基于 Cortex-M4 内核的 Kinetis 微控制器的应用研究

中文摘要

Kinetis 系列微控制器是飞思卡尔公司于 2010 年下半年推出的基于 ARM Cortex-M4 内核的微控制器，是业内首款 Cortex-M4 内核芯片。Cortex-M4 内核是 ARM 公司 2009 年下半年发布的最新的嵌入式内核。Cortex-M4 面向数字信号控制市场，具有高效并且易于使用的控制和信号处理能力。

Kinetis 系列微控制器内部集成 UART、SLCD、TSI、USB、以太网和 CAN 等模块，具有高精度的 16 位 ADC 和 12 位 DAC。Kinetis 微控制器的市场应用主要面向工业控制，应用领域包括电机控制、通讯、安防和加密等。由于飞思卡尔公司在 2010 年 11 月才提供 Kinetis 微控制器的样片，所以现在 Kinetis 的学习资料和实例程序都很少。受飞思卡尔公司委托，苏州大学飞思卡尔嵌入式研发中心承担该芯片的先期研究应用任务。

本课题选择 Kinetis 系列的 K60N512 芯片为实例，设计制作了一套 Kinetis 开发套件，包括 K60N512 核心板，Kinetis 系列扩展板和 Kinetis 芯片调试器 OSJTAG。在嵌入式构件思想指导下，本文实现了 K60N512 的常用模块的驱动，并对模块驱动进行了充分的验证。本文移植了实时操作系统 FreeRTOS，详细分析了 FreeRTOS 的移植过程并对移植进行了测试，为实时操作系统在 Kinetis 芯片中的应用提供范例和参考。为了加快 K60N512 以太网编程，本文移植了开源的嵌入式 TCP/IP 协议栈 LwIP，详细阐述 LwIP 移植和测试过程。

为了研究 Kineits 的网络应用，同时也为了验证本课题实现的软硬件平台，本文设计实现了基于嵌入式 Web 的短信猫。短信猫实现短信发送和接收，使用 FreeRTOS 调度任务，使用 LwIP 实现网络通讯，同时移植 Web 服务器用于人机界面。本文实现的 K60N512 核心板、Kinetis 系列扩展板、Kinetis 调试器 OSJTAG 和模块驱动程序已经计划在 2011 年 8 月苏州大学飞思卡尔大学计划培训中推广。

关键词：Kinetis，驱动程序，FreeRTOS，短信猫

作　　者：王超艺

指导老师：王宜怀

Research on the Freescale Kinetis Microcontroller based on Cortex-M4 processer

Abstract

Kinetis series of microcontroller, based on ARM Cortex-M4 processor, is released by Freescale in the second half of 2010, which is the first microcontroller using ARM Cortex-M4 processor. The Cortex-M4 processor is the latest embedded processor released by ARM in the second half of 2009 specifically developed to address digital signal control markets that demand an efficient, easy-to-use blend of control and signal processing capabilities.

UART, SLCD, TSI, USB, Ethernet, CAN and other common modules are integrated in the Kinetis series microcontrollers, as long as high precision 16-bit ADC and 12-bit DAC. Kinetis Microcontroller is mainly for the market of industrial control, including motor control, communication, security and encryption. Becaues the Kinetis sample is not available until November 2010, so learning materials and example programs of Kinetis are very rare. Commissioned by Freescale Company, Freescale Embeded R&D Center of Soochow University takes an early research on the applicathin of this microcontroller.

This paper take the K60N512 as an example, designs and implements a Kinetis development kit, including K60N512 core board, Kinetis expansion board and Kinetis debugger OSJTAG. Guided by the idea of embedded components, this paper implements K60N512 module driver, every driver has been sufficiently validated. This paper ports the realtime operating system FreeRTOS and analysis the port process in detail, providing an example and reference of realtime operating system in Kinetis. To speed up the K60N512 Ethernet programming, the open source embedded TCP/IP protocol stack LwIP is ported and the port process in given in detail.

To study Ethernet communication with Kinetis and to verify Kineits software and hardware, this paper designs and implements a message modem based on embeded web server. The message modem uses FreeRTOS to schedule tasks and LwIP to implement internet communication and an embeded Web server is implemented as the human

interface. The K60N512 core board, Kinetis expansion board, Kinetis debugger OSJTAG and drivers achieved by this paper have been planned to promote in the Freescale university tranning of Soochow in August 2011.

Key Words：Kinetis，Drivers，FreeRTOS，Message Modem

Written by WangChaoyi

Supervised by WangYihuai

目 录

第一章 绪论

Cortex-M 系列内核是 ARM 公司针对低功耗和高性能的嵌入式控制市场而开发的内核。Cortex-M0 和 Cortex-M3 系列芯片广泛的应用于智能仪表、智能卡、智能家电、智能玩具、短距离联网应用(Zigbee 和 NFC)、汽车电子和高效电机控制等领域。ST、TI、NXP、Atmel 和东芝等芯片设计公司都已经推出 Cortex-M3 的 MCU[1]。

Cortex-M4 是 ARM 公司于 2009 年下半年推出的内核，其性能比 Cortex-M3 提高 20%[2]。飞思卡尔的 Kinetis 系列微控制器是业内首款基于 ARM Cortex-M4 内核 32 位微控制器，资源丰富、功能强大，具有巨大的市场前景和应用价值。由于 2010 年 11 月下飞思卡尔才开始提供 Kinetis 芯片样片，而且 Kineits 是首款 ARM Cortex-M4 芯片，所以现在 Kinetis 和 Cortex-M4 的研究资料和参考设计都很少。

受飞思卡尔公司委托，本文对 Kinetis 开发方法进行深入的研究和实践，目标是推广该系列芯片在国内的应用。飞思卡尔公司已将本课题研究内容纳入大学计划。本课题设计并制作了一款 K60N512 核心板和 Kinetis 系列扩展板。在嵌入式软件工程和构件思想的指导下，本文实现了 K60N512 GPIO、UART、AD、CRC、Flash 和网络等常用模块的软件构件，并进行了充分的测试，保障了这些软件构件稳定性和可靠性。在这些工作的基础上，本文移植了应用广泛的实时操作系统 FreeRTOS，给出了详细的移植过程。本文同时移植了开源的嵌入式 TCP/IP 协议栈，给出了详细的移植过程，并对移植进行了测试。作为全文的引导，本章先介绍 Kinetis 微控制器的研究背景，接着给出本课题的设计思路，然后阐述 Kinetis 微控制器应用研究的现实意义，最后给出本文所做的主要工作和组织结构。

1.1 课题背景

飞思卡尔半导体是全球领先的半导体公司，其前身为摩托罗拉公司半导体部门，产品面向规模庞大、增长迅速的汽车、消费、工业、网络和无线市场等嵌入式半导体产品市场。飞思卡尔公司具有超过 30 年的嵌入式处理器设计经验，技术力量雄厚，产品线丰富，涵盖低端 8 位单片机到 64 位多核高端处理器，从普通 MCU 到 DSC 直至高端多核 DSP，应有尽有。飞思卡尔公司长期占据全球汽车电子处理器供应商第一位的宝座，网络通信处理器全球占有率第一位，并一度成为全球第 2 大 MCU 供应商和第 2 大 DSP 芯片供应商。

飞思卡尔半导体 2010 年下半年推出了 Kinetis 系列芯片，这是基于新 ARM Cortex-M4 内核的 90 纳米 32 位 MCU，是业内首款基于 Cortex-M4 内核的芯片，开创了其微控制器领域领先地位的新纪元。Kinetis 补充了飞思卡尔最近推出的 90 纳米 ColdFire+ MCU 系列，是业内扩展能力最强的 MCU 系列之一[3]。

“Kinetis MCU 系列代表了我们在过去多年里与飞思卡尔共同构建的坚实技术基础的良好扩展”，ARM 首席执行官 Warren East 表示。“Kinetis 系列再次证明了飞思卡尔在整个嵌入式行业内对低功耗、高性能 ARM Cortex-M 内核的加大支持，我们非常高兴能够成为面向微控制器客户群的这套广泛且创新的产品系列中的一个关键技术要素”[8]。

Kinetis MCU 采用了飞思卡尔 90 纳米薄膜存储器(TFS)技术和 FlexMemory 功能（可配置的电子可擦除、可编程、只读存储器 EEPROM）。Kinetis MCU 使用与 ColdFire+ MCU 相同的软件支持工具，使客户能够轻松地为其最终应用选择最佳解决方案。

Kinetis MCU 提供了无与伦比的可扩展性、兼容性和特性集成。通用外设、存储器映射和封装允许在 MCU 系列内和 MCU 系列之间轻松迁移，为最终产品线的扩展提供了捷径，减少产品开发成本，从而能够及时地响应市场需求。

Kinetis 系列的通用特性包括[4]：

（1）高速 16 位模数转换器；

（2）12 位数模转换器，带有片上模拟电压参考；

（3）多个高速比较器和可编程增益放大器；

（4）低功耗触摸感应功能，通过触摸能将器件从低功耗状态唤醒；

（5）多个串行接口，包括 UART；

（6）强大灵活的定时器；

（7）片外系统扩展和数据存储选项，包括 SD 主机、NAND 闪存、DRAM 控制器和飞思卡尔 FlexBus 互连方案。

K60N512 系列包括一系列高度集成的 MCU，提供最高达 180MHz 时钟频率和 IEEE1588 以太网控制器，用于工业自动化环境中的精确实时的时间控制。硬件加密支持多个算法，以最小的 CPU 负载提供快速、安全的数据传输和存储。系统安全模块包括安全密钥存储和硬件篡改检测，集成用于电压、频率、温度和外部传感（用于物理攻击检测）的传感器。

低功耗已渗透到 Kinetis MCU 设计的方方面面。Kinetis 微控制器均支持全存储器（闪存/RAM/EEPROM）和低至 1.71V 的模拟外设操作，最终延长便携式设计中的电池使用寿命[4]。Kinetis 微控制器共有 10 种低功耗操作模式，允许设计人员优化外设活动和恢复时间。低功耗实时时钟、低漏电流唤醒单元和低功耗定时器为飞思卡尔的客户增加了更多的灵活性，实现了在低功耗状态下的连续系统操作。

1.2 Cortex-M 内核

英国的 ARM 公司是嵌入式微内核世界当中的佼佼者。ARM 公司一直以来都是自己研发微内核架构，然后将这些架构的知识产权授权给各个芯片厂商，精简的 CPU 架构，高效的处理能力以及成功的商业模式让 ARM 公司获得了巨大的成功，使它迅速占据了 32 位嵌入式微内核的大部分市场份额。微软宣布其下一代操作系统产品将支持 ARM 内核，这将改变 PC 领域的芯片格局，对 Intel 和 AMD 造成威胁。

目前，随着对嵌入式系统的要求越来越高，作为其核心的嵌入式微内核的综合性能也受到日益严峻的考验，最典型的例子就是伴随 3G 网络的推广，对手机的本地处理能力要求很高，现在一个高端的智能手机的处理能力几乎可以和几年前的笔记本电脑相当。为了迎合市场的需求，ARM 公司也在加紧研发他们最新的 ARM 架构，Cortex 系列就是这样的产品。在 Cortex 之前，ARM 核都是以 ARM 为前缀命名的，从 ARM1 一直到 ARM11，之后就是 Cortex 系列了。Cortex 在英语中有大脑皮层的意思，而大脑皮层正是人脑最核心的部分，ARM 公司如此命名正有此含义[10]。

1.2.1 Cortex-M 内核特点

Cortex 系列属于 ARMv7 架构，这是 ARM 公司最新的指令集架构，ARMv7 架构是在 ARMv6 架构的基础上诞生的。该架构采用了 Thumb-2 技术，Thumb-2 技术是在 ARM 的 Thumb 代码压缩技术的基础上发展起来的，并且保持了对现存 ARM 解决方案的完整的代码兼容性。Thumb-2 技术比纯 32 位代码少使用 31％的内存，减小了系统开销。同时能够提供比已有的基于 Thumb 技术的解决方案高出 38％的性能。ARMv7 架构还采用了 NEON 技术，将 DSP 和媒体处理能力提高了近 4 倍，并支持改良的浮点运算，满足下一代 3D 图形、游戏物理应用以及传统嵌入式控制应用的需求。此外，ARMv7 还支持改良的运行环

境，以迎合不断增加的 JIT(Just In Time)和 DAC(Dynamic Adaptive Compilation)技术的使用。另外，ARMv7 架构对于早期的 ARM 内核软件也提供很好的兼容性[10]。

ARMv7 架构定义了三大分工明确的系列：“A”系列面向尖端的基于虚拟内存的操作系统和用户应用；“R”系列针对实时系统；“M”系列对微控制器、低成本和低功耗应用。

ARM Cortex-M 处理器系列是一系列可向上兼容的高能效、易于使用的处理器，这些处理器旨在帮助开发人员满足未来的嵌入式应用的需要。这些需要包括以更低的成本提供更多功能、更好的代码复用和更高的效能。Cortex-M 系的应用领域主要针对成本和功耗敏感嵌入式领域，包括智能测量、人机接口设备、汽车和工业控制系统、大型家用电器、消费性产品和医疗器械等。

1.2.2 Cortex-M3 内核

Cortex-M3 处理器是一个低功耗的处理器，具有门数少，中断延迟小，调试容易等特点。它是为功耗和价格敏感的应用领域而专门设计的，应用范围可从低端微控制器到复杂 SoC。Cortex-M3 处理器使用了 ARMv7-M 体系结构，是一个可综合的、高度可配置的处理器。它包含了一个高效的哈佛结构三级流水线，可提供 1.25DMIPS/MHz 的性能。

为降低器件成本，Cortex-M3 处理器采用了与系统部件紧耦合的实现方法，来缩小芯片面积，其内核面积比现有的三级流水线内核缩小了 30%。Cortex-M3 处理器实现了 Thumb-2 指令集架构，具有很高的代码密度，可降低存储器需求，并能达到非常接近 32 位 ARM 指令集的性能。

ARM Cortex-M3 系列是为那些对开发费用非常敏感同时对性能要求小的嵌入式应用(如微控制器、汽车车身控制系统和各种大型家电)所设计的，主要面向单片机领域，可以说是 51 单片机的完美替代品。Cortex-M3 内核框图见图 1-1。

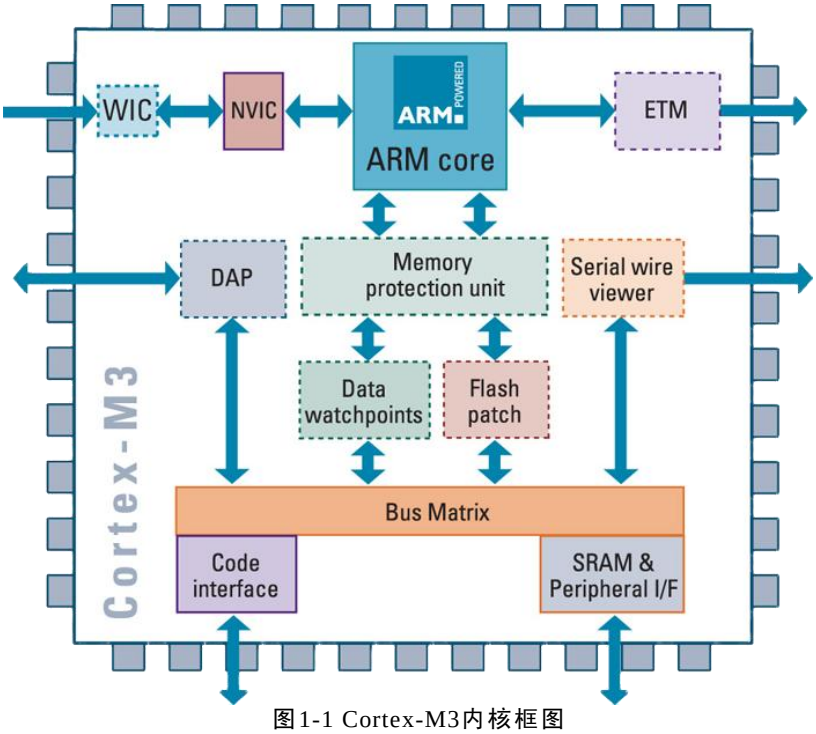


图1-1 Cortex-M3内核框图

Cortex-M3 的速度比 ARM7 快三分之一，功耗低四分之三，并且能实现更小芯片面积，利于将更多功能整合在更小的芯片尺寸中。Cortex-M3 内核结合了执行 Thumb-2 指令的 32 位哈佛微体系结构和系统外设，包括 Nested Vectored Interrupt Controller 和 Arbiter 总线。该技术方案在测试和实例应用中表现出较高的性能：在 180nm 工艺下，芯片性能达 1.2DMIPS / MHz，时钟频率高达 100MHz。

在工控领域，用户要求具有更快的中断速度，Cortex-M3 采用了 Tail-Chaining 中断技术，完全基于硬件进行中断处理，最多可减少 12 个时钟周期数，在实际应用中可减少 70%中断。目前已经面市的 Cortex-M3 芯片有意法半导体的 STM32，流明诺瑞（已经被 TI 收购）的 LM3S，恩智浦的 LPC1700 和爱特梅尔的 SAM3U 等。

1.2.3 Cortex-M4 内核

Cortex-M4 内核是在 Cortex-M3 内核的基础上发展起来的，其性能比 Cortex-M3 提高 20%。Cortex-M4 在 Cortex-M3 基础上强化了运算能力，新加了浮点运算控制器、DSP 和并行计算等。ARM 希望把 Cortex-M4 用于数字信号控制市场，也就是既有微控制器的控制能力，又有 DSP 的处理能力，主要应

用领域包括马达控制，电力与能源管理，嵌入式音频处理，以及工业自动化[11]。Cortex-M4 具有一个单时钟周期乘法累加（MAC）单元、优化的单指令多数据（SIMD）指令、饱和运算指令和一个可选的单精度浮点运算单元（FPU）。

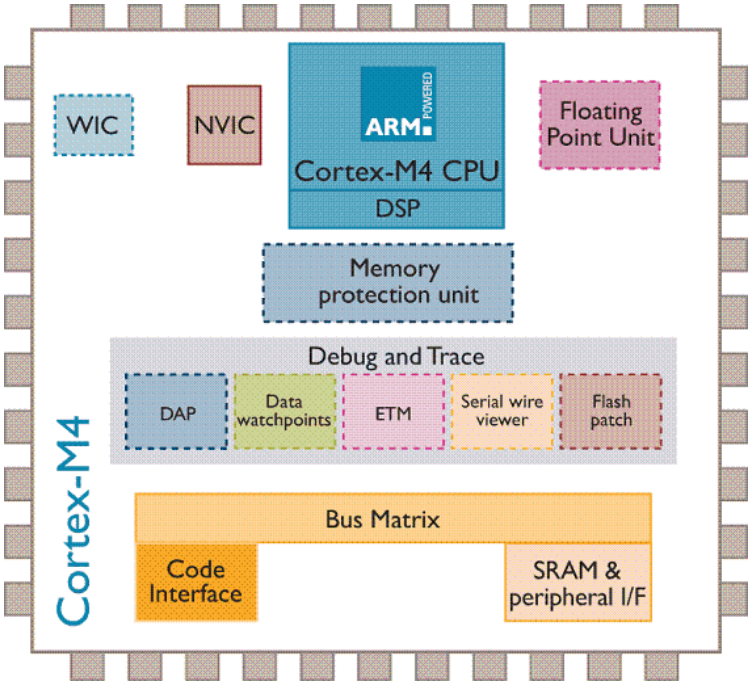


图1-2 Cortex-M4内核框图

这些数字信号处理功能基于一系列 ARM Cortex-M 系列内核所采用的创新技术，包括：高性能 32 位内核，可达 1.25DMIPS/MHz；Thumb-2 指令集，提供最佳的代码密度和一个嵌套向量中断控制器，能完成出色的中断处理。此外，该内核还提供了一个可选的内存保护单元（MPU），提供低成本的调试/追踪功能和集成的休眠状态，以增加灵活性[11]。嵌入式开发者可以快速设计并推出令人瞩目的终端产品，具备最多的功能以及最低的功耗和尺寸。Cortex-M4 内核框图见图 1-2。目前已经面市的 Cortex-M4 芯片有飞思卡尔的 Kinetis 和恩智浦的 LPC4300。

1.3 设计思路

本课题着眼于 Kinetis 系列芯片在国内的推广与应用，设计了一套 K60N512 核心板、Kinetis 扩展板、Kinetis 调试器 OSJTAG 和各个模块的软件驱动，并成功移植实时操作系统 FreeRTOS 到 Kinetis 芯片上，为 Kinetis 的学习和研发提供便利。

此外，为了方便使用 Kinetis 系列的以太网模块，本文还移植了开源的嵌入式 TCP/IP 协议栈 LwIP，并在 FreeRTOS 和 LwIP 的支持下，实现了微型的嵌入式 Web 服务器。

硬件设计中，考虑到 K60N512 芯片封装为 144MAPBGA，而且工作频率为 100MHZ，所以采用四层电路板实现。四层电路板绘制难度大，加工费用高，为了降低整体硬件成本，并且提高硬件的灵活性和扩展性，本文设计了核心板加扩展板的硬件模型。首先使用四层电路板实现 K60N512 的最小系统，并引出 K60N512 所有功能引脚。然后，使用两层电路板实现 Kinetis 扩展板，扩展板通过扩展接口与核心板对接。扩展板上实现了 Kinetis 系列 MCU 的常见模块，包括小灯模块，UART 模块，LCD 模块，USB 模块，CAN 收发模块和以太网模块等。K60N512 的核心板将来可以直接用于实际项目中，Kinetis 扩展板也可以用于评估 Kinetis 系列的其他芯片。这种硬件设计模型，大大降低了硬件设计成本，同时提高了硬件的灵活性和扩展性。

软件设计中，本文首先详细分析了 Kineits MCU 的启动过程，实现了 Kinetis MCU 的启动代码。然后在嵌入式构件思想指导下，实现了各个模块的软件构件，并编写模块测试实例，充分验证了驱动程序正确性。

嵌入式产品开发中，经常使用嵌入式实时操作系统来加快产品开发速度，同时提供产品稳定性和实时性。因此，本文移植了广泛使用的实时操作系统 FreeRTOS。FreeRTOS 操作系统是完全免费的操作系统，具有源码公开、可移植、可裁减、调度策略灵活的特点，可以方便地移植到各种单片机上运行，其最新版本为 6.1.1 版[9]。FreeRTOS 提供的功能包括：任务管理、时间管理、信号量、消息队列、内存管理、记录功能等，可基本满足常见系统的需要。FreeRTOS 内核支持优先级调度算法，每个任务可根据重要程度的不同被赋予一定的优先级。FreeRTOS 的内核可根据用户需要设置为可剥夺型内核或不可剥夺型内核。

为了研究 K60N512 芯片的网络应用，同时也为了验证本课题设计的硬件和移植的操作系统 FreeRTOS 以及 TCP/IP 协议栈，本文第五章实现了嵌入式 Web 方式的短信猫。短信猫实现短信收发，内部实现了嵌入式 Web 服务器用于人机交互。

1.4 课题意义

Cortex-M4 内核是 2009 年下半年推出的，作为 ARM 公司面向中高端微控制

器市场的主打内核，在 Cortex-M3 内核广泛应用的基础上，势必对目前微控制器市场格局造成冲击。目前 Cortex-M4 已经授权给各大微控制器生产厂商，包括飞思卡尔、恩智浦、意法半导体和德州仪器等。目前飞思卡尔和恩智浦已经推出了使用 Cortex-M4 内核的芯片。而飞思卡尔的 Kinetis 系列芯片则是业内首款使用 Cortex-M4 内核的芯片。

由于 Cortex-M4 内核和 Kinetis 系列芯片都是最新推出的产品，目前针对它们的应用和研究都非常少。目前 Kinetis 系列芯片已经大规模推出样片，工程师和研究人员急需一整套的硬件软件的平台进行学习和研究。本课题受飞思卡尔公司委托，设计实现了 Kinetis 系列的代表芯片 K60N512 的核心板和 Kinetis 系列扩展板以及 Kinetis 系列芯片的调试器 OSJTAG。硬件部分分析实现了芯片最小系统和各模块驱动电路。软件部分实现了各模块的驱动程序，此外还移植了实时操作系统 FreeRTOS 和 TCP/IP 协议栈 LwIP。

本课题的研究内容已经纳入飞思卡尔大学计划，课题实现的硬件和软件计划于飞思卡尔 2011 年 8 月在苏州大学举行的研讨会上推广。其中本课题实现的 Kinetis 系列芯片的调试器 OSJTAG 已经批量生产。课题实现的软硬件资源将有助于 Kinetis 系列芯片在国内高校的推广，同时为企业产品研发提供借鉴参考，更为广大嵌入式爱好者提供前沿嵌入式产品的学习研究平台。

本文在硬件设计过程中使用的核心板加扩展板的硬件实现方法，大大增加了硬件灵活性和扩展性，这种设计方案借鉴了飞思卡尔官方推出的 TOWER 系统的架构。本课题实现的 K60N512 的核心板采用四层板布线，最终顺利调试通过，是对多层电路板设计和布线的很好实践，为其他研究设计人员提供借鉴。

软件设计过程中，本文始终遵循嵌入式软件构件思想，所有驱动代码和功能代码均依照构件化设计的规范进行，是对嵌入式软件工程思想和设计方法的很好实践。本课题移植的嵌入式实时操作系统 FreeRTOS 是一个优秀的开源实时操作系统，已经广泛应用到多款芯片中。本课题的成功移植一方面有助于 FreeRTOS 在 Cortex-M4 内核芯片上的应用，另一方面也为其他嵌入式操作系统移植提供参考。为了更好的使用 Kinetis 芯片内部的以太网模块，本课题还移植了开源的 TCP/IP 协议栈，并将 FreeRTOS 和 LwIP 集成到一起。TCP/IP 协议栈将大大方便 Kinetis 芯片的网络程序开发，由于 LwIP 运行需要 FreeRTOS 支持，这也是对 FreeRTOS 的应用实践。

此外，为了扩展 K60N512 芯片的网络应用，也为了验证本课题实现的硬件和软件平台，本文设计实现了嵌入式 Web 方式的短信猫。短信猫可以实现短信收发，通过 Web 页面的方式和用户交互，使用方便。目前市场中的短信猫多为串口或 USB 口通讯模式，使用过程中，需要在厂商提供的二次开发包的基础上封装 UI 界面才能使用。本文实现的短信猫内部实现了嵌入式 Web 服务器，内部可以解析 HTTP 协议。用户连接网线后，直接通过浏览器和短信猫交互，使用更灵活更人性。短信猫中使用华为公司的 GSM/GPRS 模块 EM310 进行短信收发。短信猫的设计和实现过程涉及了 GSM/GPRS 通讯、嵌入式 Web 服务器和使用内部 Flash 实现的虚拟文件系统等知识点，是对 K60N512 芯片的集成应用的很好实践。

1.5 本文工作和论文结构

1.5.1 本文工作

本文的主要工作安排如下：

1）硬件开发板的设计与制作

（1）阅读 K60N512 参考手册，绘制 K60N512 核心板电路图。由于 K60N512 封装为 MAPBGA144，所以核心板使用 4 层电路板布线。

（2）分析 Kinetis 系列芯片功能模块电路，实现各模块电路，并绘制 Kinetis 扩展板。扩展板中包含 Kinetis 系列芯片常见模块，如 USB，CAN，网络，LED 和 LCD 等。扩展板使用 2 层电路板实现。

（3）分析 Kinetis 芯片调试器 OSJTAG 电路，绘制调试器电路板。

（4）制作电路板，购买元件，手工焊接电路板。

（5）电路板评估调试。

2）底层驱动软件的编写与调试

（1）按照 K60N512 手册给出的编程方法和时序要求，在深刻理解各个模块通讯原理的基础上，遵循嵌入式构件思想编写健壮的、规范的、可重用的和可维护性较好的各模块底层构件，包括 GPIO、UART、AD、CRC 和网络等。

（2）设计每个模块的测试实例，配合本课题实现的的 PC 方测试程序，验证软硬件构件的正确性。

3）FreeRTOS 和 LwIP 移植

（1）分析 FreeRTOS 结构，移植 FreeRTOS 到 K60N512 芯片中，对移植结果

进行测试评估。

（2)分析 TCP/IP 协议栈 LwIP 的结构，实现 LwIP 在 K60N512 芯片上的移植，并对移植结果进行测试评估。

4）嵌入式 Web 方式短信猫实现

设计实现基于嵌入式 Web 服务器的短信猫。短信猫实现短信收发。短信猫内部使用 FreeRTOS 调度各个任务，使用 LwIP 协议栈实现网络通信，内部集成嵌入式 Web 服务器用于人机交互。

1.5.2 论文结构

第一章分析 Kinetis 应用研究的课题背景、设计思路和研究意义，并给出本文的主要工作和结构安排。

第二章详细阐述了 K60N512 核心板、Kinetis 系列扩展板和 Kinetis 调试器 OSJTAG 的硬件设计和实现，并介绍了开发板各模块使用的各种芯片的特性及支撑电路设计，最后给出了硬件设计体会。

第三章详细讨论了 K60N512 评估板驱动软件的设计，并具体给出了 UART、AD、CRC 和 Flash 模块驱动的实现过程和测试实例。

第四章移植了开源的实时操作系统 FreeRTOS 和 TCP/IP 协议栈 LwIP，给出了详细的移植过程，并对移植进行了测试。

第五章在前面章节实现的硬件和软件基础上实现了嵌入式 Web 方式的短信猫，阐述了短信猫的设计和实现过程。

最后一章对本文所做的工作进行了总结，并提出一些尚待改进的地方。

第二章 硬件设计

嵌入式产品设计包括软件设计和硬件设计。硬件设计是软件设计的基础，硬件也是最终产品运行的平台，因此硬件设计对整个产品的开发至关重要。硬件设计过程中，需要分析各模块的电气特性，做到模块结构清晰。此外，硬件设计中应该预留扩展接口和测试接口。

K60N512 芯片封装为 144MAPBGA，芯片工作主频为 100MHZ，常规的两层电路板无法实现电路布线。在构件化硬件设计思想的指导下，本文将 K60N512 芯片和外设模块分别布板，设计实现了 K60N512 核心板和 Kinetis 扩展板两块电路板。其中，K60N512 核心板使用 4 层电路板，核心板上实现了 K60N512 的最小系统，引出了所有功能引脚；Kinetis 扩展板使用 2 层电路板，扩展板上实现了 Kinetis 系列芯片的主要模块，包括 LED，LCD，UART，CAN，SD 卡，USB OTG 和以太网等模块。

K60N512 核心板可以直接用于实际项目中，而 Kinetis 扩展板也可以用于评估 Kinetis 系列的其他芯片。因此，这种核心板加扩展板的硬件设计方案，一方面降低了硬件的整体成本，另一方面也提高了系统硬件的灵活性和扩展性。飞思卡尔官方推广的快速硬件评估系统 TOWER，其硬件设计思想也是核心板加扩展板[20]。

2.1 K60N512 芯片简介

K60N512 芯片是 Kinetis 系列 K60 子系列的典型芯片，是 Kineits 系列中集成度最高的芯片。K60 系列 MCU 具有 IEEE1588 以太网、USB2.0 OTG 和硬件加密电路。K60N512 具有丰富的通讯接口、AD 转换电路和外围控制电路[7]。其特性如下：

（1）ARM Cortex-M4 内核，主频高达 100MHZ；

（2）32 路 DMA 供外设和存储器使用，大大提高 CPU 利用率；

（3）10 种低功耗模式，包括运行，等待，停止和断电；

（4）512K Flash 和 128K SRAM；

（5）集成硬件和软件看门狗，硬件加密电路和 CRC 电路；

（6）33 路单路和 4 路差分的 16 位 AD 转换，2 路 12 位 DA 转换；

（7）8 路电机控制，2 路方波解码，4 路可编程定时器；

（8）SD 卡主机控制器，6 路 UART，IIC，IIS，SPI，CAN；

（9）USB2.0 全速和高速接口，支持 OTG；

（10）IEEE1588 以太网接口，支持 MII 和 RMII 通讯；

（11）工作电压 1.71V~3.6V，工作温度-40°~105°；

（12）多达 100 路 GPIO 引脚，所有 GPIO 引脚兼容 5V 电平。

2.2 K60N512 核心板设计

K60N512 核心板实现 K60N512 芯片的最小系统电路。芯片的最小系统电路是能够使芯片运行起来所需要的最小电路，通常包括：电源电路，晶振电路，复位电路，调试接口电路和状态指示电路[13]。K60N512 的各模块功能引脚全部通过扩展接口引出。值得注意的是核心板晶振输出引脚也通过扩展接口引出，这是由于 K60N512 的以太网接口需要。以太网物理层收发器需要和 K60N512 使用同一个晶振[5]。

由于 K60N512 芯片封装为 144MAPBGA 封装，因此，核心板使用 4 层电路板。笔者也是第一次接触多层电路板布线，其间遇到了诸多困难，最终查阅多方面资料，参考了多块多层电路板，在指导老师和同学的帮助下，最终 K60N512 核心板顺利布通，做板后也顺利调试通过。

2.2.1 电源电路

K60N512 工作电压为 3.3V，扩展接口中直接引入 3.3V 供芯片使用。K60N512 电源电路如图 2-1 所示。这里 AD 模块的参考电源直接接输入电源。电源引脚添加了多个 0.1μF 的高通滤波电容，用于稳定电源，使芯片工作更加稳定。在硬件布线时，这些电容应该尽量靠近电源引脚，以实现滤波功能。电源模块预留了测试点，同时在状态接口电路中，使用小灯来指示电源状态。

2.2.2 晶振电路

晶振用于产生芯片工作时钟。K60N512 内部集成多用途时钟产生器（Multipurpose Clock Generator，MCG）模块，用于将晶振输入时钟倍频至系统所需时钟。K60N512 共需要两个晶振，一个是芯片的主晶振，用于产生芯片和外设的工作时钟，另一个是实时定时器（Real Timer Counter，RTC）的晶振。本文中，系统主芯片时钟使用 50MHZ 有源晶振，RTC 时钟使用 32.768KHZ 无

源晶振，图 2-2 为系统晶振电路。在硬件布线时需要注意晶振附近不能走高频信号，晶振应该尽量靠近晶振输入引脚。晶振一旦不能正常工作，芯片将无法工作。晶振实际上负责给芯片提供心跳。

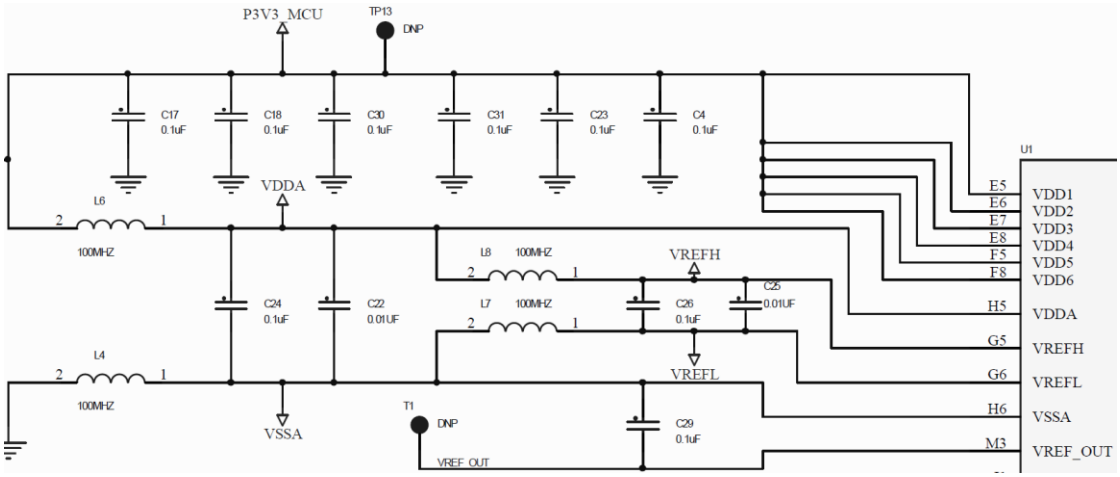


图2-1 K60N512电源电路

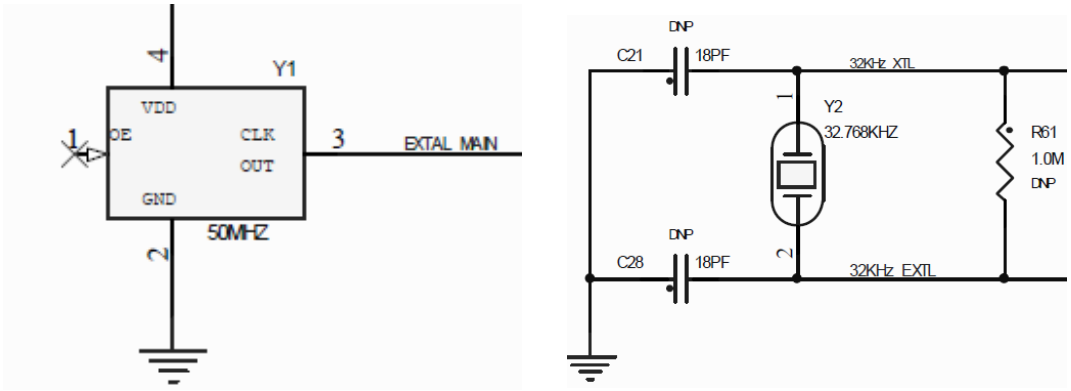


图2-2 K60N512晶振电路

2.2.3 复位电路

复位电路用于产生复位电平，使芯片复位。K60N512 复位电平是低电平有效，设计中在状态指示电路中添加了复位状态指示灯。为了防止运行中干扰信号，复位信号添加了外部上拉电阻，并在复位按键处添加高通滤波电容，用于稳定复位信号。复位电路如图 2-3 所示。

2.2.4 JTAG 电路

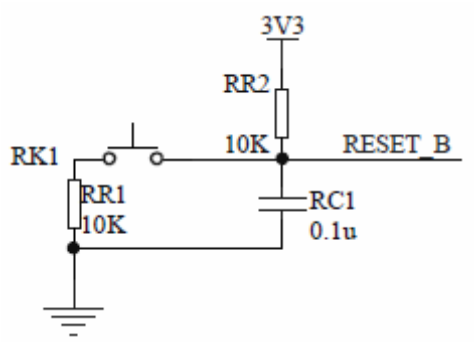


图2-3 K60N512复位电路

Kinetis 芯片使用的是 ARM Cortex-M4 内核，该内核内部集成了 JTAG（Joint Test Action Group）接口，通过 JTAG 接口可以实现程序下载和调试功能。图 2-4 为 JTAG 接口电路。

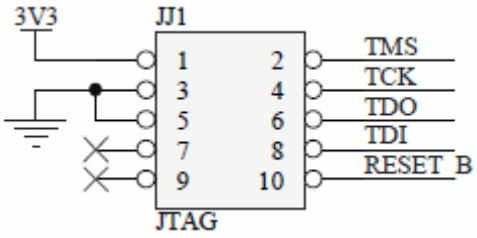


图2-4 K60N512 JTAG电路

2.2.5 状态指示电路

为了验证核心板是否正常工作，本设计在最小系统之外添加了状态指示电路，实际上是在 GPIO 口接了小灯，如果芯片能使小灯闪烁则说明核心板可以正常工作。其中 D8 和 D9 接的是普通 IO 口，D10 指示复位状态，D11 指示电源状态。图 2-5 为状态指示电路。

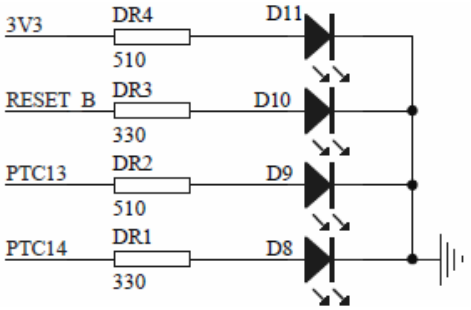


图2-5 K60N512状态指示电路

2.2.6 扩展接口电路

扩展接口将 K60N512 所有功能引脚引出，是 K60N512 核心板和 Kinetis 扩展板的接口。扩展接口还负责给核心板提供电源。图 2-6 为扩展接口电路。

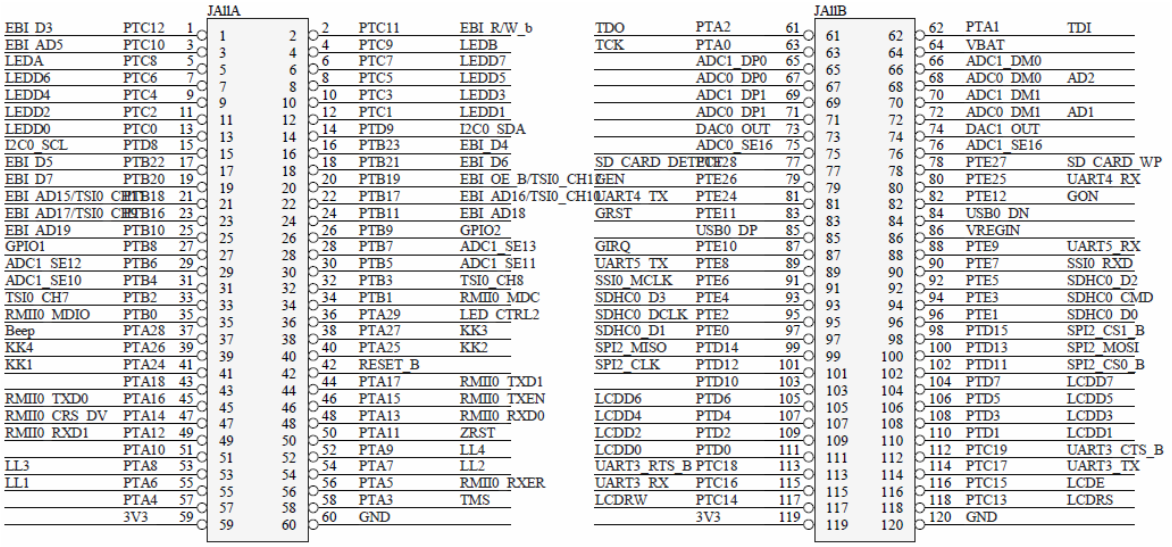


图2-6 扩展接口电路

2.3 Kinetis 扩展板设计

Kinetis 扩展板上设计了 Kinetis 系列芯片的大部分模块，其中包括 LED，CAN，UART，SD 卡，USB OTG，以太网和 LCD 等模块。扩展板使用两层电路板，在电路板布局和布线时，尽量做到了模块结构清晰，走线简单直接。在保持扩展接口一致的情况下，Kinetis 扩展板可以用于评估 Kinetis 系列的所有芯片。

2.3.1 电源转换电路

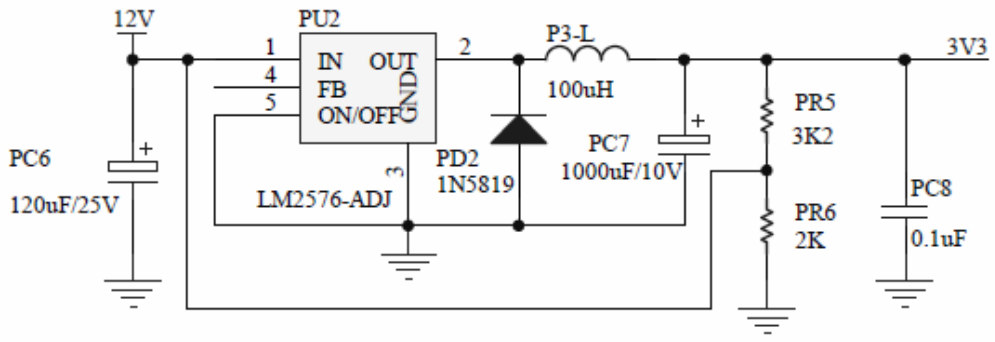


图2-7 电源转换电路

扩展板提供 12V 电源输入插头，板上使用 LM2576 芯片将 12V 电压转换

至 3.3V 电压。LM2576 产生的 3.3V 电源提供核心板和扩展板上的所有元件，LM2576 可以提供 3A 的输出电流[21]，足够核心板和外设使用。图 2-7 为电源转换电路。

2.3.2 UART 模块

UART 模块实现 TTL 电平和 232 电平之间的相互装换，转换芯片使用 MAX3232。由于使用简化的 232 通讯，232 通讯只使用了 RxD 和 TxD 信号线，没有使用硬件握手信号，因此，一片 MAX3232 芯片可以完成 2 路 232 通信。扩展板上引出了 UART3 和 UART5 两路 UART。图 2-8 为 UART 模块电路。

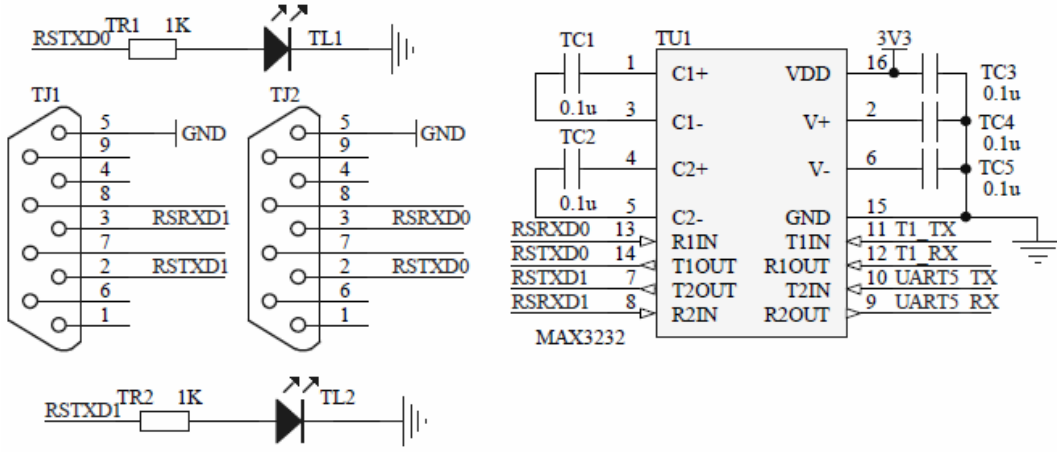


图2-8 UART电路

2.3.3 SD 卡模块

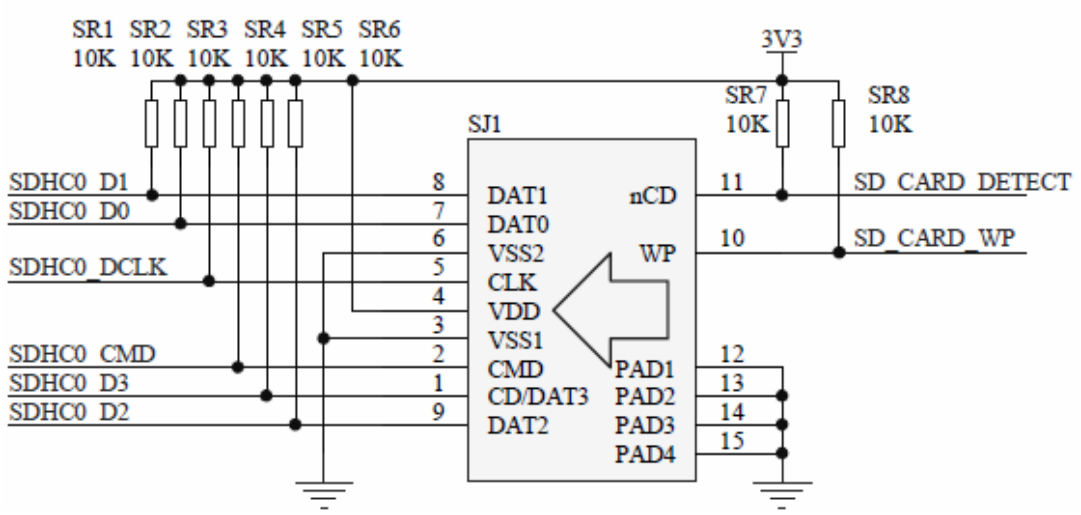


图2-9 SD卡电路

Kinetis 系列芯片内部集成 SD 主机控制器（Secure Digital host controller），支持 MMC 卡，SD 卡和 SDIO 卡。图 2-9 为 SD 卡模块电路。

2.3.4 USB OTG

Kinetis 系列芯片集成 USB OTG 模块，可以实现 USB 主机和从机通讯，通讯速率高达 12Mbps，USB OTG 使用 MAX3353 作为物理层驱动器，电路如图 2-10 所示。

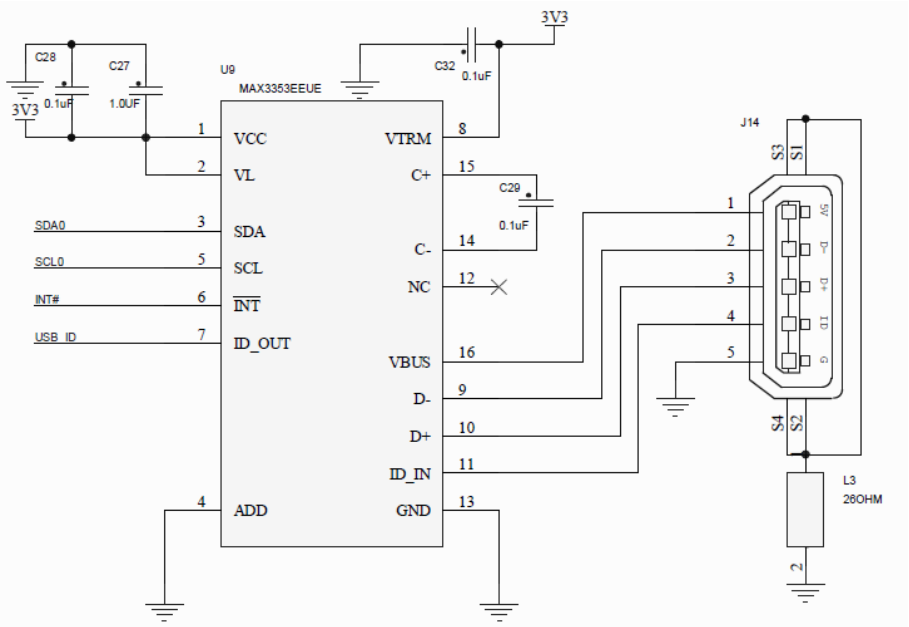


图2-10 USB OTG电路

2.3.5 液晶和数码管模块

扩展板上设计数码管和液晶模块。液晶使用四行双色的液晶 YM1602，可以显示 2 行汉字，每行 16 个。数码管为双字共阴性数码管，型号为 LG3922CH。接口电路如图 2-11 所示。

2.3.6 以太网模块

Kinetis 集成 10/100M 以太网控制器，支持 RMII 和 MII 通讯，当使用 RMII 通讯时使用核心板输出的 50MHZ 晶振。以太网物理层收发器使用 Micre 公司的 KSZ8041NL。KSZ8041NL 支持 RMII 和 MII 通讯，可以自动辨别交叉线和直通线，支持 10M 和 100M 通讯，具有丰富的配置功能。KSZ8041NL 和 K60N512 的电路连接见图 2-12。

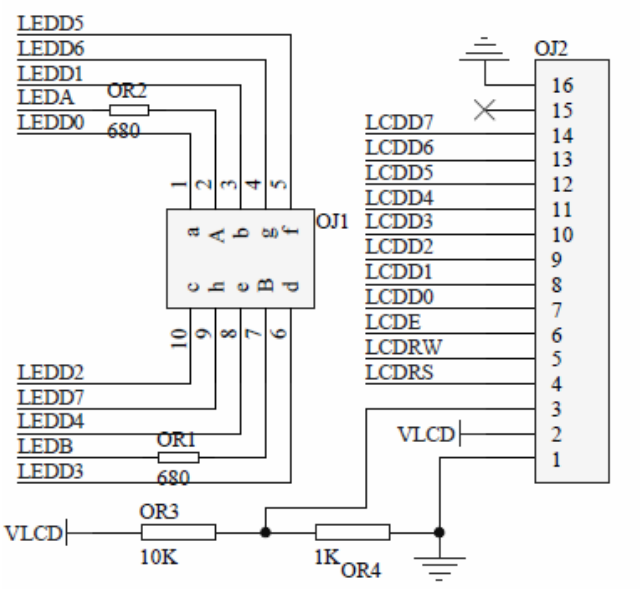


图2-11 液晶和数码管电路

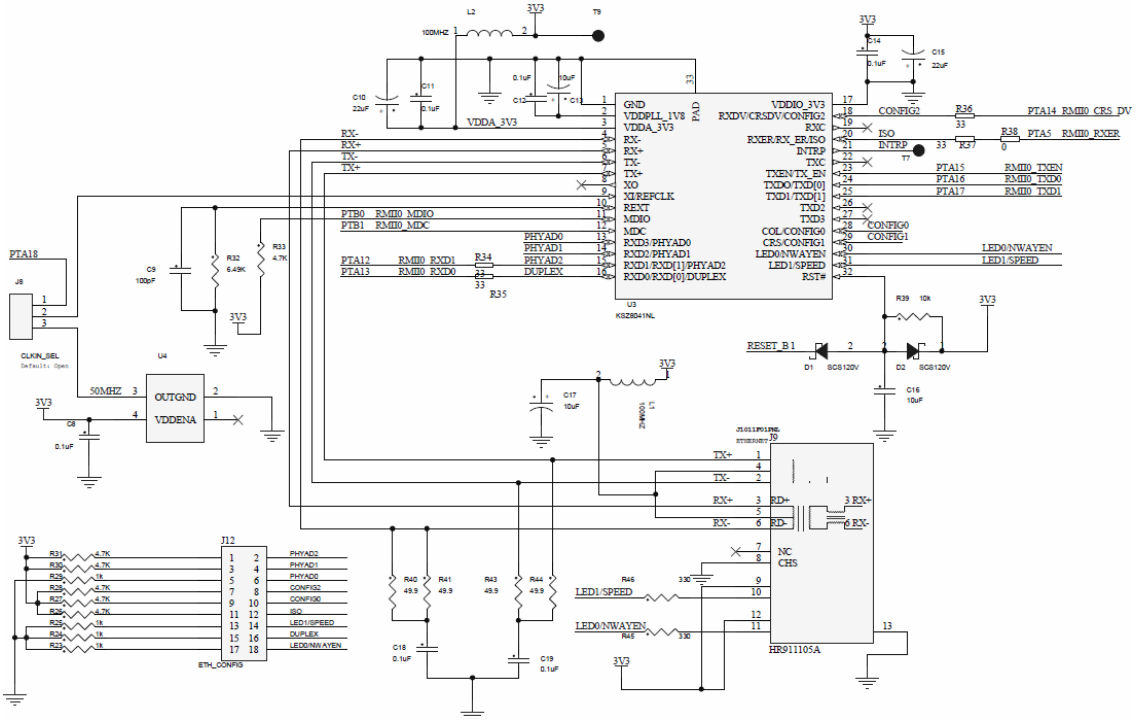


图2-12 以太网物理层电路

2.3.7 CAN 模块

K60N512 芯片内部集成 2 路 CAN 模块，模块需要外接物理层收发器，本设计使用 TJA1050 CAN 高速收发器。CAN 协议分为数据链路层和物理层。物

理层又分为物理信令层，物理媒体连接层和媒体相关接口层。K60N512 内部实现数据链路层和物理信令层。因此，只需要外接实现了物理媒体连接层和媒体相关接口层的 CAN 收发器即可实现 CAN 通讯。TJA1050 电路如 2-13 所示。

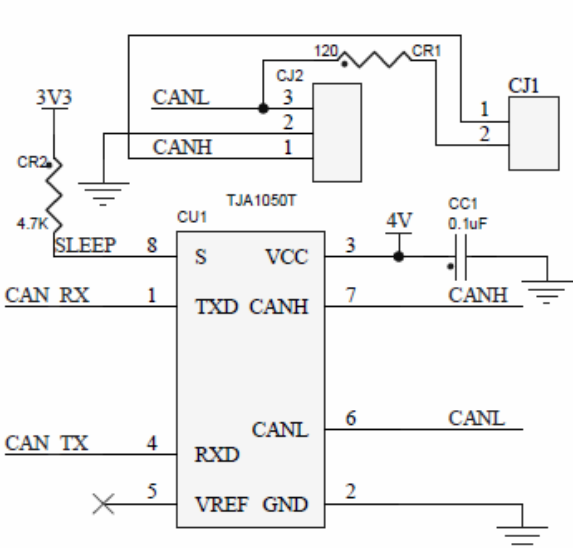


图2-13 CAN收发器电路

2.2.8 其他电路

除上述模块外，扩展板上还设计了小灯，键盘和蜂鸣器等模块。电路如图 2-14 所示。

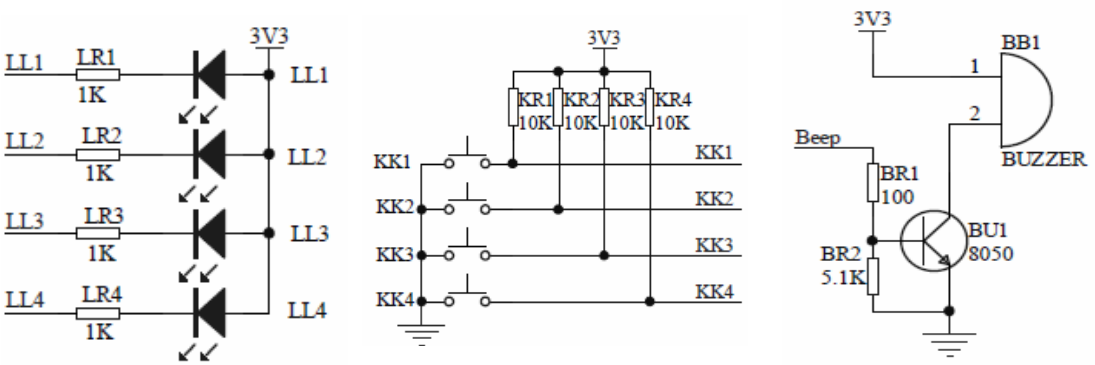


图2-14 小灯、键盘和蜂鸣器电路

2.4 Kinetis JTAG 调试器

Kinetis 内核为 Cortex-M4，内部集成 JTAG 调试接口。飞思卡尔官方发布开源的 OSJTAG 调试器，Kinetis 芯片同时支持 SEGGER 公司生产的 J-Link 调试器。本课题根据飞思卡尔官方发布的调试器电路图成功制作了 Kinetis 调试

器 OSJTAG。本课题制作的 OSJTAG 调试器在使用过程中工作稳定，可以在飞思卡尔官方的开发环境 CodeWarrior10.1 中使用，同时也可以在 IAR 环境中使用。本课题所有代码均使用 OSJTAG 在 K60N512 上实际调试过。

OSJTAG 是开源的 JTAG 调试器，而 J-Link 是商业应用的 JTAG 调试器，二者速度有着巨大的差距，在目标程序体积增大后速度差距更加明显。OSJTAG 的下载速度约是 J-Link 的 1/5。此外，J-link 有许多实用的应用程序，如 J-Flash。因此，本课题也制作 J-Link 的接口转接板来进行 K60N512 的调试。由于 OSJTAG 和 J-Link 的 JTAG 接口引脚定义不同，所以需要通过转换接口来转接[22]。图 2-15 为 OSJTAG 和 J-Link 的 JTAG 接口定义。

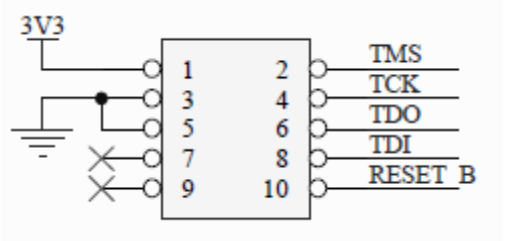


图2-15a OSJTAG接口定义

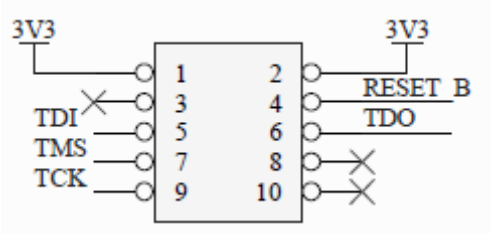


图2-15b J-Link接口定义

2.5 硬件测试

硬件电路板绘制结束后，将生产文件发给厂家生产。通常一周时间，电路板制作结束。然后要对电路板进行焊接和测试。电路板制作出来后，就进入了焊接阶段。电路板的质量受各种因素影响，包括电路板绘制的质量，厂家的生产质量，购买的元件质量和元件的焊接质量等。所以，在进行硬件电路板焊接测试时，要考虑多方面因素。

进入焊接阶段后，首先要购买质量上乘的元件。通常的元件会有原装和国产的区分，明显的是二者价格不同，有时甚至相差很大。原装的是原厂生产的元件，国产的是国内仿制或者二次回收的元件。电路板测试阶段应该尽量购买原装元件，这样可以忽略元件质量问题。

电路板的焊接技巧也是影响电路板质量的一大因素。这需要长期的焊接经验和调试经验积累。本课题使用的 K60N512 芯片封装为 144MAPBGA，这种封装无法手工焊接，因此，本课题委托电路板生产厂家对 K60N512 芯片使用机器进行焊接。其余元件均为笔者手工焊接调试。

2.5.1 K60N512 核心板测试

电路板测试过程中，最主要的测试工具就是万用表，特别是万用表的短接档。万用表的短接档可以测两根线路是否短接，如果短接万用表内蜂鸣器会响。拿到核心板后，首先测量电路板电源和地是否短路。在确保电源和地没有短路后，再测量 MAPBGA 的部分引脚是否和扩展接口接通，这是由于电路板使用四层板，表面无法看出线路走线。确保线路连通后，开始焊接元件。核心板上电容和电阻的封装均使用 0603 封装，体积很小，在焊接过程中要小心翼翼。所有元件焊接完成后，再次测试电源和地是否短接，这是因为焊接过程有可能造成电源和地连通，特别是进行电容焊接时，因为电容通常一端是电源，另一端是地。确保焊接完成的电路板的电源和地没有连通后，就可以连接调试器，进行调试了。调试中，首先使用 J-Link 调试器，然后使用本文设计的 OSJTAG 调试器，成功将 GPIO 程序下载到 K60N512 芯片中。核心板上闪烁的状态灯，证明核心板测试通过。

2.5.2 Kinetis 扩展板测试

Kinetis 扩展板的焊接和测试是按照模块进行的。首先必须测试电源模块，这必须在接插核心板之前进行。电源使用万用表电压档测试，扩展板也留出了电源指示灯。电源测试通过以后，就可以将核心板插到扩展板当中，这样才可以进行扩展板上功能模块的测试。扩展板上的模块测试是结合模块驱动程序完成的。测试中遇到困难的是网络模块。网络模块物理层收发芯片 KSZ8041NL 是 QFN32 封装，这种封装引脚在芯片底部，焊接时需要使用热风枪辅助进行，通常这种芯片都是机器焊接。笔者在焊接 KSZ8041NL 的造成了虚焊，在调试几天后才发现这个错误。

2.5.3 测试体会

通过本课题的 K60N512 核心板和 Kinetis 系列扩展板的设计、实现、焊接和测试，笔者对硬件设计和测试积累了一些经验和体会。

（1）需求分析

硬件设计开始之前，首先应该进行详细的需求分析，明确系统功能。对于评估系统，应该预留充分的测试接口和扩展接口，以备测试和扩展。

（2）模块分解

每个系统的硬件都可以分解为若干子模块，如最小系统、电源和液晶等。遵循硬件构件化设计原则，尽量做到模块功能独立，模块之间接口明晰，模块接口电路处引出测试点。

（3）电路评估

在电路的原理图设计阶段，各模块电路应该进行评估。常用的评估工具是万能板。通过万能板和导线搭建评估电路。这样可以保证硬件电路的正确性，防止硬件设计出错。同时，对于每种芯片，应该先确认是否容易购买。对于功能相同的芯片，更常用更容易购买的应该优先选择。

（4）PCB 设计

PCB 布板时，首先要考虑整体外形和元件布局。布局时遵循模块清晰原则，模块标注模块名称，关键信号使用丝印字体标注出来，这样一方面便于焊接，一方面也有利于测试。PCB 走线时，要考虑信号的电磁兼容性，模拟信号和数字信号应该隔离。电路板的走线应该在保证线间距的情况下尽量宽。电源信号和地信号应该宽于普通信号线。

2.6 本章小结

本章的主要工作总结如下：

（1）按照嵌入式硬件构件思想，实现了 K60N512 的核心板，给出核心板关键模块电路。核心板采用四层电路板布线。

（2）设计实现了 Kinetis 系列扩展板，详细阐述了各模块电路。设计实现了 Kinetis 芯片的调试器 OSJTAG。

（3）结合课题开发过程，归纳出了硬件开发板的测试方法和步骤，并阐述本课题硬件的测试体会。

第三章 模块驱动程序设计

本课题内容是实现 K60N512 的软硬件评估平台，硬件部分实现 K60N512 评估板和调试器，软件部分实现各模块驱动程序，并对驱动程序进行充分测试。第二章实现了 K60N512 核心板和扩展板的硬件设计部分，本章在嵌入式构件思想的指导下，详细阐述 K60N512 各模块驱动程序的实现过程，将驱动程序封装成规范的、可重用的、具有嵌入式构件特性的底层软件构件，并给出每个功能模块的测试例程和验证过程。

3.1 驱动程序设计原则

驱动程序驱动硬件模块，实现模块的初始化和模块基本功能。驱动程序直接同硬件打交道。模块驱动程序的实现必须完全按照硬件手册说明，直接操作模块寄存器。在嵌入式软件构件思想中，驱动程序即底层软件构件。在对底层软件构件进行设计时，最重要的是要对构件的共性和个性进行分析，抽取出构件的属性和对外接口函数，使得当一个软件构件应用到不同系统时，仅需修改构件的头文件，而尽量不要改动或尽少改动构件的源程序文件。

1）K60N512 底层软件构件设计原则

芯片每个模块均有特定的软件构件来驱动。模块软件构件由头文件和源文件组成。头文件中给出构件驱动函数声明和功能引脚宏定义，源文件中实现驱动程序，对于某些内部函数，只在源文件中出现，并且以 static 声明。驱动程序之间不能互相调用，业务功能由高层函数实现，如 AD 驱动函数不能调用 UART 的驱动函数。函数名称应该具有一定意义，通常使用模块加功能的形式，如 AD_Init 表示 AD 模块初始化函数。驱动程序实现过程中，严禁使用全局函数进行参数传递或结果返回。全局函数的值只能在主函数和中断函数中修改。

2）K60N512 底层软件构件的内容安排

K60N512 功能强大，内部集成了几乎所有工控领域常用的功能模块。限于篇幅有限，本章实现了几个具有典型意义的功能模块的底层软件构件，包括 UART、AD、CRC 和 Flash。本章对这些模块的软件构件给出了详细的实现过程，每个模块都给出了测试实例。K60N512 的以太网构件实现详见第四章 LwIP 移植部分。本章讨论的驱动程序实现方法对微控制器驱动程序的实现具有参考

意义。

3.2 K60N512 启动代码实现

在进行模块驱动程序实现之前，首先必须清楚 K60N512 的启动过程，实现 K60N512 的启动代码。目前大多基于 ARM 芯片的系统都是一个比较复杂的片上系统，多数硬件模块都是可配置的，可以通过软件来设置其需要的工作状态。因此在运行用户的应用程序之前，需要由一段代码来完成对系统的初始化。这一段代码就称为启动代码[23]。启动代码类似 PC 机中的 BIOS ROM 中的代码加上引导加载程序，如 GRUB 和 LILO。对于实现 Linux 的 ARM 芯片中，启动代码往往由 BootLoader 实现，常用的 BootLoader 有 UBoot，RedBoot。

由于芯片启动代码直接面对内核和硬件控制器进行编程，一般都是用汇编语言实现。在芯片上电复位后，需要设置中断向量表、初始化各模式堆栈、设置系统时钟频率等。启动代码通常使用汇编语言实现。启动代码完成系统初始化然后跳转到用户 C 程序。在 ARM 设计开发中，启动代码的编写是一个极重要的过程。启动代码随具体的目标系统和开发系统有所区别，但通常包含以下部分：中断向量表定义，地址重映射及中断向量表的复制，堆栈初始化，系统时钟频率设置，中断寄存器的初始化和进入用户程序等部分。

K60N512 允许将中断向量放置在 Flash 或者 RAM 中，但是上电时中断向量只能在地址 0x0000_0000 处。上电后，K60 首先从地址 0x0000_0000 处取栈地址，从地址 0x0000_0004 处取复位向量地址，然后跳转至复位向量地址处执行[5]。K60N512 芯片启动流程见图 3-1。

K60N512 芯片可以从 RAM 和 Flash 中执行代码，在从 RAM 中执行速度更快，因此，通常在启动后，将数据和中断向量表复制到 RAM 空间中。

3.2.1 关闭看门狗

复位向量中首先清零所有 CPU 通用寄存器，关闭总中断。然后关闭看门狗。看门狗在嵌入式设计中特别重要，它可以在芯片代码跑飞或死机的情况下复位芯片。嵌入式产品往往 24 小时全天候运行，为了保证程序一直运行正常必须在产品正式发布时使能看门狗。而在程序调试阶段，为了保证程序执行流程，往往先关闭看门狗中断。K60N512 的看门狗控制寄存器是只写一次寄存器，即上电后只能对其进行一次写入，如果想再次写入必须先解锁看门狗。看门狗

的解锁步骤是向看门狗的解锁寄存器是连续写入 0xC520 和 0xD928，两次写入不能超过 20 个时钟周期，否则解锁失败并产生看门狗中断。看门狗解锁后，通过配置看门狗控制寄存器的 WDOGEN 位来关闭看门狗。

如果程序在完成启动之后要使用看门狗，首先要解锁看门狗然后设置 WDOGEN 位来使能看门狗。看门狗使能以后，会在溢出时间超时后产生看门狗复位。程序中必须在溢出超时前“喂狗”。“喂狗”后看门狗模块重新计时，K60N512 的“喂狗”步骤是是向看门狗刷新寄存器连续写入 0xB480 和 0xA602。两次写入间隔不能超过 20 个时钟周期。

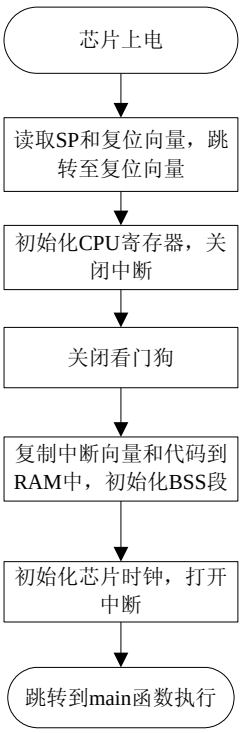


图3-1 K60N512芯片启动流程

3.2.2 复制中断向量表到 RAM 中

代码在 RAM 中执行的速度比在 Flash 中执行高。通常 ARM 芯片都会将中断向量表复制到 RAM 中。未初始化的数据段（BSS）应该清零。其中涉及到几个 Link 文件中定义的变量，包括：Flash 中断向量表地址 VECTOR_ROM，RAM 中断向量表地址 VECTOR_RAM。

Link 文件是在程序链接阶段使用，是链接器的命令文件。Link 文件定义了芯片存储器的分配和可执行代码地址空间的分配。通过修改 Link 文件可以将可执行代码链接到芯片 RAM 中或 Flash 中[24]。通常，在项目开发阶段，使用

将目标程序链接到芯片 RAM 中的 Link 文件，这样可以借助调试器直接在芯片 RAM 中调试程序。由于 RAM 下载代码更快，而且在 RAM 中运行代码也更快，所以这种方式可以大大方便程序调试，而且一定程度上延长了芯片 Flash 的使用寿命。在代码成熟以后，使用将目标程序链接到 Flash 中的 Link 文件，然后利用调试器将目标程序下载到芯片 Flash 中。本课题的代码调试就混合使用了这两种调试方式。

3.2.3 初始化芯片时钟

芯片主时钟是利用 MCG 模块中的 PLL 模块，通过倍频板上 50MHZ 的有源晶振得到的。在 Kinetis 芯片内部存在 3 种不同时钟：内核时钟，总线时钟和 Flash 时钟。内核时钟是基本时钟，其他时钟均从内核时钟分频得到。Kinetis 手册推荐了 3 种时钟配置方式，分别将内核时钟配置为 50MHZ，96MHZ 和 100MHZ。本课题实现了 SysClocKInit 函数来实现时钟初始化，函数定义如下。

```
//函数名称：SysClcoKInit
//函数功能：初始化系统时钟
//入口参数：
//   wantclcok：芯片初始化后的内核时钟，单位为 MHz：50，96 或 100
//   crystal_val：硬件电路中主晶振的频率，单位 MHz
//出口参数：无
void SysClocKInit（uint8 wantclcok，uint8 crystal_val）;
```

3.2.4 跳转至 main 函数执行

上述三步执行结束后，芯片已经进入了正常运行状态，可以执行 main 函数中用户定义的代码了。代码中直接调用 main 函数即可。

到此为止，芯片开始执行用户功能代码，芯片启动阶段结束。

3.3 UART 软件构件设计

串行通信是设备通讯时经常采用的一种通讯方式，由于接口和通信协议简单，因而在计算机串行通信领域得到了广泛的应用。UART 通信可采用单工、全双工或半双工的通信方式，数据传输格式为标准不归 0 传号/空号数据格式，通常使用的波特率（单位为 bps）有 1200、1800、2400、4800、9600、19200、38400、57600、115200 等。UART 可以提供奇偶校验保证数据正确性。为使信号传输得更远，美国电子工业协会 EIA 制订了串行物理接口标准 RS-232C。

RS-232C 采用负逻辑，-15V～-3V 为逻辑“1”，+3V～+15V 为逻辑“0”[25]。RS-232C 最大的传输距离是 30m，通信速率一般低于 20Kbps。UART 模块的驱动构件包括构件头文件和相应的源文件，主要完成对 UART 总线上数据帧的传输功能。

3.3.1 UART 构件实现

按照软件构件思想设计的 UART 头文件，包含下层构件头文件的#include 语句、用以描述 UART 属性的宏定义语句以及对外接口函数原型说明。由于 K60N512 芯片内部有 6 个 UART 接口，因此所有函数均包含 uartno 参数表示 UART 的接口号。

```
#ifndef __UART_H__
#define __UART_H__
#include “common.h”
#include “MK60N512VMD100.h”
#define RecFull(uartno) (UART_S1_REG(uartno) & UART_S1_RDRF_MASK)//接收缓冲区满
#define SendEmpty(uartno) (UART_S1_REG(uartno) & UART_S1_TDRE_MASK)//发送缓冲区空
void uart_init (int uartno, int sysclk, int uartclk);//初始化 UART
char uart_getchar (int uartno);//接收一个字符
void uart_putchar (int uartno, char ch);//发送一个字符
int uart_getstr (int uartno,char*,int);//接收 N 个字符
void uart_putstr (int uartno, char*,int);//发送 N 个字符
#endif /* __UART_H__ */
```

UART 是一种通讯接口，主要功能就是接收和发送数据，因此设计 UART 构件时设计了 UART 初始化函数，接收发送 1 个字符函数和接收发送 N 个字符函数。下面详细阐述初始化函数和接收发送 1 个字符的函数。N 个字符操作函数只是在 1 个字符操作函数的基础上又进行了封装。

1）UART 初始化函数

UART 初始化函数实现了 UART 数据格式，波特率，UART 模块时钟使能等工作；

（1）使能 UART 模块复用引脚。K60N512 芯片每个引脚都有几个复用功能，要使用 UART 功能，必须首先使能引脚的 UART 功能；

（2）使能 UART 模块时钟。K60N512 每个模块时钟均有控制开关，当模块不使用时，关闭时钟可以降低芯片功耗；

（3）设置 UART 数据格式、校验方式和停止位位数。通过设置 UART 模

块控制寄存器 C1 实现；

（4）设置 UART 数据通讯波特率。通过设置 UART 模块的波特率寄存器实现。需要注意的是 UART0~1 的时钟是从外设时钟分频得到。UART2~5 的时钟是从内核时钟分频得到的；

（5）设置是否允许接收和发送中断。通过设置 UART 模块的 C2 寄存器的 RIE 和 TIE 位实现。如果使能中断，必须首先实现中断服务程序；

（6）使能 UART 接收机和发送机。通过设置 UART 模块的 C2 寄存器的 RE 和 TE 位实现。

2）UART 接收一个字符

UART 模块状态寄存器 1 中的 RDRF 位标志接收是否完成，当接收完成后，RDRF 位为 1，UART 数据寄存器中收到一个字节数据。读取数据寄存器中数据后 RDRF 自动清零。

3）UART 发送一个字符

UART 模块状态寄存器 1 中的 TDRE 位标志发送缓冲区是否可用。当发送缓冲区可用时，TDRE 位为 1。将要发送的一个字节数据赋给 UART 数据寄存器后，UART 模块自动将该字节发出。

接收 N 个字符和发送 N 个字符的函数，是在一个字符的操作函数上进行了封装，这里不在赘述。

3.3.2 UART 构件测试

UART 构件测试可以通过两种方式实现：查询方式和中断方式。查询方式是通过不断调用接收函数，查询接收标志是否有效来判断是否成功接收数据。中断方式通过中断服务程序实现。当成功接收一个字节后，芯片自动跳转至中断服务程序，中断服务程序中调用接收一个字节的函数来接收数据。

本文分别采用两种方式进行测试。测试程序功能都是将从 PC 发来的数据发回 PC。如 PC 发送“helloworld”，则芯片发回“helloworld”。

测试发现 K60N512 的 UART 可以稳定的工作在 115200bps 下，由于串行发送时有起始位和停止位，综合数据速率为 11.52KB/S。当波特率上调至 256000 时，发现丢包。图 3-2 为本课题实现的 UART PC 端测试程序界面。

图3-2 UART模块测试软件

3.4 AD 软件构件设计

计算机或微控制系统能够处理的信息是数字量，而工业生产环境中被监控或检测的往往是像温度、压力和流量之类的连续变化的模拟量，这就需要 AD 转换模块采样这些模拟信息并转换成数字信息供内核处理。K60N512 内部集成的高性能的 AD 模块转换精度为 16 位，最高时钟频率 50MHz，最快四个时钟周期完成采样，采样速率高达 12.5MHZ[4]。AD 模块的软件构件包括构件头文件和相应的源文件。

3.4.1 AD 构件实现

K60N512 内部 AD 模块特点如下：

（1）最高高达 16 位的线性逐次逼近算法 AD；

（2）共有 4 对差分 AD 和 24 路单路 AD；

（3）多种输出模式：差分 AD：16 位、13 位、11 位和 9 位模式；单路 AD：16 位、12 位、10 位和 8 位模式；

（4）支持硬件平均值；

（5）支持转换完成中断和硬件求平均值中断；

（6）支持单次转换和连续转换；

（7）四种可选输入时钟源；

（8）支持自校准；

（9）模块工作时钟最高 50MHZ，最高转换频率 12.5MHZ。

按照软件构件思想设计的 AD 头文件，包含底层构件头文件的#include 语句、用以描述 AD 属性的宏定义语句和对外接口函数原型说明。

```
#ifndef __AD_H__
#define __AD_H__
#include “common.h”
#include “MK60N512VMD100.h”
#define ConvertEnd (ADC1_SC1A & ADC_SC1_COCO_MASK)//转换完成标志
void AD_init ();//初始化 AD 模块
uint32 ADValue (uint8 channel);//读取某一路 AD 转换结果
uint32 ADMid (uint8 channel);// 读取某一路 AD 转换结果，并进行中值滤波
#endif /* __ AD _H__ */
```

AD 模块主要功能是进行模数转换，这里设计了 AD 模块初始化，读取某一路转换结果和中值滤波的驱动函数。

1）AD 初始化函数

AD 初始化函数使能 AD 模块，配置 AD 模块时钟源和工作频率，并对 AD 模块进行自校准，这是 Kinetis 独有的特性。通过自校准，AD 模块可以得到非常高的精度。

（1）自校准 AD 模块。这一步必须在任何 AD 转换之前进行；

（2）配置配置寄存器（CFG），选择总线时钟作为时钟源，并进行四分频；

（3）配置状态和控制寄存器 2（SC2），选择软件转换触发，禁用比较功能；

（4）配置状态和控制寄存器 3（SC3），选择单次转换模式，使能硬件平均功能，选择 8 个采样平均；

（5）配置状态和控制寄存器（SC1:SC1n），选择转换是否是单端模式，禁止转换完成中断；

（6）配置增益放大寄存器，禁止增益放大功能。

2）AD 一路转换函数

AD 转换函数读取参数 channel 指定的通道的 AD 转换结果。

（1）设置 SC2 寄存器选择软件触发转换方式；

（2）设置 SC1 寄存器中的 ADCH 字段，设置要进行转换的 AD 通道号，并禁止转换完成中断；

（3）查询 SC1 的 coco 字段，当 coco 字段为 1 时，转换结束；

（4）返回 AD 模块的转换结果寄存器的值。

2）AD 中值滤波转换函数

中值滤波转换函数在单次转换函数的基础上进行封装，选择 3 次转换的中间值作为转换结果，这是最基础的滤波方式。常用的滤波方式还有均值滤波，即将 N 次转换的平均值作为转换结果。但是 K60N512 的 AD 模块内置硬件平均功能，使用硬件电路完成均值，均值数可配置为 4、8、16 和 32。硬件均值相对于软件滤波更加稳定高效。本设计的 AD 初始化函数中，使用 8 次硬件均值，即每次转换的结果已经在硬件上实现了 8 次的均值滤波。

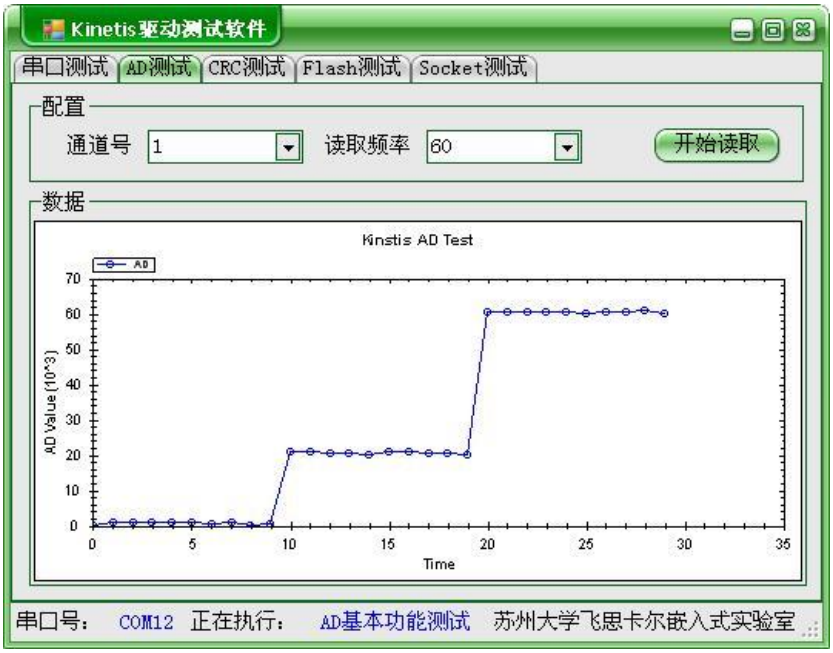


图3-3 AD模块测试软件

3.4.1 AD 构件测试

AD 模块的测试实例设计为：循环采集通道 20 引脚上的电压模拟量，采用中值滤波，并将转换得到的 16 位 AD 值通过 UART0 发往 PC 进行校验，以高高低低的字节顺序发送到 PC 的串口调试工具显示。图 3-3 为本课题实现的 PC 端 AD 模块测试软件。

在主函数中调用“ADMid(20);”，AD 模块使用的高参考电压为 3.3V，低参考电压为 0V。首先通道 20 引脚输入 3.3V 电压，理论上 AD 转换值应为(3.3/3.3)*65535=65535（十六进制为 0x0FFFF），实际采集的 AD 值稳定后为 0x0FFFA。然后通道 20 引脚输入 2.0V 电压，理论上 AD 转换值应为(2.0/3.3)*65535=39718（十六进制为 0x9B26），实际采集的 AD 值稳定后显示为 0x9B20。最后，通道 20 引脚输入 0V 电压，理论转换值应为(0/3.3)*65535=0（十六进制为 0x0000），AD 转换值为 0x0000，理论值与结果值吻合，证明转换正确。由于实际得到的 AD 值与理论值仅有 2‰左右的误差，在允许范围内，所以测试证明 AD 构件运行正确。

3.5 CRC 软件构件设计

CRC 校验是一种常用的差错检验技术，数据帧的接收方用一个生成多项式除以数据帧的内容计算出余数，并将所计算出的余数与发送方存储在数据帧中的检验码进行比较。在数据存储和数据通讯领域，CRC 无处不在。著名的通讯协议 X.25 的 FCS（帧检错序列）采用的是 CRC-CCITT。ARJ、LHA 等压缩工具软件采用的是 CRC32，磁盘驱动器的读写采用了 CRC16，通用的图像存储格式 GIF、TIFF 等也都用 CRC 作为检错手段[26]。

CRC 校验具有三个优点：侦错能力强、系统消耗小、使用简单。CRC 校验保护的单位是数据块。数据块的大小根据实际情况而定。每一个数据块均被看作是一个二进制多项式，即所有系数均为二进制（即 1 或 0）的多项式。例如，X^5+X^2+1 就是一个二进制多项式。二进制多项式的一个最大特点是：所有的二进制的位流（比特流）均可用一个二进制多项式表示。例如，101101 即可表示为：X^5+X^3+X^2+1。

由于CRC校验应用广泛，通常的嵌入式领域的C语言标准库中集成了CRC校验的库函数，但是这是使用软件生成的。我们知道 CRC 校验运算量大，运算中大量使用乘除法。因此，Kinetis 系列芯片内部集成了 CRC 校验的硬件生成电路，大大加快了 CRC 校验码的生成和匹配速度。

K60N512 内部 CRC 模块特性如下：

（1）CRC 硬件电路，支持 16 位和 32 位的转换寄存器；

（2）生成多项式和初始种子值可以设定；

（3）可配置硬件上实现对转换结果进行异或；

（4）32 比特的 CPU 寄存器编程接口。

3.5.1 CRC 构件实现

按照嵌入式软件构件思想，本文将 K60N512 构件封装为 CRC.c 和 CRC.h 两个文件。其中功能函数包括 CRC_Config，CRC_16 和 CRC_32 三个函数。

```
#ifndef __CRC_H__
#define __ CRC _H__
#include “common.h”
#include “MK60N512VMD100.h”
int CRC_Config(uint32 poly,uint32 tot,uint32 totr,uint32 fxor,uint32 tcrc);
uint32 CRC_16(uint32 seed,uint8_t *msg, uint32 sizeBytes);
uint32 CRC_32(uint32 seed,uint8_t *msg, uint32 sizeBytes);
#endif
```

CRC 模块功能是生成 CRC 校验码，K60N512 内部的 CRC 模块支持 16 位和 32 位 CRC。因此设计了 CRC 初始化和 16 位 CRC 转换和 32 位 CRC 转换函数。

1）CRC 配置函数

配置函数配置 CRC 模块的转换多项式，写转换模式，读转换模式，是否进行异或和 CRC 转换长度。返回 0 则表示配置成功。

其中转换多项式使用二进制表示。如多项式为 X^4+X^3+1 对应的二进制为 0b11001。多项式为 X^12+X^5+1，对应的二进制为 0b1000000100001，十六进制表示为 0x1021。

2）16 位 CRC 转换

用以下的步骤计算 16 位的 CRC 校验码：

（1）清除 CTRL[TCRC]比特来激活 16 比特的 CRC 模式。

（2）设置 CRC WAS 位为 1 来写 CRC 种子值。

（3）对 CRC 数据寄存器的低 16 位写入 16 位的种子值，数据寄存器的高 16 位没有使用。

（4）设置 CRC WAS 位为 0 来写 CRC 数据值。

（5）写入数据到 CRC 数据寄存器后，CRC 模块立即将转换结果放在 CRC 数据寄存器中，其中 CRC 数据寄存器的高 16 位没有使用。

（6）当所有的值都被写入后，直接从 CRC 数据寄存器低 16 位中读取 16

位的 CRC 校验码。

3）32 位 CRC 转换

用以下的步骤计算 32 位的 CRC 校验码：

（1）清除 CTRL[TCRC]比特来激活 32 比特的 CRC 模式。

（2）设置 CRC WAS 位为 1 来写 CRC 种子值。

（3）对 CRC 数据寄存器写入 32 位的种子值。

（4）设置 CRC WAS 位为 0 来写 CRC 数据值。

（5）写入 32 位数据到 CRC 数据寄存器后，CRC 模块立即将转换结果放在 CRC 数据寄存器中。

（6）当所有的值都被写入后，直接从 CRC 数据寄存器读取 32 位的 CRC 校验码。

图3-4 CRC模块测试软件

3.5.2 CRC 构件测试

CRC 构件测试中使用串口交互，可以使用 PC 自带的超级终端连接 K60 芯片，PC 发送命令给 K60，K60 将转换结果发回 PC。测试实例中分别对 CRC16 和 CRC32 进行了测试。图 3-4 为本课题实现的 PC 端 CRC 模块测试软件。CRC 模块的运算结果与 PC CRC 运算器生成的校验码完全一样。

3.6 Flash 软件构件设计

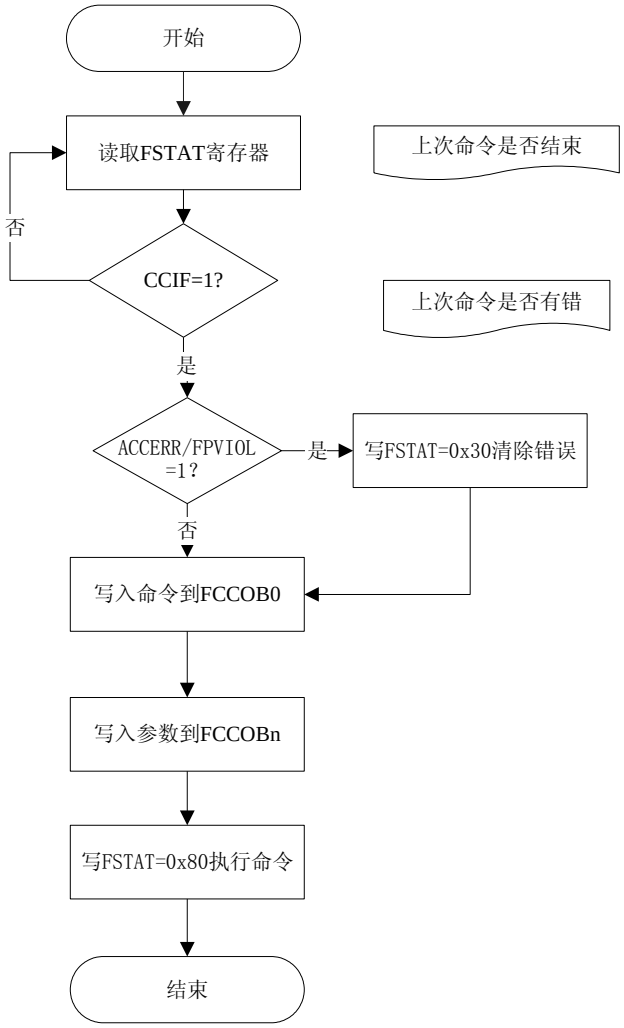


图3-5 Flash控制器命令执行流程

Flash 存储器的特点是断电后保持数据，它在微控制器中的作用类似通用计算机中的硬盘。微控制器使用 Flash 存储器来存储代码和数据，有时也将系统运行状态参数保存在 Flash 中。K60N512 内部集成 512K 的 Flash，分为 256 个扇区，每个扇区 2K 大小。Flash 存储器的特点是只能将数据从 1 写为 0，不能从 0 写为 1。Flash 存储器写入的最小单位为 4 个字节，擦除的最小单位为 1 个扇区，即 2K。

3.6.1 Flash 构件实现

Flash 存储器内部集成 Flash 控制器，控制器可以接收并执行命令。Flash

命令有长字写入，扇区写入，扇区校验，扇区擦除和整体擦除等。Flash 控制器执行命令必须按照严格的流程。K60N512 的 Flash 控制器命令执行流程见图 3-5。

按照软件构件思想设计的 Flash 模块驱动头文件，包含底层构件头文件的 #include 语句、用以描述 Flash 属性的宏定义语句以及对外接口函数原型说明。这里主要封装了 Flash 模块初始化，Flash 扇区擦除和 Flash 长字写入函数。

```
#ifndef __FLASH_H__
#define __ FLASH _H__
#include “common.h”
#include “MK60N512VMD100.h”
#define CMD_Pro_LongW 0x06 //长字写入命令
#define CMD_EraseSector 0x09 //页擦除命令
#define FlashCMDEnd (FTFL_FSTAT & CCIF) //Flash 命令执行结束
void flash_init ();//初始化 Flash 模块
uint32 flashwrite(uint32 destaddr,uint32 count,char const *buffer);//Flash 长字写入
uint32 flasherasesector(uint32 block_start);//Flash 扇区擦除
#endif /* __ FLASH _H__ */
```

1）Flash 初始化

Flash 初始化函数主要将 Flash 控制器初始化准备接收命令状态，首先清除 Flash 命令结果标志，清空 Flash 命令并禁止 Flash 保护。

2）Flash 长字写入命令

首先将 Flash 长字写入命令 0x06 写入到 FCCOB0 中，然后将要写入地址的低 24 位写入到 FCCOB1~FCCOB3 中，再将要写入的数据写入到 FCCOB4~FCCOB7 中。这里 24 位地址可以表达 16M 空间，足够表达 Kinetis 系列所有芯片 Flash。通过对 FCCOB 寄存器的写入，配置了所要进行的命令和命令的参数，然后可以执行图 3-5 所示的命令执行流程来完成命令。

Flash 页擦除命令和长字写入命令类似，只是配置的命令号和命令参数不同，这里不再赘述。

3.6.2 Flash 构件测试

Flash 构件测试是通过 K60N512 的串口和 PC 交互实现的。单片机代码给出了 Flash 长字写入、Flash 页擦除和 Flash 数据读取的接口。测试结果表明驱动函数运行正常。图 3-6 为本课题实现的 PC 端 Flash 模块测试软件。

图3-6 Flash模块测试

3.7 本章小结

本章的主要工作总结如下：

（1）详细分析 K60N512 芯片启动流程，实现了 K60N512 芯片启动代码。

（2）在嵌入式软件构件思想指导下实现了 K60N512 的常用基本模块的驱动函数，其中包括 UART、AD、CRC 和 Flash 模块。

（3）对每个驱动模块给出了测试实例和测试步骤，充分验证了驱动函数的正确性和稳定性。

第四章 FreeRTOS 和 LwIP 的移植

当嵌入式系统比较复杂的时候，需要软件中间件来进行管理，这个中间件就是嵌入式操作系统。嵌入式操作系统的功能包括硬件隔离，内存管理，进程管理，文件系统管理，网络管理等。当今有很多流行的嵌入式操作系统，比如嵌入式 Linux，eCos，vxworks，Palm，Windows CE，μc/os 等[27]。但是一些嵌入式系统的存储空间很有限，控制设备也不是很复杂，因此不适合用上述这些操作系统。很多使用 8 位或 16 位单片机为微控制器的嵌入式系统，传统的开发方法是不用任何操作系统，采用主循环配合中断的模式开发，这类系统一般控制对象比较单一，控制过程也比较简单，若采用操作系统反而会影响效率。若控制对象和控制过程变得复杂，采用这种方法就不是很适合，比如嵌入式 web 服务器。嵌入式 Web 服务器根据运行时客户端浏览器的请求动态响应。此时为了提高系统响应速度提高 CPU 综合利用率，就需要引入易于使用，效率高且资源要求低的嵌入式实时操作系统作为支撑。

K60N512 内部有 128K 的 RAM 和 512K 的 Flash，不足以支持 Linux 或 Window CE 等大型嵌入式操作系统的移植，但对于微型的嵌入式实时操作系统已经足够，如常见的 μc/os-II，eCos 和 FreeRTOS 等。本章选择移植 FreeRTOS 实时操作系统和 LwIP 协议栈。这两个都是开源的优秀项目，并且已经移植到了很多嵌入式内核中，在很多应用中验证了它们的健壮性。

4.1 FreeRTOS 在 K60N512 上的移植

4.1.1 FreeRTOS 简介

FreeRTOS 是由 InterNiche 公司于 2005 年开发的一个开源的免费多任务实时操作系统，目前最新版本是 6.1.1[9]。它很适合一些资源有限的小型嵌入式系统。用户只需要定义创建一系列任务，每个任务会在各自的堆栈空间中运行，任务可以睡眠，也可以被唤醒[42]。FreeRTOS 易于学习，移植也相对简单，目前已经移植到 PIC、Coldfire、AVR 和 ARM 等多数常见内核中。用户不用花很大功夫就可以灵活运用它，FreeRTOS 的最小配置仅需要约 0.5K 的 RAM 资源和约 1K 的 FLASH 资源[28]，运行效率高。

4.1.2 FreeRTOS 和其他 RTOS 的比较

常见的嵌入式操作实时操作系统有 μCOS-II，linux 和 FreeRTOS，表 4-1 对这 3 种实时操作系统进行了比较。

表 4-1 常见嵌入式实时操作系统比较

特性	FreeRTOS	μCOS-II	Linux
信号量，队列	支持	支持	支持
任务调度方式	优先级，时间片	优先级调度	优先级，时间片
TCP/IP 协议栈	LwIP，uIP	μC-IP	所有功能组件
任务数	无上限（链表方式）	64（任务表）	无上限
版权	GPL，代码公开	用于产品时，需支付版权费	GPL，代码公开
其他优点	微内核 RTOS，结构简单	文档丰富，支持库丰富	开源资源免费，可靠，稳定
缺点	文档不够	需支付版权	系统资源要求高

4.1.3 FreeRTOS 移植

FreeRTOS 目前已经移植到了各种常见的内核中，包括 Coldfire、HCS12、PIC、AVR、ARM7、ARM9 和 Cortex-M3 等。其官方网站上给出了多个芯片的移植范例，这些范例涵盖常见的开发平台，如 Keil、IAR、Eclipse、Codewarrior 和 ADS 等[9]。对于官方已经成功移植的芯片，可以通过复制 FreeRTOS 的范例快速的在自己的开发板上运行 FreeRTOS。然而由于 Cortex-M4 内核是在 2009 年下半年推出，而 Kinetis 系列芯片是在 2010 年下半年推出的，目前 FreeRTOS 官方还没有提供这款芯片的移植范例。

1. FreeRTOS源代码结构

要移植 FreeRTOS 首先要弄清楚它的源代码结构。FreeRTOS 的源代码树见表 4-2。FreeRTOS 和 μCOS-II 非常类似，都属于微型的实时操作系统，其移植过程也很类似。由于 FreeRTOS 文档资料比较少，本文移植过程主要参考 μCOS-II 的移植手册，同时参考了 FreeRTOS 针对其他芯片的的官方范例。

表 4-2 FreeRTOS 源代码结构

内核无关	源文件	croutine.c，list.c，queue.c，tasks.c
	头文件	croutine.h， FreeRTOS.h， list.h， mpu_wrappers.h，portable.h，projdefs.h，queue.h，StackMacros.h，task.h
内核相关	Port.c，portasm.s，portmacro.h	

如表 4-2 中所示，FreeRTOS 的核心代码是与内核无关的四个 C 文件，在

移植过程中，与内核无关的代码不需要更改。移植过程主要集中在 3 个文件里面：portmacro.h，port.c，portasm.s。其中 portmacro.h 主要包含与编译器相关的数据类型的定义、堆栈类型的定义、几个宏定义和函数说明。而 port.c 中则包含与移植有关的 C 函数，包括堆栈的初始化函数、任务调度器启动函数、临界区的进入与退出函数和时钟中断服务程序等。portasm.s 中则包含与移植有关的汇编语言函数，包括上下文切换、开/关中断和任务切换等[33]。

2. portmacro.h

本文件包含编译器相关的数据类型的定义、堆栈类型的定义、几个宏定义和函数说明。移植代码如下，其中开关中断和进入退出临界区的函数体在 portasm.s 文件中实现。注意其中的 portSTACK_GROWTH 表示栈的生长方向。如果内核栈是从高地址向低地址生长则定义为 1，否则为-1。

```
#define portCHAR          char    //字符类型
#define portFLOAT          float //浮点类型
#define portDOUBLE        double //双精度浮点数
#define portLONG          long    //长整数
#define portSHORT         short    //短整数
#define portSTACK_TYPE unsigned portLONG //栈中数据类型
#define portBASE_TYPE    long //内核字长
#define portSTACK_GROWTH (-1 ) //栈生长方向，-1：递减，1：递加
#define portDISABLE_INTERRUPTS()       vPortSetInterruptMask() //关中断
#define portENABLE_INTERRUPTS()        vPortClearInterruptMask() //开中断
#define portENTER_CRITICAL()           vPortEnterCritical() //进入临界区
#define portEXIT_CRITICAL()             vPortExitCritical() //跳出临界区
```

3. portasm.s

Portasm.s 中实现了开关中断的汇编函数和进入退出临界区的汇编函数。开中断代码如下：

```
vPortSetInterruptMask: /*关中断函数*/
CPSID   i    //关闭中断
bx r14    //函数返回
vPortSetInterruptMask: /*开中断函数*/
CPSIE   i    //打开中断
bx r14    //函数返回
vPortSetInterruptMask: /*进入临界区函数*/
PUSH {XPSR} //标志寄存器压栈
CPSID   i    //关闭中断
```

```
bx r14    //函数返回
vPortSetInterruptMask: /*退出临界区函数*/
POP {XPSR} //标志寄存器出栈
bx r14    //函数返回
```

进入临界区后会关闭中断，禁止调度器，但这里必须将标志寄存器压栈，这样才能在退出临界区时将标志寄存器恢复到进入临界区之前的标志状态。

4. port.c

Port.c 文件是移植过程中修改量最大的文件，它实现了任务栈初始化函数 pxPortInitialiseStack，调度开始函数 xPortStartScheduler，系统滴答中断函数 xPortSysTickHandler 和申请软件中断函数 vPortYieldFromISR。

任务栈初始化函数用于初始化上下文切换之前的栈结构。调用任务栈初始化函数后再调用软件中断函数 vPortYieldFromISR 即可以实现上下文切换。

```
portSTACK_TYPE    *pxPortInitialiseStack(    portSTACK_TYPE    *pxTopOfStack,
pdTASK_CODE pxCode, void *pvParameters )
{
//仿真栈结构，栈结构和中断发生时一样
//这样申请软件中断就能实现上下文切换
pxTopOfStack--;
*pxTopOfStack = 0x01000000;  //标志寄存器压栈
pxTopOfStack--;
*pxTopOfStack = ( portSTACK_TYPE ) pxCode;//PC 压栈，类似中断返回函数地址
pxTopOfStack--;
*pxTopOfStack = 0;  //LR 压栈
pxTopOfStack -= 5;  // R12, R3, R2  和  R1 压栈
*pxTopOfStack = ( portSTACK_TYPE ) pvParameters; //参数是通过 R0 来传递的
pxTopOfStack -= 8;  // R11, R10, R9, R8, R7, R6, R5 和 R4 压栈
return pxTopOfStack;
}
```

调度开始函数 xPortStartScheduler 用于使能调度器，其主要负责初始化滴答中断，即负责初始化 FreeRTOS 的心跳函数。

```
portBASE_TYPE xPortStartScheduler( void )
{
extern uint32 VECTOR_RAM[];                  //引用 RAM 中的中断向量表地址
VECTOR_RAM[15]=(UINT32) xPortSysTickHandler;//设置滴答中断服务函数
SYST_RVR = (configCPU_CLOCK_HZ / configTICK_RATE_HZ) - 1UL;
//使能滴答定时器，并且使能定时器中断
```

```
SYST_CSR=portSYSTICK_CLK | portSYSTICK_INT | portSYSTICK_ENABLE;
//执行第一个任务
vPortISRStartFirstTask();
//不会执行到这里
return 0;
}
```

系统滴答中断函数 xPortSysTickHandler 用于将滴答计数加 1，并根据配置判断是否进行上下文切换。

```
void xPortSysTickHandler( void )
{
    unsigned long ulDummy;
    //如果使用可剥夺调度，则强制进行上下文切换
    #if configUSE_PREEMPTION == 1
        *(portNVIC_INT_CTRL) = portNVIC_PENDSVSET;
    #endif
    ulDummy = portSET_INTERRUPT_MASK_FROM_ISR();
    {
        vTaskIncrementTick();//滴答计数加 1
    }
    portCLEAR_INTERRUPT_MASK_FROM_ISR( ulDummy );
}
```

软件中断函数 vPortYieldFromISR 实现软件中断，软件中断通过设置 SCB_ICSR 寄存器向系统中断控制器申请软件中断。

```
void vPortYieldFromISR( void )
{
    //申请软件中断
    SCB_ICSR = portNVIC_PENDSVSET;
}
```

实现了 K60N512 对应的 portmacro.h，portasm.s 和 port.c 后，FreeRTOS 的移植就结束了。FreeRTOS 提供了很多配置选项对操作系统进行配置。通过配置选项可以优化操作系统性能，减小最终可执行文件的体积。下面分析 FreeRTOS 的配置选项。4.1.4 节对移植后的 FreeRTOS 进行了测试。

5. FreeRTOS配置

FreeRTOS 提供了诸多配置项，通过这些配置项可以打开或禁用某些特性。通常在项目中使用 FreeRTOS 时，会添加 FreeRTOSConfig.h 文件对 FreeRTOS 进行配置。

下面是本文移植的 FreeRTOS 的配置代码。

```
#define configUSE_PREEMPTION 1    //使用可剥夺的调度算法
#define configUSE_IDLE_HOOK    0 //不使用空闲任务钩子函数
#define configMAX_PRIORITIES    ( ( unsigned portBASE_TYPE ) 5 ) //最多的优先级数目
#define configUSE_TICK_HOOK    0//不使用滴答钩子函数
#define configCPU_CLOCK_HZ   ((unsigned long)96000000)//CPU 工作频率
#define configTICK_RATE_HZ ((portTickType)1000)//滴答中断频率 1000HZ
#define configMINIMAL_STACK_SIZE ((unsigned short )80)//任务栈的最小值
#define configTOTAL_HEAP_SIZE ((size_t)(19 * 1024))//所有任务堆的最大值
#define configMAX_TASK_NAME_LEN (12 )//任务名称的最大长度 12 个字节
#define configUSE_TRACE_FACILITY 1//使能 TRACE 功能
#define configUSE_MUTEXES 1//使用 Mutex
#define configUSE_COUNTING_SEMAPHORES 0//不使用计数信号量
#define INCLUDE_vTaskPrioritySet 1 //包含设置任务优先级 API
#define INCLUDE_uxTaskPriorityGet 1 //包含获取任务优先级 API
#define INCLUDE_vTaskDelete 1//包含删除任务优先级 API
#define INCLUDE_vTaskCleanUpResources 0//不包含清除资源 API
#define INCLUDE_vTaskSuspend 1//包含任务挂起 API
#define INCLUDE_vTaskDelayUntil 1//包含任务 DelayUntil API
#define INCLUDE_vTaskDelay 1//包含任务 Delay API
```

4.1.4 FreeRTOS 移植测试

FreeRTOS 移植成功结束，需要对移植的代码进行测试。FreeRTOS 官方提供了多个测试源文件。详见表 4-3。添加相应的测试源文件即可以测试相对应的操作系统功能。本设计还使用了另外一个源文件 partest.c 来测试移植的 FreeRTOS。

表 4-3 FreeRTOS 测试源代码

文件名	说明
BlockQ.c	任务队列测试
comtest.c	串口通讯测试
death.c	任务 Kill 测试
dynamic.c	任务同步测试
events.c	事件测试
flash.c	小灯闪烁测试
flop.c	浮点数测试
integer.c	整型数测试

print.c	输出测试
semtest.c	信号量测试

Partest 是一个经典的名字，源自于并行端口测试“parallel port test”，FreeRTOS 的很多测试代码中都包含它。但现在它的功能已经改变，它主要功能是测试 LED 小灯。

Partest 头文件如下：

```
#ifndef PARTEST_H
#define PARTEST_H
void vParTestInitialise(void);//初始化小灯
//设置小灯状态
void vParTestSetLED( unsigned portBASE_TYPE uxLED, signed portBASE_TYPE xValue );
//闪烁小灯
 void vParTestToggleLED( unsigned portBASE_TYPE uxLED );//小灯闪烁
#endif
```

各个函数的实现类似第三章 UART 构件的实现，另外添加了临界区的处理。

测试主函数建立了 3 个任务，每个任务控制一盏小灯闪烁。小灯任务的代码如下：

```
static void vLEDFlashTask( void *pvParameters )
{
	portBASE_TYPE	LEDNo =*(( portBASE_TYPE*) (pvParameters));
	for(;;)
	{
		vParTestToggleLED(LEDNo );
		vTaskDelay( 500 );
	}
}
```

4.2 LwIP 在 K60N512 上的移植

4.2.1 LwIP 简介

LwIP 是瑞士计算机科学院开发的一个开源的 TCP/IP 协议栈,目前最新版本为 1.4.0[18]。它主要关注的是怎么样减少内存的使用和代码的大小，这样就可以让 LwIP 适用于资源有限的小型平台，如嵌入式系统[39]。为了简化处理过

程和内存要求，LwIP 对 API 进行了裁减。

LwIP 是轻型(Light Weight)IP 协议，有无操作系统的支持都可以运行。LwIP 实现的重点是在保持 TCP 协议主要功能的基础上减少对 RAM 的占用，一般它只需要几 K 字节的 RAM 和 30K 左右的 ROM 就可以运行[29]，这使 LwIP 协议栈适合在低端的嵌入式系统中使用。

其主要特性如下[30]。

1、支持多网络接口下的 IP 转发；

2、支持 ICMP 协议；

3、包括实验性扩展的 UDP 协议；

4、包括阻塞控制、RTT 估算、快速恢复和快速转发的 TCP 协议；

5、提供专门的内部回调接口(Raw API)，用于提高应用程序性能；

6、支持 socket。

表 4-4 常见嵌入式 TCP/IP 协议栈比较

协议栈	LwIP	uIP	μc/IP	Linux TCP/IP 协议
操作系统相关性	操作系统无关性，可在有无操作系统下独立运行	操作系统无关性，只能在无操作系统下独立运行	针对μC/OS系统设计，但可移植到其它操作系统	与 Linux 系统紧密相关
移植难度	比较容易地移植到各种操作系统上	可移植到各种不同的操作系统，但较麻烦	只能在与μC/OS系统兼容的嵌入式系统中运行	只能在与Linux系统兼容的嵌入式系统中运行
存储使用量	40KB ROM 和几 KB RAM	几 KB ROM 或几百字节 RAM	30~60KB	70KB~150KB
裁减难度	可裁减	可裁减	可裁减	裁减优化有难度
TCP/IP 协议	部分 TCP/IP 协议	功能简单	部分 TCP/IP 协议	完整的 TCP/IP 协议
Socket	支持	不支持	支持	支持
对应用层的支持度	多种方式与应用程序通信，如邮箱、消息队列、内部回调函数接口等	应用层接口较复杂，功能简单，这些都限制了uIP在一些较高要求场合下的应用	对网络应用程序的支持不足，μC/IP在文档支持与软件升级管理上还有较多不足，功能还需完善	参考的资料较多，并有庞大的开源社区提供支持

LwIP 中所有 TCP/IP 协议栈都在一个进程当中，这样 TCP/IP 协议栈就和

操作系统分开了。而应用层程序既可以使用单独的进程，也可以驻留在 TCP/IP 进程中。如果应用程序是单独的进程，可以通过操作系统的邮箱，消息队列等方式和 TCP/IP 进程进行通信。如果应用层程序驻留在 TCP/IP 进程中，那应用层程序就利用内部回调函数接口(Raw API)和 TCP/IP 协议栈通信，LwIP 的系统结构见图 4-1。整个 TCP/IP 协议栈都在同一个任务(TCPIP_thread)中，应用层程序既可以是独立的任务，也可以在 TCPIP_thread 中利用内部回调函数接口(Raw API)和 TCP/IP 协议栈通信[29]。

除了 LwIP 以外，现在常见的嵌入式 TCP/IP 协议栈还有 uIP，μc/IP 和 Linux TCP/IP 协议栈。这几种协议栈的比较见表 4-4[31,43]。从表 4-4 中可以看出，uIP 不支持 socket，LwIP 功能更强大。而 μc/IP 只能和 μc/OS 配合使用，Linux TCP/IP 协议栈对硬件要求太高。本课题已经移植了 FreeRTOS 操作系统，综合比较，LwIP 更加适合 K60N512 芯片。

4.2.2 LwIP 结构

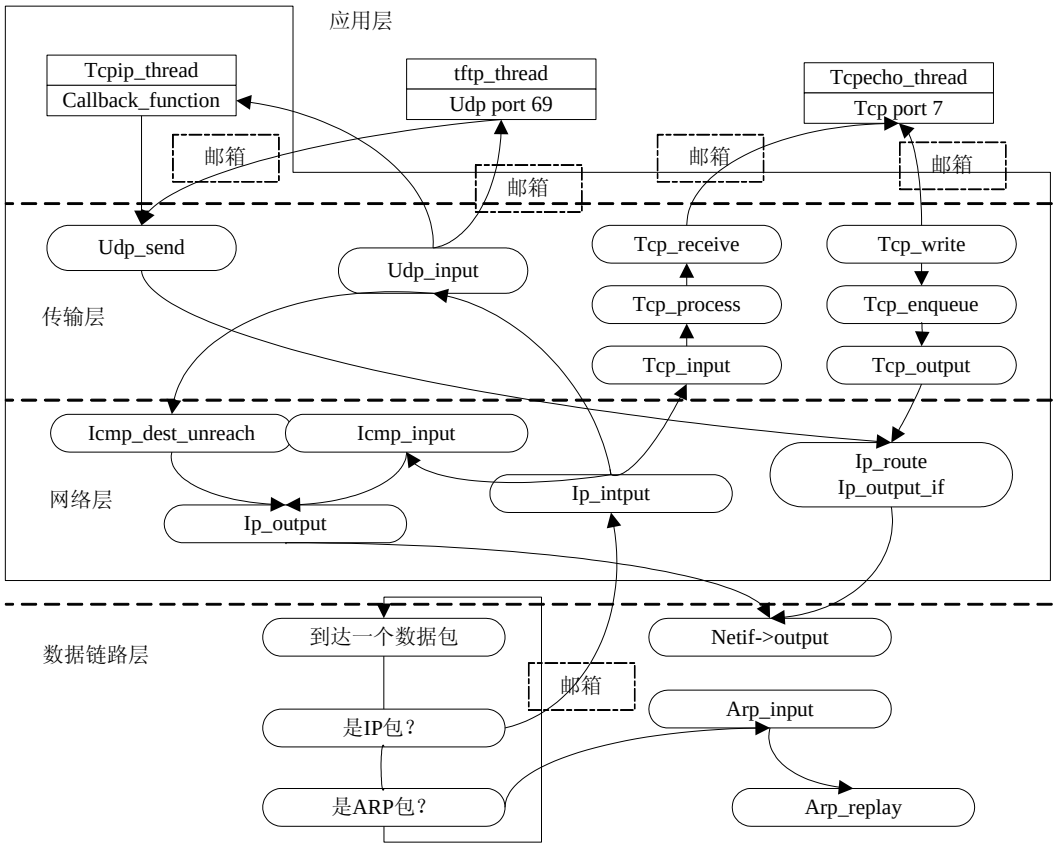


图4-1 LwIP系统结构图

LwIP 实现了链路层以上的功能，其核心代码主要位于网络层和运输层。应用层协议实现不是 LwIP 的研究范畴，但为了验证协议栈，LwIP 源代码中附带了简单的应用层协议，如 HTTP 协议。LwIP 能够发送、接收并转发数据包，但是不处理 IP 分片包，也不能处理携带 IP 参数选项的数据包。但这对大多数的网络应用来说没有问题[29]。LwIP 系统结构见图 4-1。

4.2.3 LwIP 移植

LwIP 是微型的 TCP/IP 协议栈，它的核心代码是与硬件无关的，核心代码工作在数据链路层之上。对 LwIP 的核心功能抽象后，可以知道 LwIP 工作在硬件之上，同时又是在操作系统支持下运行。因此要移植 LwIP 需要进行两方面工作，首先需要移植操作系统支持代码，此外还要移植数据链路层以下的代码。

LwIP 为了能方便的一直到不同操作系统中，没有使用特定操作系统的系统调用和数据结构，而是在操作系统和 LwIP 之间增加了一个操作系统模拟层。操作系统模拟层为操作系统服务提供了统一的接口[44]。将 LwIP 移植到一个新的操作系统时，只需要实现特定操作系统的操作系统模拟层。没有操作系统的环境也可以实现虚拟的操作系统模拟层来移植 LwIP[32]。操作系统模拟层对操作系统服务如定时器、进程同步和消息传递机制等提供了统一的接口。原则上，移植 LwIP 到其他的操作系统上，对于特定操作系统只要实现其操作系统模拟层，操作系统接口层对应源文件 sys_arch.c。

移植 LwIP 还需要实现网卡接口层。网卡接口层实现了数据链路层通讯和物理层通讯，这部分和硬件关系密切，LwIP 同样将它抽象出来。

1. 操作系统模拟层

操作系统模拟层（sys_arch）存在的目的主要是为了方便 LwIP 的移植，它在底层操作系统和 LwIP 之间提供了一个接口。移植 LwIP 到一个新的目标系统时，只需修改这个接口即可。

sys_arch 为 LwIP 提供信号量（semaphores）和邮箱（mailboxes）两种进程间通讯方式（IPC）。sys_arch 还提供了多线程支持。对于仅需要基本功能的用户来说，可以不去实现多线程。除了上文所述的 sys_arch 源文件需要实现的功能外，LwIP 还要求用户提供几个头文件，这几个头文件包含 LwIP 使用的宏定义。下文将详细讲述 sys_arch 源文件及头文件的实现。

sys_arch 中主要实现信号量，邮箱和多线程。本文移植中采用 FreeRTOS 中的信号量表示 LwIP 中信号量，使用队列实现邮箱，使用任务实现多线程。类型定义在 sys_arch.h 文件中，移植代码如下：

```
#define SYS_MBOX_NULL (xQueueHandle)0
#define SYS_SEM_NULL   (xSemaphoreHandle)0
#define SYS_DEFAULT_THREAD_STACK_DEPTH configMINIMAL_STACK_SIZE
#define SYS_THREAD_NULL NULL
typedef xSemaphoreHandle sys_sem_t;
typedef xQueueHandle sys_mbox_t;
typedef xTaskHandle sys_thread_t;
```

信号量采用 FreeRTOS 中的二值信号量。邮箱用于消息传递，使用 FreeRTOS 中队列实现，允许多条消息投递到这个邮箱，也可以每次只允许投递一个消息。线程使用 FreeRTOS 中的任务实现。

下文首先分析 sys_arch 源文件中 9 个功能函数的实现。这 9 个函数见表 4-5。

表 4-5 sys_arch 源文件中的功能函数

函数类型	函数名	说明
信号量函数	sys_sem_new	新建信号量
	sys_sem_free	释放信号量
	sys_sem_signal	发送信号
	sys_arch_sem_wait	等待信号
邮箱函数	sys_mbox_new	新建邮箱
	sys_mbox_free	释放邮箱
	sys_mbox_post	投递邮件
	sys_arch_mbox_fetch	收取邮件
线程函数	sys_thread_new	新建线程

1）sys_sem_new

该函数建立并返回一个新的信号量。参数 count 指定信号量的初始状态。

```
sys_sem_t   sys_sem_new(u8_t count)
{
    xSemaphoreHandle   xSemaphore;
    portENTER_CRITICAL();
    if(count == 0)   // 初始状态
    {
        xSemaphoreTake(xSemaphore,1);
    }
    vSemaphoreCreateBinary( xSemaphore );
    portEXIT_CRITICAL();
```

```
    if( xSemaphore == NULL )
        return NULL;
    else
        return xSemaphore;
}
```

2）sys_sem_free

该函数释放信号量，直接释放信号量内存。

3）sys_sem_signal

该函数发送一个信号，使用 FreeRTOS 的 xSemaphoreGive 释放信号量。

4）sys_arch_sem_wait

该函数等待指定的信号并阻塞线程。该函数的参数 timeout 指定等待时间。当 timeout 为 0，线程会一直被阻塞至收到指定的信号；非 0，则线程仅被阻塞至指定的 timeout 时间（单位为毫秒）。在 timeout 参数值非 0 的情况下，返回值为等待指定的信号所消耗的毫秒数。如果在指定的时间内并没有收到信号，返回值为 SYS_ARCH_TIMEOUT。

5）sys_mbox_new

该函数建立一个空的邮箱。

```
sys_mbox_t sys_mbox_new(int size)
{
    xQueueHandle mbox;
    mbox = xQueueCreate( size, sizeof( void * ) );//新建一个队列实现邮箱
    return mbox;
}
```

6）sys_mbox_free

该函数释放一个邮箱。

7）sys_mbox_post

该函数投递一则消息到指定的邮箱中。

8）sys_arch_mbox_fetch

该函数阻塞线程直至邮箱收到至少一条消息。最长阻塞时间由 timeout 参数指定（与 sys_arch_sem_wait()函数类似）。msg 是一个结果参数，用来保存邮箱中的消息指针（即*msg=ptr），它的值由这个函数设置。“msg”参数有可能为空，这表明当前这条消息应该被丢弃。返回值表示等待的毫秒数或者 SYS_ARCH_TIMEOUT－－如果时间溢出的话。

9）sys_thread_t sys_thread_new(void(*thread)(void *arg), void *arg, int prio):

该函数启动一个由函数指针 thread 指定的新线程，arg 将作为参数传递给 thread()函数，prio 指定这个新线程的优先级。返回值为这个新线程的 ID，ID 和优先级由底层操作系统决定，FreeRTOS 中 ID 值越小，任务优先级越低。sys_arch 中维持 tasks 链表表示当前所有线程，新建的线程被分配到链表末尾。

除了 sys_arch.c 源文件中的这几个函数外，操作系统模拟层还需要几个头文件 cc.h，cpu.h，sys_arch.h[30]。

cc.h 文件定义了芯片基础类型，包括 u8_t，s8_t，u16_t，s16_t，u32_t，s32_t 和 mem_ptr_t。

cpu.h 定义芯片大小端，Kinetis 芯片为小端，所以定义 BYTE_ORDER 为 LITTLE_ENDIAN。

2. 网卡接口层

对于 LwIP 的网卡接口层，LwIP 的作者已经为我们完成了绝大部分，我们只需在作者设计好的框架内完成与底层硬件相关的部分即可，没有多少可自由发挥的空间。网卡接口层框架见图 4-2。这些函数功能包括初始化底层网络接口、链路层请求发送数据、初始化网卡、接收线程、获得一帧数据、获得数据包的长度、读取数据包和发送数据包等。

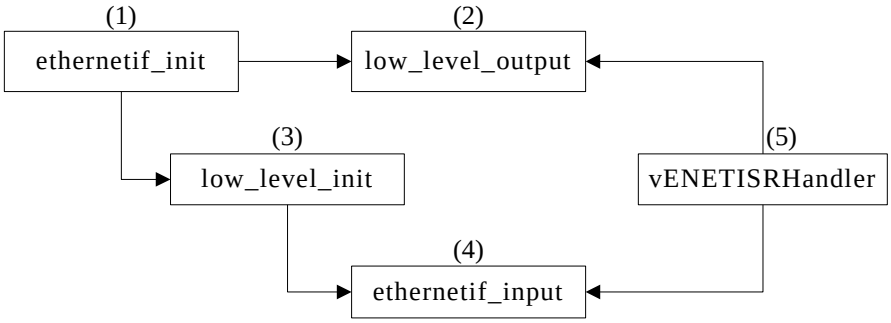


图4-2 LwIP网卡接口层框架

K60N512 芯片内部集成以太网控制器，但是需要外接物理层收发器。本设计选择 KSZ8041NL。KSZ8041NL 是单电源供电的 10Base-T/100Base-TX 物理层收发器，它可作为 MII/RMII 接口来收发数据。其独有的混合信号设计不但可以扩大发送信号的距离，与此同时还可以降低功耗。在进行网络连接时，KSZ8041NL 可以自动辨别交叉线缆和直通线缆[34]。KSZ8041NL 代表着更新级别特性和性能的收发器。对用于 10BASE-T/100BASE-TX 应用的物理层收发器

来说，KSZ8041NL 是极佳的选择。

本设计使用 RMII 方式通信，KSZ8041NL 与 K60N512 的电路连接见图 4-3。这里需要注意，KSZ8041NL 和 K60N512 使用同一个 50MHZ 的晶振。

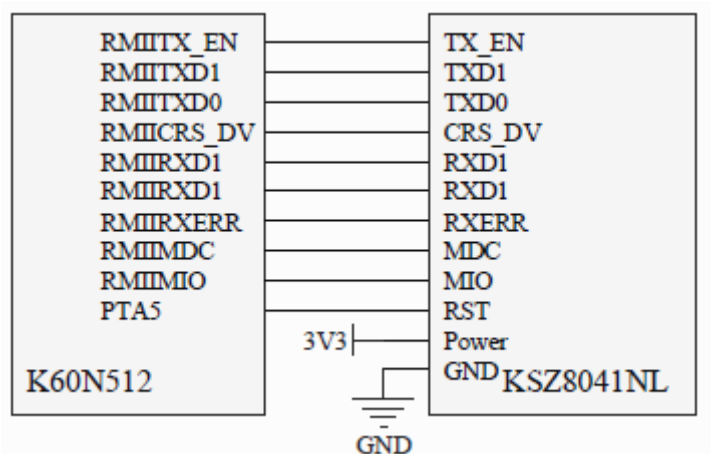


图4-3 KSZ8041NL和K60N512电路连接

网卡接口层源文件为 ehternetif.c，它实现了图 4-2 中的 5 个函数。下面依次分析这几个函数的功能和实现过程。

1）初始化网卡-ehternetif_init

这个函数的源码如下。

```
err_t ethernetif_init(struct netif*netif)
{
    static struct ethernetif ethernetif;
    netif->state=&ethernetif;
    netif->name[0]=IFNAME0;
    netif->name[1]=IFNAME1;
    netif->output=ethernetif_output;
    netif->linkoutput=low_level_output;//注册链路层发送函数
    ethernetif.ethaddr=(struct eth_addr*)&(netif->hwaddr[0]);
    low_level_init(netif);
    etharp_init();
    sys_timeout(ARP_TMR_INTERVAL,arp_timer,NULL);
    return ERR_OK;
}
```

ethernetif 是一个结构体，用于描述底层网络硬件设备。这个结构体唯一不可或缺的成员就是网卡的 MAC 地址，它是 LwIP 用于响应 ARP 查询的核心数据。这个函数向 LwIP 注册实际发送函数。这个函数并不是真正的发送函数，

它是通过调用注册的函数完成信息包发送的。ethernetif->ethaddr 指针指向 netif 中保存的网卡 MAC 地址。初始化网卡，建立稳定的物理连接链路并建立接收线程。这个函数直接与底层硬件打交道，LwIP 仅能为其提供一个参考结构，需要结合实际硬件情况重新设计实现。

2）链路层发送函数-low_level_output

该函数将协议栈发来的链路层发送请求实现。主要工作是将高层要发送的数据填充到发送链表中等待，然后检查网卡信号量是否已经阻塞。如果没有则通知以太网控制器将发送缓冲区中数据发出。否则进行等待，当等待超时后，返回错误信息。这里使用了 FreeRTOS 中的二值信号量 xRxENETSemaphore 来同步网卡发送请求。其执行流程见图 4-4。

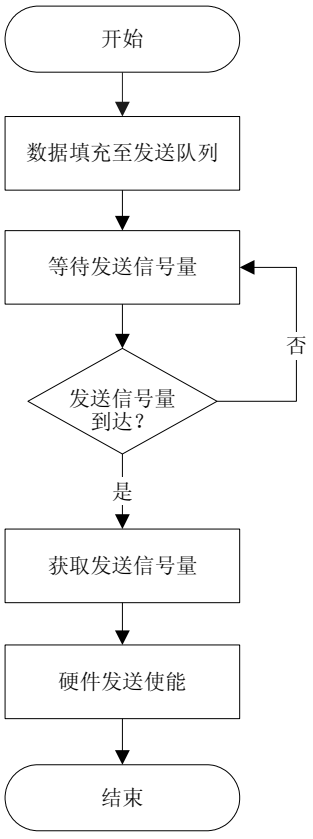


图4-4 链路层发送函数执行流程

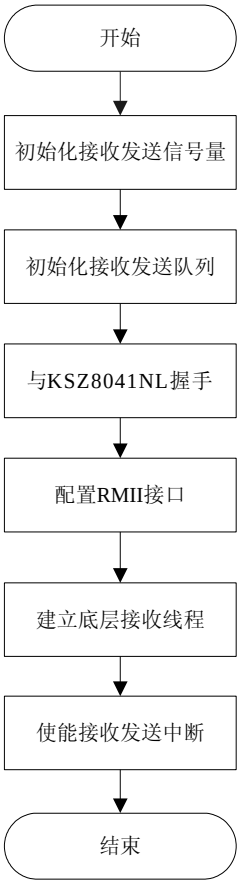


图4-5 底层接口初始化函数执行流程

3）底层接口初始化-low_level_init

这个函数是真正的初始化底层网卡和链路层通讯的函数。它的主要工作包括配置底层网卡 MAC 地址，使能 RMII 接口，初始化 KSZ8041NL 芯片，使能

K60N512 以太网的收发中断和初始化收发部分使用的信号量等工作。它还建立了底层接收数据线程，线程对应的函数体为 low_level_input。这个函数是底层初始化的核心部分。其执行流程见图 4-5。

4）线程接收数据函数-ethernetif_input

这个函数的函数体是一个标准的 FreeRTOS 的线程函数体。接收线程是在 low_level_init 中建立的。线程接收数据函数等待以太网接收中断发来的信号量 xRxENETSemaphore。当 xRxENETSemaphore 到达时，表示链路层有一帧数据到达。然后解析收到的数据帧，如果是 IP 帧则向上层协议发送数据；如果是 ARP 帧，则更新 ARP 表。线程接收函数执行流程见图 4-6。

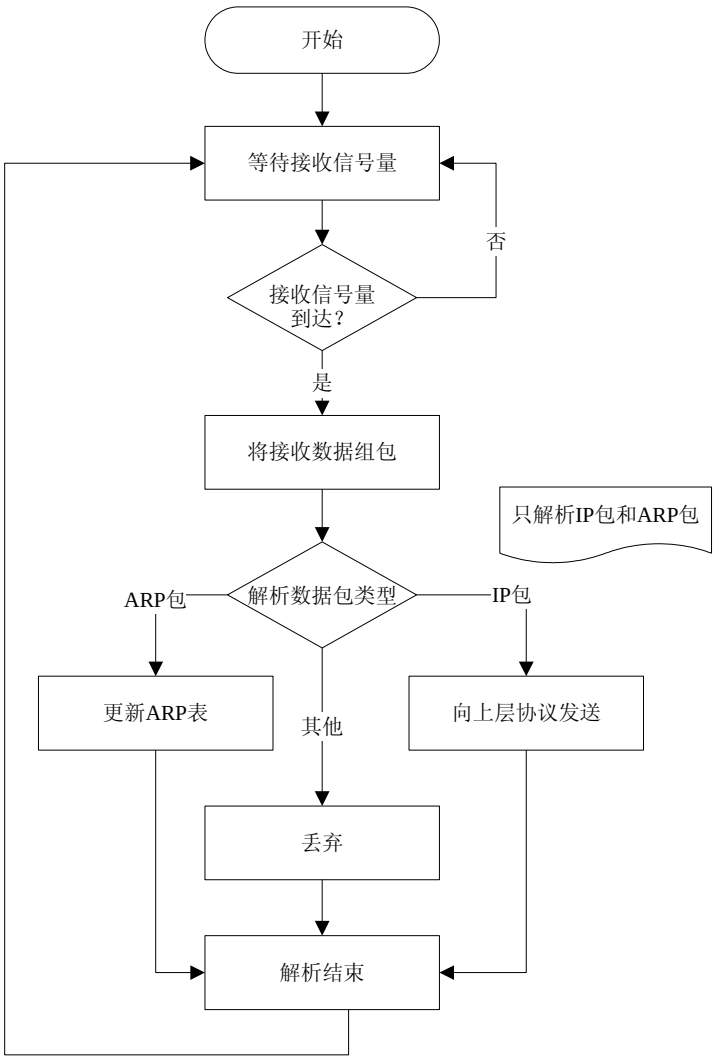


图4-6 线程接收数据函数函数执行流程

5）以太网中断服务函数-vENETISRHandler

本设计中使能了发送和接收中断来和 LwIP 进行同步。中断会在发送完成

和接收完成时触发。中断服务函数首先判断中断类型，如果是接收完成中断，则投递接收信号量 xRxENETSemaphore，通知 ethernetif_input 接收线程有一帧数据收到；如果是发送完成中断，则投递发送信号量 xTxENETSemaphore，表示当前网卡空闲，可以发送下一帧数据。

4.2.4 LwIP 测试

本文对移植后的 LwIP 协议栈进行了测试，首先进行了最简单的 ping 测试。测试结果见图 4-7。其中 K60N512 的 IP 地址设置为 192.168.0.6。Ping 测试是最简单的网络测试，主要用于测试网络是否连接。图 4-7 所示，PC 向 K60N512 发送了 4 个数据包，每个包 32 个字节。包的往返时间小于 1ms，丢包率为 0%。

```
D:\>ping 192.168.0.6

Pinging 192.168.0.6 with 32 bytes of data:

Reply from 192.168.0.6: bytes=32 time<1ms TTL=255
Reply from 192.168.0.6: bytes=32 time<1ms TTL=255
Reply from 192.168.0.6: bytes=32 time<1ms TTL=255
Reply from 192.168.0.6: bytes=32 time<1ms TTL=255

Ping statistics for 192.168.0.6:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss)
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms
```

图4-7 LwIP协议栈的ping测试

本文还进行了 socket 测试。测试使用 TCP 通讯，K60N512 使用 1026 号端口。Socket 通讯测试中建立了 tcpecho 任务，该任务监听 1026 端口，当连接建立后，将接收到的数据发回。Tcpecho 任务的执行流程见图 4-8。

这里实现的 tcpecho 任务类似于 tcp socket 服务器，服务器接收客户端的连接请求，然后接收客户端的数据并将接收到的数据发回客户端。这里所有的操作仅仅在运输层中完成，没有对接收到的 TCP 数据包进行解析。应用层的协议实现就是对 TCP 数据包的内容进行解析执行，本文第五章实现了简单的 Web 服务器。Web 服务器就是在 80 号 TCP 端口监听数据，将接收到的 TCP 数据包按照 HTTP 协议进行解析，然后发送 HTTP 应答。

通过 ping 测试和 socket 通讯测试，证明本文移植的 LwIP 协议栈和 FreeRTOS 实时操作系统工作稳定。

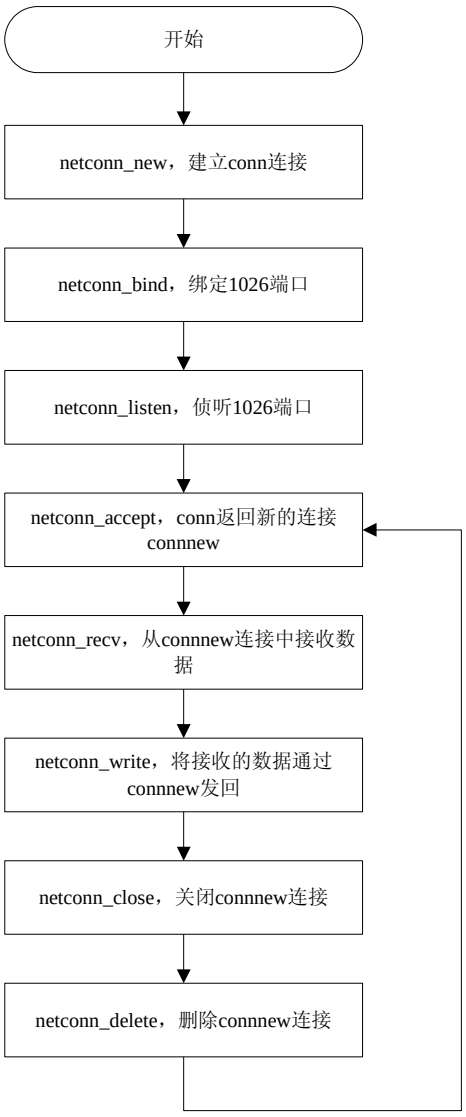


图4-8 socketecho任务执行流程

4.3 本章小节

本章的主要工作总结如下：

（1）分析了 FreeRTOS 代码结构，分析其移植流程，成功移植 FreeRTOS 到 K60N512 芯片中，并对移植进行了测试。

（2）分析了开源 TCP/IP 协议栈 LwIP 的代码结构，分析了它的操作系统模拟层和网卡接口层移植过程，成功移植 LwIP 到 K60N512 芯片中，并对移植进行了测试。

第五章 基于嵌入式 Web 短信猫的设计

本章在前面章节实现的软硬件基础上，设计实现了基于嵌入式 Web 的短信猫。短信猫是可以进行短信收发的设备。本文设计的短信猫通过 Web 页面访问，将用户填写的信息发到指定手机中。市场上现在的短信猫多为串口通信或者 USB 口通信方式，在使用过程中需要在厂家提供的二次开发包的基础上进行 UI 层封装才能使用[35]。本文设计的嵌入式 Web 短信猫内部集成嵌入式 Web 服务器，可以同时支持多个用户访问，使用简单。另外，本设计还提供了 DLL 供用户进行二次开发。二次开发的 DLL 封装了 socket 通信，对于有二次开发需求的用户可以在很短的时间内实现短信发送功能。

5.1 短信猫设计

5.1.1 系统需求

短信猫中关键部分是短信收发设备。本设计选择华为公司的 EM310 作为通讯模块。硬件部分实现 EM310 驱动电路，使用 UART 接口和 K60N512 通讯。

软件部分使用 FreeRTOS 作为操作系统调度各个任务。网络协议栈使用 LwIP 协议栈，协议栈之上移植嵌入式 Web 服务器用于人机交互。系统模型见图 5-1。

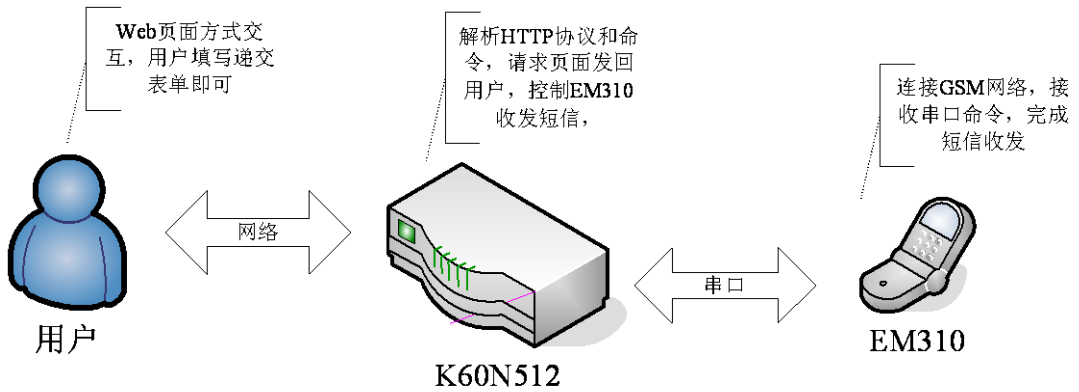


图5-1 短信猫系统模型

5.1.2 EM310

EM310 是一款兼容 GSM/GPRS 通信模块，广泛用于远程数据采集、远程测试和无线 POS 机等领域。EM310 模块内嵌 TCP/IP 协议栈，可以有效缩短产

品开发周期。

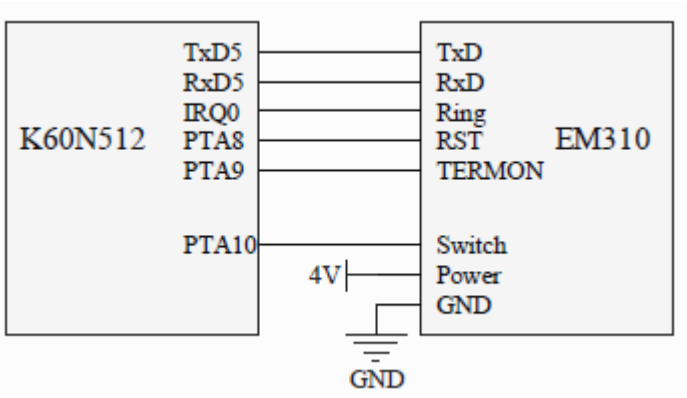


图5-2 EM310和K60N512连接图

在价格敏感的终端上使用该模块可有效降低终端制造成本。EM310 和 K60N512 的连接电路见图 5-2。其中 RST 为 EM310 复位引脚，TERMON 为使能引脚，Switch 为电源开关引脚，Ring 为振铃引脚，振铃引脚接 K60N512 的 IRQ 引脚，当有数据到达时会触发 IRQ 中断。

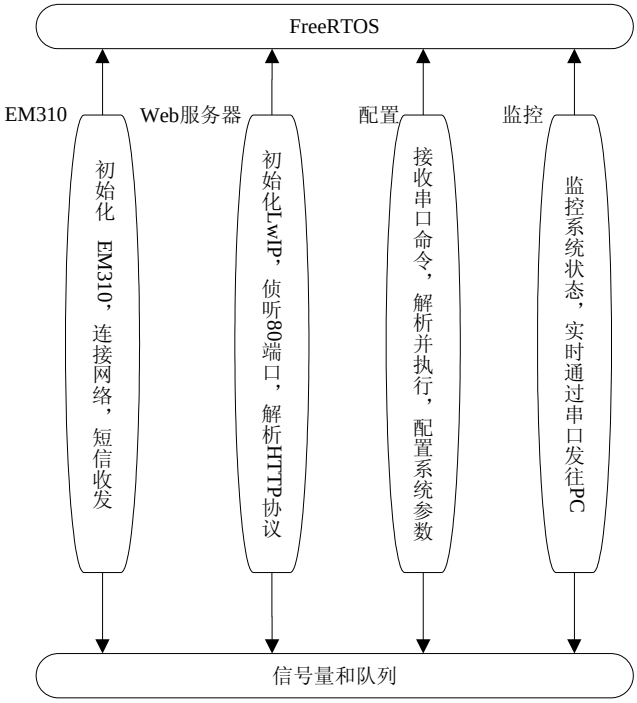


图5-3 系统任务结构

5.1.3 系统任务结构

短信猫软件部分划分了四个任务，每个任务功能单一明确。任务之间通过信号量进行同步和通信。一个任务驱动 EM310，实现短信收发；一个任务负责

网络通讯，接收 Web 请求，并解析执行 HTTP 请求；第三个任务接收串口控制命令，对系统状态进行配置；第四个任务实时输出系统状态，是一个监控任务。系统任务结构见图 5-3。

5.1.4 系统执行流程

由于系统划分为四个任务，每个任务功能单一。下面分析每个任务的执行流程。

1）EM310 任务

该任务作为 EM310 的驱动任务，实现 EM310 的初始化，短信收发，短信删除和查看等功能。该任务内部维持了收发队列，通过命令信号量和 Web 服务任务通讯。该任务执行流程见图 5-4。

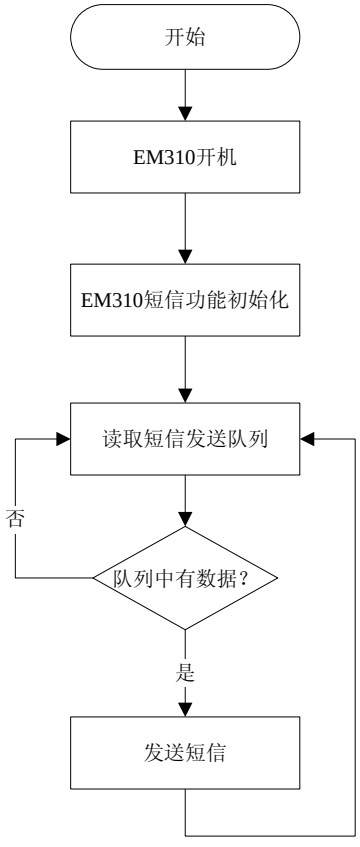


图5-4 EM310任务执行流程

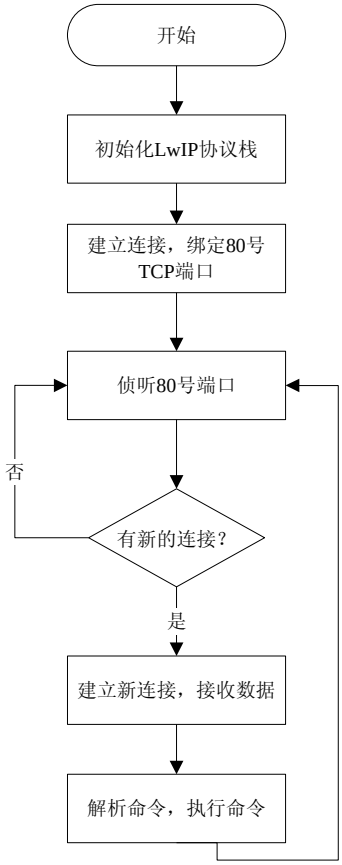


图5-5 Web服务器任务执行流程

2）Web 服务器任务

该任务实现了嵌入式 Web 服务器，根据用户请求，动态生成 HTML 返回用户请求执行结果。HTML 页面传回用户请求。用户与系统之间的交互通过 JavaScript 脚本实现。其中主要用到了 HTML 中的 Request 和 Response 原语。

当用户要发送短信时，首先填写 HTML 表单，然后提交给 Web 服务器。芯片接收 TCP 80 端口传来的请求，解析并执行请求，并将执行结果反馈给用户。图 5-5 给出了该任务的执行流程。

3）配置任务

该任务接收串口发送来的命令，解析然后执行命令。该任务提供了配置接口，可以对系统属性进行配置。系统可配置属性包括系统 IP 地址，MAC 地址，Web 服务器端口号和是否使能监控等。

4）监控任务

该任务实时扫描其他任务状态，输出系统状态和命令执行结果。该任务用于监控系统运行状态。

5.2 短信猫实现

5.2.1 EM310 驱动程序

EM310 支持 GSM 模块的标准的 AT 命令，所有的命令均通过串口通讯完成[47,48]。本文按照嵌入式构件思想指导，封装了 GPRS.h 和 GPRS.c 两个文件。这部分代码的实现主要是根据 EM310 的 AT 命令手册。GPRS.h 文件定义如下：

```
#ifndef __GPRS_H__
#define __GPRS_H__
long InitGPRSModem();//初始化 EM310
long CloseGPRSModem();//关闭 EM310
long GetMsgCenterNo(char * MsgCenter);//设置短信中心
long SetMsgCenterNo(char * MsgCenter);//获取短信中心
long GetMsgCount();//获取短信数目
long ReadMsg( char*msg, unsigned int   Index);//读取第 Index 条短信
long DelMsgByIndex( int Index);//删除第 Index 条短信
long SendMsg(char * SendNo, char* Message,int Type);//发送短信，type 为发送模式
#endif /*__GPRS_H__*/
```

5.2.2 嵌入式 Web 服务器移植

Web 服务器使用 HTTP 协议通讯，是建立在 80 号端口的 TCP 通讯[37]。精简的 HTTP 协议本身并不复杂[36,40]。Web 服务器的执行流程见图 5-5。本设计中将所有 HTML 文件和图片文件均存储在 K60N512 的内部 Flash 中。本设计没有移植文件系统，其中的 GetFileFromFlash 函数可以根据文件名查找到存储

文件的地址和文件长度，这样就虚拟了文件系统。Web 服务器的代码如下。

```
#define HTTP_OK    "HTTP/1.0 200 OK\r\nContent-type: text/html\r\n\r\n"
static void vProcessConnection( struct netconn *pxNetCon )
{
static portCHAR *pDynamicPage;
unsigned portLONG    filelength;//请求的文件长度
struct netbuf    *pxRxBuffer;
portCHAR             *pcRxString;
unsigned portSHORT usLength;
//读取接收到的数据，装载 netbuf
pxRxBuffer = netconn_recv( pxNetCon );
//从 netbuf 中将数据存到 pcRxString 中，
//此时 pcRxString 类似：GET /send.html?phone=13588888888&msg=hello...
netbuf_data( pxRxBuffer, ( void * )&pcRxString, &usLength );
//判断是否是 GET 命令
if( !strncmp( pcRxString, "GET", 3 ) )
{
      //发送 HTTP OK 头
       netconn_write( pxNetCon, HTTP_OK, strlen(HTTP_OK), NETCONN_COPY );
       //处理 requeststring，如 phone=13588888888&msg=hello
        processcmd(pcRxString);
       //在 Flash 中查找请求的 HTML 页面，取得回发的地址和长度
      GetFileFromFlash(pcRxString,&pDynamicPage,&filelength);
       //将查找到的 HTML 页面发回 PC
       netconn_write( pxNetCon, pDynamicPage,filelength, NETCONN_COPY );
}
netbuf_delete( pxRxBuffer );
netconn_close( pxNetCon );
}
```

5.2.3 系统界面

短信猫系统 Web 页面见图 5-6 所示。其中包含短信发送，短信查询，短信删除和联系方式等页面。

图5-6 短信猫系统界面

5.3 本章小结

本章的主要工作总结如下：

（1）采用华为 EM310 为无线模块，实现了嵌入式 Web 方式的短信猫。短信猫可以进行短信收发。

（2）在 LwIP 协议栈的基础上，实现了微型的 Web 服务器。服务器中存储了 Web 页面，用户通过 Web 页面和短信猫交互。

第六章 总结与展望

6.1 总结

飞思卡尔公司 Kinetis 微控制器 K60N512 资源丰富、功能强大，具有较高的市场潜力和应用价值，本文受飞思卡尔公司委托，对 Kinetis 芯片的硬件和软件驱动实现进行研究。本文选取 Kinetis 系列的代表芯片 K60N512，深入研究了该芯片的详细功能和开发方法，并实现了其上常用模块的软、硬件构件的设计。在各模块驱动程序实现的基础上移植了实时操作系统 FreeRTOS 和嵌入式 TCP/IP 协议栈 LwIP。并在 FreeRTOS 和 LwIP 支持下实现了嵌入式 Web 方式的短信猫。短信猫以 Web 页面方式和用户交互，实现短信收发。本文的主要工作总结如下：

（1）采用飞思卡尔 Kinetis 系列 ARM Cortex-M4 内核微控制器 K60N512，设计并制作了 K60N512 的核心板，核心板采用四层板实现，核心板上主要包含 K60N512 的最小系统电路。本文还实现了 Kinetis 系列扩展板，扩展板上集成 Kinetis 系列常见模块，如 USB，网络和 CAN 等。此外，硬件设计中还实现了 Kinetis 芯片的调试器 OSJTAG，目前 OSJTAG 调试器已经批量生产。本课题实现的所有硬件已经通过测试，并计划于 2011 年 8 月份在苏州大学举行的嵌入式研讨会上推广。

（2）驱动程序开发是嵌入式软件开发的一大内容。在嵌入式软件构件思想指导下，本课题探讨了驱动程序开发的原则，实现了 K60N512 常见模块的驱动，如 GPIO、UART、AD、CRC、Flash 和网络等。限于篇幅有限，而且驱动程序开发原理相通，本文并没有一一阐述。

（3）本文成功移植了实时操作系统 FreeRTOS 和嵌入式 TCP/IP 协议栈 LwIP，测试证明二者在 K60N512 中运行稳定。并在 FreeRTOS 和 LwIP 支持下实现了嵌入式 Web 方式的短信猫。短信猫内部集成嵌入式 Web 服务器，通过 Web 页面方式和用户交互，实现短信收发功能。短信猫的设计一方面研究探讨了 K60N512 的网络应用，另一方面也验证了本文移植的 FreeRTOS 及 LwIP 和课题实现的硬件平台。

6.2 展望

虽然已经达到课题预期的研究目的，但是由于毕业设计时间有限，工作量繁重，本课题中还有一些不完善的地方需要做进一步的加强和改进：

（1）软件部分没有提供文件系统支持，移植开源的 FatFS 可以实现 FAT 文件系统。

（2）短信猫中实现的嵌入式 Web 服务器相对简单，没有提供 CGI 支持，目前只解析 GET 命令，下一步实现完整的 CGI 支持。

（3）在 FreeRTOS 和 LwIP 支持下，可以移植和实现更多网络应用层协议，如 FTP，SMTP 等[46]。

参考文献

[1] ARM. Cortex-M Series [EB/OL]. http://www.arm.com/products/processors/cortex-m /index.php, 2009

[2] ARM. Cortex-M4 Processor [EB/OL]. http://www.arm.com/products/processors/cortex-m/cortex-m4-processor.php, 2009

[3] Freescale Semiconductor Inc. Kinetis Microcontrollers[EB/OL]. http://www.freescale.com /webapp/sps/site/homepage.jsp?code=KINETIS, 2010

[4] Freescale Semiconductor Inc. K60 Sub-Family RM.pdf[DB/OL]. http://www .freescale.com, 2010

[5] Freescale Semiconductor Inc. Kinetis Peripheral Module Quick Reference.pdf [DB/OL]. http://www.freescale.com, 2010

[6] Freescale Semiconductor Inc. TWR-K60N512 UM.pdf[DB/OL]. http://www.freescale.com, 2010

[7] Freescale Semiconductor Inc. K60 Data Sheet.pdf[DB/OL]. http://www.freescale.com, 2010

[8] 飞思卡尔推出混合信号 ARM Cortex-M4 微控制器系列 Kinetis[EB/OL]. http://www.freescale.com.cn/media/2010/0622.asp，2010

[9] The FreeRTOS Project[EB/OL]. http://www.freertos.org, 2010

[10] 华清远见嵌入式培训中心. Cortex 系列 ARM 内核介绍[EB/OL]. http://www.hqyj.com，2009

[11] ARM. Cortex-M4 Technical Reference Manual.pdf[DB/OL]. http://www.arm.com，2009

[12] 屯娜. 飞思卡尔 MCF52259 微控制器的应用研究[D]. 苏州大学硕士学位论文，2010.6

[13] 王宜怀，刘晓升. 嵌入式技术基础与实践[M]. 清华大学出版社，2007

[14] 李盛，张扬. 嵌入式通信设备驱动程序设计标准化[J]. 电子科技大学学报，2005，34(3):355-358

[15] 谢希仁. 计算机网络[M]. 电子工业出版社，2006

[16] Adam Dunkels. Full TCP/IP for 8-bit architectures[C]. Proceedings of the 1st

international conference on Mobile systems, 2003:85-98

[17] 王宜怀，陈建明，蒋银珍. 基于 32 位 ColdFire 构建嵌入式系统[M]. 电子工业出版社，2009

[18] lwIP--A Lightweight TCP/IP stack [EB/OL]. http://savannah.nongnu.org/projects/lwip, 2004

[19] DUNKELSA. Design and implementation of the lwIP TCP/IP stack[EB/OL]. http://www.sics.se/, 2004

[20] Freescale Semiconductor Inc. Tower System[EB/OL]. http://www.freescale.com, 2008

[21] National Semiconductor. LM2576 DataSheet.pdf[DB/OL]. http://www.national.com, 2004

[22] Freescale Semiconductor Inc. OSBDM User's Design&Troubleshooting Guide.pdf[EB/OL]. http://www.freescale.com, 2010

[23] 万永波，张根宝，田泽，杨峰. 基于 ARM 的嵌入式系统 Bootloader 启动流程分析[J]. 微计算机信息，2005，21(11):90-92

[24] IAR Systems. IAR C/C++ Development Guide.pdf[DB/OL]. http://www.iar.com, 2010

[25] 维基百科. RS-232[EB/OL]. http://zh.wikipedia.org/wiki/RS-232，2010

[26] 维基百科. 循环冗余校验[EB/OL]. http://zh.wikipedia.org/wiki/循环冗余校验，2010

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

[28] 何巍，何建忠. 开源 RTOS 内存管理机制分析和改进[J]. 计算机工程，2010，(10):67-69

[29] 董向阳. 基于 ARM 的 LwIP 协议栈的研究与移植[D]. 哈尔滨理工大学计算机科学与技术学院，2009

[30] 陈伟. 基于 ARM 的轻量级 TCP/IP 协议栈的移植及应用[D]. 山东轻工业学院电子信息与控制工程学院，2009

[31] 张齐，劳炽元. 轻量级协议栈 LwIP 的分析与改进[J]. 计算机工程与设计，2010，(10):2169-2171，2256

[32] Wei Chen, Shu-Bo Qiu, Ying--Chun Zhang. The Porting and Implementation of Light-Weight TCP/IP for Embedded Web Server[C]. 2008. WiCOM '08. 4th International Conference on Wireless Communications, Networking and Mobile Computing, 2008:1-4

[33] Bo Qu, Daowei Fan. Design of remote data monitoring and recording system based on ARM[C]. 2010 2nd International Conference on Industrial and Information Systems (IIS), 2010:252-254

[34] Micrel. KSZ8041NL datasheet.pdf[DB/OL]. http://www.micrel.com, 2006

[35] 百度百科. 短信猫. http://baike.baidu.com/view/425158.htm，2010

[36] 雷必成. 嵌入式系统中 TCP/IP 协议的精简与实现[J]. 微计算机信息，2006，22(6):107-109

[37] James F, Kurose, Keith W Ross 著，陈鸣等译. 计算机网络——自顶向下方法与 Internet 特色[M]. 北京：机械工业出版社，2005.6

[38] Raj Kamal 编著，陈曙晖等译. 嵌入式系统——体系结构、编程与设计[M]. 北京：清华大学出版社，2005.5

[39] 郑巨明，张和生，贾利民，梁玉庆. 基于 μC/OS-Ⅱ和 LwIP 的嵌入式以太网接口设计[J]. 计算机测量与控制，2009，(11):2238-2242

[40] 祝叶. 嵌入式以太网远程测控系统通用平台的开发及应用[D]. 苏州大学硕士学位论文，2009.6

[41] Joseph Yiu 著，宋岩译. ARM Cortex-M3 权威指南[M]. 北京航空航天大学出版社，2009.7

[42] 刘滨，王琦，刘丽丽. 嵌入式操作系统 FreeRTOS 的原理与实现[J]. 单片机与嵌入式系统应用，2005，(7):7210

[43] 胡志强. 嵌入式 TCP/IP 协议研究与 ARM 实现[D]. 陕西：西北农林科技大学，2007

[44] 焦海波. μC/OS-II 平台下的 LwIP 移植笔记[EB/OL]. http://bbs.elecfans.com，2006

[45] 高长艳. 嵌入式 TCP/IP 协议的研究与实现[D]. 吉林：中国科学院研究生院(长春光学精密机械与物理研究所)，2006

[46] 汪小燕，连晓平，董燕等. 基于 TFTP 协议的嵌入式系统开发方法设计与实

现[J]. 华中科大学学报(自然科学版)，2006，34(12)：56-58

[47] 深圳华为技术有限公司．华为 EM310 无线模块产品描述.pdf [DB/OL]. http://www.huawei.com.cn，2009

[48] 深圳华为技术有限公司．华为 EM310 无线模块 AT 命令手册.pdf [DB/OL]. http://www.huawei.com.cn，2009

[49] 高珀珀，邵时．低端嵌入式设备 Web 服务器的研究与实现[J]. 计算机工程，2005，31(10)：219-221

[50] 周桂兵．基于精简 TCP/IP 的嵌入式 WebServer 平台研究[D]. 武汉理工大学，2006

[51] 陈伟．基于单片机嵌入式 Web 服务器技术的研究及实现[D]. 西安科技大学，2006

[52] 张曦煌，柴志雷．嵌入式 Web 服务器中 CGI 的特点及实现[J]. 小型微型计算机系统，2003，24(11):2046~2048

公开发表的学术论文及参与的主要科研项目

[1] 王超艺，常赛，王宜怀. 飞思卡尔 S08 系列机器码文件下载软件的开发. 计算机应用与软件，编号：A10090149（已录用）

[2] 常赛，王超艺，王宜怀，倪敬飞. 基于 MC13233 的 Zigbee 物理层分析与设计. 计算机应用与软件，编号：A10120070（已录用）

[3] 王超艺，王宜怀. 基于红外传感器的自寻迹小车控制系统的设计. 电子工程师，2008 年 11 期

[4] 王超艺（项目负责人）. S08/S12/Coldfire BDM 调试器. 苏州大学大学生课外学术科研基金项目（重点项目）

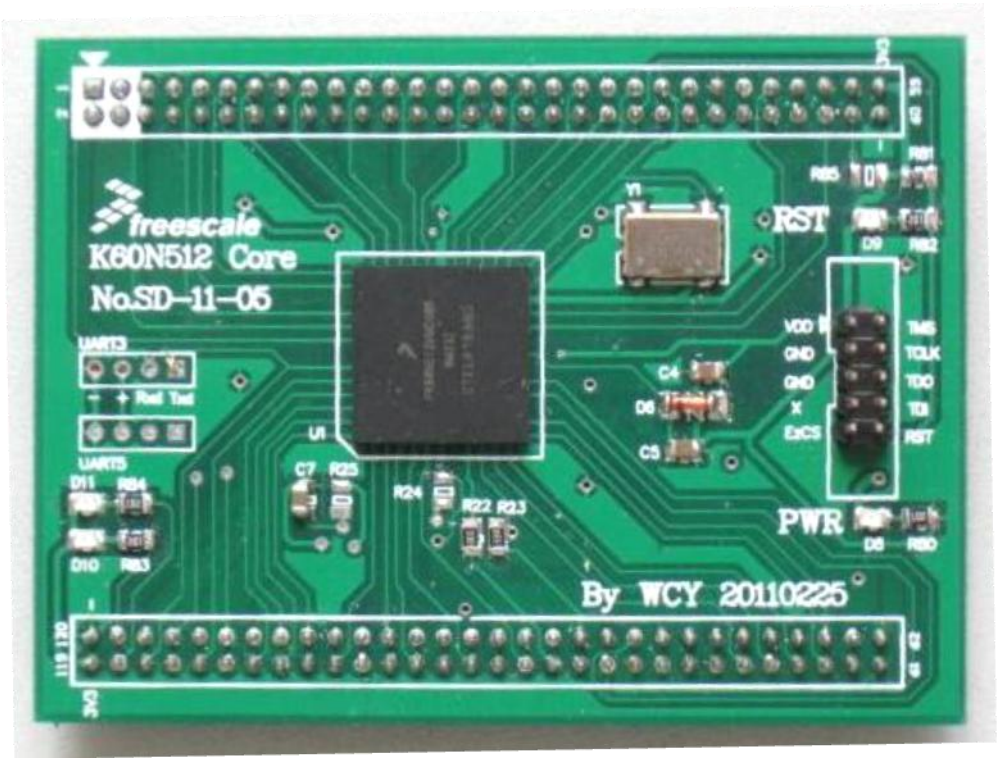
[5] 参与设计开发 ColdFire 系列 MCU 调试器软件 V1.0，软著登字第 0167893 号

[6] 参与第四届飞思卡尔全国大学生智能车大赛，并获得华东区二等奖

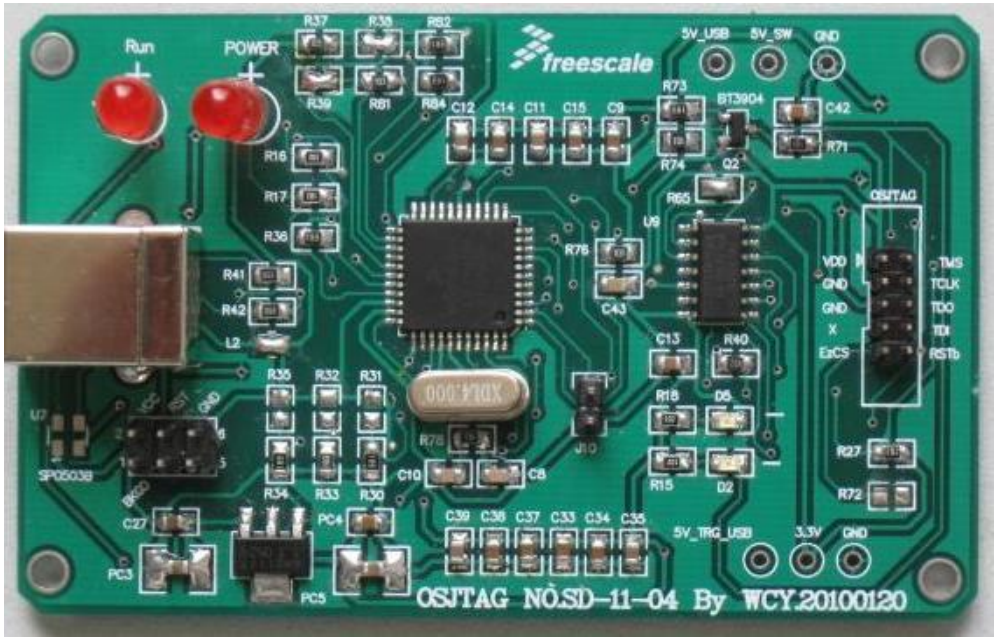
[7] 参与第三届飞思卡尔全国大学生智能车大赛，并获得华东区三等奖

[8] 参与首届长三角嵌入式系统创新设计应用竞赛，并获得三等奖

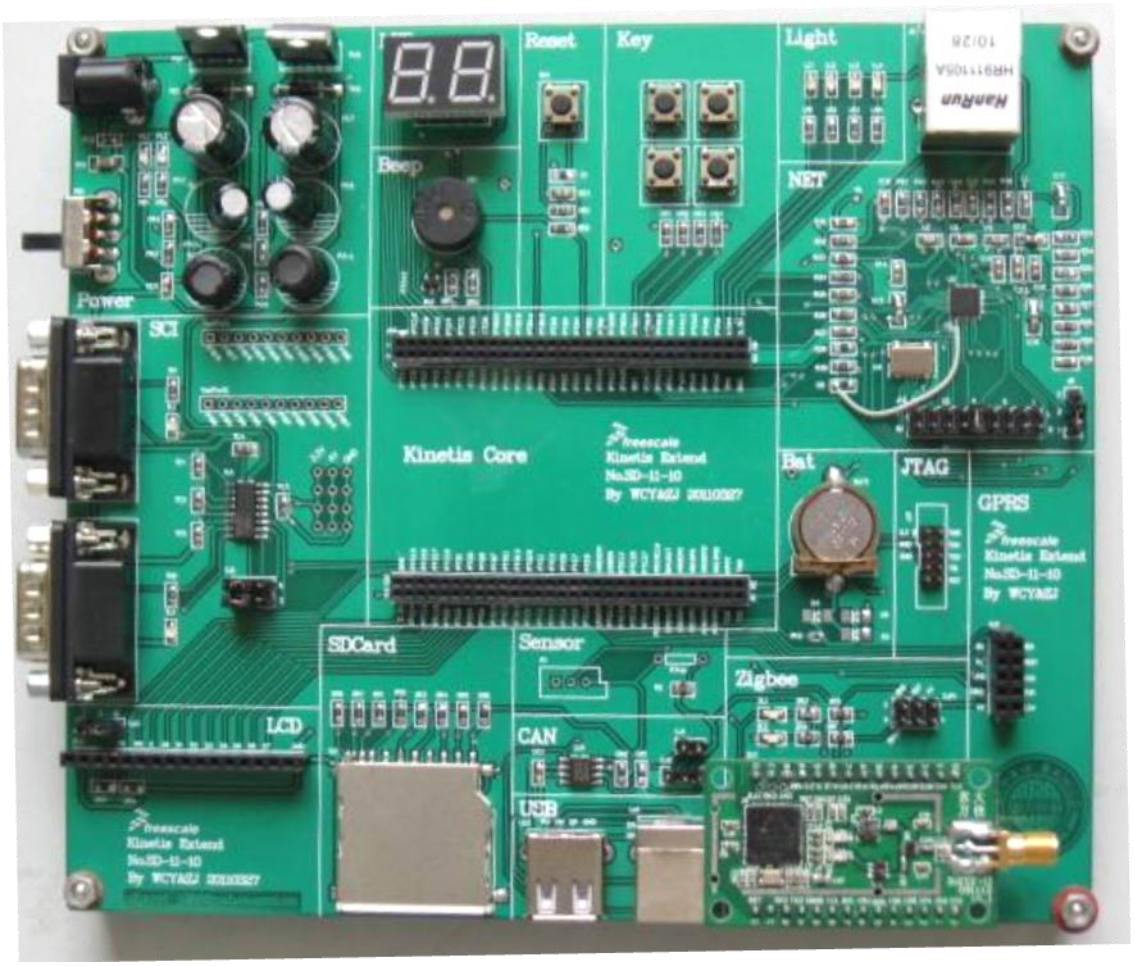
附录 A K60N512 核心板实物图



附录 B Kinetis 调试器 OSJTAG 实物图



附录 C Kinetis 系列扩展板实物图



致谢

三年时间飞逝而过，在我的研究生生涯即将结束之际，首先衷心感谢我的导师王宜怀教授，正是因为他的悉心指导和谆谆教诲，我的论文才得以完成。王老师丰富的科研经验、严谨的治学态度以及务实的工作作风，激励我不断努力学习和工作。特别是在论文的选题、设计和撰写阶段，王老师提出了很多珍贵的意见，使论文顺利完成。在此我向王老师表达我最诚挚的谢意。

三年中大部分时间都是在实验室度过的，这是一个温馨融洽的集体，感谢实验室的兄弟姐妹们。感谢李翠霞、朱乐乐、周杰、姚丹丹和陈爱兵在论文修改期间提供的支持和帮助。

感谢飞思卡尔公司的马莉女士在课题研究期间给予的支持和鼓励。感谢苏州高源科技有限公司和苏州鑫来科技有限公司在我读研期间给予的锻炼机会和人力物力的支持。

感谢我的家人，感谢我的女朋友，在我的学习和研究期间给予的精神和物质支持。

最后，感谢评审论文的各位专家学者。