

浙江大学

硕士学位论文

基于CK510嵌入式软件开发平台的研究与设计

姓名：夏雨

申请学位级别：硕士

专业：电路与系统

指导教师：严晓浪

20070501

摘 要

随着我国集成电路设计水平的不断提高,我国已经拥有自主研发 32 位高性能嵌入式 CPU 的能力,CK510 是浙江大学和杭州中天微系统有限公司联合研发,拥有自主知识产权的一款 32 位高性能嵌入式 CPU。为了缩短基于 CK510 的用户产品方案的研发周期,扩大用户群,需要研发一整套“CK510 嵌入式软件开发平台”。

“CK510 嵌入式软件开发平台”主要功能模块包括工具链、操作系统内核、硬件驱动程序、程序库以及嵌入式图形界面 GUI。有了这些组件用户才能开发高质量、高可靠的 CK510 体系结构软件,才能快速研发出适合市场需求的高质量产品。

论文在分析比较现有的基于开放源代码的命令行工具链和集成开发环境的优缺点的基础上,设计实现一个功能强大具有集成开发环境的基于 CK510 的嵌入式软件开发平台。该开发平台有以下模块: C/C++ 语言和 CK510 汇编语言源代码编辑器、CK510 C/C++ 交叉编译工具链(C/C++ Compiler、Assemble & Linker)、CK510 体系结构二进制代码在线调试器、CK510 体系结构 eCos-2.0 实时操作系统(内核、程序库等)以及轻量级图形用户支持系统 MiniGUI。CK510 体系结构的嵌入式软件开发平台可适合应用于复杂系统的开发或者多个开发团队的合作开发,例如一个应用软件复杂,具有多个任务,并且相互之间关系紧密的系统。

整个项目由中天微系统有限公司软件组全体组员共同完成,本人在整个项目中分担的任务是 eCos 操作系统的移植以及图形用户界面 GUI 的实现,因此本论文后半部分将对这两部分内容进行重点叙述,对于其它的模块只作简单介绍。

关键词: CK510, eCos, 移植, 软件开发平台, MiniGUI

ABSTRACT

As the standard of IC design in China continuously improves, we are now able to develop 32-bit high performance embedded CPU. CK510 is a 32-bit high performance embedded CPU co-developed by Zhejiang University and Hangzhou C-Sky Microsystems Co., Ltd with independent intellectual property. In order to shorten the development time of the user application solution based on CK510 and broaden the amount of CK510 users, it is essential to develop a complete set of “Embedded Software Development Platform for CK510”.

Main functional components of Embedded Software Development Platform include tool chain, operating system kernel, hardware drivers, user interface for IDE and middleware. The powerful tool chain, OS kernel and middleware enable the users to develop software for CK510 with high quality and reliability, thus lead to quick development of high quality products to meet the demand of market.

The thesis analyzed and compared the pros and cons of existing open source command line tool chain and realized a powerful software development system for CK510. The system includes the following modules: Source code editor for C/C++ and assembly, Tool chain (compiler, assembler, linker and other binary analysis tools), on-line binary code debugger, eCos-2.0 for CK510 port. The software development system for CK510 is suitable for developing complex systems or multi-group codevelopment, e.g, a system with very complex applications and multi-task which are tightly coupled to each other.

The whole project is completed by the contribution of all members of C-SKY software group. My role in the project is the porting of eCos-2.0 and development of GUI for CK510, so the bottom half of this thesis is focused on the two parts the project, and the other modules will be referred simply.

Keywords: CK510, eCos, Transplant, Software Development Platform, MiniGUI

第 1 章 绪 论

1.1 嵌入式系统

随着多媒体技术、通讯技术相结合的信息时代的快速发展和互联网的广泛应用，我们从 PC 时代过渡到了后 PC 时代。后 PC 时代的主要特征就是用于工作和生活的小型数字智能嵌入式设备得到广泛的应用。嵌入式技术全面渗透到日常生活的每个角落，同人们的日常生活紧密结合。特别是计算机、通信和消费类电子产品随着信息技术的快速发展进一步走向融合。3C 产品的结合和一体化，已成为信息技术和信息产业发展的重要趋势。

嵌入式系统^[1]可以被定义为：以应用为中心、以计算机技术为基础、软硬件可裁减、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。

嵌入式系统的出现至今已经有 30 多年的历史了，嵌入式技术也经历了几个发展阶段。进入 90 年代后，以计算机和软件为核心的数字化技术取得了迅猛发展，不仅广泛渗透到社会经济、军事、交通、通信等相关行业，而且深入到家电和娱乐、艺术、社会文化等各个领域，掀起了一场数字化技术革命。多媒体技术与 Internet 的应用迅速普及，消费电子、计算机、通信一体化趋势日趋明显，嵌入式技术再度成为一个研究热点。纵观嵌入式技术的发展，大致经历了以下 4 个阶段^[2]。

第一个阶段是以单芯片为核心的可编程控制器形式的系统，同时具有与监测、伺服、指示设备相配合的功能。

第二阶段是以嵌入式 CPU 为基础、以简单操作系统为核心的嵌入式系统。

第三阶段是以嵌入式操作系统为标志的嵌入式系统。

第四阶段是以基于 Internet 为标志的嵌入式系统，这是一个正在迅速发展的阶段。目前大多数嵌入式系统还孤立于 Internet 之外，但随着 Internet 的发展以及 Internet 技术与信息家电、工业控制技术等结合日益密切，嵌入式设备与 Internet 的结合将代表着嵌入式技术的真正未来。

总体看来，嵌入式系统具有便利灵活、性价比高、嵌入式强等特点，可以嵌入到现有任何信息家电和工业控制系统中。从软件角度来看，嵌入式系统具有不可修改性、系统所需配置要求较低、系统专业性和实时性较强等特点。

后 PC 时代是个真实的阶段，而且是一个可以预测的时代。嵌入式系统就是与这一个时代紧密相关的产物，它将拉近人与计算机的距离，形成一个人机和谐的工作与生活的环境。从某个角度来看，嵌入式系统在传统的工业控制和商业管理领

域已经具有广泛的应用空间,如智能工控设备、POS/ATM机、IC卡等;在家庭领域^[1]更具有广泛的应用潜力,如机顶盒、数字电视、WebTV、网络冰箱、网络空调等众多消费类电子设备以及医疗保健类电子设备等;还有在媒体手机、袖珍电脑、掌上电脑、车载导航器等方面的应用,将极大地推动嵌入式技术深入到生活和工作的方方面面。此外,它在娱乐、军事方面的应用潜力也是巨大的,而且是有目共睹的。

1.2 嵌入式系统开发所面临的问题

嵌入式软件开发有别于桌面软件系统开发的一个显著的特点,是它一般需要一个交叉编译和调试环境^[4],即编辑和编译软件在主机上进行(如在PC机的Windows操作系统下),编译好的软件需要下载到目标机上运行(如在一个PowerPC的目标机上的VxWorks操作系统下),主机和目标机建立起通讯连接,并传输调试命令和数据。由于主机和目标机往往运行着不同的操作系统,而且处理器的体系结构也彼此不同,这就提高了嵌入式开发的复杂性。

由于嵌入式硬件系统千差万别,软件模块和系统资源也多种多样,要使系统能正常工作,软件开发者必须对目标系统具有完全的观察和控制能力,例如硬件的各种寄存器、内存空间,操作系统的信号量、消息队列、任务、堆栈等。嵌入式操作系统种类繁多,如VxWorks、Linux、Nucleus、WinCE等等,即使在一个公司之内,也会同时使用好几种嵌入式操作系统。如果需要同时调试多种类型的板子,每个板子上又运行着多个任务或进程,那复杂性是可想而知的。此外,嵌入式系统变化更新比较快,对开发时间要求比较紧,尤其是消费类产品更是如此。

作为嵌入式系统开发的支撑技术,嵌入式软件开发平台必须拥有一套功能强大的嵌入式软件集成开发工具以满足系统开发各个阶段的需求,即集程序源代码编辑分析、编译、汇编反汇编、链接、二进制文件分析和系统调试等功能于一体,同时还必须拥有使用方便,界面友好等优点。

鉴于以上的问题,嵌入式软件开发平台必须具备以下特性,才能满足日新月异的嵌入式系统开发市场:

- 友好,便捷的编辑,编译和调试界面
- 支持多种操作系统
- 支持多任务调试
- 对目标系统具有完全的观察和控制能力,例如硬件的各种寄存器、内存空间,操作系统的信号量、消息队列、任务、堆栈等。
- 支持硬件系统的软件仿真器

1.3 嵌入式CPU及其软件开发平台

计算机应用可以分为桌面计算、服务器和嵌入式计算机三大类。桌面计算强调定点和浮点数据类型的程序性能，不注重程序大小和处理器的功耗。服务器主要是指科学和工程计算、数据库、个人文件服务和网络应用的多用户服务系统。与前两类不同，嵌入式计算机是指嵌入在其他设备中不直接可见的计算机，这些设备可以是日常生活中的电子设备，如打印机、网络交换机，也可能是手持电子设备，视频游戏等。嵌入式计算机除了要有很强的功能，其功耗、体积、成本、可靠性、实时性、速度、处理能力等方面均受到应用要求的约束。

利用嵌入式处理器和一些专用的电路来开发嵌入式系统是嵌入式系统设计的一个重要趋势。嵌入式处理器是各类面向用户、面向产品、面向应用的嵌入式系统的核心部件。

嵌入式处理器由早期的 8 位 CPU（如 Intel 8051，PIC 等），16 位 CPU，已经步入了 32 位高性能处理器时代。目前，32 位高性能嵌入式处理器主要有 ARM、MIPS、Power PC 和 Motorola 68000 系列等，主要产商有 Motorola、ARM、MIPS、TI 和 Hitachi 等公司^[5]。国内也有很多公司开发了自主知识产权的嵌入式处理器，如杭州中天微系统有限公司的 CK510、CK520 和 CK640 等的 CK-CORE 系列 CPU。

其中以 ARM，MIPS 等嵌入式 CPU 为代表，嵌入式 CPU 提供商一般都有自己的整套嵌入式软件开发平台，以方便开发人员快速开发出正常稳定工作的应用系统，例如 ARM 公司的 ADS 等。许多第三方公司也提供商品化的开发平台，例如国外的 Metrowerks 的 CodeWarrior 集成开发环境、WindRiver 的 Bench 开发环境、QNX 的 Momentics 开发套件和 Nuclear 等产品，以及国内的 CoreTek 的 DeltaSystem 开发套件，冰河的 SIdView IDE 等产品。

1.4 CK510 32位CPU及其软件开发平台

CK-CORE处理器芯片是面向嵌入式系统和SOC应用领域的一系列高性能、低功耗的处理器，由浙江大学和杭州中天微系统有限公司联合自主研发，和摩托罗拉 MCORE CPU的指令集相兼容，但其体系结构和微体系结构作了显著改进，还增加了许多优越特性^[6]。

CK510是第一代CK-CORE处理器芯片，具有以下优越特性：

- 采用单发射，无序执行，按序退休机制。
- 具有7级流水线，且大部分指令都在一个CPU时钟里完成。
- 集成了片上指令缓存和数据缓存。指令与数据缓存大小可根据应用情况方便重构。

- 提供内存保护机制，用于保护存储器系统，它提供存储器管理单元的简单交替，并且简化了硬件和软件的结构
- 采用许多低功耗设计技术，并且提供低功耗管理机制。STOP/DOZE/WAIT 指令使处理器进入低功耗工作模式
- 采用AMBA总线，通用性强，易于SOC集成
- 使用国际JTAG标准接口用于硬件调试

为了方便用户开发高质量，高可靠的CK510体系结构软件，快速研发出适合市场需求的高质量产品，大大缩短基于CK510的用户产品方案的研发周期，有必要研发一整套“CK510嵌入式软件开发平台”。

1.5 论文主要研究内容和组织结构

论文通过分析比较现有的基于开放源代码的命令行工具链和集成开发环境的优缺点，设计实现一个功能强大的具有集成开发环境的 CK510 体系结构的嵌入式软件开发平台。

论文第 2 章主要介绍国内外嵌入式软件开发平台的研究现状和发展趋势，开发平台的结构模型，并对当前几个成熟的产品进行分析比较。第 3 章介绍了嵌入式软件开发平台的总体设计框架和实现机制。第 4 章详细分析和介绍了 eCos-2.0 操作系统的移植到 CK1000evb 开发平台的过程。第五章介绍了用户图形界面 GUI 的实现过程。第 6 章对这次嵌入式软件开发平台的设计进行简单总结，并对开发平台的研究和发展方向做了展望。

第 2 章 嵌入式软件开发平台概述

2.1 嵌入式软件开发平台的发展趋势

早期,嵌入式系统硬件技术的飞速发展和体系结构的不断改进,在很长一段时间内对嵌入式系统软件工程师而言,似乎并未产生很大影响,他们仍然沿用传统的软件设计方法和调试方法。近几年,随着硬件复杂性的增加和对系统功能和性能需求的大幅度提高,特别是要求系统具有灵活方便的网络连接、轻巧节电的无线移动和功能强大的多媒体信息处理等能力,使软件开发工作量急剧增长,甚至可以占到全部工作的 70~80%以上^[7]。这样的系统不再是单枪匹马可以完成的,需要集合一个开发团队协同工作才能完成。这也导致了嵌入式软件开发平台的几种发展趋势:

1. 平台化。

为了帮助嵌入式开发人员缩短集成时间,提高产品开发效率并加快上市进程,嵌入式软件供应商纷纷针对特定应用提供平台^[7]。

其中,风河的一大产品系列就是特定市场平台(MSP),其中包括网络设备平台(PNE)、消费设备平台(PCD)、工业设备平台(PID)、汽车信息平台(PCI)和面向军事/航空应用的关键安全平台(PSC)。每个特定市场平台均包含 VxWorks 实时操作系统,以及面向特定市场的中间件。风河这种集成化嵌入式平台不再是单一的操作系统或支离破碎的模块,而展示了平台的发展趋势。

此外,科银京成的道系统(DeltaSystem)也是一种展现相同趋势的平台,它集成了科银京成的嵌入式操作系统 DeltaOS、内核 DeltaCore、嵌入式 TOP/IP DeltaNET、嵌入式文件系统 DeltaFILE 以及嵌入式图形接口 DeltaGUI 等。

2. 标准化。

标准化也是嵌入式软件发展的一个重要趋势。风河公司认为:“加快产品上市进程和降低成本方面是如今企业高层(CxO)面临的两项关键压力,而嵌入式软件标准化可在这方面为客户提供帮助。

如在汽车电子方面,欧洲的汽车制造商们规定,将 OSEK 标准作为汽车嵌入式控制器开发的公共平台的应用编程接口(API),目前已经有风河、Accelerated Technology 和 Integrated Systems 等多家公司推出兼容的操作系统。

在军事/航空方面,国际航空无线电委员会(RTCA)制定了有关航空系统和设备软件的 DO-178b 标准。越来越多国家将该标准作为民航领域必须的资格认证。

在电信市场, IEEE 制定了 POSIX 标准, 这种标准描述了操作系统的调用服务接口, 用于保证编制的应用程序可以在源代码一级上在多种操作系统上移植运行。如凯思昊鹏的 Hopen 操作系统就支持这个标准。

而在消费电子方面, 日立、松下电器、NEC、飞利浦、三星、夏普、索尼和东芝等八家知名电子公司共同创立了消费电子 Linux 论坛(CELF), 以推广 Linux 在消费电子产品方面的应用, 目前该论坛的会员公司超过 50 家。该论坛在 2004 年年中发布了 1.0 版本的 Linux 规范。

3. 行业化

一个新的电子产品的研发成功是电子产品厂商和嵌入式软件开发平台厂商共同努力的结晶。嵌入式软件开发平台厂商做的工作越多, 电子产品厂商所必需做的工作就越少。嵌入式软件开发平台厂商必须从电子产品厂商的应用需求中提取共同的特性, 并将这些特性融入自己的开发平台之中, 只有这样才能使之在一定范围内具有适用性。

显然, 相同行业对于嵌入式软件开发的需求具有许多共同的特性, 行业领域是界定需求特征的重要依据。面向不同的行业应用, 对嵌入式软件开发平台进行有针对性的定制, 就比面向所有的行业提供相同的软件开发平台具有更好适应能力。也就是说, 通过面向行业应用, 嵌入式软件开发平台厂商可以为电子产品厂商提供更多的辅助功能和支持。

4. Linux 势不可挡

尽管在智能电话操作系统市场上, Linux 的份额仍然偏小。但就整个嵌入式软件市场而言, 代理 Monta Vista 公司嵌入式操作系统产品的北京麦克泰公司总经理何小庆认为, Linux 的应用前景非常令人看好。

例如, 此前一直坚决反对开放源软件的风河公司 2004 年 3 月宣布, 到 2005 年初将与 Linux 开发商 Red Hat 公司合作开发一种基于 Linux 的嵌入式开发平台。风河将推出的是 Linux 版本的网络设备平台(PNE Linux Edition)。业内人士甚至将此举看作 Linux 操作系统光辉时代来临的一个信号。

2.2 嵌入式软件开发平台模型

目前嵌入式软件开发平台模型主要由以下几个高度集成的部分组成:

1. 运行在宿主机的强有力的交叉开发工具和实用程序;
2. 运行在目标机上的高性能、可裁减的实时操作系统;
3. 运行在目标机操作系统之上的用户应用程序中间件;
4. 链接宿主机和目标机的多种通信方式, 如以太网、并口线、ICE 等;
5. 运行在宿主机的, 强大的、可扩展的集成环境用户界面。

各部分结构如图 2.1

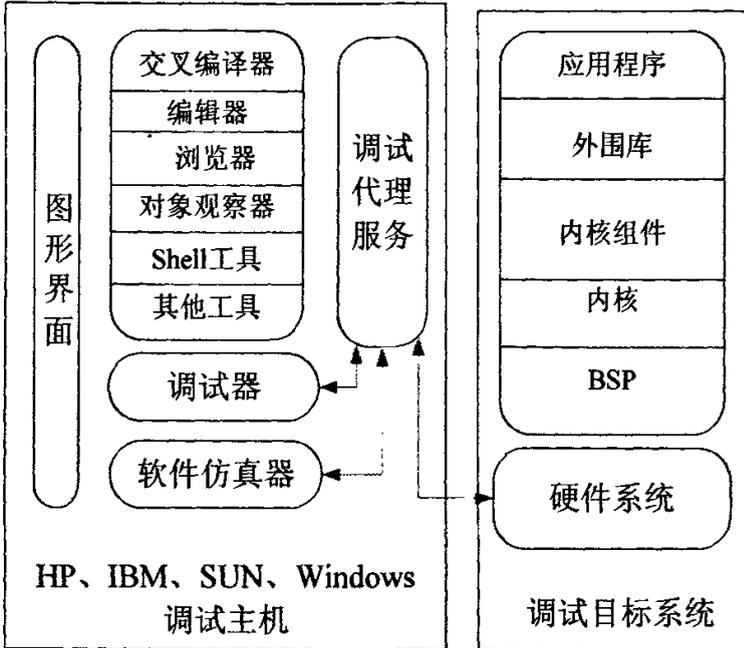


图 2.1. 嵌入式软件开发平台模型

2.3 当前几种嵌入式软件开发平台

2.3.1 WindRiver Workbench

Workbench是WindRiver公司开发的一套功能完备的嵌入式软件开发平台。它是WindRiver基于早先版本Tornado开发环境和VxWorks嵌入式操作系统产品研发而成。新一代Workbench开发平台继承了其原有的Tornado集成开发平台的一贯优势，并且功能更加强大，由于新采用了先进的Eclipse软件框架结构^[8-11]，从而使整个系统更加开放和易于扩展。

Workbench是当前嵌入式软件开发领域中功能非常强大的一个集成开发环境，它适合应用于复杂系统的开发或多个开发团队的合作开发，比如一个复杂的系统需要用到多种CPU或多种目标操作系统，或者应用软件本身非常复杂，具有多个任务，并且相互之间关联紧密，或者多个项目组之间需要进行协同开发和软件模块共享，或者企业涉及到了从硬件开发，到软件开发，再到生产测试的全过程。其优点主要体现在七“多”上，即多任务、多目标、多模式、多OS、多CPU、多连接形式、多主机环境

2.3.2 ARM ADS

ARM ADS 的英文全称为 ARM Developer Suite，是 ARM 公司推出的新一代

ARM 集成开发工具^[12], 用来取代 ARM 公司以前推出的开发工具 ARM SDT。ADS 使用 CodeWarrior IDE 集成开发环境, 现代集成开发环境的一些基本特性如源文件编辑器语法高亮, 窗口驻留等功能在 ADS 中得以体现。

ARM ADS 支持所有 ARM 系列处理器包括最新的 ARM9E 和 ARM10, 可以在 Microsoft Windows XP/2000/Me 以及 RedHat Linux 上运行。

ARM ADS 由六部分组成:

- 代码生成工具 (Code Generation Tools)

代码生成工具由源程序编译、汇编、链接工具集组成。ARM 公司针对 ARM 系列每一种结构都进行了专门的优化处理, 这一点除了作为 ARM 结构的设计者的 ARM 公司, 其他公司都无法办到, ARM 公司宣称, 其代码生成工具最终生成的可执行文件最多可以比其他公司工具套件生成的文件小 20%。

- 集成开发环境 (CodeWarrior IDE from Metrowerks)

CodeWarrior IDE 是 Metrowerks 公司一套比较有名的集成开发环境, 有不少厂商将它作为界面工具集成在自己的产品中。CodeWarrior IDE 包含工程管理器、代码生成接口、语法敏感编辑器、源文件和类浏览器、源代码版本控制系统接口、文本搜索引擎等, 其功能与 Visual Studio 相似, 但界面风格比较独特。

- 调试器 (Debuggers)

调试器部分包括两个调试器: ARM 扩展调试器 AXD (ARM eXtended Debugger)、ARM 符号调试器 armsd (ARM symbolic debugger)。AXD 基于 Windows9X/NT 风格, 具有一般意义上调试器的所有功能, 包括简单和复杂断点设置、栈显示、寄存器和存储区显示、命令行接口等。Armsd 作为一个命令行工具辅助调试或者用在其他操作系统平台上。

- 指令集模拟器 (Instruction Set Simulators)

用户使用指令集模拟器无需任何硬件即可在 PC 机上完成一部分调试工作。

- ARM 开发包 (ARM Firmware Suite)

ARM 开发包由一些底层的例程和库组成, 帮助用户快速开发基于 ARM 的应用和操作系统。具体包括系统启动代码、串行口驱动程序、时钟例程、中断处理程序等。

- ARM 应用库 (ARM Applications Library)

ADS 的 ARM 应用库完善和增强了 SDT 中的函数库, 同时还包括一些相当有用的提供了源代码的例程。

用户使用 ARM ADS 开发应用程序必须选择配合 Angel 驻留模块或者 JTAG 仿真器进行, 目前大部分 JTAG 仿真器均支持 ARM ADS。ARM ADS 的零售价为 5500 美元, 如果选用不固定的许可证方式则需要 6500 美元。

2.3.3 SIdView

SIdView 是建华科技公司资深开发人员研制的国内少有的嵌入式软件开发平台之一，它为广大嵌入式开发工程师提供了一整套完备的开发、调试方案。目前支持 ARM 系列内核的开发。

SIdView 采用主机—目标机交叉开发模式，通过仿真器支持在线仿真调试。主要有以下特点：

- 可视化的工程管理功能
- GCC C/C++交叉编译链接工具
- 兼容性
全面兼容 ARM 的 SDT、ADS 系列开发工具，支持 ARMice 硬件仿真器。
- 操作系统支持
SIdView 拥有强大的支持 uCosII、uClinux 的调试功能，主要有内核跟踪调试，驱动调试，动态任务级调试，线程调试，模块调试，链接库调试。
- 强大的代码编辑器

2.3.4 各种开发平台优缺点

WindRiver Workbench 可以说是嵌入式软件开发平台的顶尖产品之一了，它不仅支持了一般开发平台提供的功能，支持许多嵌入式处理器硬件目标机，提供高效率的实时操作系统 VXWorks 等功能，还拥有了嵌入式软件开发中许多强大的工具，例如软件逻辑分析仪，系统对象检查工具等。但是在 Linux 应用日益频繁的今天，Workbench 却没有对 Linux 操作系统的支持，而且其昂贵的价格也是许多嵌入式开发商望而却步。

ARMADS 采用了 CodeWarrior 作集成环境界面对源代码编辑，程序调试等方面提供了很大方便；但是该开发平台目前只支持 ARM 体系结构，在目前多种多样的嵌入式应用中，未免显得有些单薄了；而且 ADS 只提供了有限的 ARM 开发包，对嵌入式操作系统没有很好的支持。

SIdView 是国内少有的嵌入式软件开发平台之一，它支持了目前应用广泛的 Linux，uCOSII 等操作系统，但是在硬件平台的支持方面，也只有 ARM 体系结构，从而缩小了其在嵌入式系统应用中的发展前景。

第 3 章 CK510 嵌入式软件开发平台的系统架构

3.1 引言

嵌入式系统软件平台的多样性要求嵌入式软件开发平台设计过程中要充分考虑系统的灵活性、可移植性和伸缩性。本论文在参考国内外成熟的嵌入式软件开发平台的基础上，采用了层次化、模块化的思想，使最终实现的 CK510 嵌入式软件开发平台具有很好的独立性和可扩展性，满足了市场对嵌入式软件开发平台系统的不同需求。

本章详细介绍 CK510 嵌入式软件开发平台的结构和各功能模块的划分，并对系统运行原理和流程做了简要介绍。

3.2 CK510 处理器体系结构

CK510 是中天微系统有限公司自主研发的一款基于 MCORE 指令集，面向嵌入式应用的高性能、低功耗的处理器。CK510 具有可扩展指令，可配置硬件资源，可重新综合，易于集成等优点^[13]。其结构如图 3.1

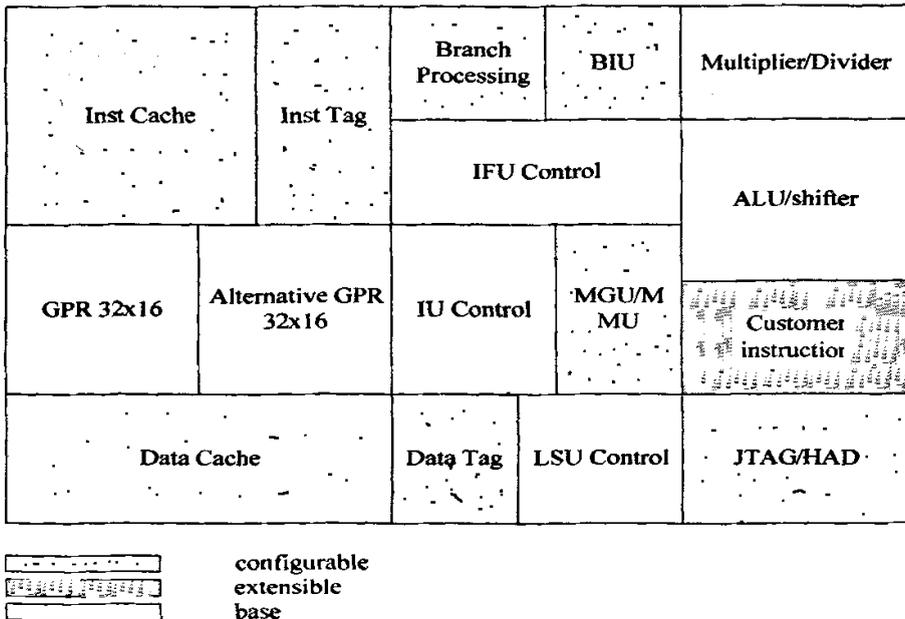


图 3.1. CK510 微体系结构

CK510 作为高性能的精简指令集计算机结构 (RISC) 的处理器，具有 16 位指令长度，7 级流水线；支持 Big Endian 和 Little Endian；拥有高性能的指令缓存和

数据缓存。

3.2.1 CK510指令集

CK510 的基本指令集由 98 条指令，可以分 4 大类，见表 3.1

表 3.1. CK510 指令集

Function Category	Function Sub-category	Instruction List
Load/Store		lrw, ld[bhw], ldq, ldm, st[bhw], stq, strn
Computational	Arithmetic	Abs, addi, addc, addu, rsubi, mult, mulsh, divu, divs, subi, subc, subu, rsub, ixh, ixw
	Arithmetic Involve Condition Bit	lnct, incf, decf, decgt, declt, decne, dect, clrf, clrt
	Logical	And, andi, andn, not, or, xor, cmphs, cmplt, cmplti, cmpne, cmpnei, tst, tstnbz
	Byte And Bit Manipulation	Bclri, bgeni, bgenr, brev, bseti, btsti, bmaski, psrset, psrclr, xtrb0, xtrb1, xtrb2, xtrb3, ffi, sextb, zextb, sexth, zexth
	Shift And Rotate	Asr, asrc, asrl, lsr, lsrc, lsri, lsl, lsli, lslc, rotli, xsr
	Data Movement	Mov, movi, movf, mvc, movt, mvcv, mfcrl, mterl
Branch /Jump	Conditional	Bf, bt
	Unconditional	Br, bsr, jsr, rte, jmp, jsri, rfi, jmpj
Special	Power	Doze, stop, wait
	Others	Bkpt, dly4, sync, trap

3.2.2 CK510流水线结构

CK510 的 7 级流水线结构由 IF, IS, RF, EX, LF, LS, WB 组成。如图 3.2

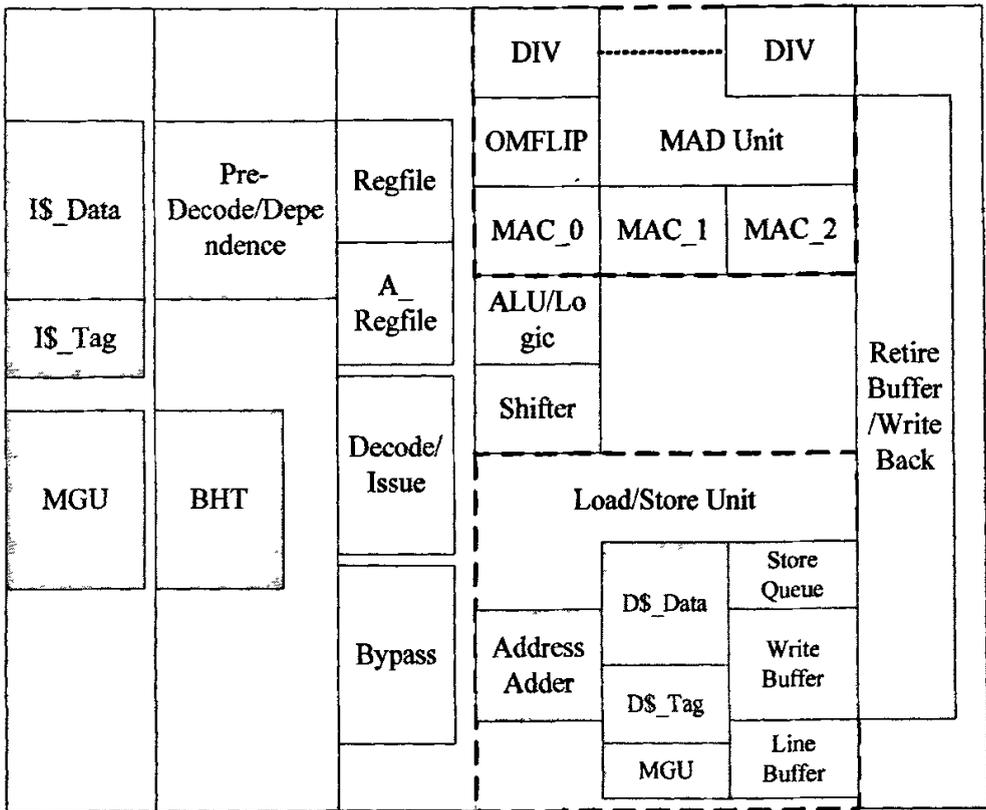


图 3.2. CK510 流水线体系结构

指令提取单元 (IFU)，由 IF 和 IS 两级流水线级组成，配备 8KB 高速缓存，采用关键指令先驱和发射已经添加指令暂存器等手段来增加指令的发射效率，利用先进二级指令跳转预测，全局和局部结合的记录来减少因为转移指令造成的流水线空闲。

存储单元 (LSU)，由 LF 和 LS 两级流水线级组成，支持不间断流水线，开辟 8 个写缓冲区，具有内部 Bypass 机制，利用快速退休加快指令的执行速度。

指令执行单元 (IU)，由 RF, EX 和 WB 三级流水线级组成，利用指令依赖表格和操作数前馈来有效地处理数据竞争，实现高性能的指令发射。高性能的指令退休通过 4 个退休缓冲器组的非序回收，并行回收，按序退休，快速退休来实现。

3.2.3 CK510编程模型

CK510 定义了两种编程权限：超级用户模式和一般用户模式。两种模式下有着各自不通的 CPU 资源访问权限，如图 3.3

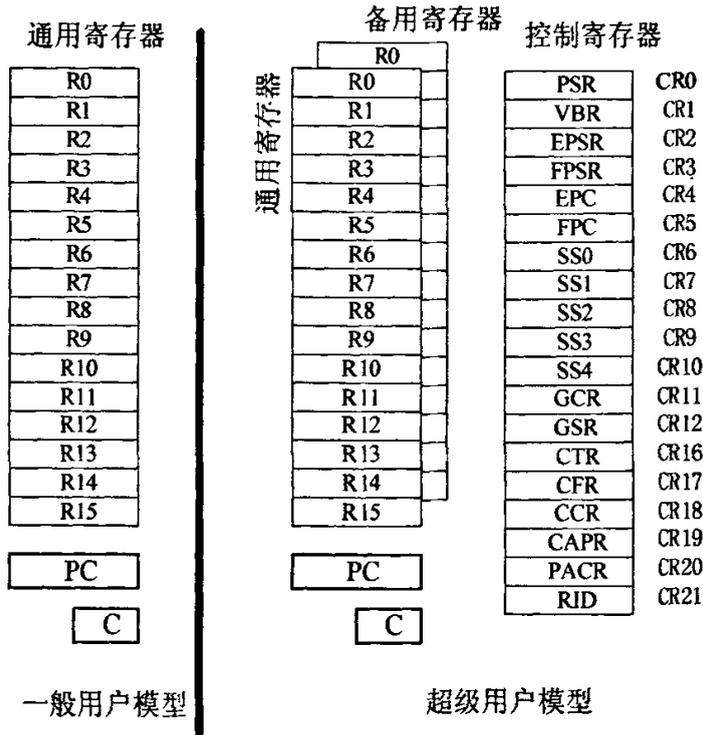


图 3.3. CK510 寄存器资源

一般用户模式只能访问 16 个 32 位通用寄存器、32 位程序计数器 (PC)、条件码 / 进位标志位 (C bit) 和用于 MAC 操作的 HI, LO 寄存器。超级用户模式可以访问所有的寄存器，从而可以利用控制寄存器进行管理。因此，通常用户程序不能访问一些受保护的系统的信息，运行于一般用户模式；而操作系统负责管理调度用户程序并提供系统调用，运行于超级用户模式。程序状态寄存器 (PSR) 中的 S 位是核心模式使能位，它决定了当前运行的指令是在何种模式下。大多数指令在两种模式都可以执行。但是一些对系统有重要影响的指令只有在超级用户模式下才能被访问。TRAP #n 指令为一般用户模式提供了访问操作系统服务程序的可控制接口。

在中断或异常发生时，处理器从用户模式转为核心模式。中断异常处理电路把 PSR 的当前值保存在 EPSR 或 FPSR 影子寄存器中，然后置 PSR 中的 S 位，使处理器进入核心模式。从中断异常返回时，执行 RTE (从异常返回) 或 RTI (从中断返回)。

备用寄存器通过设置 PSR 里的控制位 AF 来选择，当 AF 被清 0，使用原来的通用寄存器；当 AF 被置位，交替寄存器取代通用寄存器发挥和通用寄存器一样的功能，这可以大大节省中断时存取进程上下文所需的时间。在 5 个暂时寄存器用来在核心模式下处理异常。VBR 寄存器用来改变中断异常向量的地址，GCR、

GSR 两个寄存器用于全局控制和状态。超级用户模式下才能访问的寄存器用 MFCR 和 MTCR 指令来存取。

3.3 CK1000evb开发板简介

CK1000evb 是一块以 32 位高性能嵌入式 MCU—CKMCU 为核心的评估板，是我们本次研究的硬件平台。CKMCU 包含 CK510 处理器，并且集成了内部 SRAM、DMA 控制器、MAC、串口、LCD 控制器等多个外围 IP，采用流行的 AMBA 总线来连接各个外围 IP，带宽比较大的连接到 AMBA AHB 总线，带宽相对较小的连接到 AMBA APB 总线，APB 总线和 AHB 总线通过 bridge 连接。它的结构图如图 3.4 所示：

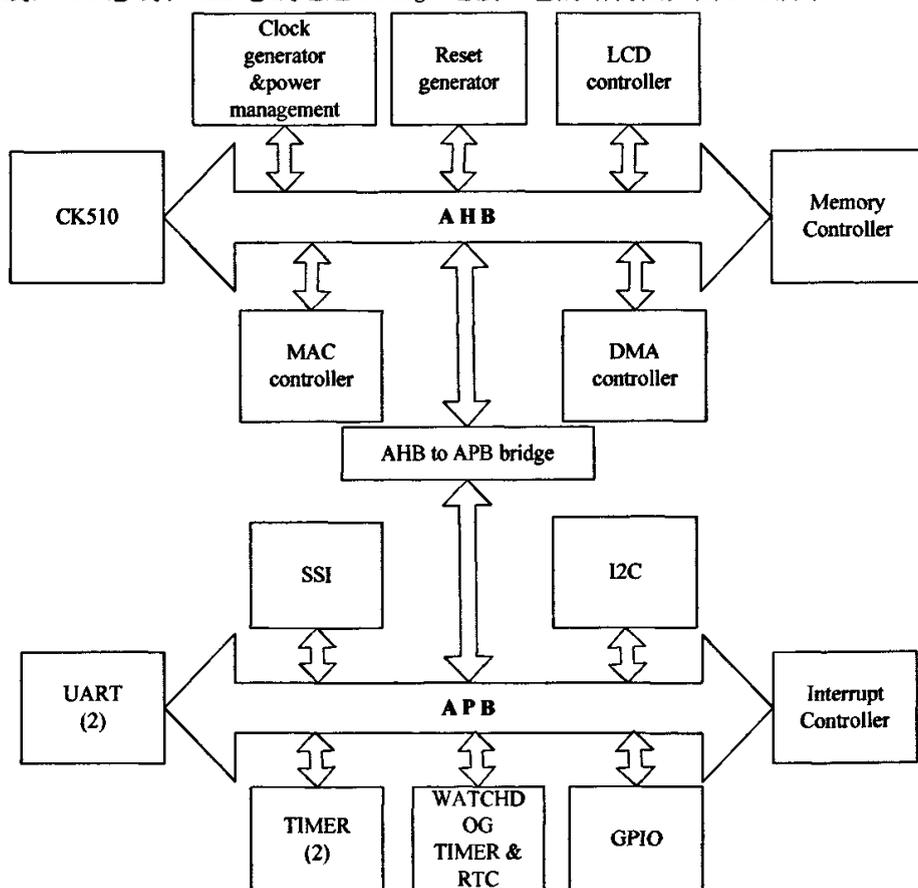


图 3.4 CKMCU 结构图

3.4 CK510嵌入式软件开发平台的总体设计

3.4.1 模块结构设计

CK510 嵌入式软件开发平台是基于高性能 32 位嵌入式 CPU CK510 而设计的，是编写 CK510 嵌入式实时应用程序的完成的开发平台。它给嵌入式开发人员提供了一个不受硬件资源限制的开发和调试环境。

整个开发平台可以分割为两部分，即宿主机运行部分和目标板部分。如图 3.5 所示，主机部分运行用户界面和 CK510 交叉编译工具，目标机部分运行操作系统和用户应用程序，主机和目标机之间通过调试代理服务程序通信。

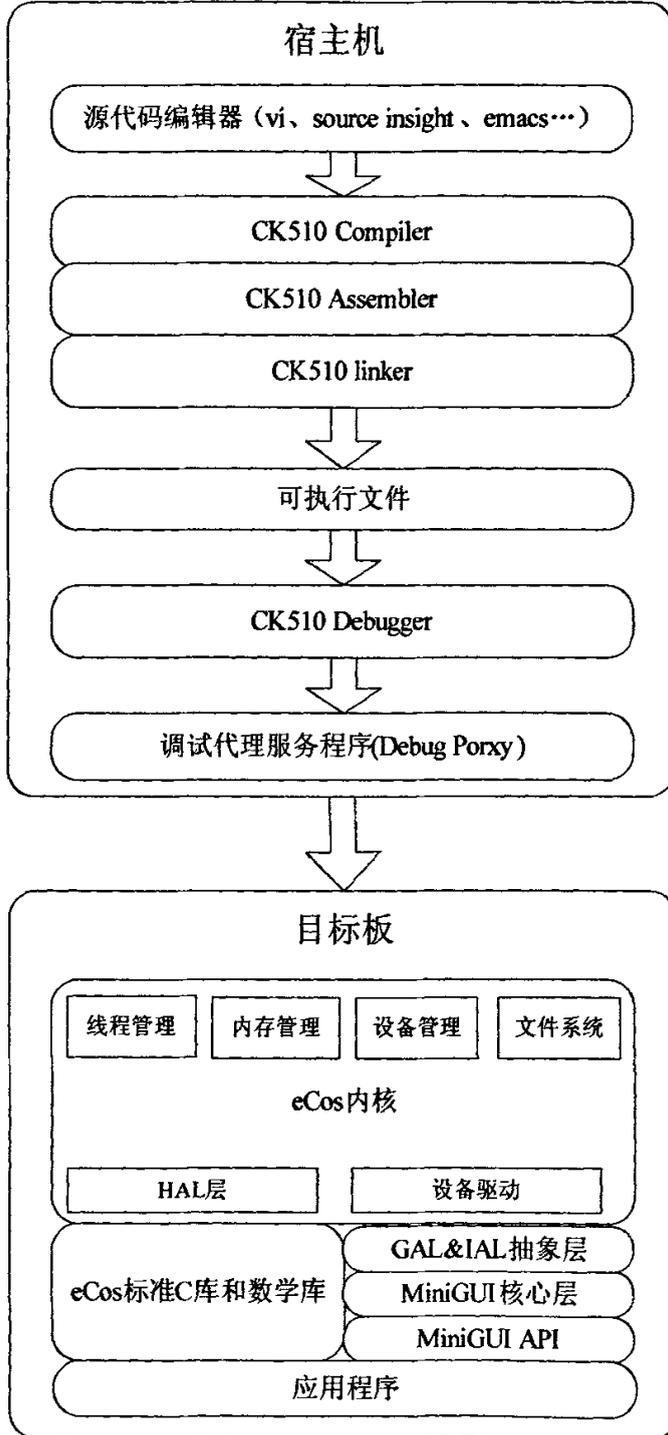


图 3.5 CK510 嵌入式软件开发平台模块结构

3.4.2 开发平台功能模块

1) 源代码编辑器

该模块提供了强大的用户源代码编辑，浏览功能，例如关键字的高亮显示、编辑提示、子函数跟踪和查找等功能。

2) C/C++交叉编译工具链 C/C++ Compiler、Assemble & Linker

开发平台为用户提供支持了 CK510 C/C++语言的交叉编译器^[14-15]，同时也支持 CK510 汇编语言的编译。交叉编译工具包括 GNU^[16]的 C/C++编译器^[17-20]，汇编器^[21]和链接器^[22-23]等核心模块，并且针对 CK510 体系结构的指令系统、流水线进行改进优化。

3) CK510 Debugger

CK510 Debugger 是一个源代码集成调试器^[24]，支持任务级和系统级调试，支持混合代码和汇编代码显示。

CK510 Debugger 沿用了 GNU Debugger 的源代码，并且进行了 CK510 体系结构的定制。用户可以使用多种调试手段对应用程序进行调试：如设置断点、查看变量的值、反汇编代码、直接看寄存器的值等。

4) 调试代理服务程序 Debug Proxy^[25-31]

调试代理服务程序是主机的调试程序和 CK510 目标机通信时的通信代理，主要负责将调试程序的调试命令进行协议转换，并且转发到目标机，并从目标机处接收硬件平台的运行数据，从而转发给调试代理服务程序。。

5) CK-eCos 操作系统

CK-eCos 指的是已经完成 CK510 体系结构移植的 eCos-2.0 操作系统，有了操作系统内核和程序库的支持，它们是应用程序的基础。

6) CK-GUI 图形用户界面

CK-GUI 图形用户界面指的是已经完成 CK510 体系结构移植的 MiniGUI 轻量级图形库，有了图形库的支持，用户可以开发出各种高质量图形界面应用程序。

第 4 章 eCos 操作系统移植

4.1 概述

对于嵌入式软件开发平台而言,开发一套面向嵌入式系统的实时操作系统是必须的,这里我们选择开源的 eCos-2.0 操作系统^[32]为开发基础,基于硬件验证开发平台为 CK510 (32 位高性能 CPU) 为核心的硬件评估板,进行一系列的操作系统开发及其之上的程序库实现。

嵌入式可配置实时操作系统 eCos(Embedded Configurable Operation System)是 RedHat 推出的面向嵌入式领域的嵌入式实时操作系统。它是一个源码开放的操作系統,具有良好的开放性、兼容性和可扩展性,可移植性强。适合于深度嵌入式应用。由于 eCos 使用开放源码的形式,降低了嵌入式产品的开发成本,eCos 在嵌入式领域中已经得到广泛应用。

eCos 操作系统的核心是一个功能全面灵活的可配置嵌入式实时微内核,按照多线程模式设计,没有用户态与内核态之分,支持实时嵌入式应用所需的线程原语、同步原语、时钟管理、任务调度机制等。eCos 中断处理机制将中断处理分为两个层次,与中断向量直接相连的中断服务程序 ISR 和滞后中断服务程序 DSR,有效提高了系统的实时性。除此之外,eCos 内核还提供了一般嵌入式应用支持,包括文件系统、内存管理、任务间通信、例外处理等。

eCos 内核是一个抢占式多任务实时内核,提供两种方式来访问内核所提供的服务,一种是应用程序直接调用内核提供的 API 接口函数,例如 `cyg_thread_create` 和 `cyg_mutex_lock` 等;另一种方式是使用 eCos 兼容层提供的标准函数,eCos 有许多软件包提供对现有 API 的兼容,如 POSIX 和 μ ITRON,应用程序可以调用一些标准函数如 `pthread_create` 来使用内核服务。

可裁剪、可配置近几年来成为衡量嵌入式操作系统优劣的一个重要指标,eCos 是一个面向应用的可定制实时操作系统,它的一个独特之处在于可配置性和配置机制,特有的组件定义语言 CDL(Component Definition Language),为开发人员提供了大量配置选项实现对底层操作系统的配置,满足了不同需求的嵌入式应用,典型的配置选项如调度器的类型、任务优先级的数目、内存分配方案等。通过 eCos 提供的专用图形化配置工具,开发人员可以非常方便的在系统源代码级进行配置和裁减,实现对底层操作系统的细粒度配置,有效地降低了操作系统代码的大小。一个简单的 eCos 系统配置可以仅占用几十 KB 的存储空间,非常适合于深度嵌入式系统应用。

eCos 的开源特性使其具有非常多的扩展功能，比如 POSIX 兼容层、各种文件系统、TCP/IP 协议栈、嵌入式 Web 服务器等，这些扩展组建为用户提供了丰富的编程接口。

ckcore-ecos-2.0 是在 eCos-2.0 基础上移植开发出来的操作系统，除了 eCos 原有的优点外，还具有更高效的底层操作模块、便捷的驱动程序开发接口等特点，同时 ckcore-ecos-2.0 为用户应用程序提供方便的开发接口和高效的执行环境，并且采用 MiniGUI 做为图形用户界面。

4.2 eCos系统结构

4.2.1 eCos层次结构

eCos 被设计成由几个主要的软件组件^[33]（如内核和 HAL）组成的可配置的体系结构。这样做的基本目的就是能利用这些可重用的软件组件来开发完整的嵌入式系统；同时也使得用户可以根据自己应用的特定需求来配置整个操作系统，创建一个最适合系统应用需求的 eCos 映象，软件中只包含需要的组件，因此大小是最精简的。eCos 系统的层次结构可以用图 4.1 来表示：

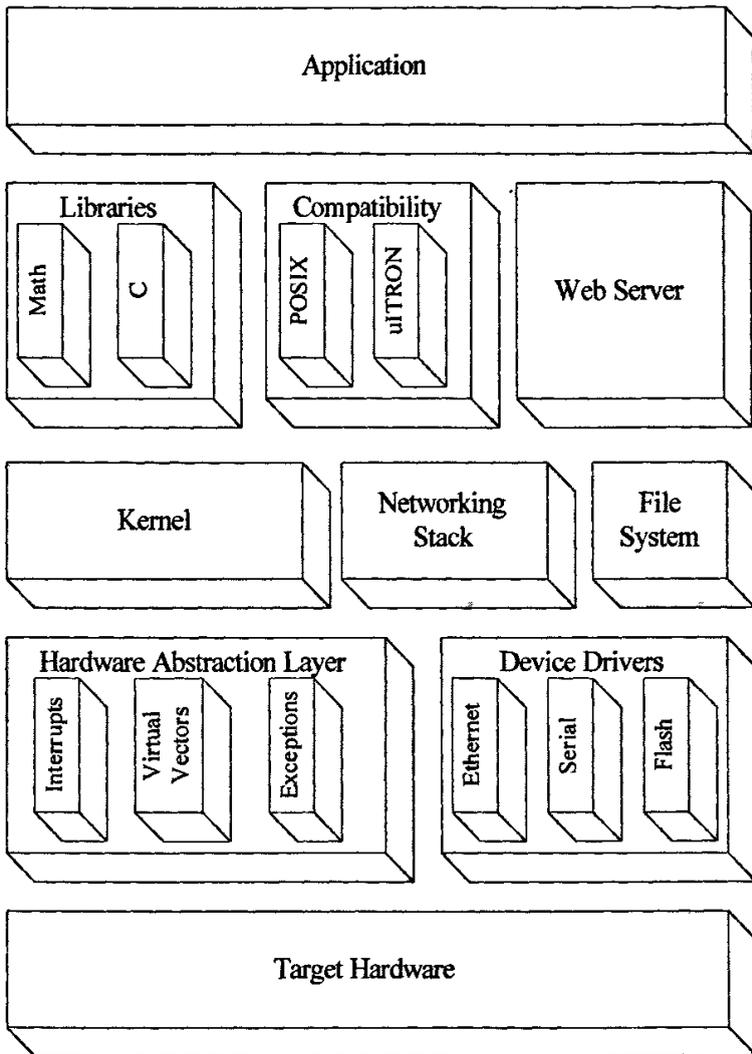


图 4.1 eCos 系统层次结构图

从图中可以看到,硬件抽象层 HAL 以及驱动程序层都是直接和硬件相关联的,它们也是移植的重点,它们屏蔽了 eCos 系统中依赖于体系结构的相关特性,向上层提供了一个统一的接口。在它们之上是内核、网络协议栈以及文件系统,其中内核是操作系统的核心,主要的功能有线程管理、内存管理、文件管理、设备控制等。在这三个模块之上有库程序, eCos 自带有 C 库和数学库,里面包含了大部分常用的库函数;还有兼容层,兼容层的作用是按照一套标准(如 POSIX, 该规范定义了一些 API, 这些 API 详细说明了应用程序如何与操作系统接口)来封装内核的功能实现规范的 API 函数,从而隔离了 eCos 内核,用来与底层 eCos 核进行接口,这样一些调用标准 API 的应用程序无须修改可以直接在 eCos 上运行;此外 eCos 在这一层面上还实现了一个 GoAhead 嵌入式 Web 服务器,用以克服 SNMP 用作远端管理的不足之处。最顶层的就是用户的应用程序了。显然,移植的工作

主要集中于底层，即 HAL 层和设备驱动程序层，尤其是一些接口函数的实现。

4.2.2 eCos组件库

eCos 具有高度的可配置性，而配置和裁减的基本单位是包，组件库(Component Repository)是一个目录结构，其中包含了在 eCos 安装时创建的所有包。组件框架中提供了一个包管理工具，他提供了对组件库中的包进行增加、更新和删除的功能。eCos 的根目录中存放了所有 eCos 发布的文件，其中子目录 package 就是用来存放组件库的，在这个目录下还有一个数据库 ecos.db，他是由包维护工具进行管理的，这个数据库文件中包含了组件库中所有包的详细信息。eCos 组件库目录的结构图如下：

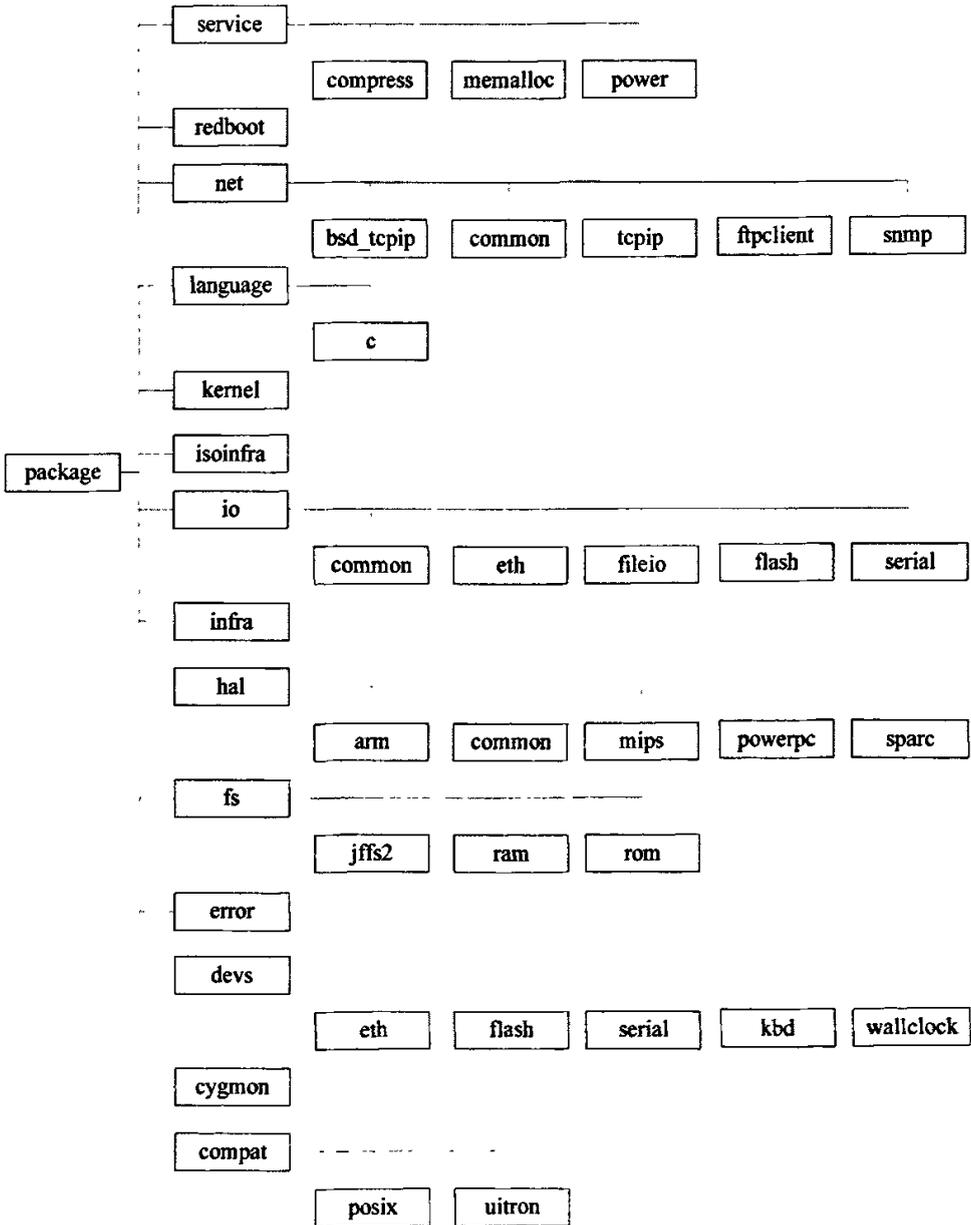


图 4.2 eCos 组件库目录结构图

从上面的结构图中可以看到，eCos 组件库下面包含了 14 个子目录，下面是对这些目录结构的详细说明：

- **compat** 目录中的文件用于兼容 POSIX 和 uITRON 的软件包
- **cygmon** 是用于调试监视器 **cygmon** 的包
- **devs** 是设备驱动程序的包，例如串口、以太网、flash 等
- **error** 是公共的错误和状态码包，在不同的包中为错误和状态的报告提供了公共的接口
- **fs** 是文件系统支持包，目前包含了 **romfs**、**ramfs** 以及 **jffs2** 三种文件系统的支持
- **hal** 是目标硬件平台的 HAL 包，包含了目标硬件的支持代码
- **infra** 是包含了公共类型、宏、代码跟踪、断言和启动选项的 eCos 基础包
- **io** 是独立于硬件设备的公共 I/O 系统支持包，包括以太网、flash、串口等，是设备驱动程序的基础
- **isoinfra** 实现了 ISO C 库和 POSIX 的包
- **kernel** 包含了内核中的核心功能，包括调度器、信号量、线程等
- **language** 包含了 ISO C 库和数学库，这样在应用程序中就可以使用标准 C 库和浮点数学库中的函数
- **net** 是网络支持包，包括网络协议 TCP、UDP、IP、SNMP 以及基于 UCD-SNMP 项目的代理支持库
- **redboot** 是调试用的 **redboot rom** 监视器
- **services** 包括了内存分配、压缩和解压缩的支持库

4.3 HAL层的移植

4.3.1 概述

HAL^[32]屏蔽了 eCos 系统中依赖于体系结构的相关特性，它向上层提供一个统一的接口，使得其它组件具有很好的可移植性。基本上说，硬件抽象层 HAL 对处理器结构和系统硬件平台进行了抽象，负责对目标系统硬件平台进行操作和控制，它具有统一的 API 接口，封装了用于实现所需功能的特定硬件操作。根据所描述的硬件对象的不同，通常将硬件抽象层分为体系结构抽象层、变体抽象层以及平台抽象层三个子层。

- 体系结构抽象层(Architecture HAL)。eCos 支持多种不同体系结构的处理器，

如 Motorola、PowerPC、ARM 等，体系结构抽象层对 CPU 的基本结构进行抽象和定义，通常包括中断的交付处理、上下文切换、CPU 的启动以及该处理器特定的指令系统等。

- 变体抽象层(Variant HAL)。同一体系处理器中特定类型的处理器具有不同的特征，变体指的是该处理器在此系列中的特殊性，例如 Cache、MMU 和 FPU(浮点部件)等方面与同系列处理器的基本结构存在差异，变体抽象层对这些特征进行了抽象。
- 平台抽象层(Platform HAL)。根据具体的嵌入式应用需要，同一处理器平台的外围硬件可能存在较大差异，平台抽象层即对当前系统的特定硬件平台进行抽象，包括平台启动、芯片选择与配置、外围电路配置、定时设置、I/O 寄存器访问以及中断控制等。

4.3.2 建立对应的目录结构

HAL 层的移植是整个移植的重点，由于 eCos 不支持我们的 ckcore，所以移植的过程就是实际上是增加一种目标体系结构的支持。

上面在介绍组件库的时候已经提到 hal 目录包含了各个硬件平台的支持代码，很显然我们的 CPU 还不在于支持之列，因此我们要建立自己的目录结构。首先我们要找一个和我们 cpu 比较接近的移植模板，比如 m68k 或者 powerpc，拷贝它们的目录，改名为 ckcore，建立我们自己的目录结构。当然，里面包含了 arch 子目录，代表了我们的 CPU 的体系结构抽象层，由于我们的 CPU 没有变体，所以只要建立平台抽象层就可以了，我们建立 ck1000evb 目录，对应我们的 ck1000evb 开发板。建立好的目录结构如图 4.3 所示，这里我们省略了各目录中代表版本号的中间目录 v2.0:

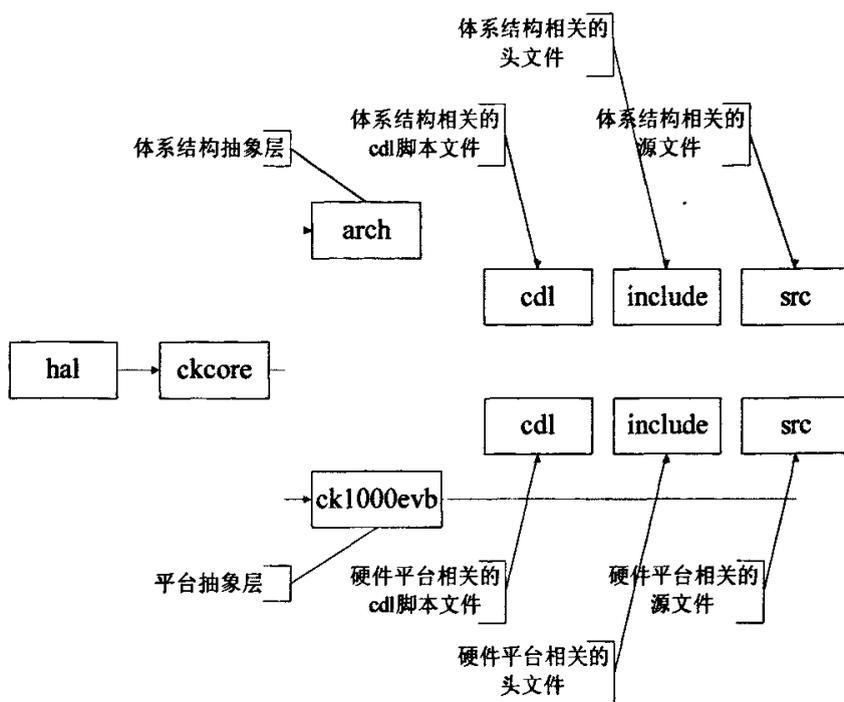


图 4.3 ckcore 目录结构图

4.3.3 初始化部分的移植

初始化部分的代码指的是内核启动以前的引导代码，以及内核启动调用的初始化接口函数^[34]。这部分的代码量不大，但是非常关键，因为只有先做好了初始化的工作，内核及应用程序才能正常运行。这部分实现的主要文件及接口函数主要有：

- **vector.S**: 这是在 `arch/src/`目录下，包含系统的引导代码，相当于 PC 机上的 `head.S`，这部分代码做的主要工作主要是为内核初始化作一些准备工作，比如说设置初始 `psr` 寄存器的值、创建异常向量表、开启指令 `cache` 和数据 `cache`、初始化堆栈、清空 `bss` 段。接着跳转到平台初始化函数 `hal_platform_init` 进行硬件平台相关的初始化，然后跳转到 `cyg_hal_invoke_constructors` 执行各个类的构造函数，最后跳转到 `cyg_start` 进入内核的初始化。这个文件全部用 `CKCORE` 的汇编代码来实现的。
- **hal_platform_init()**: 在 `ck1000evb/hal_aux.c` 文件中，这个函数的主要功能是实现 `ck1000evb` 开发板的初始化，这个函数首先调用 `hal_vsr_init()`和 `hal_isr_init()` 初始化异常和中断向量表，这两个函数的作用在后面异常和中断一节中会详细介绍，接着初始化中断控制器，然后调用调用公共函数 `hal_if_init()`完成虚拟向量表的初始化，最后通过宏 `HAL_DIAG_INIT()`调用 `hal_diag_init()`完成诊断输

出的初始化。

- `hal_diag.c`: 在 `ck1000evb/src/` 目录中, 主要的功能是实现诊断输出的底层串口驱动, 诊断输出 `diag_printf` 是 `eCos` 在真正的串口设备驱动还没有初始化以前在初始化的过程中通过串口输出信息的一种方式, 类似于 `linux` 内核中的 `printk`。由于 `diag_printf` 中的调用都通过虚拟向量表来实现, 因此这个文件主要有两个目的, 实现了串口的基本操作函数, 同时把它们填入虚拟向量表的相应位置。
- `cyg_hal_invoke_constructors()`: 在 `arch/src/hal_misc.c` 文件中, 由于 `eCos` 内核是由 C++ 编写, 因此必须在初始化内核以前调用内核中全局或静态对象所属类的构造函数来初始化这些对象。这个函数的功能就是按顺序依次调用链接时存放在 `CTOR` 段中的构造函数, 初始化所有的对象。
- `HAL_CLOCK_INITIALIZE(_period_)`、`HAL_CLOCK_RESET(_vec_, _period_)`、`HAL_CLOCK_READ(_pvalue_)`: 在 `arch/include/hal_intr.h` 中这三个接口宏和定时器 `timer` 相关, 作用分别是用 `_period_` 初始化 `timer`, 重置 `timer` 使其重新开始计数、读取当前 `timer` 的计数值并保存到 `_pvalue_` 中, 内核会调用这三个宏实现系统时钟 `real_time_clock`。

4.4 异常和中断的移植

4.4.1 概述

异常和中断^[32]的移植其实是 HAL 层移植的一部分, 我们为了表述方便把它单独列为一节。

异常是在进程执行过程中出现的一个同步事件, 中断是在进程执行过程中出现的一个异步时间, 它们都打断了正常的指令流程, 如果不能正确的处理异常和中断, 系统就会有崩溃的危险。由于每种体系结构都有不同的异常和中断机制, 因此异常和中断的移植是必须的。

`eCos` 中有两种主要的异常处理方法。一种是 HAL 和内核异常处理(HAL and Kernel Exception Handling)的结合, HAL 提供通用的硬件异常处理, 然后将控制权交给扩展异常支持的应用程序。默认的配置方法就是这样。第二种方法是应用程序异常处理(Application Exception Handling)。它由应用程序全权控制所有异常, 并将向量服务程序与硬件直接绑定。显然, 第二种由应用程序控制异常处理的方法比较灵活, 可以根据不同的应用程序选择不同的异常服务函数, 但是牺牲了稳定性, 因此在我们的移植过程中, 选择了第一种异常处理方法。

eCos 用一种分离中断处理方案将中断处理分为两部分。第一部分是 ISR；第二部分是延迟的服务例程(Deferred Service Routine,DSR)。这个方案通过减少中断服务例程内的时间来使系统具有最小的中断延时，其理念就是使得中断在 ISR 中执行的时间最小化。在该方案中，在 DSR 执行时，中断是使能的，使得系统可产生更高级别的中断，并使其在低级别的中断响应过程中可以得到处理。

eCos 提供了一种适用于所有 HAL 架构的标准中断处理方法，就是在 VSR 中填入默认的中断处理程序，然后再根据中断向量号执行相应的中断服务程序，这和上面第一种异常处理方法类似。

4.4.2 CKCORE的异常和中断机制

在 CKCORE 的体系结构中，第 2 号控制寄存器 cr2 是向量基地址寄存器(Vector Base Register)，简称 VBR，这个寄存器用于存放异常向量表的基地址。该寄存器内容低 11 位必须为 0，也就是说向量表必须按照 4k 地址对齐。设置好向量基地址后，异常或中断发生时，会根据发生的异常或中断号，去查找对应的向量，具体如下：向量地址=异常（中断）号*4+vbr。因此，系统初始化时，在 vector.S 中要正确设置 vbr 的内容，指向我们的异常向量表。

在 CKCORE 的体系结构中，支持 512 个字节的向量表包含 128 个异常向量。开始的 31 个向量是用作在处理器内部识别的向量。第 32 个向量是留给软件的，用作指向系统描述符的指针。其余 96 个向量都是留给外部设备的，比如在 ck1000evb 开发板上 32 号向量分配给 GPIO，40 号向量分配给了定时器 timer0，而 uart0 的中断向量号则是 46 等。

CKCOE 将中断分成普通中断和快速中断两种，两种中断处理略有区别。首先介绍异常和普通中断处理，异常和普通中断处理过程很相似，但也有区别，当异常发生时，CPU 会将当前执行指令的 pc（程序计数器）和程序状态寄存器 psr 分别放入控制寄存器 epc 和 epsr，然后将异常号乘以 4 再加上 vbr 的值得到异常向量地址，再取出异常向量送入 pc，同时，psr 会更新，表明当前处于异常处理状态下，相应的异常向量号也要置出，当异常处理完成后返回时，使用 rte 指令，这条指令会将 epc 和 epsr 分别拷贝回 pc 和 psr 寄存器。普通中断发生时，过程和异常处理一样，但是，送入 epc 的不是发生中断时执行指令的 pc，中断发生一定要等待 CPU 正在执行的指令完成，然后将下一条指令的 pc 送入 epc，其余过程就和异常处理相同，返回也是用 rte 指令。

快速中断发生时，会将 CPU 下面即将执行的指令的 pc 送入控制寄存器 fpc，程序状态寄存器 psr 送入控制寄存器 fpsr，其余处理过程和普通中断处理相似，但返回时要调用 rfi 指令，该指令将 fpc 和 fpsr 分别拷贝回 pc 和 psr。

4.4.3 异常的移植

上面提到 eCos 支持两种异常处理方法，显然，第二种由应用程序控制异常处理的方法比较灵活，可以根据不同的应用程序选择不同的异常服务函数，但是牺牲了稳定性，因此在我们的移植过程中，选择了第一种异常处理方法。实现的主要接口函数有：

- `hw_vsr_default`: 在 `arch/src/vector.S` 中，用汇编实现，所有异常的入口函数，主要功能先保存现场，在我们的体系结构中指的是保存 `epc`、`epsr` 以及 16 个通用寄存器 `r[0: 15]` 到当前线程的堆栈中，因为在异常服务程序中可能会使用和修改这些寄存器。接着从 `psr` 寄存器中取出异常向量号，然后作为参数传递给函数 `hal_default_exception_handler()` 然后由内核来根据向量号处理异常。最后从堆栈中恢复发生异常以前的现场，程序继续运行。
- `hal_vsr_init()`: 在 `ck1000evb/hal_aux.c` 文件中，主要作用是初始化异常向量表，把前 31 个表项都填入默认异常入口函数 `hw_vsr_default`，把后面 96 个表项都填入默认中断入口函数 `hw_vsr_interrupt`。
- `hal_default_exception_handler()`: 在 `arch/src/hal_misc.c` 文件中，主要的功能就是准备好参数调用内核 API 函数 `cyg_hal_deliver_exception()`，把异常交给内核来处理。

实现了以上接口函数以后，eCos 下异常的处理流程可以用图 4.4 表示：

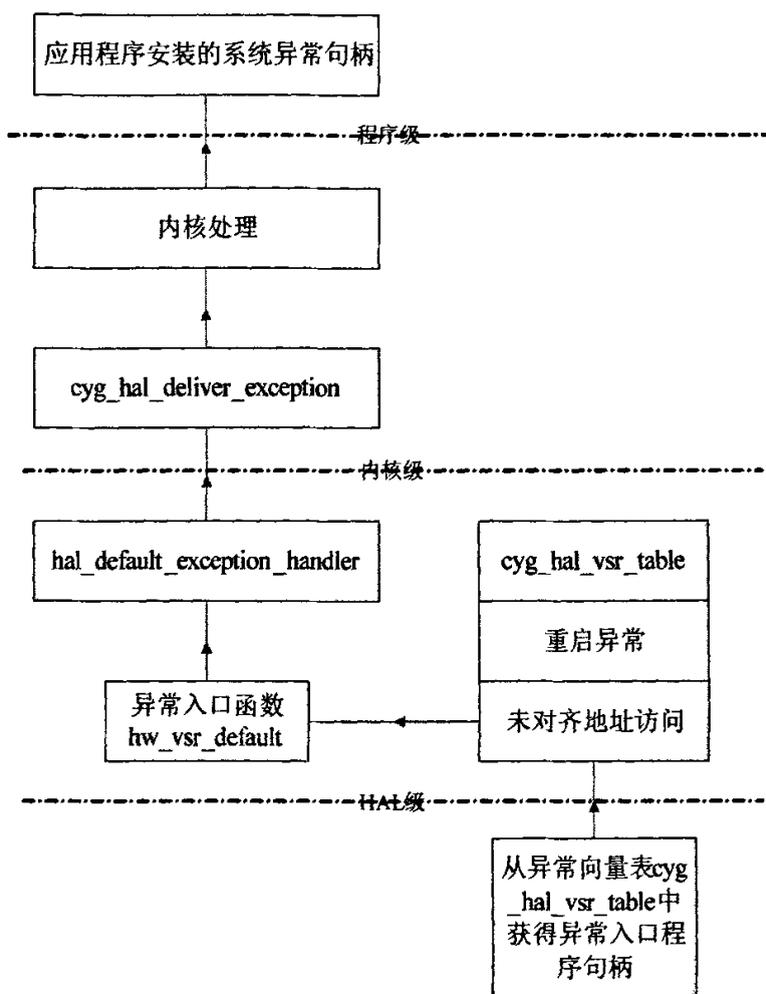


图 4.4 eCos 异常处理执行流程

最后应用程序安装的系统异常句柄指的是当采用这种异常处理方法时，内核中会有异常处理句柄指针数组 `cyg_exception_handler *exception_handler[CYGNUM_HAL_EXCEPTION_COUNT]`;括号中的宏是 hal 定义的异常数量，每个指针指向具体的异常处理句柄。应用程序可以通过内核 API 函数 `cyg_exception_set_handler()` 来设置具体向量的异常服务句柄，设置成功会替换指针数组中的指针向量，以后内核就会调用该句柄来处理异常。

4.4.4 中断的移植

- `hw_vsr_interrupt`：在 `arch/src/vector.S` 文件中，用汇编实现，类似于 `hw_vsr_default`，所有普通中断的入口函数，首先保存现场，接着根据 `psr` 寄存器中中断向量号查找中断例程向量表 `cyg_hal_interrupt_handlers`，执行相应的中断服务程序 `ISR`，然后跳转到内核服务函数 `interrupt_end()`，这个函数执行延

时中断服务 ISR 以及释放调度器,进行线程调度等。最后返回到 `hw_vsr_default` 中,恢复中断前的现场,程序继续执行。

- `hw_vsr_fastautovec`: 快速中断的入口函数,其它同上。
- `hal_isr_init()`: 在 `ck1000evb/hal_aux.c` 文件中,主要实现了中断向量表 `cyg_hal_interrupt_handlers[CYGNUM_HAL_ISR_COUNT]` 的初始化,括号中的宏代表中断的个数,在我们的体系结构中是 96 个。这个函数把所有的中断向量表都指向默认句柄 `hal_default_isr()`。

除了这两个函数以外,在 `hal` 中还定义了很多接口宏供上层调用,以下的宏定义均在头文件 `arch/include/hal_intr.h` 中:

- `HAL_ENABLE_INTERRUPTS()`、`HAL_DISABLE_INTERRUPTS(_old_)`、`HAL_RESTORE_INTERRUPTS(_prev_)`: 这三个宏定义实现的功能分别是把 `psr` 寄存器中断有效控制位 `IE` 置一使能中断,把 `psr` 寄存器中断有效控制位 `IE` 清 0 关闭中断,用参数 `_prev_` 中的对应位设置 `psr` 寄存器的 `IE` 位。
- `HAL_TRANSLATE_VECTOR(_vector_,_index_)`: 由于在我们的体系结构中,中断向量使用的是 32 号到 127 号向量,而在中断向量表 `cyg_hal_interrupt_handlers[]` 中却是从 0 开始计算的,即第一个中断 GPIO 的向量号是 32,而在中断向量表中却是第一个,因此这个宏实现的功能就是把实际的向量号 `_vector_` 转化为中断向量表中的偏移量 `_index_`。
- `HAL_INTERRUPT_MASK(_vector_)`、`HAL_INTERRUPT_UNMASK(_vector_)`: 和上面的三个宏不同,这两个宏是控制单个中断向量的使能与关闭的,它们的功能分别是设置中断控制器,使能和关闭对应参数 `_vector_` 对应的中断向量。
- `HAL_INTERRUPT_IN_USE(_vector_,_state_)`: 查询对应参数 `_vector_` 对应的中断向量表 `cyg_hal_interrupt_handlers[]` 中的表项是不是默认服务程序 `hal_default_isr()`,如果是说明该表项还没有被其它程序使用,设置 `_state_` 为 0;反之说明该表项已经被其它程序注册,设置 `_state_` 为 1。
- `HAL_INTERRUPT_ATTACH(_vector_,_isr_,_data_,_object_)`: 向系统注册中断,将中断例程 `_isr_` 填入与 `_vector_` 对应的中断向量表,同时把传递给该 ISR 的数据 `_data_` 填入对应的数据表 `cyg_hal_interrupt_data`,把该 ISR 传递给内核使用的信息 `_object_` 填入对应的信息表 `cyg_hal_interrupt_objects`。当该中断发生时,入口程序 `hw_vsr_interrupt` 会调用该 `_isr_`,同时把 `_data_` 和 `_object_` 作为参数传入。
- `HAL_INTERRUPT_DETACH(_vector_,_isr_)`: 实现和上面的宏相反的功能,向系统注销 `_vector_` 对应的中断例程 `_isr_`,重新填入默认例程 `hal_default_isr()`。

实现了上述接口函数和接口宏以后,我们以普通中断为例,eCos 中断流程如图 4.5 所示:

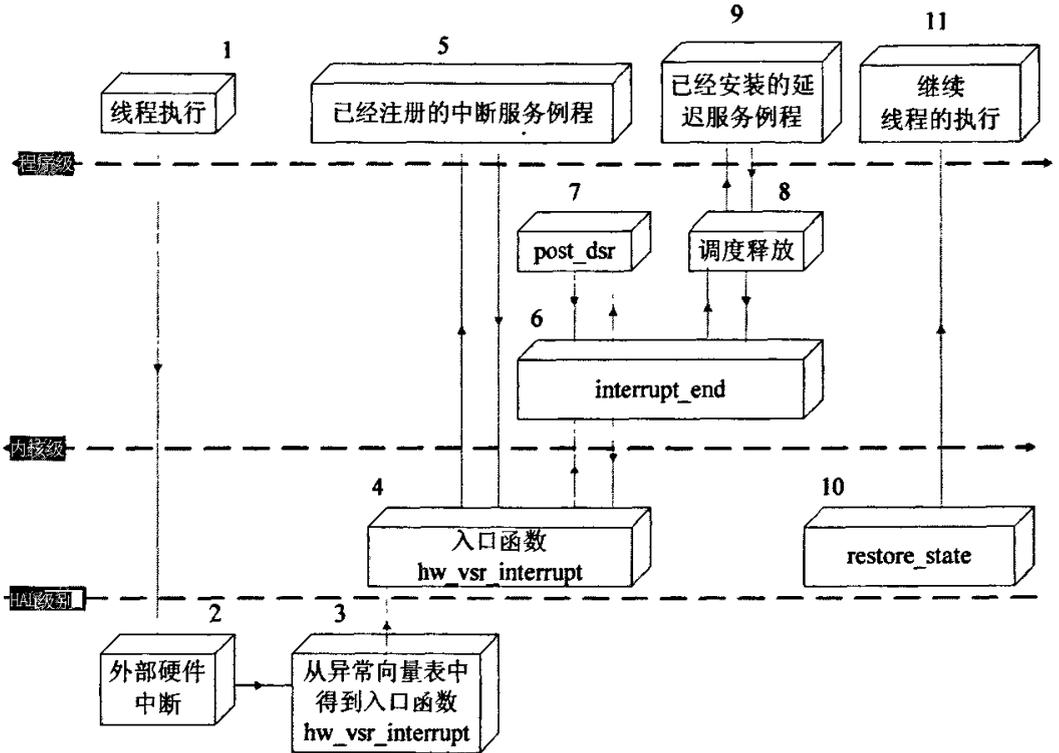


图 4.5 eCos 中断执行流程

- 1) 图 7.5 所示的第一个是线程的执行；
- 2) 发生外部中断；
- 3) 处理器在 VSR 表 hal_vsr_table 中查找，以决定执行中断向量服务程序的位置。在我们的 hal 层中，所有中断都用同一个默认中断入口函数 hw_vsr_interrupt；
- 4) 执行 hw_vsr_interrupt，保存现场；
- 5) 从 hw_vsr_interrupt 中根据中断向量号跳转到与每个向量绑定的中断服务程序，HAL 使用以数组实现的 3 个表来维护 ISR 所需要的信息，分别是
 - hal_interrupt_handlers: 包含由应用程序安装的中断服务例程的地址
 - hal_interrupt_data: 包含传递到 ISR 的数据
 - hal_interrupt_objects: 包含内核级使用的信息，该信息对应用程序是屏蔽的
- 在代码中根据中断向量号，查找这三个表，第一个表是执行的函数，后两个表的内容是函数的参数；
- 6) 从 ISR 返回后，调用 interrupt_end 函数，从这里开始进入 DSR 部分，可以开启中断了，这是在 inrt.cxx 中定义的公共函数；
- 7) 如果需要执行 DSR，则 interrupt_end 调用 pset_dsr 函数；

- 8) 从 `pset_dsr` 返回后, 函数 `interrupt_end` 解锁调度器;
- 9) 如果调度器的锁变量为 0, 那么 DSR 就执行, 如果调度器的锁变量大于 0, 那么对调度器进行了锁定的线程在这里执行;
- 10) 恢复线程上下文;
- 11) 线程继续执行。这里要注意的是 `interrupt_end` 可能启动调度器, 进行线程调度, 现在运行的线程很可能不是原来那个线程了。

4.5 线程相关的移植

4.5.1 概述

在传统的操作系统中, 进程是系统资源分配的基本单位, 也是调度运行的基本单位^[35]。但是在现代操作系统中引入了“线程”的概念, 它是进程执行运算的最小单位, 即 CPU 调度运行的最小单位。这样, 进程的功能就发生了变化, 它只是资源分配的单位, 而运行调度则是由“线程”来完成。

每个进程都自行管理虚拟内存、I/O 句柄和内存管理单元 MMU 等资源, 防止该进程被其它进程破坏, 但是这在提高了安全性的同时也降低了系统运行效率, 因为其它进程无法直接访问该进程的资源。为了提高进程执行的效率, 操作系统研究人员又设计出了一种可以独立执行, 但没有独立地址空间的轻量级进程, 这就是线程。

对比于进程, 线程的优点是需要维护的状态信息少, 线程创建和环境切换开销小, 能显著提高应用程序性能; 缺点是高精度的锁定策略引起较高的同步开销, 在性能方面有一定的损失, 线程之间受到的“MMU 保护”很少或者没有, 应用程序健壮性降低。但是对于嵌入式实时操作系统来说, 上述的缺点并不能构成困扰, 这是因为:

- 线程之间数据的访问控制可以通过精心设计的各种同步机制来保证;
- 实时性任务对于执行顺序的严格要求是第一要素, 性能方面的损失并不是主要矛盾。

下面两幅图很好的概括了进程和线程之间在资源占用上的区别:

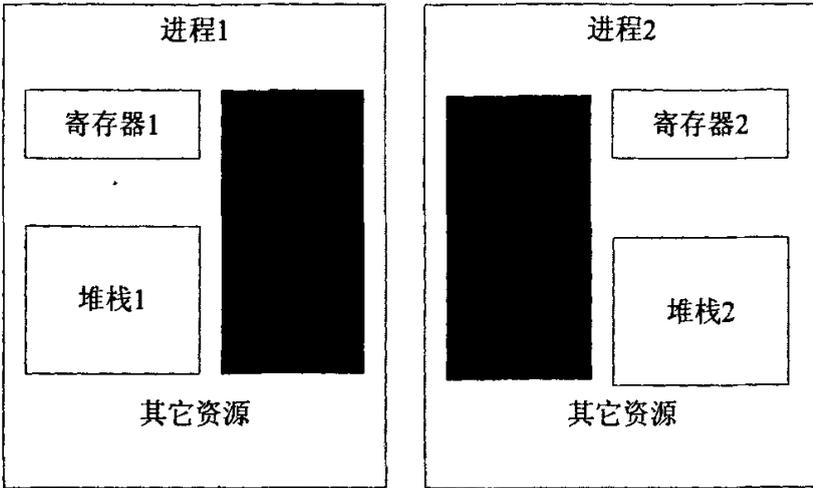


图 4.6 进程资源示意图

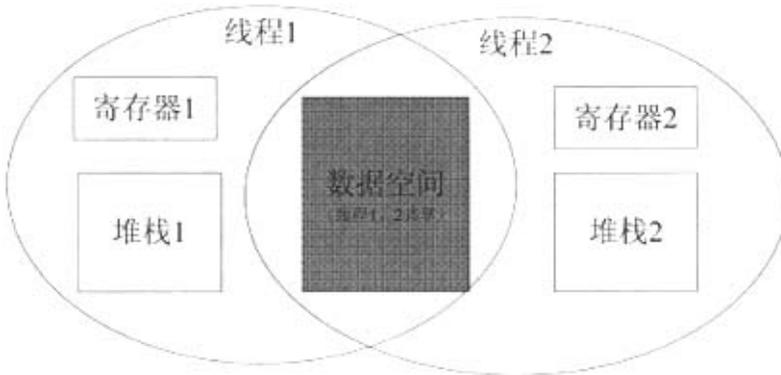


图 4.7 线程资源示意图

在 eCos 中没有进程的概念，对于执行的目标任务划分是按照线程来进行的。通过将一目标任务划分为多个线程，并通过同步单元来控制多个线程的执行顺序，并由调度器按照调度策略来调度和切换不同的线程来执行来完成所需要的目标任务。

4.5.2 移植所作的工作

由于线程调度是在内核中进行，因此和线程相关的操作函数大多在内核代码中无须移植。但是还是有部分和硬件相关的接口宏需要在 HAL 层中实现，这些宏都在头文件 arch/include/hal_arch.h 中定义。

- HAL_THREAD_INIT_CONTEXT(_sparg_, _thread_, _entry_, _id_): 这个宏的功能是用所给的参数初始化一个新线程的堆栈，其中参数 _sparg_ 是新堆栈的首地址，_entry_ 一个函数指针，指向新线程的入口程序，而 _thread_ 指向该线程的

对象，并作为参数传递给将要运行的入口程序，`_id` 则用来初始化其它的寄存器，可以作为调试的手段。

- `HAL_THREAD_LOAD_CONTEXT(_tsptr_)`: 载入一个新线程并开始运行，参数 `_tsptr_` 指向该线程的堆栈地址，由于涉及到底层寄存器的操作，这个宏是用内嵌汇编代码实现的，原理是用从上面初始化好的堆栈中按顺序取出值来设置各个寄存器，最后跳转到新 `pc` 处开始运行。
- `HAL_THREAD_SWITCH_CONTEXT(_fsptr_, _tsptr_)`: 进程切换的底层代码实现，同样是用内嵌汇编代码编写，`_fsptr_` 指向当前线程的堆栈，`_tsptr_` 指向目的线程的堆栈，原理是首先把当前线程的上下文保存到 `_fsptr_` 指向的堆栈中，然后用上一个宏的方法把目的线程载入运行。

4.6 设备驱动的开发

4.6.1 eCos 设备驱动的结构分析

操作系统的设备驱动程序^[36]通常包含以下内容：

- 1) 提供一些基本的 I/O 函数。它们负责完成以下工作：初始化和配置设备、从设备收发数据、控制设备、处理设备中断等。
- 2) 向内核注册设备。
- 3) 调用系统函数，进行设备管理。操作系统内核应提供函数支持驱动程序的同步、计时、内存管理、缓冲区管理、设备名空间及资源管理等

eCos 为应用程序提供了一组统一的 API 用于进行 I/O 操作；但是，eCos 也允许应用程序绕过统一的设备驱动程序接口，直接访问硬件。

使用统一的 API 进行 I/O 操作，可以对应用程序屏蔽设备的差别，使应用程序开发者忽略设备 I/O 操作的细节，专心进行应用程序开发，这一方面利于开发过程中分工的细化，另一方面也有利于增强代码的可移植性和可维护性，从而提高开发效率，缩短开发周期。但是，通用往往以效率为代价，统一的 API 增加了系统调用开销，延长了系统处理设备 I/O 的时间，对于一些简单的嵌入式设备，在应用程序中直接读写设备端口可能会更高效；统一的 API 在处理一些特殊设备时，缺乏灵活性，例如对显示卡的读写操作和对鼠标的读写操作完全不一样，使用统一的 API 为操作系统增加图形显示支持就显得极为笨拙。

eCos 设备驱动由 DEV 包和 I/O 包^[37]组成，它们与系统其它部分的关系如图 4.8 所示：

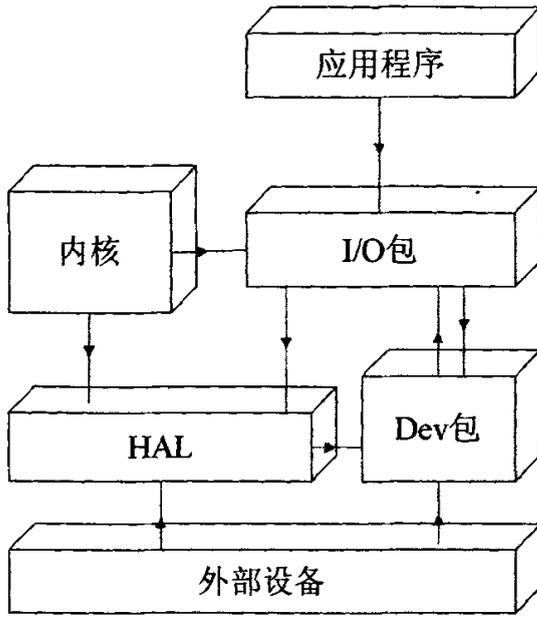


图 4.8 eCos 设备驱动体系结构

- 应用程序：使用 eCos 设备驱动的应用程序通过 I/O API 访问外设；
- eCos 内核：操作系统的核心，向设备驱动提供调度、时钟、同步、中断等内核服务的支持。但是，在某些嵌入式应用中，部分内核服务并不是必需的。为了减少代码空间，在配置 eCos 时可以选择将 kernel 包裁剪掉，这时，设备驱动所必需的内核服务由 HAL 层提供。
- HAL：HAL 包包含所有与平台相关的代码，如上下文切换和寄存器访问等。当操作系统配置不包含内核的时候，由 HAL 向设备驱动提供必需的同步与中断服务。
- I/O 包：I/O 包提供抽象的设备 I/O 操作支持。I/O 包为应用程序提供 I/O API，应用程序访问设备时使用逻辑设备名，每个设备都对应一个唯一的逻辑设备名。eCos 设备驱动支持分层的结构模型，即一个设备可以建立在另一个设备之上。这样，一个物理设备就可能对应多个逻辑设备，如“/dev/ser0”和“/dev/tty0”分别为建立在串口 0 上的两个逻辑设备。
- Dev 包：Dev 包提供设备 I/O 操作的底层实现，直接操作硬件。I/O 包和 Dev 包之间的界限并不是很严格，设备在 Dev 包中的代码甚至可以完全上移到 I/O 包中实现。但是在驱动程序设计时合理地划分 I/O 包代码和 Dev 包代码，可以增强代码的可重用性。例如，如果要在多个体系平台上为同一种设备（如串口）开发设备驱动程序，那么，将所有对逻辑设备的操作放在 I/O 包中，而只在 Dev 包中编写具体平台的 I/O 操作实现代码，就可以大大减少开发工作量。

4.6.2 eCos设备驱动的实现方法

我们知道，Linux 使用文件操作表来支持对设备的 I/O 操作，文件操作表中的每一项都对应着一个 API，编写 Linux 设备驱动就是用实际的操作函数填充文件操作表的各个项。eCos 使用设备表(DevTable)和设备 I/O 表(DevIOTable)来管理设备，每个设备的设备表项和设备 I/O 表项分别由 DEVTAB_ENTRY 和 DEVIO_TABLE 两个宏生成^[37]，和 Linux 类似，eCos 设备驱动的实现就是用实际的 I/O 操作函数填充这两个宏。

4.6.2.1 DEVTAB_ENTRY宏

设备表项由宏 DEVTAB_ENTRY 创建，每次使用 DEVTAB_ENTRY 宏添加一个设备表项，DEVTAB_ENTRY 宏的定义为 DEVTAB_ENTRY(*I*, *_name*, *_dep_name*, *_handlers*, *_init*, *_lookup*, *_priv*)，其中 *I* 为设备表项的“标签”，用于在符号表中标识该设备表项。*_name* 为设备的逻辑设备名，可以是任何有意义的字符串，一般以设备名前加一个 dev 前缀命名，例如 dev/ser0。*_dep_name* 用于分层设备，为该设备所依赖设备的逻辑设备名，如果该设备不依赖任何设备，该域为 NULL。

_handlers 为指向设备 I/O 表的指针，它连接了设备表项和设备 I/O 表。设备 I/O 表中包含了设备 I/O 操作函数的入口地址，这些函数直接操作物理设备，通过 *_handlers* 域，I/O 包的各个 I/O API 可以调用这些函数完成具体的 I/O 操作。因此 *_handlers* 也是连接逻辑设备和物理设备的桥梁。

_init 是设备的初始化函数，它负责初始化硬件和绑定中断。而 *_lookup* 为应用程序调用 `cyg_io_lookup()` 时设备的处理函数，它实际上提供了设备“打开”的操作，在应用程序使用设备前，确保设备为可用状态，并执行进行设备复位，清理缓冲区的操作。

_priv 为私有数据。*_priv* 是 eCos 设备驱动程序中一个很有用的域。设备表项中的其它域都有确定的类型，只有 *_priv* 为 void 类型的指针。不同的设备具有不同的特性，管理设备需要的信息也因而不尽相同，由于 *_priv* 为 void 类型的指针，因此它可以携带任何类型的指针，这使得设备表项具有了灵活的可扩展性，可以保存更丰富的设备相关信息。例如，鼠标驱动程序需要保存鼠标的当前坐标值和鼠标允许移动的范围，除此之外，还要保存鼠标的中断向量、中断处理句柄等必要的信息。这时，可以考虑为鼠标驱动程序定义一个数据结构，专门保存上述信息，在使用 DEVTAB_ENTRY 宏时，将这个数据结构作为一个指针传递给 *_priv*，这就

扩展了鼠标的设备表项，方便了设备管理。

4.6.2.2 DEVIO_TABLE宏

设备 I/O 表项由 DEVIO_TABLE 宏来创建，每次使用 DEVIO_TABLE 宏添加一个设备 I/O 表项。DEVIO_TABLE 宏的定义为：

```
DEVIO_TABLE(_I, write, read, select, get_config, set_config)
```

_I 为设备 I/O 表的“标签”。write、read、get_config 和 set_config 分别为读、写、获取 I/O 信息、设置 I/O API 的回调函数入口地址。对于设备不提供的操作，相应的域为 NULL。

结合以上的分析，eCos 设备驱动的实现方法可以分为以下几个步骤：

- 1) 在深入了解设备硬件特性基础上，选择适当的中断模型，定义保存设备专门信息的数据结构；
- 2) 调用 DEVTAB_ENTRY 宏和 DEVIO_TABLE 宏向系统注册设备；
- 3) 为 init、lookup、write、read、get_config 和 set_config 分别编写设备相关的代码；如果设备使用了中断，根据所选择的中断处理模型，为设备编写中断处理函数。

4.6.3 设备驱动的具体实现

eCos 的设备可基本分为字符设备、块设备和网络设备三种。目前 eCos 的字符设备主要包括鼠标、键盘、串口等；块设备包括硬盘、闪存等。网络设备在 eCos 里做专门的处理，eCos 支持的网络协议和驱动程序之间有专门定义的数据结构传递数据。针对我们的 ck1000evb 开发板，本次移植实现的驱动程序有串口、以太网卡、Flash。其中 Flash 的驱动将在后面 jffs2 文件系统中介绍，这里我重点阐述串口和以太网卡驱动的具体实现过程。

4.6.3.1 串口驱动的实现

串口属于字符设备，它是开发板与宿主机通信的主要手段，也是调试程序的有力工具。因此，我在移植过程中把串口的驱动放在了首位来开发。

上面提在 I/O 包中已经实现了逻辑串口设备相关的很多功能，只留了一些底层相关的接口函数需要我们在 Dev 包中实现。串口驱动的 I/O 层驱动在 package/io/serial/目录下，而且在 I/O 层驱动中已经通过 DEVIO_TABLE 设置了抽象串口设备 I/O 表项，也就是说由于串口设备的共性，所有的设备可以共用一个

设备 I/O 表项, 这样我们就不要再单独调用 DEVIO_TABLE 宏来产生设备 I/O 表项了。此外在 I/O 层驱动中还实现了两个针对串口设备的宏来抽象串口设备的功能, 简化驱动的开发, 如下所示:

- SERIAL_FUNS(*_l*, *_putc*, *_getc*, *_set_config*, *_start_xmit*, *_stop_xmit*): 这个宏实际上定义了一个底层函数统一接口, 其中 *_l* 是标签, 后面的参数都是底层的串口 I/O 操作函数, 需要在 Dev 包中实现。
- SERIAL_CHANNEL_USING_INTERRUPTS(*_l*, *_funs*, *_dev_priv*, *_baud*, *_stop*, *_parity*, *_word_length*, *_flags*, *_out_buf*, *_out_buflen*, *_in_buf*, *_in_buflen*): 这个宏抽象了一个使用中断的串口通道, 第一个参数是标签, 第二个参数是上面宏所代表的函数跳转表, 第三个参数 *_dev_priv* 指向具体设备的私有数据结构, 接下来的五个参数 *_baud*, *_stop*, *_parity*, *_word_length*, *_flags* 分别表示串口的波特率、停止位、奇偶校验、数据位长短以及额外的标志位, 最后四个参数代表了输入和输出缓存的地址和大小。

我们所作的工作就是在 Dev 包中加入我们的串口驱动目录, 实现调用上面两个宏所需要的函数以及调用 DEVTAB_ENTRY 宏来注册设备。具体如下所示:

- *ckcore_serial_init*: 对应 DEVTAB_ENTRY 宏中的 *_init* 域, 串口初始化函数, 实现的主要功能包括串口中断服务程序的注册、调用 *ckcore_serial_config_port* 函数进一步设置串口的各个寄存器的值。
- *ckcore_serial_config_port*: 这个函数首先解析传进来的参数, 包括串口的波特率、停止位和奇偶校验, 然后根据这些参数来设置串口的各个寄存器。
- *ckcore_serial_lookup*: 对应 DEVTAB_ENTRY 宏中的 *_lookup* 域, 执行“打开”串口的操作。
- *ckcore_serial_ISR*、*ckcore_serial_DSR*: 串口中断的服务程序 ISR 和延迟服务程序 DSR。

下面的五个函数依次对应 SERIAL_FUNS 宏中的各个域:

- *ckcore_serial_putc*: 从串口输出一个字符。
- *ckcore_serial_getc*: 从串口得到一个输入的字符。
- *ckcore_serial_set_config*: 调用 *ckcore_serial_config_port* 设置串口。
- *ckcore_serial_start_xmit*、*ckcore_serial_stop_xmit*: 使能和禁止串口发送数据。

4.6.3.2 以太网卡驱动的实现

eCos 的以太网网络模型^[38]主要由 Dev 包底层驱动程序、I/O 包高层驱动程序、网络层协议、应用层协议、传输层协议和网络应用程序组成。它的层次结构图如下所示:

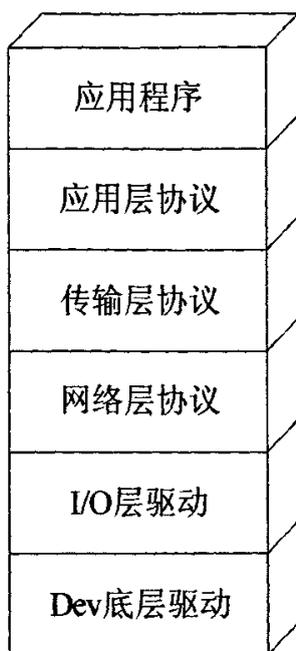


图 4.9 eCos 网络模型

eCos 网络设备的管理和其它设备有很大区别，它不是通过上面 `DEVTAB_ENTRY` 宏和 `DEVIO_TABLE` 宏来注册设备，而是自己在 I/O 包中实现了一套设备注册和管理方法。eCos 网卡 I/O 层驱动代码保存在 eCos 源代码的 `packages\io\eth` 文件路径下，主要文件有 `eth_drv.h`、`netdev.h` 和 `eth_drv.c`。下面介绍两个重要的宏—`NETDEVTAB_ENTRY` 和 `ETH_DRV_SC`，这两个宏定义了设备入口表登记和底层驱动程序的函数统一接口。

每个网络设备通过 `NETDEVTAB_ENTRY` 宏来向系统注册，它的定义为 `NETDEVTAB_ENTRY(_l, _name, _init, _instance)`，参数 `_l` 为设备表名；参数 `_name` 为设备名；`_init` 为初始化函数名；`_instance` 是用宏 `ETH_DRV_SC` 定义的底层驱动接口表名字。

底层驱动程序函数接口表由另一个宏 `ETH_DRV_SC` 来实现，它的定义为 `ETH_DRV_SC(sc, priv, name, start, stop, control, can_send, send, recv, deliver, poll, int_vector)`，参数 `sc` 为专用底层驱动函数接口表的变量名；参数 `priv` 为具体设备的私有数据指针；参数 `name` 为逻辑设备名，应用程序可以通过 `/dev/name` 来访问该设备；从参数 `start` 开始到 `poll` 为止都是底层 I/O 操作函数，需要在底层驱动中实现；最后一个参数 `int_vector` 代表设备的中断向量号。

eCos 正是利用这两个宏登记了网卡底层驱动的主要函数。下面就以介绍数据发送与接收过程来说明主要函数的作用。

网卡数据的发送过程是当网卡准备好并启动后，由上层应用程序首先发出发送数据的命令。然后调用 `can_send()` 函数判断网卡是否空闲可以发送数据。如果网卡

正忙,请求上层数据等待;如果网卡空闲,即调用 `end()` 函数发送数据到网卡的硬件发送口中。由于发送中断已开放,当网卡发送完成后,程序会进入发送中断 ISR 函数。ISR 函数主要功能是判断此次发送过程是否成功。如果不成功往上层程序报告错误信息;如果发送成功,只对硬件设备进行中断响应,清除中断信号,返回等待下一次中断信号。

网卡数据的接收过程是当网卡准备好并启动后,网卡的接收中断已开放,等待网卡硬件接收中断信号。当有数据被接收后,程序会进入网卡接收 ISR 中断服务程序。ISR 接收中断函数主要负责判断接收过程是否成功以及对硬件设备中断信号的响应和清除。如果接收过程不成功,程序则立刻返回不作任何处理;如果接收成功,程序则调用 DSR 延迟服务函数。在 DSR 中断服务函数中,程序已不占操作系统中的核调度资源,因而 DSR 函数与 `deliver()` 函数执行了比较花费时间的数据搬移工作,此时数据将被搬到 `sg_list` 的上层链表中,并通知上层有数据已被接收。

在我们的驱动代码中,首先调用 `ETH_DRV_SC` 宏来实现底层驱动函数接口表,当然具体的 I/O 操作函数必须由我们自己实现,它们的函数名和作用如下所示:

```
ETH_DRV_SC(ckcore_eth_sc, // 变量名
            &ckcore_eth_priv_data, // 设备私有数据结构指针
            "eth0", // 设备逻辑名
            ckcore_eth_start, // “打开”设备
            ckcore_eth_stop, // “关闭”设备
            ckcore_eth_control, // 解析参数并设置设备状态
            ckcore_eth_can_send, // 查询是否可以发送
            ckcore_eth_send, // 发送一个数据包
            ckcore_eth_recv, // 接受一个数据包
            ckcore_eth_deliver, // 查询设备状态并调用相关处理函数
            ckcore_eth_int, // 调用 ckcore_eth_deliver 轮询设备状态
            ckcore_eth_int_vector); // 返回设备中断向量号
```

然后调用 `NETDEVTAB_ENTRY` 向系统注册设备,

```
NETDEVTAB_ENTRY(ckcore_eth_netdev,
                 "ckcore-eth",
                 ckcore_eth_init,
                 &ckcore_eth_sc);
```

其中第一个参数是标签名,第二个参数是实际设备名,第三个参数 `ckcore_eth_init` 是设备的初始化函数,负责设备中断服务函数的注册、创建定时器来统计设备信息、设置网卡的各个寄存器以及设置网卡的物理地址等。最后一个

参数就是上面创建的函数接口表变量的地址。

4.7 文件系统的实现

一个操作系统最重要的部件莫过于进程管理模块和文件系统^[39]了，现代操作系统中一般都包含这两个组件。eCos 中自带了三种文件系统，分别是 RAM 文件系统、ROM 文件系统和 JFFS2 日志闪存文件系统^[40]。RAM 和 ROM 文件系统都是在内存中模拟的文件系统，RAM 文件系统由于它的易失性，一般只是作为 VFS（虚拟文件系统）来使用，而 ROM 文件系统是只读的，相对来说，JFFS2 文件系统是针对闪存的文件系统，可读写而且数据能保存，在嵌入式系统中有广泛的应用，因此我们选择 JFFS2 作为我们的文件系统。

4.7.1 JFFS2文件系统简介

JFFS2 是一个由 RedHat 推出的开源项目，它是一个日志结构(log-structured)的文件系统，包含数据和原数据(meta-data)的节点在闪存上顺序的存储。JFFS2 之所以选择日志结构的存储方式，是因为对闪存的更新应该是 out-of-place 的更新方式，而不是对磁盘的 in-place 的更新方式。

JFFS2 将文件系统的数据和原数据以节点的形式存储在闪存上，具体来说节点头部的定义如下：

幻数屏蔽位	节点类型
节点总长度	
节点头部CRC校验码	

幻数屏蔽位：0x1985 用来标识 JFFS2 文件系统。

节点类型：JFFS2 自身定义了三种节点类型，

- JFFS2_NODETYPE_INODE: INODE 节点包含了 i-节点的原数据(i 节点号, 文件的组 ID, 属主 id, 访问时间, 偏移, 长度等), 文件数据被附在 INODE 节点之后。除此之外, 每个 INODE 节点还有一个版本号, 它被用来维护属于一个 i-节点的所有 INODE 节点的全序关系。
- JFFS2_NODETYPE_DIRENT: DIRENT 节点就是把文件名与 i 节点对应起来。在 DIRENT 节点中也有一个版本号, 这个版本号的作用主要是用来删除

一个 dentry。具体来说，当我们要从一个目录中删除一个 dentry 时，我们要写一个 DIRENT 节点，节点中的文件名与被删除的 dentry 中的文件名相同，i 节点号置为 0，同时设置一个更高的版本号。

- JFFS2_NODETYPE_CLEANMARKER: 当一个擦写块被擦写完毕后，CLEANMARKER 节点会被写在 NOR flash 的开头，或 NAND flash 的 OOB(Out-Of-Band) 区域来表明这是一个干净，可写的擦写块。

节点总长度：包括节点头和数据的长度。

节点头部 CRC 校验：包含节点头部的校验码，为文件系统的可靠性提供了支持。

JFFS2 的挂载过程分为四个阶段：

- 1) JFFS2 扫描闪存介质，检查每个节点 CRC 校验码的合法性，同时分配了 struct jffs2_inode_cache 和 struct jffs2_raw_node_ref
- 2) 扫描每个 i 节点的物理节点链表，标识出过时的物理节点；对每一个合法的 dentry 节点，将相应的 jffs2_inode_cache 中的 nlink 加一。
- 3 找出 nlink 为 0 的 jffs2_inode_cache，释放相应的节点。
- 4 释放在扫描过程中使用的临时信息。

4.7.2 FLASH设备驱动的实现

由于 JFFS2 是针对闪存的文件系统，所以必须要以 FLASH 作为承载介质。在我们的 ck1000evb 开发板上已经安装了 SST 公司的，型号 39VF3201，大小 4M 的 FLASH 芯片，在挂载文件系统以前，必须在 eCos 操作系统中实现它的驱动程序。

FLASH 属于块设备，它的驱动相对于串口和网卡来说比较简单，因为各个厂商生成的 FLASH 芯片差别很小，因此绝大部分的功能都可以在 I/O 包中抽象出来实现，只留了少量的涉及到具体硬件 I/O 操作的接口函数需要每个设备在 Dev 包中实现。

FLASH 的 I/O 操作比较特殊，对 FLASH 读可以直接像读 memory 一样直接读取，但是对它写就必须先把原先的数据擦除，执行一个 erase 操作，然后发送写命令，才能开始写数据。

FLASH 设备的 I/O 层驱动在 package/io/flash 目录中，主要的文件有 flash.c 和 flashiodev.c，在 flash.c 文件中实现了 FLASH 设备本身的一些抽象操作函数，如 flash_init、flash_erase、flash_program 等。而 flashiodev.c 则实现了 FLASH 设备作为一个块设备应该支持的一些 I/O 函数，可以理解为更上一层的函数，如 read、write、select 等，并通过宏 BLOCK_DEVIO_TABLE 和宏 BLOCK_DEVTAB_ENTRY 向系统注册该块设备。另外在 I/O 层中还定义了一个代表 FLASH 的抽象数据结构 struct flash_info，每个设备要根据自己的型号在初始化时设置这个数据结构。

而对于 Dev 层来说，开发具体 FLASH 设备的驱动就是实现 flash.c 中所要调用的全部底层 I/O 操作函数。具体的函数和功能如下：

- flash_hwr_init: 实现 FLASH 芯片的初始化，检查型号和配置的一致性，然后根据型号设置全局 struct flash_info 结构变量 flash_info。
- flash_query: 检查 FLASH 芯片型号和配置是否一致。
- flash_erase_block(void* block, unsigned int size): 从片内地址 block 开始，擦除 size 字节大小的数据。
- flash_program_buf(void* addr, void* data, int len): 从地址 data 开始复制 len 字节数据到片内地址 addr 处。

4.7.3 JFFS2 文件系统的挂载

实现了 FLASH 设备驱动以后，就可以尝试挂载 JFFS2 文件系统了，首先在图形配置工具中加入 JFFS2 filesystem 包以及 Zlib compress and decompress 包，因为 JFFS2 文件系采用了压缩技术存储。此外还要在 FLASH 驱动包中选择对应块设备的逻辑名、片内起始地址以及总的大小，即使得 JFFS2 能够通过设备逻辑名来访问 FLASH，默认的设备名是“/dev/flash1”。

在宿主机平台下使用 mkfs.jffs2 工具创建 JFFS2 文件系统映象，然后通过网卡或者串口下载到 FLASH 芯片中，下载的地址必须和配置时设置的 FLASH 块设备片内起始地址一致。应用程序可以通过函数 mount(“/dev/flash1, “/”, “jffs2”)把设备挂载到根节点，然后就可以正常的读写文件和目录了，比如在映象中根目录下有一个 test.c 文件，应用程序可以直接用 open(“/test.c”, O_RDWR)以可读写方式打开目标文件，然后用 read 和 write 函数读写该文件。

第 5 章 嵌入式GUI的移植

随着嵌入式系统的广泛应用, PDA、机顶盒、DVD/VCD 播放机及 WAP 手机已经迅速普及。图形用户界面 (GUI) ^[41] 的广泛流行, 是当今计算机技术的重大成就之一。它极大的方便了非专业用户的使用, 因此实时嵌入式系统对 GUI 的需求越来越明显, 而这一切均要求有一个高性能、高可靠性的 GUI 的支持。

由于嵌入式系统对实时性要求非常高, 对 GUI 的要求也更高。这些系统一般不希望建立在庞大累赘的、非常消耗系统资源的操作系统和 GUI 之上, 比如 Windows 或 X Window, 太过庞大和臃肿。这样, 对轻型 GUI 的需求更加突出。另外嵌入式系统往往是一种定制设备, 它们对 GUI 的需求也各不相同。有些系统只要求一些图形功能, 而有些系统要求完备的 GUI 支持, 因此, GUI 也必须是可定制的。嵌入式系统对 GUI 的基本要求包括轻型、占用资源少、高性能、高可靠性以及可配置。

5.1 GUI的选择

目前市场是比较常用的嵌入式 GUI 系统主要有: MiniGUI^[42]、MicroWindows、OpenGUI 及 QT/Embedded 等。下面简单的介绍这几种 GUI 系统。

- **MiniGUI:** MiniGUI 是由原清华大学教师魏永明先生开发的, 一种面向嵌入式系统或者实时系统的图形用户界面支持系统。它可以运行在任何一种具有 POSIX 线程支持的 POSIX 兼容操作系统上, 也是国内最早出现的几个自由软件之一。
- **MicroWindows:** MicroWindows 是一个著名的开源嵌入式 GUI 软件, MicroWindows 提供了现代图形窗口系统的一些特性。MicroWindows 采用分层设计方法, 基本上用 C 语言实现。它的主要缺点是作为一个窗口系统, 窗口处理功能还需要进一步完善, 比如缺少控件的支持、键盘和鼠标的驱动还很不完备。
- **OpenGUI:** OpenGUI 是基于 Linux 操作系统的一个功能强大的图形库, 它基于一个用汇编实现的 x86 图形内核, 提供了一个高层的 C/C++ 图形接口。OpenGUI 提供了二维绘图原语、消息驱动的 API 及 BMP 文件格式支持, 同时支持鼠标和键盘事件。它的优点内核用汇编实现, 运行速度快, 但是这也让可移植性受到了影响。

- **QT/Embedded:** QT/Embedded 是著名的 QT 库开发商 Trolltech 开发的面向嵌入式系统的 QT 版本。这个版本的主要特点是可移植性强，但是该系统不是开源的，如果要使用这个库，需要支付昂贵的授权费用。

基于以上分析和比较，我们决定采用 MiniGUI 作为我们的用户图形界面，它的主要优点是：

- ◆ 遵循 GPL 条款的纯自由软件。
- ◆ 跨操作系统支持，具体包括普通嵌入式 Linux/uClinux、VxWorks、eCos、uC/OS-II、pSOS、ThreadX、Nucleus、OSE 等，同时还提供 Win32 平台上的 SDK 开发包，方便嵌入式应用程序的开发和调试。
- ◆ 多运行模式支持，为了适应不同的操作系统运行环境，MiniGUI 可配置成三种运行模式：MiniGUI-Threads、MiniGUI-Processes 及 MiniGUI-Standalone。
- ◆ 内建资源支持。我们可以将 MiniGUI 所使用的资源，诸如位图、图标和字体等编译到函数库中，该特性可提高 MiniGUI 的初始化速度，并且非常适合 uCOS-II/ThreadX 等无文件系统支持的实时嵌入式操作系统。
- ◆ 提供了完备的多窗口机制和消息传递机制。
- ◆ 提供常用的控件类，包括静态文本框、按钮、单行和多行编辑框、列表框、组合框、菜单按钮、进度条、滑块、属性页、工具栏、树型控件、月历控件、旋钮控件、酷工具栏、网格控件、动画控件等。
- ◆ 对话框和消息框支持。
- ◆ 多字符集和多字体支持。
- ◆ 多种汉字输入法的支持，包括内码、全拼、五笔等。
- ◆ BMP、GIF、JPEG 等常见图象文件的支持。
- ◆ Windows 的资源文件支持，如位图、图标、光标等。
- ◆ 可配置性，可根据项目需求进行定制配置和编译。
- ◆ 小巧，包含全部功能的库文件大小在 300KB 左右。
- ◆ 采用分层结构设计，具有很好的移植性。

5.2 MiniGUI 的软件架构

5.2.1 基于 MiniGUI 的嵌入式系统结构

MiniGUI 的可移植性是通过合理的软件架构^[43]来实现的，通过抽象层和底层的操作系统分离开来，如图 5.1 所示：

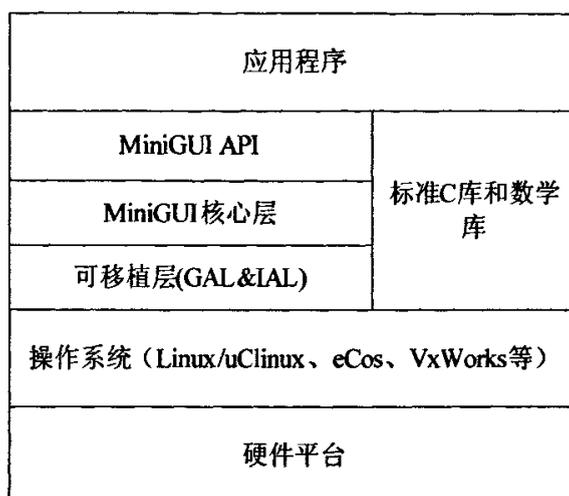


图 5.1 MiniGUI 和嵌入式操作系统关系图

基于 MiniGUI 的应用程序一般通过 标准 C 库和数学库、操作系统和驱动程序接口以及 MiniGUI 自身提供的 API 来实现自己的功能；MiniGUI 中的“可移植层”可将特定操作系统及底层硬件的细节隐藏起来，而上层应用程序则无需关心底层的硬件平台输出和输入设备。

在 MiniGUI 系统的开发中，引入了了图形和输入抽象层（Graphics and Input Abstract Layer, GAL 和 IAL）的概念。抽象层的概念类似 Linux 内核虚拟文件系统的概念。它定义了一组不依赖于任何特殊硬件的抽象接口，所有顶层的图形操作和输入处理都建立在这些抽象接口之上。而用于实现这一抽象接口的底层代码称为“图形引擎”或“输入引擎”，类似操作系统中的驱动程序。这实际是一种面向对象的程序结构。利用 GAL 和 IAL，开发的 GUI 系统可以在支持不同输入设备的各种硬件平台上运行。

另外，MiniGUI 特有的运行模式概念，也为跨操作系统的支持提供了便利。

5.2.2 MiniGUI 的运行模式

如前所述，和 Linux 这样的类 UNIX 操作系统相比，一般意义上的传统嵌入式操作系统具有一些特殊性。举例而言，诸如 uClinux、uC/OS-II、eCos 等操作系统，通常运行在没有 MMU（内存管理单元，用于提供虚拟内存支持）的 CPU 上；这时，往往就没有进程的概念，而只有线程或者任务的概念，这样，MiniGUI 的运行环境也就大相径庭。因此，为了适合不同的操作系统环境，我们可将 MiniGUI 配置成三种不同的运行模式^[42]：

- **MiniGUI-Threads**：运行在 MiniGUI-Threads 上的程序可以在不同的线程中建立多个窗口，但所有的窗口在一个进程或者地址空间中运行。这种运行模式主

要用来支持大多数传统意义上的嵌入式操作系统，比如 VxWorks、ThreadX、Nucleus、OSE、pSOS、uC/OS-II、eCos 等等。当然，在 Linux 和 uClinux 上，MiniGUI 也能 MiniGUI-Threads 的模式运行。

- MiniGUI-Lite。和 MiniGUI-Threads 相反，MiniGUI-Lite 上的每个程序是单独的进程，每个进程也可以建立多个窗口，并且实现了多进程窗口系统。MiniGUI-Lite 适合于具有完整 UNIX 特性的嵌入式操作系统，比如嵌入式 Linux、VxWorks 6 等 6。
- MiniGUI-Standalone。这种运行模式下，MiniGUI 可以以独立任务的方式运行，既不需要多线程也不需要多进程的支持，这种运行模式适合功能单一的应用场合。比如在一些使用 uClinux 的嵌入式产品中，因为各种原因而缺少线程支持，这时，就可以使用 MiniGUI-Standalone 来开发应用软件。

显然，对于只有线程的 eCos 操作系统，我们选择 MiniGUI-Threads 这种运行模式。

5.2.3 MiniGUI 窗口系统

在 MiniGUI 中窗口组织为层次体系结构的形式。根窗口作为所有窗口的祖先，除了根窗口以外的所有窗口都有父窗口，每一个窗口都可能有子窗口、兄弟窗口、祖先窗口和子孙窗口等。在同一级的窗口可以重叠，但是某个时刻只能有一个窗口输出到重叠区域。

MiniGUI 中有三种窗口类型：主窗口、对话框和控件窗口（子窗口）。主窗口通常包括一些子窗口，这些子窗口通常是控件窗口，也可以是自定义窗口类。应用程序还会创建其它类型的窗口，例如对话框和消息框。对话框本质上就是主窗口，应用程序一般通过对话框提示用户进行输入操作。

5.2.4 MiniGUI 通讯机制

MiniGUI 下的通讯是一种类似于 Win32 的消息机制，对于运行在线程模式的 MiniGUI 版本，线程间的消息传递模型如下图所示，其中的 Desktop 线程充当一个微服务器，所有的消息在 Event 线程获取出来以后就会投递给 Desktop 线程，然后再分发到目的应用程序主窗口上面，如下图所示。

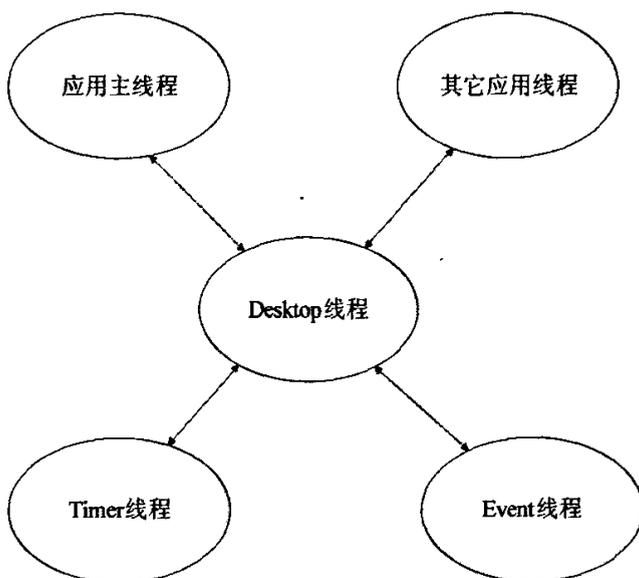


图 5.2 MiniGUI-threads 运行模式的消息通讯机制

5.3 MiniGUI 的移植过程

MiniGUI 的移植分为两个层次，一个层次是系统级的，主要指对多线程体系、消息队列等的相关移植工作、另一个层次就是可移植层的，主要指 GAL 和 IAL 的移植^[44]。

在我们的系统中，eCos 已经提供了 MiniGUI 使用的 POSIX 标准的 C 多线程支持。包括线程的建立和销毁、互斥量操作接口、信号量操作接口、条件量操作接口、信号操作接口。与 Linux 上的 POSIX 库相似，线程函数的名称均以 pthread_ 开头，信号量操作的函数均以 sem_ 开头。

从整体结构上看，MiniGUI 是分层设计^[45]的，层次结构见图 1。在最底层，GAL 和 IAL 提供底层图形接口以及鼠标和键盘的驱动；中间层是 MiniGUI 的核心层，其中包括了窗口系统必不可少的各个模块；最顶层是 API 即编程接口。

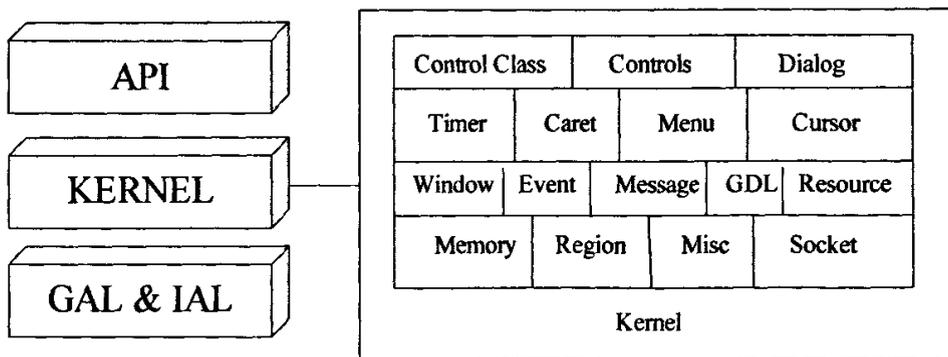


图 5.3 MiniGUI 的层次结构图

GAL 和 IAL 为 MirliGUI 提供了底层的图形接口以及输入接口，而 Pthread 是用于提供内核级线程支持的 C 函数库。利用 GAL 和 IAL，大大提高了 MiniGUI 的可移植性，并且使得程序的开发和调试变得更加容易。

在 MiniGUI-threads 的运行模式下，MiniGUI 本身运行在多线程模式下，它的许多模块都以单独的线程运行，同时，MiniGUI 还利用线程来支持多窗口。从本质上讲，每个线程有一个消息队列，消息队列是实现线程数据交换和同步的关键数据接口。一个线程向消息队列中发送消息，而另一个线程从这个消息队列中获取消息，同一个线程中创建的窗口可共享同一个消息队列。利用消息队列和多线程之间的同步机制，实现微客户服务器机制。

综上所述，本次移植的重点是 GAL 和 IAL，具体来说，我们的 ck1000evb 开发板上通过 LCD 控制器接了一块 SHARP 的、分辨率 320×240、16 位色的 TFT 液晶屏，一个太阳花的串口鼠标，我们的目标就是在 MiniGUI 上实现它们的图形引擎和输入引擎。

5.3.1 图形抽象层 GAL 的移植

我们选择的 MiniGUI 版本号是 1.3.3，这个版本已经包含了对 eCos 操作系统的支持，并且在图形引擎中有“ecoslcd”图形引擎，它采取了一种特殊的绕过驱动程序直接通过 HAL 访问 LCD 设备的方法。在图形引擎中留下了几个接口函数需要我们在 HAL 层中实现：

- `struct lcd_info`: 这个数据结构代表了一个指定的 LCD 设备，需要我们在 HAL 层中实现，里面的成员变量应该包含 `ecoslcd` 所引擎需要的全部信息，如分辨率、颜色深度、显存地址等。
- `lcd_init(int depth)`: 这个函数的功能是根据参数指定的颜色深度初始化对应的 LCD 控制器。
- `lcd_getinfo(struct lcd_info *info)`: 这个函数把 LCD 设备的参数如分辨率、色深等写入指针 `info` 所指向的对象。

首先在 eCos 主目录，`package/hal/ckcore/src/`中创建 `lcd_support.c` 文件，在 `package/hal/ckcore/include/`中添加 `lcd_support.h` 文件，然后在 `lcd_support.h` 中定义数据结构 `lcd_info`，它的成员变量和含义如下：

```
struct lcd_info {
    short height, width; // 屏幕的分辨率
    short bpp;           // 颜色深度
    short type;         // RGB 类型
    short rlen;         // 每一行的字节数
```

```
void *fb;           // framebuffer 显存地址  
};
```

针对我们的实际情况，在 `lcd_getinfo` 函数中把分辨率设成 `height=240`，`width=320`；`bpp=16`；RGB 类型指的是颜色分配策略，在我们的系统中采用的是 RGB555 的分配方法，即对 16 位色深来说，最高 1 位不用，然后红、绿、蓝三色各 5 个比特；用于我们采用 16 位色深，即每个像素需要两个字节来表示，所以一行需要 320×2 即 640 个字节来显示，所以 `r1en=640`；而最后的指针指向全部二维数组：`static short framebuffer[240][320]`，这个数组中的每一个元素代表了一个像素，对数组的操作直接反映到 LCD 显示屏上，这也符合 framebuffer 的原理。

5.3.2 输入抽象层 IAL 的移植

我们采用的是串口鼠标，因此不需要在 eCos 中额外的增加底层驱动程序，不过由于串口 0 已经被使用，鼠标只能接在串口 1 上，在原先的串口驱动中要再次调用 `SERIAL_CHANNEL_USING_INTERRUPTS` 宏来增加一个串口信道以及调用 `DEVTAB_ENTRY` 宏增加 `"/dev/ser1"` 设备，其它的 I/O 操作函数可以和串口 0 共享。在我们的开发板上没有接键盘，因此只需实现鼠标的输入引擎就可以了。

而在 MiniGUI 众多的 IAL 输入引擎中，已经包括了对老式 Microsoft 规范的串口鼠标的支持，不过要根据我们的波特率、数据位、停止位和奇偶校验在 IAL 初始化时进行设置。根据厂商的说明书，这个鼠标的工作波特率是 1200，8 位数据位、1 位停止位、无奇偶校验，而在 MiniGUI 中不应该直接操作串口设备，而是调用 `termios` 接口函数来设置，如 `tcsetattr` 函数，由于工作量很小，就不再这里展开。移植完成后鼠标输入流程图如下所示：

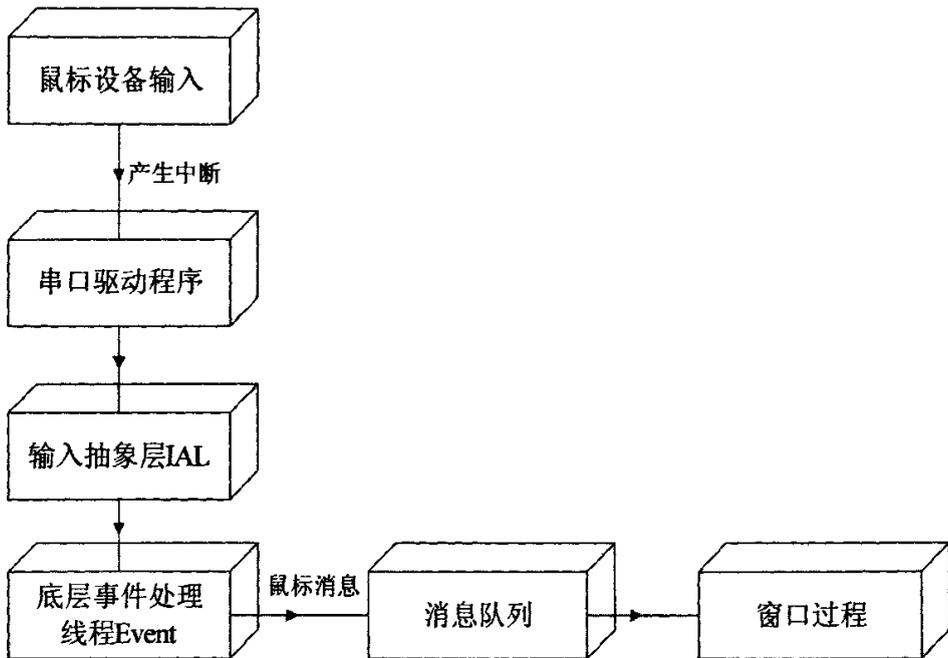


图 5.4 MiniGUI 鼠标输入流程图

5.3.3 配置文件的修改

前面提到我们已经移植 JFFS2 文件系统，但是在 MiniGUI 配置^[42]的过程中还是选择内建资源模式，因为这样在应用程序中可以减少很多文件 I/O 操作，加快 MiniGUI 的初始化速度。

因此需要对内建式配置相关的文件 `libminigui-1.3.3/src/sysres/mget.c` 依据我们的情况作相应的修改。

首先是系统配置项：`static char *SYSTEM_VALUES[] = {"ecoslcd", "console", "/dev/termios1", "MS"}`，以上四个字符串的含义是 GAL 图形引擎；IAL 输入引擎；IAL 对应的设备逻辑名，我们之所以使用 `"/dev/termios1"` 而不是 `"/dev/ser1"` 是因为需要用到 `termios` 操作，所以不用原始设备而用逻辑设备；鼠标类型。

然后是图形引擎配置项：`static char *ECOSLCD_KEYS[] = {"defaultmode"}`、`static char *ECOSLCD_VALUE[] = {"320x240-16bpp"}`，代表了我们的 LCD 显示屏的参数。

其它字符等键值不需修改，直接用默认的就就可以了，当然用户可以根据需要自行调整。

5.4 MiniGUI 的运行

完成了上述的移植以后,我们成功编译了 MiniGUI 的库程序,然后从 mde-1.3.0 中选择了扫雷程序作为测试案例,首先用编译 bomb.c 源文件产生 bomb.o 目标文件,然后把 bomb.o 和 MiniGUI 图形库和 eCos 库编译链接在一起生成 elf 可执行文件 bomb.elf,生成 elf 文件主要是为了方便调试。

然后用 mkfs.jffs2 工具把扫雷程序所需要的资源文件(都在 bomb 目录的 res 文件夹下)转换为 JFFS2 文件系统映象,我们取名为 bomb.img,并通过网卡下载到 FLASH 芯片中。

最后用调试代理服务程序通过 JTAG 接口把可执行文件 bomb.elf 下载到 ram 中运行,注意我们已经在 bomb.c 主函数中添加了 mount("/dev/flash1, "/", "jffs2") 语句,保证在执行其它代码以前先把 JFFS2 文件系统挂载上去。

程序调试成功后运行扫雷游戏的效果图如下:

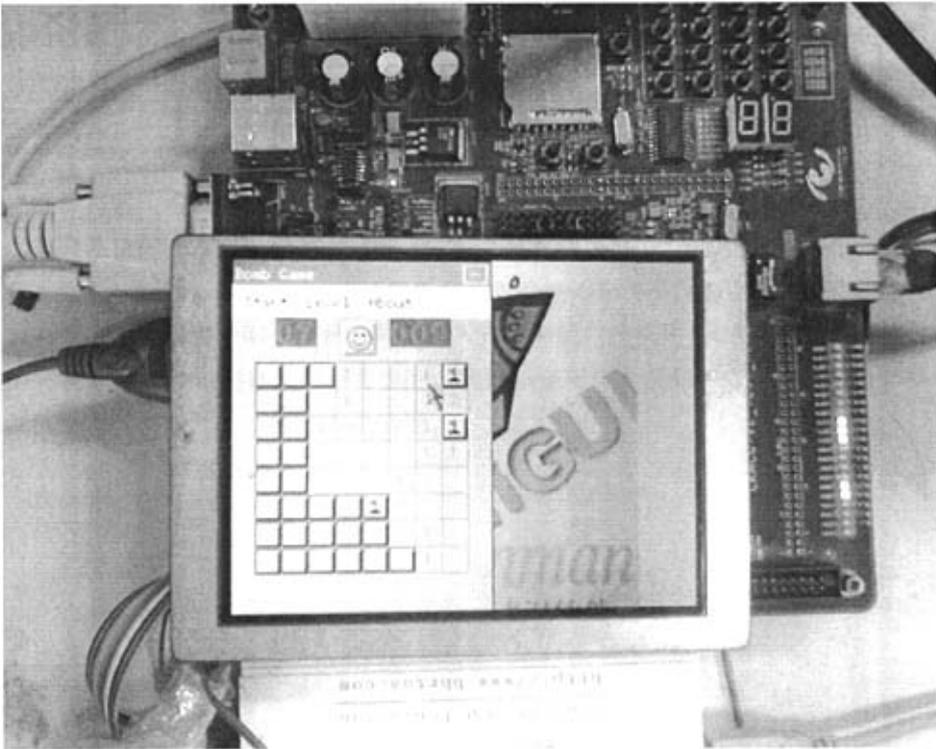


图 5.5 扫雷游戏效果图

第 6 章 总结与展望

6.1 总结

论文在分析和研究国内外嵌入式软件开发平台的基础上,设计一个功能完善、面向嵌入式领域、具有良好扩展性的基于 32 位高性能嵌入式 CPU CK510 的软件开发平台框架,在全体组员的共同努力下完成了整个平台的搭建和调试工作。

通过对嵌入式软件开发平台的研究,我加深了对嵌入式系统的理解,对如何在一个目标平台上运行操作系统 eCos、如何实现设备驱动程序、如何移植图形系统 MiniGUI. MiniGUI 编程环境等都有了一定程度的经验。但就对整个嵌入式系统而言,这些工作是极其有限的。它只是证实了在某一个方面,某一个应用层次上的一种可行的解决方案。

虽然本文的实现还存在一定的局限性,但整体来说,对于构建一个以其它目标平台为基础的同类应用具有极大的参数价值。

6.2 展望

作为嵌入式系统开发的支撑技术,嵌入式软件开发平台有着良好的应用前景。这是因为目前嵌入式系统越来越庞大,越来越复杂。另一方面,CK510 作为国有自主知识产权的嵌入式 CPU 领域的新秀,随着整体开发平台的建立与完善,它将会有更广阔的应用空间。当然,随着嵌入式产品的广泛应用,CK510 嵌入式软件的开发平台升级与完善是必不可少的,我们还需要作大量的工作,比如软件仿真器的开发和集成开发环境 IDE 的实现。

致 谢

首先要感谢我的导师严晓浪教授，严老师渊博的知识、丰富的阅历、严谨的治学态度和求是的科学作风对我影响至深，使我受益匪浅。正是严老师对集成电路产业和软件开发行业的独到见解将我引入了软件设计的殿堂，严老师的言传身教将在以后很长时间内深深影响着我。在此，由衷地向严老师表示我最诚挚的感谢和敬意。

感谢杭州中天微系统有限公司给我提供了学习和研究的环境，感谢公司的葛海通总经理在我的研究过程中给予的悉心指导和大力帮助。感谢我们软件组的组长李春强师兄，他扎实的基本功和丰富的项目经验保证了整个项目的正确走向，感谢全体组员刘兵、李靖、彭信民和刘丹青，在他们的共同努力下，实现了开发平台其它的模块，保证了整个项目的顺利完成。同时还要感谢公司其它同学和同事在整个项目期间对我们的帮助和指点。

感谢浙江大学超大规模集成电路研究所对我的培养，在超大所的学习和研究的经历将使我受益终生。

最后，感谢我的父母和家人，你们默默的付出和无私的奉献永远是我不断前进的动力。

参 考 文 献

- [1] 郭兵(2002). 嵌入式软件开放式集成开发平台体系结构研究. 电子科技大学计算机科学与工程学院博士学位论文。
- [2] 吴朝晖(2000). 纵谈嵌入式技术. 《微电脑世界》. (49). pp.8-10.
- [3] 李小将, 樊天晴(2002). 嵌入式系统在信息家电中的应用. 《计算机工程》. (2).
- [4] 康宇峰 (2004). 嵌入式系统开发面临的问题与集成开发环境的应用. 风河系统公司论文。
- [5] 周全, 窦振 (2004). 为嵌入式软件建立统一软件系统框架的方法. 电子产品世界网站论文。
- [6] 杭州中天微系统有限公司用户文档(2004). CK510 Datasheet.
- [7] 风河系统公司 (2004). 嵌入式软件开发跨入平台时代. 电子产品世界网站论文。
- [8] Chris Lüer (2003) . Evaluating the Eclipse Platform as a Composition Environment . 3rd International Workshop on Adoption-Centric Software Engineering Portland, Oregon.
- [9] 刘洪星, 谢玉山 (2005). Eclipse 开发平台及其应用. 《武汉理工大学学报》. 27(2). pp.89-92.
- [10] 朱舟, 马范援 (2003). Eclipse--下一代 IDE 开发环境. 《微型电脑应用》. 19(5). pp.62-63.
- [11] 黄凯 (2005). 基于 Eclipse 体系的构件开发管理平台的设计实现. 《科学技术与工程》. 5(14). pp.994-997,1005..
- [12] ARM 开发工具综述. <http://www.mcu99.com/Article/ARM/200601/844.html>.
- [13] 杭州中天微系统有限公司(2004). CK510 用户手册。
- [14] W. W. Hwu, R. E. Hank, D. M. Gallagher, S. A. Mahlke, D. M. Lavery, G. E. Haab, J. C. Gyllenhaal, D. I. August (1995) . Compiler Technology for Future Microprocessors. Proceedings of the IEEE. 83(12). pp. 1625-1640.
- [15] Trung N. Nguyen, Zhiyuan Li, David J. Liljia(1994). *Compiler Support for the Efficient Use of Cache Coherence Directories*.
- [16] GNU/The Free Software Foundation Website. <http://www.gnu.org/>.

- [17]GCC On-Line Documentation. <http://gcc.gnu.org/onlinedocs/>.
- [18]任珊虹, 赵克佳, 赵雄芳(1995). GCC 的中间语言及后端信息的转换. 《计算机工程与应用》. (2). pp.74-82.
- [19]Arthur Griffith, 胡恩华. *GCC 技术参考大全*. 清华大学出版社.
- [20]Richard M. Stallman(1999). *Using and Porting the GNU Compiler Collection*. the Free Software Foundation.
- [21]GNU Gas On-Line Documentation.
<http://www.gnu.org/software/binutils/manual/gas-2.9.1/as.html>.
- [22]GNU BFD On-Line Documentation.
<http://www.gnu.org/software/binutils/manual/bfd-2.9.1/bfd.html>.
- [23]GNU LD On-Line Documentation.
<http://www.gnu.org/software/binutils/manual/ld-2.9.1/ld.html>.
- [24]GNU GDB On-Line Documentation.
<http://www.gnu.org/software/gdb/documentation/>.
- [25]Martin Berg, Fredrik Cederquist, David Hamilton, Jenny Johansson, Anita Vindblom(2004). A Case Study of How to Add a Plugin Framework. *System Architecture and Integration*.
- [26]Robert Chatley, Susan Eisenbach, Jeff Magee(2004). *Component Deployment*. Springer Berlin / Heidelberg. pp.97-112.
- [27]Alexander Kleiner, Thorsten Buchheim(2004). *A Plugin-Based Architecture for Simulation in the F2000 League*. Springer Berlin / Heidelberg. pp.434-445.
- [28]S. A. Mahlke, W. Y. Chen, R. A. Bringmann, R. E. Hank, W. W. Hwu, B. R. Rau, M. S. Schlansker(1993). A Model for Compiler-Controlled Speculative Execution. *ACM Transactions on Computer Systems*. 11(4).
- [29]Xiaoming Ju, Ce Shi, Qingdong Yao, Zhejiang Univ (2004). Compiler support for reducing the complexity of register file in media processors. *Embedded Processors for Multimedia and Communications*. 19. pp.143-152.
- [30]Akihiko INOUE, Hiroyuki TOMIYAMA, Takanori OKUMA, Hiroyuki KANBARA, Hiroto YASUURA(1998). Language and Compiler for Optimizing Datapath Widths of Embedded Systems. *IEICE TRANSACTIONS on Fundamentals of Electronics, Communications and Computer Sciences*. E81-A(12). pp.2595-2604.

- [31] Barry SHACKLEFORD, Mitsuhiro YASUDA, Etsuko OKUSHI, Hisao KOIZUMI, Hiroyuki TOMIYAMA, Hiroto YASUURA(1996). An Integrated Processor Synthesis and Compiler Generation System. IEICE TRANSACTIONS on Information and Systems. E79-D(10). pp.1373-1381.
- [32] Anthony J. Massa, 颜若麟, 孙晓明, 尤伟伟, 林巧民. 嵌入式可配置实时操作系统 eCos 软件开发. 北京航空航天大学出版社.
- [33] eCos On-Line Documentation.
<http://ecos.sourceware.org/docs.html>
- [34] 秦怀峰. 面向嵌入式 PC 的 eCos 支持与完善. 西北工业大学硕士学位论文.
- [35] 王京起, 黄健, 沈中杰. 嵌入式可配置实时操作系统 eCos 技术及实现机制. 电子工业出版社.
- [36] 魏永明, 骆刚, 姜君. Linux 设备驱动程序. 中国电力出版社.
- [37] 秦怀峰, 施笑安, 周兴社, 谷建华 (2003). eCos 设备驱动程序设计分析. 《微电子学与计算机》. 7. pp 8-11.
- [38] 赵楚莹, 尹俊勋, 梁伟豪 (2006). ECOS 嵌入式系统的 S3C2510 以太网驱动程序设计. 《微计算机信息》. 22(42). pp 110-112.
- [39] 周永红, 王玲玲 (2005). 如何构建 eCos 嵌入式系统. 《单片机与嵌入式系统应用》. 11. pp 79-81.
- [40] JFFS2 On-Line Documentation. <http://sourceware.org/jffs2/jffs2-html>
- [41] 崔树林. 构建一种嵌入式图形应用平台. 北京科技大学硕士论文.
- [42] MiniGUI On-Line Documentation. <http://www.minigui.org/>.
- [43] 闫玉忠, 石理 (2003). 嵌入式 Linux 的 MiniGUI 研究和移植. 《单片机与嵌入式系统应用》. 6. pp 4-6.
- [44] 周杰, 李齐, 邵惠鹤 (2005). 基于 MC9328MXL 的 MiniGUI 的移植与开发. 《控制工程》. 12(增刊). pp 196-199.
- [45] 于孝辉, 陈秋艳, 李国义 (2006). 嵌入式 Linux 下 MiniGUI 的移植与软件开发. 《辽宁工学院学报》. 26(2). pp 90-92.