务发送信号

ASR(Asynchronous Signal Routine)异步信号子程序，类似于一个应用的 ISR。当一个信号发送给一个任务时，这个任务的执行就会被 ASR 中断。任务都有信号标志（signal flag）,来通知另一个任务重要状况的发生。每个任务都跟 32 个信号标志相联系。从 RTEMS_SINGNAL_0 到 RTEMS_SINGNAL_31。一个或多个信号标志组成一个信号集（signal set）。一个 signal set 发送给一个任务,它就是 posted，但是没有被处理，这个 signal set 就是 pending 的。

虽然 ASR 跟 ISR 很相似,但是也有不同,主要表现为:

ISR 是硬件处理器调度的；ASR 是 RTEMS 调度的。

ISR 不在任务的上下文中执行；ASR 在任务的上下文中执行。

ISR 被激活时，中断向量号作为参数传递；ASR 被激活时，信号集作为参数传递。

ASR 有任务的执行模式，并且这个模式可以与当前的任务不同；ISR 不像任务一样执行，所以，就没有任务的执行模式。

主要的操作：

■ 建立一个 ASR

使用 rtems_signal_catch 命令可以为调用者任务建立一个 ASR。ASR 的地址和执行模式在这个命令中定义。并且它的执行模式跟任务的模式是可以不同的。一个任务可以通过设置 rtems_signal_catch 命令中的 ASR 的地址为 NULL 来使 ASR 无效，并且舍弃该任务的所有 pending 信号。任务也可以通过设置任务的执行模式为 RTEMS_NO_ASR 来使 ASR 无效。

■ 发送一个信号

使用 rtems_signal_send 命令可以使任务和 ISR 发送信号给一个目标任务。发送一个信号给任务不会影响任务当前的状态。如果这个任务当前不是执行的任务，

那么信号先保持 pending 状态。当任务被执行时，信号才会被 ASR 来处理。同 ISR, ASR 也是可以嵌套的。

■ 处理 ASR

ASR 具有一些实现软件中断的能力。软件中断和硬件中断是并行处理的。当 ASR 返回到 RTEMS, 被中断任务的执行模式,执行路径恢复到进入 ASR 之前的情况。

#### 4.3.2.10 分区管理（Partition Manager）

分区管理是提供在固定大小的单元里动态分配内存的管理。主要提供的命令有：

- rtems_partition_create – 创建一个分区
- rtems_partition_ident – 获取分区的 ID
- rtems_partition_delete – 删除一个分区
- rtems_partition_get_buffer – 从分区中获取一个缓冲区
- rtems_partition_return_buffer – 把缓冲区返回到分区中

一个分区就是把物理上邻接的内存区域划分成固定大小的缓冲区，并且这些缓冲区可以被动态的分配和回收。分区是以缓冲区链表形势进行管理的。可以从空缓冲区链的头部获得缓冲区，回收时缓冲区返回到链的末尾。当缓冲区处于空闲状态时，RTEMS 为每个缓冲区分出 8 字节作为空缓冲区链表。当缓冲区被分配了，那么整个缓冲区都可以被应用所使用。

主要的操作：

■ 创建一个缓冲区

通过 rtems_partition_create 命令创建一个分区。同时分区的名字，起始地址，长度，缓冲区的大小都在这个命令中定义。RTEMS 为每个分区分配一个分区控制块（PTCB）,来管理每一个新创建的分区。每个分区中的缓冲区的数量可以通过分区长度和缓冲区的大小来决定。

■ 获区分区的 ID

当一个分区创建后，RTEMS 就为它产生一个唯一的分区 ID，直到分区被删除。通过，rtems_partition_ident 可以获得分区的 ID。分区 ID 在调用其他分区命令时会用到。

■ 获取一个缓冲区

通过 rtems_partition_get_buffer 命令可以获取一个缓冲区。如果获取成功，那么就立即返回 successful 码。如果没成功，就返回 unsuccessful。

■ 释放一个缓冲区

通过 rtems_partition_return_buffer 命令可以释放缓冲区，并且这个缓冲区返回

到空缓冲区链的尾部。如果要释放的缓冲区之前还没在这个分区中被分配，那么就返回一个 error 码.

■ 删除一个分区

通过 rtems_partition_delete 命令可以把一个分区删除。当一个分区被删除了，这个分区的 PTCB 也要返回到空 PTCB 表中。一个已经分配了缓冲区的分区是不可以被删除的。如果要删除他，就会返回一个 error 码。

#### 4.3.2.11 区域管理（Region Manager）

区域管理是在不同大小的单元里动态分配内存的管理。主要的命令有：

- rtems_region_create – 创建一个区域
- rtems_region_ident – 获取区域的 ID
- rtems_region_delete – 删除一个区域
- rtems_region_extend – 扩展一个区域
- rtems_region_get_segment – 从区域中获取一段
- rtems_region_return_segment – 把一段返回到区域中
- rtems_region_get_segment_size – 获取段的大小

一个区域就是把一个物理上邻接的内存空间通过用户自定义的边界划分成大小不同的段，这些段可以动态的被分配和回收。每个段的大小是用户定义的页的大小的倍数。例如，一个区域中的页是 256 字节，需要的段是 350 字节，那么就分配一个 512 字节的段。区域采用不同大小内存块的双链表管理。采用“first-fit”算法。从链表的头检查，如果匹配就分配。当一个段返回到区域时，这个空闲的块就跟他邻接的（如果也是空闲的话）块接合，成为一个更大的空闲块。由于各个块是可以自动接合的，所以一个区域的段数也是动态变化的。

主要的操作：

■ 创建一个区域

通过 rtems_region_create 命令可以创建一个区域。用户可以选择使用 FIFO 还是任务优先级算法作为任务等待队列中的任务调度算法。RTEMS 为每个区域分配一个区域控制块（RNCB）来管理新建的这个区域。

■ 获取一个区域的 ID

当一个区域创建后，RTEMS 就为它产生一个唯一的区域 ID，直到这个区域被删除。通过 rtems_region_ident 可以获得区域的 ID。区域 ID 在调用其他的区域命令时会用到。

■ 区域扩展

通过 rtems_region_extend 命令可以为一个已经存在的区域添加内存。调用者定义要添加的内存的大小和起始地址。

■ 获取一个段

通过 rtems_region_get_segment 可以从一个特定的区域中获取一个段。如果这个区域有足够的空闲区域，那么这个段就获取成功。如果没有合适的段被分配，有下列 3 种可能的情况：

缺省情况下，调用者任务将会一直等待直到获得了需要的段。

如果定义了 RTEMS_NO_WAIT 选项，任务将不会做任何等待，立即返回一个 error 吗。

定义一个任务等待的极限时间,当等待的时间超过了这个极限,就返回一个 error 码，不再等待。

在任务等待队列中的任务可以按照 FIFO 或是优先级顺序排列的，这个由用户定义。

■ 释放一个段

通过 rtems_region_return_segment 命令可以使一个分配的段返回到区域中。这个段跟它两边邻接的段接合成为一个长度更长的空闲段。然后，等待队列中的第一个任务检查现在是否有一个段可以满足要求。如果有，就分配段，并且该任务被激活。如果没有，继续等待。

■ 获取一个段的长度

通过 rtems_region_get_segment_size 命令可以获得一个特定段的大小。

■ 删除一个区域

通过 rtems_region_delete 命令可以把一个区域删除。当一个区域被删除了，这个区域的 RNCB 也要返回到空 RNCB 表中。一个已经分配了段的区域是不可以被删除的。否则，就会返回一个 error 码。

#### 4.3.2.12 双端口内存管理（Dual-Ported Memory Manager）

双端口内存管理提供了一个双端口内存区域（DPMA）内外地址转换的机制。主要的操作：

- rtems_port_creat – 创建一个端口
- rtems_port_ident – 获得端口的 ID
- rtems_port_delete – 删除一个端口
- rtems_port_external_to_internal – 外部地址转换成内部地址
- rtems_port_internal_to_external – 内部地址转换成外部地址

双端口内存区域（DPMA）是被一个特定的处理器拥有的 RAM 块，并且这个 RAM 块还可以被其他的处理器访问。拥有 RAM 块的处理器通过内部地址访问内存，而其他的处理器通过外部地址来访问。因此，RTEMS 定义了一个特殊的端口作为内部和外部地址映射。

主要的操作：

■ 创建一个端口

通过 rtems_port_create 创建一个 DPMA 的端口。用户定义这个端口的内部外部关系。RTEMS 为每个新创建的 DPMA 端口分配一个端口控制块 DPCB 来管理新建的端口。

■ 获取端口的 ID

当一个端口被创建的时候，RTEMS 就为每个端口分配一个唯一的 ID。可以通过 rtems_port_ident 命令来获取。端口 ID 在调用其他的端口命令时用到。

■ 转换地址

rtems_port_external_to_interna 命令可以把一个特定端口的外部地址转换成内部地址；rtems_port_internal_to_external 命令可以把一个特定端口的内部地址转换成外部地址。

■ 删除一个 DPMA 端口

通过 rtems_port_delete 命令可以把一个端口从系统中删除。当一个端口被删除后，它的 DPCB 返回到 DPCB 的空闲表中。

#### 4.3.2.13 I/O 管理（I/O Manager）

I/O 管理为访问设备驱动提供了一个完好定义的机制，并为组织设备驱动提供了一个结构化的方法。主要的命令为：

- rtems_io_initialize – 初始化设备驱动
- rtems_io_register_name – 登记一个设备驱动名字
- rtems_io_lookup_name – 查找一个设备驱动名字
- rtems_io_open – 打开一个设备
- rtems_io_close – 关闭一个设备
- rtems_io_read – 读设备
- rtems_io_write – 写设备
- rtems_io_control – 特殊的设备服务（跟对应的驱动有关）

每一个应用都会应用 RTEMS 的 I/O 管理来在系统配置表中定义设备驱动表。这个表包含了设备驱动的入口地址。

主要操作：

■ 登记和查询设备驱动名字

命令 rtems_io_register 使得一个特定的设备名字跟这个设备的 major/minor number 相关联。命令 rtems_io_lookup 是用来对于一个给定设备名字，来找到跟它相对应的 major/minor number。(其中，一个设备的 major number 是指定设备的设备驱动表的入口地址的索引，用来选择一个特定的设备驱动。Minor number 是用

来区分同一个驱动控制的各个设备。）

■ 访问一个设备

I/O 管理可以使得应用程序在标准方式下使用设备驱动。rtems_io_initialize, rtems_io_open，rtems_io_close, rtems_io_read, rtems_io_write, rtems_io_control 都是访问设备驱动的命令。

#### 4.3.2.14 致命错误管理器（Fatal Error Manager）

致命错误管理用来处理所有的致命的和无法挽救的错误。主要的命令：

rtems_fatal_error_occurred – 激活致命错误处理器

当 RTEMS 或是用户探测到一个致命的错误后,自动的唤起致命错误管理。虽然致命错误处理器的具体操作跟处理器的类型有关,但是通常它会屏蔽所有可屏蔽的中断, 放入一个错误码到处理器对应的堆栈或寄存器中,然后停机。其他的具体操作,可以在特定的处理器应用辅助文档中找到。

主要的操作：

■ 声明一个致命错误

当一个致命错误被发现时,rtems_fatal_error_occurred 命令就被激活。并且记录一些信息，如错误源，错误是否是内部执行产生的及错误类型等。rtems_fatal_error_occurred 不仅可以激活 RTEMS 的致命任务处理器，还可以激活用户提供的致命任务处理器。如果一个应用要求有更复杂的错误处理，就可以用户自定义相应的处理。自定义的处理要写到 RTEMS 的配置表中。rtems_fatal_error_occurred 就可以调用用户自定义的处理了。如果没有用户自定义的处理，就直接调用系统自带的处理。

#### 4.3.2.15 速率单调管理（Rate Monotonic Manager）

速率单调管理是实现周期性任务执行的设备。它可以支持那些使用速率单调调度算法来保证周期性任务即使是在短暂的超负荷的条件下也可以满足 deadline 的应用[16]。主要的命令：

- rtems_rate_monotonic_create – 创建一个速率单调周期
- rtems_rate_monotonic_ident – 获得周期的 ID
- rtems_rate_monotonic_cancel – 取消一个周期
- rtems_rate_monotonic_delete – 删除一个周期
- rtems_rate_monotonic_period – 结束当前的/开始下一个周期
- rtems_rate_monotonic_get_status – 获取周期的状态信息

当一系列任务都可以满足它们的 deadline 的时候，我们说这些任务是可调度的。速率单调调度算法（RMS）就为一系列任务提供了可调度性分析——在最坏的情况下任务是否是可调度的。RMS 根据任务的周期决定任务的优先级，周期越

短，优先级越高。当两个任务有同样的优先级时，RMS 就不区分两者。但是当同样优先级的任务在等待时，RMS 将会调度等待时间最长的那个任务。

主要的操作：

■ 创建一个速率单调周期

rtems_rate_monotonic_create 命令来创建一个速率单调周期。RTEMS 为每个周期分配一个周期控制块 PCB 来管理新建的速率单调周期。

■ 初始化一个周期

rtems_rate_monotonic_period 来建立和保持先前已经创建了的速率单调周期的周期性运行。一旦被初始化后，这个周期就一直执行直到周期过期了或被重新初始化。速率单调周期的状态会产生如下几种情况：

如果速率单调周期正在运行，调用者任务就会被阻塞。当这个周期运行完，这个周期将会被重新初始化。

如果速率单调周期没有正在运行并且没有过期，它会被初始化，调用者任务立即返回。

如果速率单调周期已经过期了，它会被用初始化，调用者任务立即返回一个 timeout 错误状态。

■ 获取周期的状态

如果命令 rtems_rate_monotonic_period 被激活了一段时间，指定的速率单调周期的当前状态将会返回。获取周期的状态不会改变周期的状态和长度。

■ 取消一个周期

通过 rtems_rate_monotonic_cancel 命令可以停止一个周期。被停止的周期可以通过 rtmes_rate_monotonic_period 命令重新初始化。

■ 删除一个速率单调周期

rtems_rate_monotonic_delete 命令来删除一个速率单调周期。如果这个周期正在运行并且没有过期，这个周期将会自动删除。PCB 也会自动返回到 PCB 空闲表中。

#### 4.3.2.16 用户扩展管理（User Extension Manager）

用户扩展管理可以使应用的开发者通过提供被系统事件激活的扩展程序来增加执行的功能。主要的命令：

- rtems_extension_create – 创建一个扩展集
- rtems_extersion_ident – 获取扩展集的 ID
- rtems_extension_delete – 删除一个扩展集

用户可以定义多个扩展集（extension set）。其中，静态扩展集可以在系统配置表中的用户扩展表中定义。这种扩展集在系统的整个生命周期中都是激活的，

而且不可以被删除。同时，用户还可通过命令来建立动态扩展集。这种扩展属于RTEMS 的对象，有名字和 ID，并且可以动态的创建和删除。

主要操作：

■ 创建一个扩展集

rtems_extension_create 命令可以创建一个新的扩展集。并且分配一个扩展集控制块（ESCB）来管理新创建的扩展集。

■ 获取扩展集的 ID

当一个扩展集被创建后，系统就给他分配一个 ID, 可以通过 rtems_extension_ident 命令来获取他的 ID,这个 ID 可以在调用其他的扩展集命令中使用。

■ 删除扩展集

rtems_extension_delete 命令可以删除一个扩展集。扩展集控制块也返回到 ESCB 空闲表中。

#### 4.3.2.17 多处理器管理（Multiprocessing Manager）

在多处理器的实时系统中，由于多处理器之间的数据和全局资源的共享，这就需要有一个有效的，可靠的通信机制来保证各个处理器之间的正常通信。多处理器通信接口层(MPCI)负责管理一系列被称为包的缓冲区。这些包用来包含一些各个节点之间传递的信息。各个处理器之间就是通过获取包，返回包，发送包，接收包等动作来实现通信的。

# 第五章 嵌入式开发平台的设计与实现

本章将讨论整个开发平台的硬件设计、选型、实现以及最重要的 RTEMS 向目标开发板上的移植工作。

## 5.1 设计需求

在对 RTEMS 进行调查、分析和研究后，本课题最终决定选用它作为该嵌入式开发平台的软件开发环境。但是还有一个问题，一个好的嵌入式开发平台不应该只有一个高稳定和高性能的 RTOS 作为其软件开发环境，还应该有一个成熟的并且集成了常用功能的硬件环境，只有这样才能使用户的开发周期尽可能的缩短从而产生更大的效益。这一点已经被今天的市场所验证[1]。

因此，本课题针对面向工业的嵌入式开发平台这一要求，针对工业上对实时性、抗干扰性、可靠性和低成本的特点，通过大量调研，本课题在进行硬件选型和最终制版工作后，成功的开发了 NEUinfo 开发板并将 RTEMS 移植到了该开发板上，最后基于该开发平台进行了一些应用程序的开发。

## 5.2 硬件选型

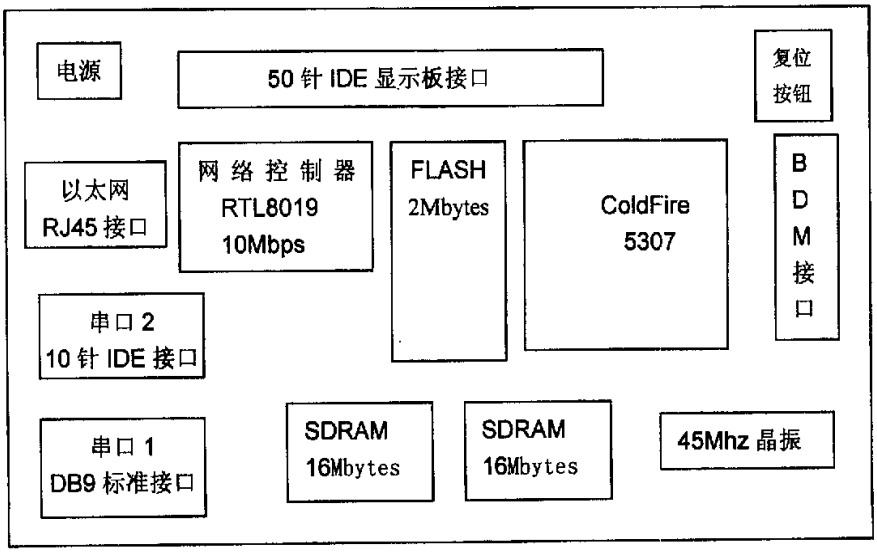


图 5.1 硬件开发平台

Figure 5.1 Hardware Development Platform

根据本课题是一个面向工业的嵌入式开发平台这一需求，拟定了硬件开发板

的基本结构：以 Motorola 公司的 MCF5307 嵌入式处理器为核心，具备网络通信功能、基本的串行通讯功能。此外，为支持系统软件开发和应用软件开发，该硬件开发板提供 BDM 调试接口。硬件开发平台功能模块如图 5.1 所示。

**CPU 选型**

CPU 选型为 Motorola 公司的 MCF5307 嵌入式处理器（Coldfire 系列）。

ColdFire 是 Motorola 公司继承 68K 系列 CPU 的嵌入式微处理器，主要面向工业控制和低成本的通讯市场。ColdFire 最大的优点是和它所相关的资源特别丰富。包括：编译器、调试器、硬件资源（原理图、PCB 图）、源代码、操作系统等等。所以说 ColdFire 和 Free 密切相关。

MCF5307 在最大时钟频率 90Mhz 下，达到 70MIPS 的运行速度。MCF5307 处理器采用了 Coldfire V3 内核，并有如下特点[4]：

- 8-Kbyte cache
- 4-Kbyte 片上 SRAM
- 整数/小数累加器 (MAC) 单元
- 除法单元
- 系统 debug 接口
- 同步和异步 DRAM 控制器
- 4 管道 DMA 控制器
- 2 个通用定时器
- 2 个串口(UART)
- $I^2C$™总线接口
- 并口 I/O
- 系统全局模块 System integration module (SIM)

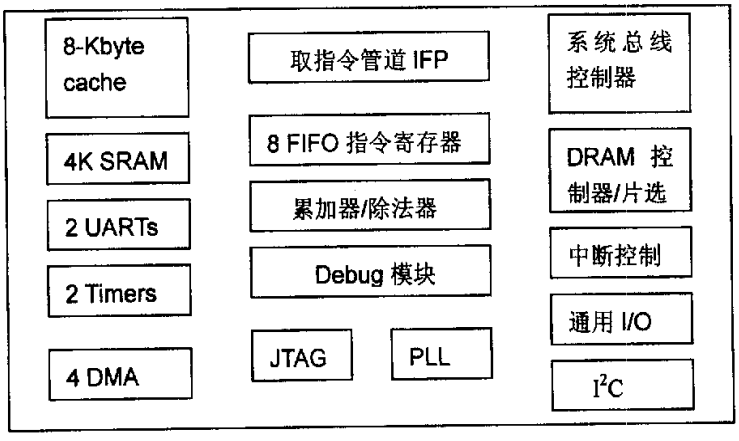


图 5.2 MCF5307 核心模块

Figure 5.2 Core Module of MCF5307

网络控制器选型

网络控制器选型为 Realtek 公司的 RTL8019AS 以太网控制器。该控制器集成了全部的网络接口功能，支持 10Mbps 以太网协议。其 10 位地址线和 16 位数据线通过地址缓冲和数据缓冲器进行总线隔离后与 MCF5307 的地址总线和高 16 位数据总线相连。MCF5307 的读写信号/RW 及其反相信号分别与 RTL8019AS 的 IORB 和 IOWB 相连，进行读写控制。

RTL8019AS 是嵌入式 Internet 应用中广泛使用的以太网络接口控制器[9]。它是和以太网传输介质相连接的硬件设备。它实现了基于 IEEE802.3 协议标准的 MAC 层（Media Access Control Layer）的所有功能，内置的双 DMA 通道和 FIFO 完成简单有效的包管理（数据帧的接收和发送）功能，本地 DMA 通道的传输速率高达 10Mbps。

该 NIC 具有下列特点：

- 适合 10Base5，10Base2，10BaseT 使用
- 与 NE2000 软件兼容
- 支持以太网全双工通信方式
- 支持三种掉电模式：休眠模式、内部时钟不间断模式和内部时钟停止模式
- 具有简单、多功能的缓冲管理功能
- 多种寻址支持
- 低功耗，采用先进的低功耗 CMOS 设计
- 多种处理器支持，可支持 8 位，16 位和 32 位的微处理器

RTL8019AS 是高度集成化的以太网控制器，它可以提供与 NE2000 相兼容的以太网适配器的解决方案，并采用全双工通信方式。全双工通信方式使帧的发送与接收能够在一条通信线路上同时进行，这使信道的带宽增加一倍。RTL8019AS 优越的性能价格比使它被嵌入式 Internet 系统广泛地使用。RTL8019AS 的内部功能模块及其关系如图 5.3 所示。

NIC 主要用来控制网络和主机间的数据传输。完成数据传输主要由以下几个模块来实现。

(1)接收解串功能模块。该模块主要用来把串行输入的比特流进行 8 位分割，然后把分割后的比特流（8 位）并行送入 16 字节的 FIFO。另外，还需要激活接收字节计数器对帧长度进行计数，以及把目的地址送往地址识别逻辑单元。如果地址识别单元拒绝该帧，则清除 FIFO 中的数据和字节计数单元的值。同时必须把串行输入的数据送往 CRC 生成逻辑，对该帧进行校验。

(2)CRC 产生校验功能模块。该模块由两个作用：当发送数据时，它用来产生校验和，并把校验和附在数据之后一起送入网络。当接收数据时，它用来校验接

收到的数据是否有错，并把错误信息存入接收状态寄存器，以供数据接收程序检查。

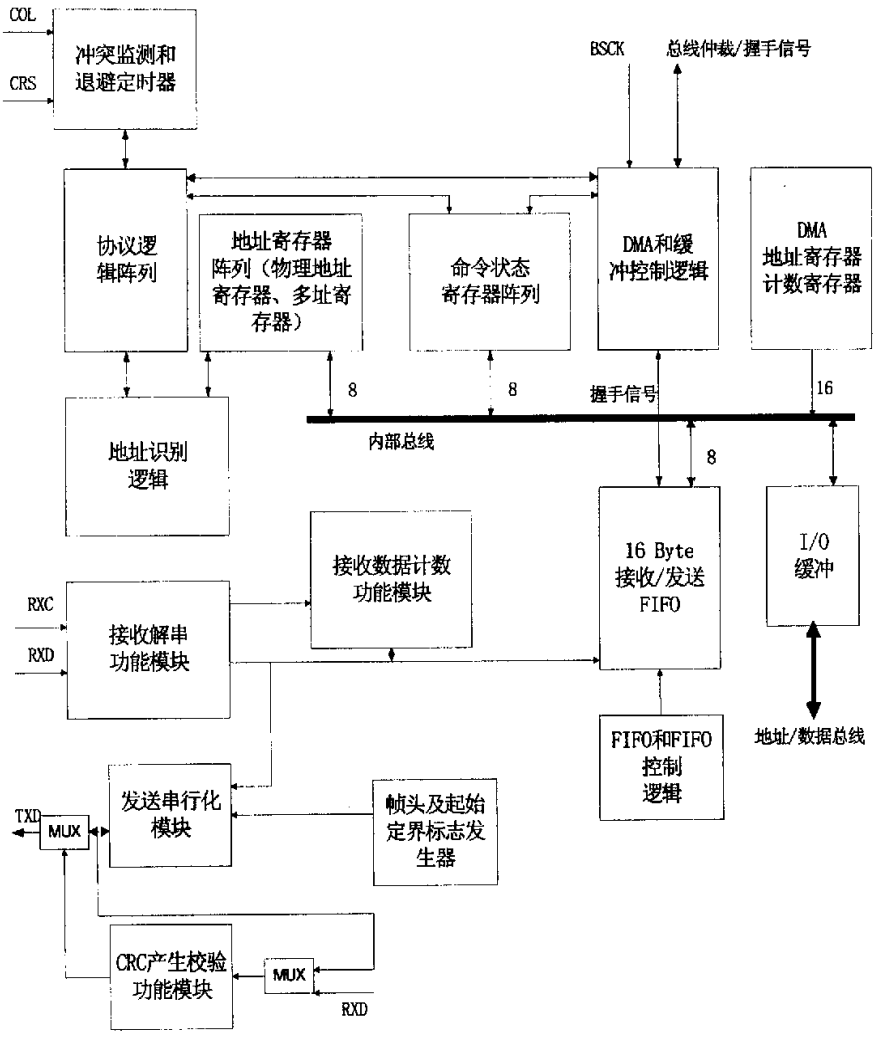


图 5.3 RTL8019AS 的内部功能模块及其关系

Figure 5.3 Interior Modules of RTL8019AS

(3)发送串行化功能模块。该功能模块负责把 FIFO 中的并行数据转变为串行数据并发送，同时把串行化数据送入 CRC 产生校验功能模块，以计算校验和，该校验和附加在数据之后发出。

(4)地址识别逻辑。该功能模块用来识别目的地址中的 6 字节长的地址并与地址寄存器组中的地址进行匹配。若不匹配，则拒绝接收该帧；若匹配，则继续接收帧。

(5)FIFO 和 FIFO 控制逻辑。NIC 只有 16 字节的 FIFO。当发送数据时，本地

DMA 把数据写入 FIFO，发送串行化模块把 FIFO 中的数据并行读走，并串行发送该数据。当接收数据时，解串功能模块把并行（8 位）写入 FIFO，再由本地 DMA 把 FIFO 中的数据存入缓冲区。FIFO 控制逻辑用来对进入 FIFO 的字节数进行计数，以便在发生溢出前能够把数据取走。

(6)协议逻辑阵列。该功能模块主要负责实现 IEEE802.3 协议，包括冲突的随机退避，装帧（加帧头）和拆帧（去帧头），以及实现接收同步。

(7)DMA 和缓冲控制逻辑。DMA 和缓冲控制逻辑用来控制两个 16 位的 DMA 通道。当接收数据时，本地 DMA 用来把接收到的数据存入接收数据缓冲环。当发送数据时，本地 DMA 把发送缓冲中的待发数据传送到 FIFO。远程 DMA 用来实现本地缓冲区和主机间的数据传输。本地 DMA 的优先级高于远程 DMA。

**FLASH ROM 选择**

FLASH ROM 选择 AMD 公司的 Am29PL160C－70R。该芯片容量是 16Mbits(2Mbytes)，并且可通过选择 BYTE#信号输入使其工作在字或字节模式（1 为字，0 为字节）。该芯片最大存取时间 70ns,工作电压 3V－3.6V。在该系统中，将 BYTE#置 1(工作于 16 位模式)。片选(CE#)和使能输出(OE#)分别接 MCF5307 的-CS0 和-OE,16 位数据线（FB_D0~FB_D15）和 20 位地址线（MA0~MA19）分别通过数据缓冲和地址缓冲接到 MCF5307 的数据总线和地址总线上。

**SDRAM 的选择**

SDRAM 为现代公司 HY57V281620HCT，大小为 128Mbits(16Mbytes)。其地址和数据总线直接连接与 MCF5307 的地址和数据总线。

**UART 选型**

UART 为 MAXIM 公司的 MAX232E（RS-232 通讯电平转换）。该芯片封装了 2 对 5V~15V 间的传送和接受电平转换，可同时支持 2 路串口通讯。其中充电电容选为 1uF。传送和接受信号分别与 MCF5307 的 TX 和 RX 相连。

**BDM 调试接口**

MCF5307 内置了调试模块，该模块提供了为方便开发调试的所有的信号接口。其中包括 8 位的 PSTDDATA[7:0]（处理器状态数据并行输出），该并行数据口以内置的 PSTCLK(1/2 处理器时钟频率)的时钟信号进行调试过程中的处理器状态数据输出。外围开发设备可通过模块提供的/BKPT 信号请求处理器中断并输出状态数据和通过 PSTCLK 时钟信号接收该状态数据。另外，该模块还提供了与外围调试设备进行命令和数据通信的全双工串行接口。系统之间通过外围设备提供的 DSCLK 时钟频率（≤1/5 处理器频率）进行通讯，分别以 DSI 和 DSO 信号作为输入和输出。该系统采用双排 26 针的连接插头与外围调试设备进行连接。

电源与接地

电源部分

整个系统除以太网控制芯片 RTL8019 的逆变部分用到 5V 电源外，其余各芯片均使用 3.3V 电源。为虑除电源的高次谐波，防止 EMI，在电源处并接 0.1uF 至 680pF 的虑波电容。另外，对于给 PVCC 端提供的 3.3V 滤波电压，在布线时应使滤波电容和电阻靠近 MCF5307，以达到较好地滤波效果。

接地部分

为避免各系统间信号干扰，按照系统划分的原理，应将数字地，模拟地，大功率地/电感地和电源地先各自分接，再一起接入大地。对于本系统，先将各系统地分别接入 DGND、AGND 和 HGND，然后一起连接到 GND。最后，在布线时，地线面应包围整个印制板，以对电路板进行屏蔽和阻挡辐射干扰。

开发板实际图

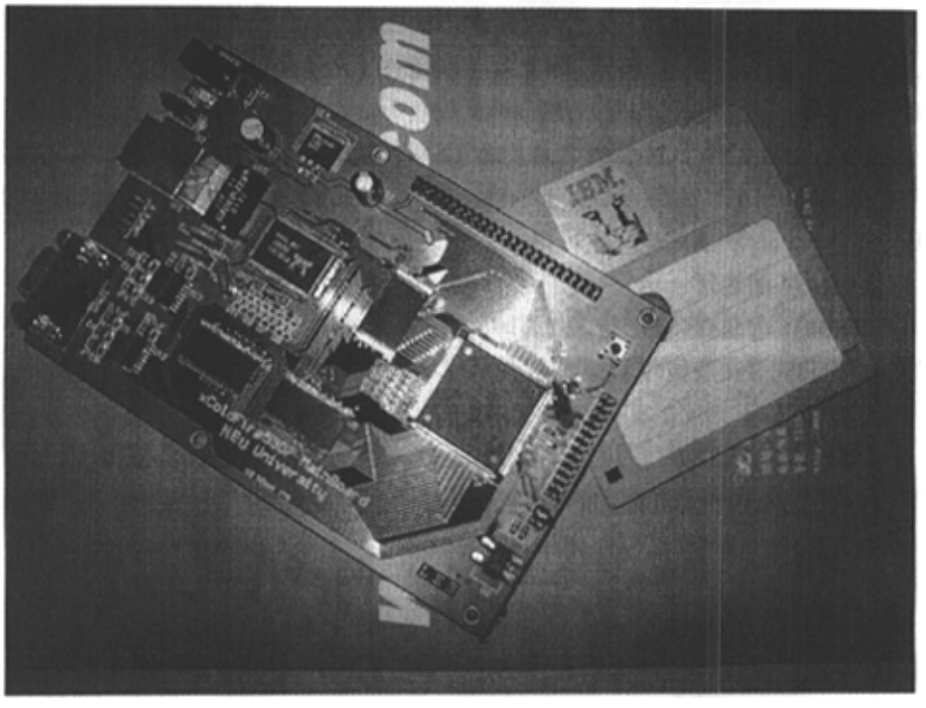

图 5.4 开发板实际图

Figure 5.4 Development Platform

## 5.3 开发环境

当今几乎所有的实时嵌入式软件系统都是在交叉编译环境中开发出来的[10]。在交叉编译环境中，软件开发都是在一个计算机系统中实现的——主机系统。开发的结果即软件系统要在目标平台上执行。目标平台的需求有可能与主机系统有

冲突，交叉编译环境就可以协调两者的关系，使得在主机系统上开发的代码可以在目标平台上能够运行并实现特定功能。下面就简要介绍一下建立 RTEMS 交叉编译环境的过程。

### 5.3.1 软件环境

环境

主机系统：Intel Pentium 4 + Redhat Linux 7.2（kernel 2.4.7）

目标平台：NEUinfo 开发板（MCF5307）

安装步骤

首先要在宿主机上安装标准 Linux 发行版，推荐使用 Red Hat，接下来就可以建立交叉编译开发环境（下载网站：www.rtems.org）。

建立交叉编译环境

安装 RTEMS 补丁的 GNU 编译调试工具集：

binutils 的 RTEMS 补丁

newlib 的 RTEMS 补丁

gcc 的 RTEMS 补丁

调试工具 gdb 的 RTEMS 补丁和 BDM 驱动。

安装 RTEMS

在本地解压缩 RTEMS，然后调用 RTEMS 的 configure 命令对你的 RTEMS 进行配置，enable-rtemsbsp=<your target board>是目标平台（开发板）的 BSP 型号。具体的配置方法可以参见 README.configure 文件。最后编译，安装 RTEMS。

安装完后，就可以在 /root/.bashrc 文件中加入语句 RTEMS_MAKEFILE_PATH=<yourRTEMS installed directory>/<your target board>设置 RTEMS 编译路径。然后就可以用 RTEMS 来编译以 RTEMS 为内核的实时应用了。

### 5.3.2 硬件环境

本课题组采用了 1 台运行 Linux 的 PC 机作服务器，被开发的目标板也挂连在网上并和这台 Linux 服务器通过串口和 BDM 相连。课题小组中的成员通过局域网用 Telnet 登录到这台 Linux 服务器上，调试、编译并生成目标代码，然后通过串口下载到开发板上。如图 5.5 所示。

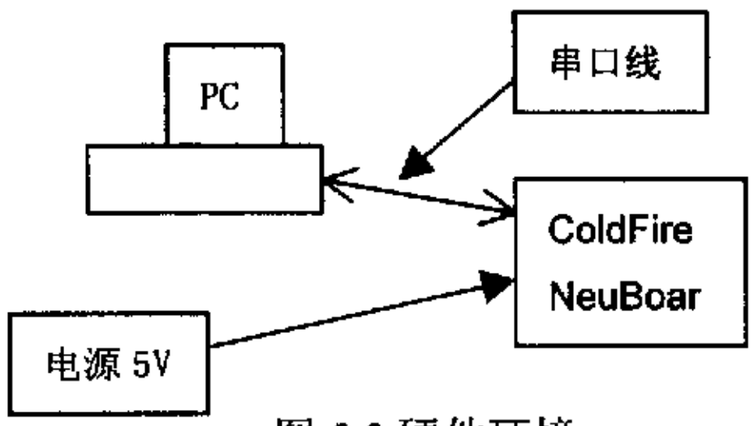


图 5.5 硬件环境

Figure 5.5 Hardware Environment

## 5.4 RTEMS 的移植

RTEMS 的移植工作主要分为 CPU 支持文件和 BSP 开发板支持包的改写。因为目前 RTEMS 还没有 MCF5307 处理器(虽然 RTEMS 支持 Coldfire 系列)的 CPU 支持文件，所以需要添加这个文件。在 CPU 支持文件里，将处理器芯片上的资源进行声明和驱动，给上层提供时钟，定时器和片上 UART 终端的调用命令。根据自己的板子进行 BSP 开发板支持包的改写，首先对开发板上资源进行声明，然后添加启动代码，添加网卡和串口的驱动。

### 5.4.1 RTEMS 的移植特点

RTEMS 采用了多重方法来加强它的可移植性。这些都是为了加强基于 RTEMS 开发的程序的可重用性。基于 RTEMS 开发的绝大部分代码都可以在所有平台上重用。RTEMS 将和硬件平台有关的代码分为 3 部分，分别是：

- CPU dependent
- Board dependent
- Peripheral dependent

**CPU dependent**

这部分代码包括了执行代码的一些基本规则，例如程序切换和中断子程序的调用。此外，该代码还包含了帮助用户使用这个 CPU 来进行应用开发的子程序。例如 CPU 的片上外围设备驱动。

**Board dependent**

这部分代码将 RTEMS 和一个特定的开发板很好的结合在一起，这部分代码表现为 BSP 和相应的驱动程序。

**Peripheral dependent**

这部分代码提供了一个可重用的外围设备驱动程序库，这个库可以被裁减以适应特定的扳子。

### 5.4.2 BSP 移植

板级支持包（Board Support Package）是 RTEMS 和在特定硬件平台下的应用之间接口设备的集合。这些设备包括硬件初始化，设备驱动，用户扩展，多处理器通信接口（MPCI）。但是，最小的 BSP 只需支持处理器复位和初始化。由于 BSP 是 RTEMS 和目标硬件之间的联结纽带，它在操作系统的移植过程中有着非常重要的作用。所有的 BSP 都在目录 c/src/lib/libbsp 下。在这个目录下，对每一个 CPU 系列还有一个子文件夹。另外，还有一些共享的文件是无论任何 BSP 都可以使用的。它们在

- c/src/lib/libbsp/shared
- c/src/lib/libbsp/CPU/shared
- c/src/lib/libbsp/CPU/BSP

在每一个 BSP 下有一系列的子文件夹，通常包括这些：

console: 一般是 BSP 的串口驱动，而不仅仅是一个控制台驱动，它处理板子上的 UARTs。

clock: 支持时钟的 tick，一个基于内核的有规则的时间。

timer: 支持定时器设备。

rtc: 支持硬件的实时时钟。

nvmem: 支持掉电不丢失数据的内存例如 EEPROM or Flash。

network:网卡驱动。

shmsupp: 在多处理器系统中，支持共享内存驱动 MPCI 层。

include: BSP 的头文件。

wrapup: 将构成 BSP 库所需要的部分打包。

下面再简要介绍一下 BSP 的几个主要组成部分。

初始化代码

当执行了 reset 命令后，初始化代码是执行的第一段代码。它的目的是为了应用程序初始化板子。每当有复位或是重起时，BSP 的初始化代码都是最先执行的代码，它为应用程序初始化目标板。

中断向量表

虽然不同的 CPU 有不同的中断向量，但是基本功能是相同的。每一个表的入口都对应一个特定的服务子程序。当一个中断出现的时候，就调用相应的服务子程序。当调用出现时，一部分切换信息由处理器自动进行存储。而为了使得中断服务子程序可以在高级语言中执行，RTEMS 将负责存储足够的切换信息。

在一些处理器上，中断向量表是一个固定的地址。如果这个地址在 RAM 里，通常 BSP 就初始化向量表使得它包含缺省服务子程序的指针。如果在 ROM 里，

用户要根据特别的步骤来填这个表。

如果中断向量表基地址可以动态的改变为任意地址，那么 RTEMS 将会分配自己的表并且安装它。MCF5307 就有这样的基地址。MCF5307 默认的中断向量表在 start.S 里声明。

片选初始化

当微处理器访问一个地址区域时，首先地址要经过解码器解码，这样处理器就知道是哪一个内存片被访问了。片选寄存器必须符合 linkcmds 的设置。在 MCF5307 的 BSP 中，ROM 和 RAM 的地址可以同时在 linkcmds 和初始化代码里找到。

全部处理器的寄存器初始化

在嵌入式系统中处理器是非常复杂的设备而且有很多的片上外围设备。对于这些设备，通常有一些特别的寄存器需要初始化。开发者可以通过参考 MCF5307 手册来得到更详细的信息，需要注意的是要非常小心的编辑它们，任何一个小小的疏忽都会造成整个系统的崩溃。

数据段重新拷贝

通过数据段的重新拷贝，从而实现：

1．将数据段.data 从 ROM 拷贝到它的 RAM 保留地址中

2．清空.bss 段（所有未初始化的数据都将被初始化为 0）。

在 BSP 中有一个 Linkcmds 文件，这个文件是用来描述为了连接程序板子所需要的内存配置。重点记录了应用的代码和数据在内存中的位置。

在一般的计算机中，都存在 RAM 和 ROM,程序总是存在于磁盘中并在 RAM 中执行。但嵌入式目标板不同，目标板会在每次重新启动时执行程序。应用程序储存在不掉电内存中，如 ROM,PROM,EEPROM,Flash 中。处理的数据储存在 RAM 中。

在嵌入式数据的存储分为三个部分：

代码部分（.text）:这一部分是程序代码，不可以改变的。放在 ROM 中。

非初始化的数据部分（.bss）：程序的非初始化的变量，在程序中可能改变，放在 RAM 中。

初始化的数据部分（.data）：初始化的程序数据，在程序周期内可能改变，要放到 RAM 中。但是，这些变量又是要预先定义的，又要放在 ROM 中。

由于初始化数据（.data）要每次都加载到 RAM 中，所以每次都要初始化。一个常用的方法就是，先把.data 部分的副本复制到 ROM 中的.text 部分的尾部。每当程序开始时，就把 ROM 中的.data 部分复制到 RAM 中的合适的区域，等待程序的调用和修改。具体过程如图 5.6[3]所示。

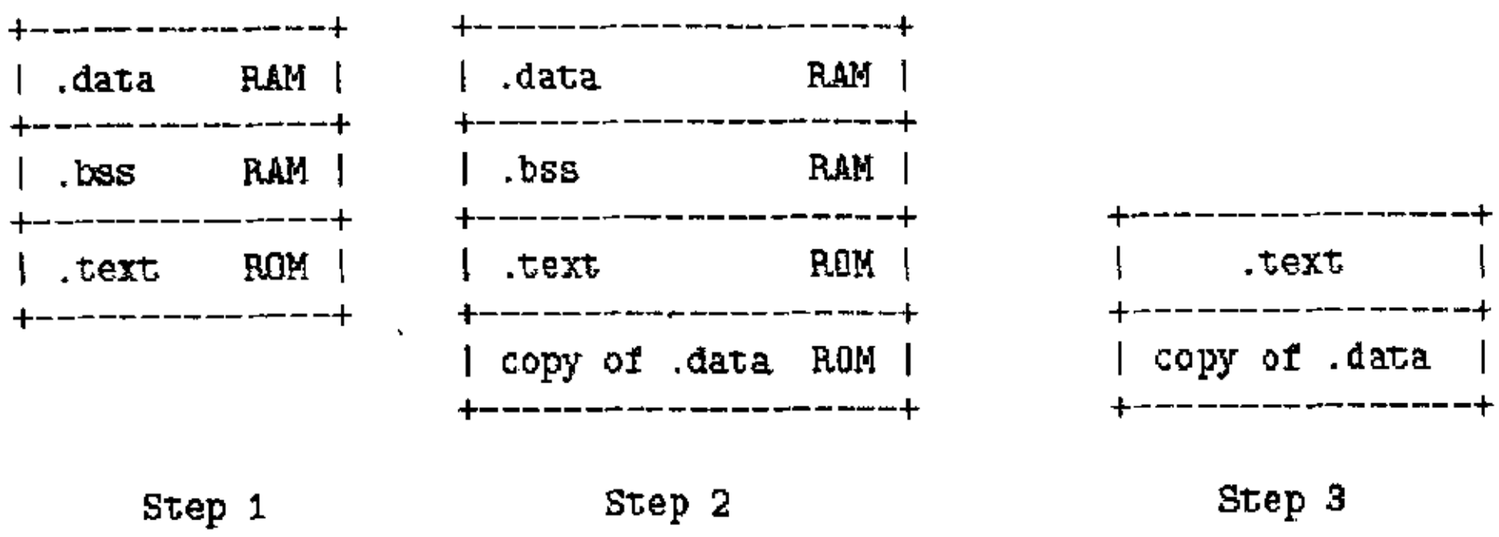


图 5.6 初始化数据示意图

Figure 5.6 Data Initialization

串口控制台驱动

串口控制台的驱动使用了 POSIX 的 Termios 支持。通常，RTEMS 将所有的串行设备认为是控制台。一个控制台可以作为标准的输入和输出，并且可以处理原始数据流。

串口的驱动可以被看成一系列的标准 C 语言库，例如 printf 和 scanf 或者直接通过系统进行标准的 read 和 wrtie 调用。有两种主要的功能模式：

1 console: 格式化的输入和输出，还有一些特别的字符识别和处理。

2 raw: 允许原始数据流处理。

可能有人认为可以用两个串口来分别处理两种不同的功能，但是 Termios 只允许有一个串口做控制台驱动。

Termios 是一个标准的中断管理，它包括在 POSIX 1003.1b 标准中。它通常提供一些 UNIX 命令，RTEMS 对 Termios 的支持有好多好处：从用户的角度来看，它提供了标准的原始操作支持，来让用户很方便的访问终端和修改配置。这些操作在 UNIX 和 RTEMS 下是完全相同的。从 BSP 开发者的角度来看，它使得开发者不需要再去处理缓冲区和互斥这样的问题。

对于 UART，通常有两种主要的工作模式：

轮循模式

在轮循模式中，处理器将会阻塞正在发送或接收的字符。这种模式并不是最有效的使用 UART 的方式。通常这是比较简单的编程方法，而且也足以满足当 BSP 出现错误的时候打印一个出错信息到控制台上。当主要用串口做调试用的时候，轮循模式是用的比较多的。在一个简单的轮循模式驱动中，程序在进行读写操作的时候将会连续的检查 UART 的状态。

中断模式

在中断模式中，处理器将不会阻塞正在发送或接收的字符。数据被放到应用程序和中断服务子程序的缓冲区里。因为串型设备是相对较慢的，所以用两个缓

冲区来存储。一个用来存储收到是数据，一个用来存储要发送的数据。当 UART 收到数据时，将会产生一个中断。中断服务子程序将会把接收到的字符放到缓冲区的队尾。当一个应用程序请求缓冲区的时候，队首的数据将会被返回。当一个应用向串口设备打印的时候，字符将被放到输出缓冲区的队尾。驱动将负责依次将这些字符放到 UART。当全部字符被传送完毕的时候，将产生一个中断。中断服务子程序将把剩余的字符以同样的方式发送。

串口驱动程序包括：

轮循模式写

轮循模式读

中断读服务程序

中断写服务程序

时钟驱动

时钟驱动的目的是给内核提供一个稳定的时钟 tick，这样 RTEMS 原语（需要时钟 tick）就可以正确的工作了。

时钟驱动程序包括：

Tick 计数器

5307 的时钟驱动提供了一个全局变量来累加到目前为止 tick 出现的次数，就是统计 tick 的中断服务子程序发生中断的次数。

初始化

初始化程序用来响应一个可以周期性出现中断的可编程硬件。通常我们用一个 timer（定时器）作为产生时钟 tick 的源。

时钟中断程序

设备应该被编程以保证是每隔 m 个微秒产生一个中断，这里 m 必须等于 BSP 配置表中的 BSP_Configuration.microseconds_per_tick。我们必须参照 MCF5307 手册来仔细选择我们的定时器时间，通常选择为微秒的 10 倍。

定时器驱动

定时器驱动是 RTEMS 用来做时间测试的。这个驱动提供了准确的基准时间，精确到微秒或一个 CPU 周期，或一个总线周期。

RTEMS 时间测试组件需要一个准确的基准时间。这样 RTEMS 时间测试组件程序就可以准确的确定目标硬件的性能并且将它的性能并和其它 RTEMS 目标硬件进行比较。定时器驱动程序包括：

定时器初始化

读定时器，返回自从上一次定时器初始化后流逝的基准时间数量。

空函数，RTEMS时间测试组件使用这个函数来测量调用一个子函数的时间消

耗。

Set _find average overhead，这个程序由RTEMS时间测试组件中的Check Time（tmck）来调用。它使得Read_timer子程序不减去初始化和读的消耗，从而准确的测出初始化和读的消耗。

## 5.5 具体移植工作

下面，本问将简述主要的移植工作，涉及到的函数将会给出函数名和主要功能。

首先，将版本为 SS-20030417 的 RTEMS 编译通过，确认当前版本有效可用。按照 5307 的需求，将没有用的 BSP 和 CPU 里的包全部删除。之后在根目录下运行./bootstrap –c 和./bootstr，编译通过。在/rtems/make/custom/下增加一个关于 5307 的配置信息文件。再次编译通过后，确认上述修改有效后，进行 BSP 和 CPU 内部函数移植修改。

RTEMS 的移植工作主要分为 CPU 支持文件和 BSP 开发板支持包的改写。因为目前 RTEMS 还没有 MCF5307 处理器(虽然 RTEMS 支持 Coldfire 系列)的 CPU 支持文件，所以需要添加这个文件。在 CPU 支持文件里，将处理器芯片上的资源进行声明和驱动，给上层提供时钟，定时器和片上 UART 终端的调用命令。根据自己的板子进行 BSP 开发板支持包的改写，首先对开发板上资源进行声明，然后添加启动代码，添加网卡和串口的驱动。

### 5.5.1 CPU

1) 寄存器移植

MCF5307/include/MCF5307.h：在该程序中对 MCF5307 片上外围设备进行声明，全部声明可参考 MCF5307 user's manual 手册。

首先，定义 5307 的 8 位，16 位和 32 位寻址模式，即基地址+偏移量。进行片上寄存器移植，包括：

- CACHE related
- SRAM related
- DAM related
- SIM System Integration Module related
- Interrupt related
- Reset
- System Protection

- Software Watchdog Related
- Pin Assignment
- Bus Master Arbitration Control
- Chip Select Related
- Parallel Port (GPIO) Module
- DRAM Controller
- UART Module Related
- M-BUS (I2C) Module
- Timer Related Register

**2)** 串口控制台移植

A. 头文件部分/MCF5307/include/MCFuart.h：该头文件声明了一个 UART 的描述结构和几个 UART 的函数，具体方法的实现在 MCFuart.c 里。

移植的 UART 功能函数包括：

MCFuart_init(MCFuart *uart, void *tty, rtems_unsigned8 intvec, rtems_unsigned32 chn);该函数验证输入的参数，并完成初始化。

MCFuart_reset(MCFuart *uart); 该函数执行硬件初始化。

MCFuart_disable(MCFuart *uart); 该函数禁止 UART。

int MCFuart_set_attributes(MCFuart *MCF, const struct termios *t); 该函数分析终端的属性，然后执行设置。

int MCFuart_poll_read(MCFuart *uart); 该函数进行轮训模式下读 UART，并且执行错误控制。

int MCFuart_interrupt_write(MCFuart *uart, const char *buf, int len); 该函数在中断模式的时候，进行缓存内容的传送。

int MCFuart_poll_write(MCFuart *uart, const char *buf, int len); 该函数在轮训模式下，按位传送缓存内容。

int MCFuart_stop_remote_tx(MCFuart *uart); 该函数停止远程的数据流。

int MCFuart_start_remote_tx(MCFuart *uart); 该函数恢复远程的数据流。

B. 具体实现部分/MCF5307/console/MCFuart.c：该程序是 UART 的操作函数的具体实现，包括：

MCFuart_init 函数的具体实现。

MCFuart_set_baudrate 设置波特率，编写 UART 的定时器来指定 BAUD 率。

MCFuart_reset 重启初始化的实现，使用 MCFuart_init 里的参数对 UART 进行硬件初始化，按照 5307 用户手册提供的算法编写的重启初始化。

MCFuart_disable 的实现，关闭 motorola 的 UART 控制器，关闭传送和接收，

如果是中断模式的就还要关闭中断。

MCFuart_set_attributes 的实现，这个函数分析终端的属性，然后对硬件进行相应的设置。

MCFuart_poll_read 的实现，进行轮训来读 UART 上的字符，并且进行错误控制，当奇偶校验和成帧误差出现的时候，返回值依赖于 termios 的输入模式标志。

MCFuart_poll_write 的实现，在轮训模式下按位传送缓冲区里的值。

MCFuart_interrupt_handler 的实现，UART 中断处理操作。

MCFuart_interrupt_write 的实现，初始化中断模式的传送缓冲区。

MCFuart_stop_remote_tx 的实现，关闭远程设备的数据流。

MCFuart_start_remote_tx 的实现，重新开始远程设备的数据流。

**3) 系统时钟移植**

/MCF5307/clock/Ckinit.c：该驱动程序将 5307 片上的 timer0 初始化成为系统时钟。包括函数：

Clock_driver_ticks 是一个单调递增的计数器，从 timer 被驱动后它累计 clock tick 数。

rtems_isr，处理 timer 中断，通过清除 timer 中断标志和声明 clock tick 给系统。

Clock_exit，关闭 timer，如果它被使能，并且将它从 5307 的中断掩码上去除。

Install_clock，以 BSP 上的时间片值作为基值（reference 值）来初始化 timer，并且设置中断处理来处理系统 clock tick。

Clock_initialize，设置时钟驱动，并使得驱动 timer 的 major/minor 值对其它驱动也有效。

**4) 系统定时器移植**

/MCF5307/timer/Timer.c：初始化 5307 上的 Timer2 作为时钟基准

外接汇编中断处理子程序 extern rtems_isr timerisr(rtems_vector_number vector)。

Timer_initialize，该函数为了准确的时钟测量初始化 timer2。

Read_timer，该函数在开始计数后，每隔一微秒读出 timer 值。

Empty_function，该函数测量 Timing Test Suite 中的 loop 花消。

Set_find_average_overhead，该函数由 RTEMS Timing Test Suite 里的"Check Timer" (tmck) test 调用，使得 Read_timer 不减去 timer 的启动停止总开销。

/MCF5307/timer/timerisr.S：产生一个时钟中断的汇编程序。

该汇编程序的操作为：

1 保护 a0 寄存器的值

2 将 timer 的 event 寄存器地址给 a0

3 清除挂起的中断，通过将 5307 的 event 寄存器的 REF 和 CAP 置 1

4 中断数加 1

5 恢复 a0 寄存器的值

6 返回

### 5.5.2 BSP

BSP(Board Support Package)就是板支持包，包括初始化启动代码和硬件驱动。BSP 是由用户提供的一组工具集，这些工具集是 RTEMS 及其应用与特定硬件平台之间的接口。这些工具集包括硬件初始化，设备驱动，用户扩展，多处理器通信接口。但是，最小的 BSP 只需支持处理器复位和初始化。进行 BSP 移植，主要是针对 RTEMS 和目标开发板上硬件的相关的部分进行改造，至于操作系统级的调用并不需要做改动。因为 RTEMS 采用了统一的代码格式来书写，我们在开发自己板子的 BSP 时可以参考已有的 BSP，特别是那些和你选用的 CPU 一样或者类似的 BSP，将会对开发工作有很大的帮助。

**1)** 板子上的原型和声明移植

\lib\libbsp\m68k\MCF5307neuinfo\include\bsp.h

这个程序包括了这个板子上的原型和声明，每一个板子都需要一个 bsp.h。大多数 bsp.h 都提供在 linkcmds 里声明的变量原型。如果这里的内存资源声明有变化，要同样修改 linkcmds 里的声明。

具体内容包括：

- 资源定义
- 片选 0 接 flash，片选掩码为 2M。
- 片选 2 接 sram，分配 1M 空间
- 注释掉片选 3
- 增加片选 4，以太网控制器
- 修改 DRAM 控制器外接的 RAM
- 将片上 SRAM 改为 4K
- 中断向量声明
- 声明 2 个串口终端
- 声明中断堆栈，中断堆栈的大小为 4*1024
- 声明 System frequency
- 声明两个 TEST 的时间持续的长度
- 声明 TIME Test 27 的，不要使用硬中断，使用软中断
- 声明一个 microsecond 的自延迟程序，用来进行驱动的时候的延迟

- 声明时钟驱动程序入口
- 引用几个外部变量，包括 BSP 配置，中断向量表地址，中断向量数
- 声明 bsp_cleanup 函数
- 声明网络驱动配置

**2)** 初始化代码

c/src/lib/libbsp/m68k/MCF5307neuinfo/star.S: 这段汇编程序的核心部分就是把程序从 ROM 中拷贝到 RAM 中，并清空 BSS 段。这段程序要结合 linkcmds 一起修改。

整个初始化流程分析

start.S 汇编语言部分是第一部分，它完成四个工作

1. 初始化堆栈
2. 清零 BSS 段
3. 禁止外部中断
4. copy 数据从 ROM 到 RAM

c/src/lib/libbsp/m68k/<your BSP>/startup/init5307.c: 系统初始化代码将会调用 boot_card(),它完成的工作：

初始化通用的 CPU Configuration Table 和其它的 Configuration Table

调用 bsp_start()

这个程序最重要的功能是确定 RTEMS 的工作区在内存中的位置完成这个程序的执行后，将返回到 boot_card()

然后调用 rtems_initialize_executive_early()初始化执行，C 库函数和所有的设备驱动，并不初始化多进程和允许中断。

调用 main()

将不再返回直到执行结束

调用 bsp_cleanup()

清除内存区，执行结束

**3)** 串口控制台移植

lib\libbsp\m68k\MCF5307buaa\console\console.c: 这个程序是 console 的驱动程序，主要功能有：

声明两个 MCFuart 结构，static MCFuart uart[2]

声明 console 的 3 种操作模式

- 0 不基于 termios 的轮训输入和输出
- 1 基于 termios 的轮训输入和输出
- 2 基于 termios 的中断式输入和输出

接下来把把在\lib\libcpu\m68k\MCF5307\console\MCFuart.c 实现的函数进行封装，提供给上层标准的内部 API

MCFuart_poll_read，轮循读函数的封装，该函数返回从 UART 读到的字符，或者-1 如果没有有效的字符。

MCFuart_interrupt_write，中断写函数的封装，该函数将写缓冲区的数据写到串口去，成功则返回 SUCCESS，否则返回-1。

console_poll_write，轮循写函数的封装，该函数在轮循模式下按位传送缓冲区里的值，成功则返回 SUCCESS，否则返回-1。

console_set_attribut，串口属性设置函数的封装，这个函数分析终端的属性，然后对硬件进行相应的设置。设置成功则返回 SUCCESS，否则返回-1。

console_stop_remote_tx 的封装，该函数关闭远程设备的数据流，成功则返回 SUCCESS，否则返回-1。

console_start_remote_tx 的封装，该函数恢复远程的数据流，成功则返回 SUCCESS，否则返回-1。

console_first_open，该函数验证输入的参数，并完成初始化，成功则返回 SUCCESS，否则返回-1。

console_last_close，该函数关闭串口设备注销其在设备表中的信息，成功则返回 SUCCESS，否则返回-1。

console_initialize，该函数初始化 console I/O，并且在 RTEMS I/O 系统中注册这个设备，成功则返回 SUCCESS，否则返回-1。

console_open，该函数打开 console 设备，将相应的 termios 回叫信号给 termios 库，成功则返回 SUCCESS，否则返回-1。

console_close，该函数关闭 console 设备，成功则返回 SUCCESS，否则返回-1。

console_read，该函数从 console 读 RAM 数据，成功则返回 SUCCESS，否则返回-1。

console_write，该函数向每一个 console 写数据，成功则返回 SUCCESS，否则返回-1。

console_control 该函数处理 console 的 I/O 控制，成功则返回 SUCCESS，否则返回-1。

**4)** 网卡驱动的移植

由本开发小组的张新灵同学完成

## 5.6 调试工作

在白箱和黑箱调试法中，由于开放源代码的编译调试器并不为开发者提供集

成开发环境，因此本课题采用了黑箱调试法，调试的时候，应先打通串口，使得开发板可以和主机进行通讯，然后再调试每一个驱动模块。

采用 RTEMS 提供的例子 hello world。首先，编译生成 hello.exe，然后启动 BDM 调试器 m68k-bdm-rtems-gdb o-otimize/hello.exe。如果在串口控制台正确打印结果：HELLO WORLD，则说明串口已经打通。

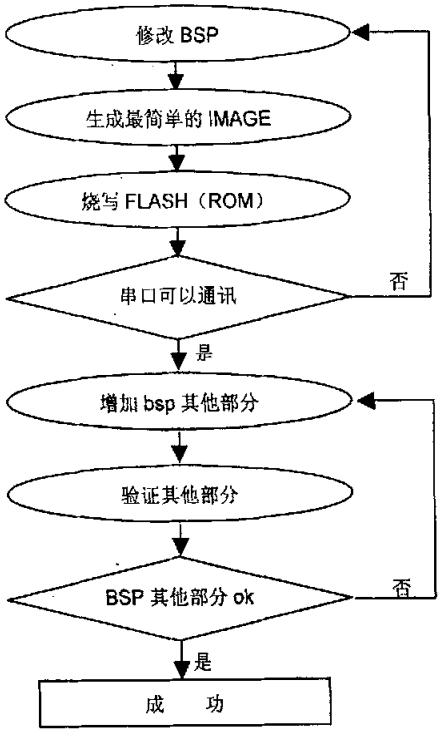


图 5.7 调试流程

Figure 5.7 Debug Flow

## 5.7 运行实现

图 5.8 为上电后从串口终端打印的系统运行图，图 5.9 为 RTEMS 的一个网络应用程序的运行图。

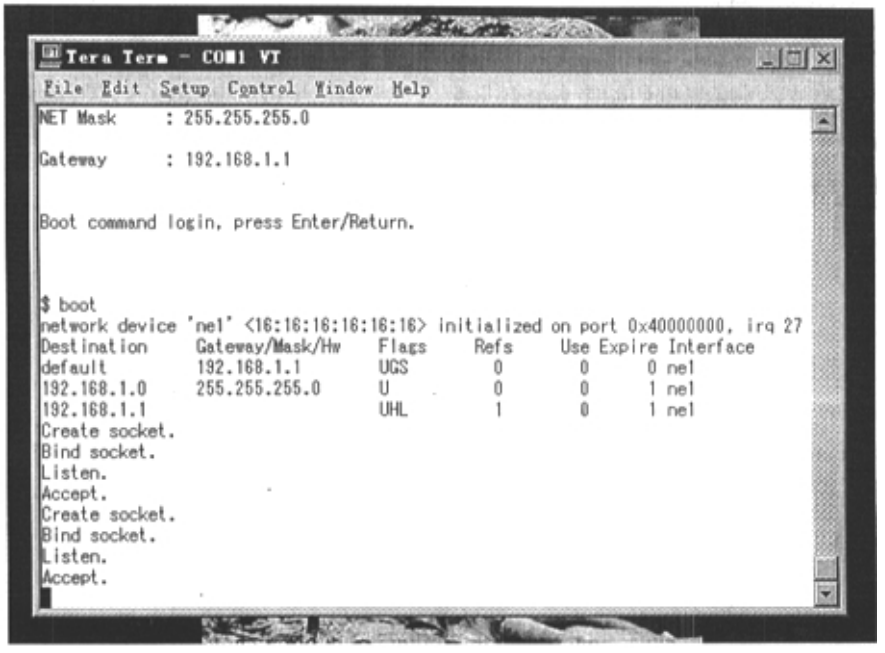


图 5.8 系统运行图

Figure 5.8 User Interface of RTEMS

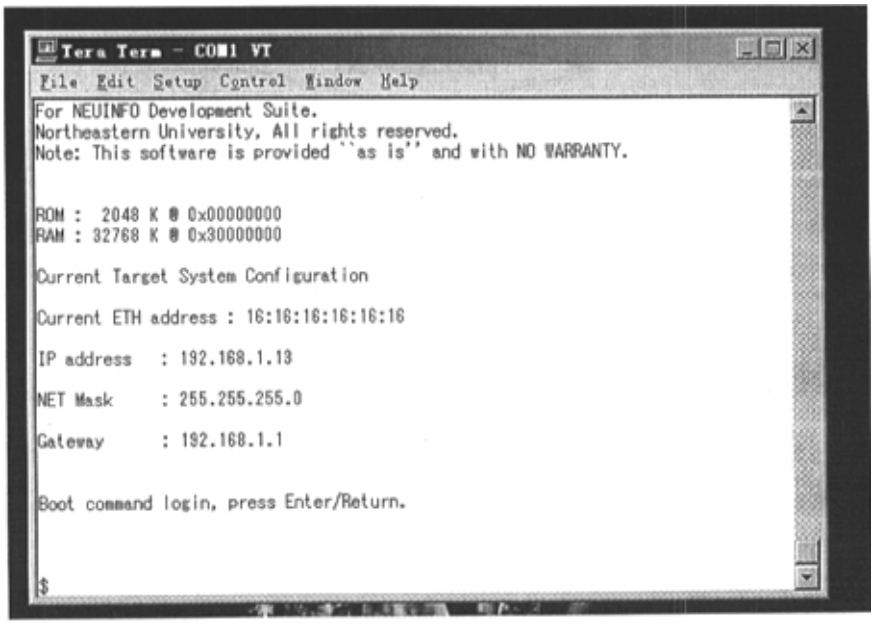


图 5.9 一个 RTEMS 应用运行图

Figure 5.9 An Application of RTEMS

# 第六章 结束语

## 6.1 论文的总结

首先，在本文的开始，从嵌入式技术的发展角度出发，分析了目前在这个领域中开发平台的重要性和必要性。针对面向工业的需求和开发平台的发展状况，提出了构建我们自己的嵌入式开发平台的目标，以便将来所开发的嵌入式的软件能有一个性能优越、开发方便、保障实时的软件开发平台。

其次，在第三、四章中，对本课题近一年来研究的支持嵌入式应用的实时操作系统及本课题最终选用的 RTEMS 操作系统做了介绍，并结合当前在嵌入式系统开发中使用的实时操作系统的主流趋势，介绍了一些实时操作系统的基本概念和原理。

然后，为了适应该开发平台的需要，对硬件平台提出了设计和实现方案。

之后，论文详细阐述了该嵌入式开发平台的设计思想与实现方法，并在自主设计的开发板上，成功实现了 RTEMS 的移植，为嵌入式软件的开发提供了强大的开发平台。目前，这个内核除向用户提供了任务创建、任务调度、任务间同步与通信和实时时钟管理等系统服务，还提供标准的 API 支持如 POSIX1003.1b, ITRON 和 RTEID，从而可以很好地满足在嵌入式软件开发方面的需求。

最后，通过实际调试证明该操作系统和硬件平台的性能是可靠的。

可以说，该嵌入式平台是对现有领域成熟技术的一次整合，这正是这个课题的创新点。就像人们最初在造火箭的时候，每一小块技术都是成熟的，但是将其进行整合并形成一个火箭却是困难的。

## 6.2 展望

经过这一年多的科研阶段，我深有感触，希望能起到抛砖引玉的作用：

一、由于 ColdFire MCF5307 处理器本身资源的限制以及 RTEMS 操作系统的局限，当前开发平台只能提供这些系统功能。随着更高档的 CPU 处理能力的提高和资源的丰富，该开发平台还会扩充新的功能。例如，如果处理器能增加浮点处理单元并且为操作系统增加一个功能强大的 shell，无疑将可以进一步提高系统的性能。

二、嵌入式实时操作系统的二次开发和嵌入式技术的应用是非常有前景的两个领域，具有深远的理论研究和实践应用价值。能否将二者结合起来，开发一种

专用于嵌入式工业软件开发的实时内核，使这个实时内核更优化地支持工业大量实时数据的处理。今后的工作将是适应嵌入式应用的潮流，完成嵌入式实时多任务内核以及嵌入式硬件开发板的组件化，作为嵌入式组件移植到真正的应用环境中。

三、随着嵌入式技术的进一步发展，嵌入式技术一定会在 21 世纪带来信息技术领域的一场革命。让我们期待着这一天的早日到来。

四、基于 RTEMS 实时操作系统的嵌入式开发平台是东北大学软件所和沈阳嵌入式工程技术中心在于戈教授的带领下对嵌入式技术领域所做的一种有意义的尝试，并有希望成功的转化为实际的工业产品，希望这种产、学、研结合的道路，对于我们的科研成果迅速转化为生产力，起到积极有效的示范作用。

# 致谢

这篇论文能够顺利的完成，首先要感谢我的父母，是他们在生活上对我无微不至的关心和在学业上对我人生道路的建议，才使得我今天可以在这里完成我的硕士论文。

我要特别感谢我的导师于戈教授。在我与于老师交往的近三年时间里，于戈老师以他渊博的知识、严谨的治学态度和雷厉风行的工作作风在学习和生活的各个方面都给予我很大的影响。他积极乐观的生活态度使我在对人生的理解上也受益匪浅，我很高兴能够在于老师的指导下完成硕士学业。

我还要特别感谢邓庆绪老师。在这个课题中，从课题研究，攻克难点到最后的论文选题、材料组织等方面，是他一直给予我细致耐心的指导。在非典肺炎流行的日子里，课题遇到了巨大的困难，是邓老师的关怀和帮助使我们走出了困境。他高度的敬业精神、一丝不苟的工作作风、平易近人的生活态度给我留下了深刻的印象，并时刻激励和鞭策着自己。

我还要感谢周兴建同学、张新灵同学和韩宗豪同学对我的帮助和指导，不论在技术上还是在论文资料的准备上他们都给了我很大的帮助。在他们那里，我体会到了团队精神的重要性。

还要特别感谢东大信息公司的邓建军工程师，在我刚刚步入嵌入式领域时给了我很好的建议和鼓励。在硬件设计和调试上，他丰富的经验给了整个课题组巨大的帮助。

感谢东大信息公司的所有老师、同学和同事，东大信息公司是一个令人振奋的集体，在这个集体中，我学到了很多的知识，取得很大的进步，并且有了一段难忘的学习和生活的经历。

最后感谢所有给予我帮助的人，是你们让我充满信心的不断迎接挑战。

# 参考文献


1. 吕京建，肖海桥著.面向二十一世纪的嵌入式系统综述[Z]. http://www.bol-system.com
2. On-Line Applications Research Corporation.《RTEMS C User's Guide》[Z].http://www.OARcorp.com. Jan. 2003
3. On-Line Applications Research Corporation [Z]. http://www.RTEMS.com. Jan. 2003
4. Motorola Inc.《MCF5307 ColdFire® Integrated Microprocessor User's Manual》. http://www.motorola.com/semiconductors. 08/2002
5. Open Source Real Time Operating Systems Overview [A]. Proc. of the 8th Int. Conf. on Accelerator & Large Experimental Physics Control Systems, 2001, San Jose, California
6. 张辉，何涛，力昌兵.嵌入式因特网产品设计思想的发展回顾和展望[J].嵌入式系统论文集，237－239.《电子产品世界》杂志社，2000 年 11 月
7. 王鹏，尤晋元，朱鹏译，Andrew S.Tanenbaum，Albert S.Woodhull 著.《操作系统：设计与实现（第 2 版）》[M] .电子工业出版社.2000 年 7 月
8. 张尧学，史美林.计算机操作系统教程[M]. 清华大学出版社,1993
9. 邹思轶.《嵌入式 Linux 设计与应用》[M]．清华大学出版社 2002.4
10. 孔祥营，柏桂枝。嵌入式实时操作系统 VxWorks 及其开发环境 Tornado[M]. 中国电力出版社，2002 年 1 月
11. Jean J Labrosse.μC/OS- II ,The Real-Time Kernel[M]. R&D Publications, Inc, 1998
12. Jean J Labrosse.Inside Real-Time Kernels [A]. Proc. of Embedded System Conf., 1999
13. David Kalinsky.A Survey of Task Schedulers [A]. Proc. of Embedded System Conf., 2000
14. Ed Klein.RTOS Design: How is Your Application Affected? [A]. Proc. of Embedded System Conf., 2000
15. William Stalling.Operating Systems Internals and Design Principles [M]. 清华大学出版社 Prentice Hall，1998
16. Mark H. Klein, Thomas Ralya, Bill Pollak, Ray Obenza and Michael Gonzalez Harbour, "A Practitioner's Handbook for Real-Time Analysis: Guide to Rate Monotonic Analysis for Real-Time Systems"[M]. Kluwer Academic Publishers,

1993

17. 邵贝贝等. 什么是嵌入式实时操作系统[J]. 单片机与嵌入式系统应用,2001(7):72-76
18. Charles Crowley. Operating Systems A Design — Oriented Approach[M]. McGraw-Hill Book Co, 1997
19. 蔡德聪. 工业控制计算机实时操作系统[M]. 清华大学出版社，1999
20. 朱巍 嵌入式实时操作系统及其应用[J].单片机与嵌入式系统应用
21. Antoine Colin. Worst-case Execution Time Analysis of the RTEMS Real-Time Operating System. Proc. of The 13th Euromicro conf. on the real time system [A], Delft, The Netherlands, June 2001