中文摘要

摘要：本课题提出的“小型化标准12导联心电图仪”，是一款手持式、低功耗、低成本，但功能完善的心电图设备。该心电图仪按功能分为主控、采集、处理和显示三大模块，其核心采用三星电子的S3C2410A ARM芯片，以Windows CE 5.0作为操作系统平台。

本文着重论述了Windows CE 5.0嵌入式操作系统的构建过程以及心电图噪声滤波算法的实现与比较。其中在系统构建方面主要描述了Windows CE 5.0操作系统平台下的Boot Loader、OAL的开发和移植，显示驱动、触摸屏驱动的开发与移植。

在滤波算法方面，由于考虑到系统的资源的有限，而且心电图数据的显示与绘制会占用较多的CPU与内存资源，为了能处理来自12导联的心电图数据，心电图的滤波算法应该主要以效率高、实时性为目标。本课题实现的滤波算法中是基于自适应模板的滤波器，利用Matlab对实际的心电图数据进行了算法的测试，通过分析与比较该滤波器对心电图滤波后的结果，发现它的滤波效果基本能满足实际的应用需要，最重要的是它的计算量非常小，执行效率很高，完全能够满足实时滤波的需求。

关键词：心电图；自适应滤波器；ARM；WinCE

分类号：TH772.4

ABSTRACT

ABSTRACT: The "miniaturized standard 12 leads ECG device" advanced in this subject is a handheld ECG device which is low-power,low-cost and has whole basic functions. This ECG device includes controlling module, gathering module and display module by functions. It's kernel uses the Samsung 2410A ARM microprocessor and takes Windows CE as it's operating system .

This paper emphasize the construction procedure of Windows CE 5.0 embeded operating system and the design and realization of ECG noises filter. The construction procedure includes developing and implanting Windows CE 5.0 Boot Loader and OAL,developing and implanting display driver and touch screen driver.

From the system view, There is limited system resource, and the displaying and drawing of ECG data need a lot of CPU and memory resources, therefore the main object of the filter is how to process the data from 12 leads in real-time. This subject realizes a filter based on Integer Coefficients digital filter and tests it by ECG data in Matlab. This filter is found be able to satisfy the requirement of practical application after analysizing and comparing the filtered result, And the most important thing is this filter need few calculations and is very efficient. It is suitable for real-time filtering.

KEYWORDS: ECG; LCD; Adaptive Digital Filter; ARM; Windows CE
CLASSNO: TH772.4

学位论文版权使用授权书

本学位论文作者完全了解北京交通大学有关保留、使用学位论文的规定。特授权北京交通大学可以将学位论文的全部或部分内容编入有关数据库进行检索，并采用影印、缩印或扫描等复制手段保存、汇编以供查阅和借阅。同意学校向国家有关部门或机构送交论文的复印件和磁盘。

（保密的学位论文在解密后适用本授权说明）

学位论文作者签名：张小楠　　导师签名：

签字日期：2007年12月4日　　签字日期：07年12月21日

独创性声明

本人声明所呈交的学位论文是本人在导师指导下进行的研究工作和取得的研究成果，除了文中特别加以标注和致谢之处外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得北京交通大学或其他教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示了谢意。

学位论文作者签名：张士辉 签字日期：2007 年 12 月 7 日

致谢

本论文的工作是在我的导师周洪利副教授的悉心指导下完成的。本论文从选题、撰写、字斟句酌的修改，一直到最后定稿，倾注了周洪利老师大量的心血。他渊博的知识、严谨的治学态度、科学的工作方法和精益求精的工作精神给了我极大的帮助和影响，使我受益匪浅。

在两年半的研究生学习期间，周老师在生活、科研两方面，都给予我无微不至的关怀。在我学习和课题研究遇到困难时给予我耐心指导，为我提供了难得的学习和锻炼机会，使我在理论水平和实践能力方面都得到了很大的提高，在此衷心感谢周老师对我的关心和指导！

陈连坤副教授在我攻读硕士研究生期间给予我无私的帮助和悉心的指导，在此也表示衷心的感谢！

涂小强同学负责本项目应用程序的设计与开发，在此过程中，我们互相学习，一起探讨项目中的难点，共同进步。实验室的石义维、张禹同学对我的学习和生活也作出了许多帮助，在此向他们表达我的感激之情。

最后还要感谢我的家人，感谢他们一如既往理解和支持，使我能够顺利完成学业。

1 引言

1.1 课题背景

心血管疾病是目前发病率和死亡率最高的疾病之一，是当今社会的第一大健康杀手。世界卫生组织指出，目前全球每年有1700万人死于心脏病和其他心血管疾病，约占全球死亡人数的三分之一，预计到2020年这个数字将有可能突破2000万，死亡原因主要是致病性心律失常和急性心梗[1]。据北京市急救中心统计，70%以上发病于家中或工作现场，大部分人因失去抢救时间而死在院外；很多病人是由于未及时发现病变延误了治疗而最终导致死亡。因此，对心血管疾病的诊断、预防是当今医学界面临的首要问题。尽早地发现心血管系统疾病征兆，及时地了解心脏病状况，对疾病的预防和及时诊治具有重要的意义。

心电信号是最早应用于医学的人体生物电之一，如今医学界人士已经可以通过对心电信号的分析研究对心血管相关病变做出预测和诊断，心电图已经是心脏疾病诊断的必不可少的工具之一。常规心电图(Electrocardiogram， ECG)是病人在医院静卧情况下由心电图仪记录下的病人心电活动，它可以从不同角度观察心脏的活动情况[2]，是临床中对心律失常、心肌梗塞、心肌炎等心血管疾病的重要诊断手段。

目前，世界各大医疗仪器的生产厂家竞相投入大量的人力、物力进行心电监护系统的开发、生产和销售，促进了监护系统的发展和普及。但是，国内监护系统的科研、生产与国际先进水平相比还存在一定的差距，高、精、尖的监护设备大部分依赖于进口，价格比较昂贵，中小医院无力购买，严重影响了监护系统在我国的应用和推广，从而制约了危重病人的救护。

另一方面，心血管疾病发作具有随机性和突发性， 医院中的心电监护设备往往体积笨重、价格昂贵、不便于携带，很难在急救状态下及时进行心电监护工作，许多患者由于不能及时得到病症诊断而延误了抢救时机，从而丧失了宝贵的生命。外科手术、查房现场、医生院内外会诊、农村基层医院出诊、野外急救等移动医疗行为，都十分需要一种小型、轻便、功能完善的心电图仪。

本课题的目标是研究并且开发一种面向中小医院和诊所，集心电信号的采集、分析、存储和显示于一体的小型化标准12导联心电图仪以实现对病人心电信号的采集和分析，为临床诊治提供有价值的诊断资料，同时对心脏疾病的早期发现和心脏功能的评估也具有十分重要的意义。

1.2 心电图仪发展现状

临床应用的数字化 12 导联同步心电图机外观和体积多种多样，但是主要分为两大类：

(1) 便携式

硬件部分与相应软件紧凑地组合成一单元模块，构成体积小、成本低，智能化程度高，使用灵活、操作方便的便携机型。适合携带外出、移动抢救等用途。记录部分为高分辨热阵式打印，液晶监控显示屏幕，可以存储若干份心电图数据，并能通过标准接口将数据传输到计算机系统中。如图 1.1 所示的是通用医疗器械公司的一款便携式心电图仪。市面上的便携式心电图机通常只打印部分导联信号或仅仅支持单道打印，显示部分的导联信号或通过切换显示其他导联信号，多数仅仅支持单色显示，无法存储长时间的心电信号或缺少存储可扩展能力，系统功耗大无法支持长时间的工作。

图 1.1 通用医疗器械 MAC1200 便携式心电图仪

(2) 计算机模式

由典型的个人计算机系统、高分辨率激光打印机、心电信息处理单元以及专门的处理软件组成，往往还具有心电系统其它检测等功能，更适合心电信息的研究和探讨，多用于固定场所和长时间监测，高分辨彩色显示器可以监控心电信号采集的质量和监护异常心电图，并可从所获取的信息中人工选择需要的心电信号进行存储、打印等处理。具有自动测量功能，由于计算机备有海量存储器，可以存储大量的原始心电数据，尤其适合建立心电信息数据库。但是计算机模式的 12 导联心电图机价格昂贵，体积大，不适合于外出携带和移动抢救。图 1.2 所示是

通用医疗器械公司的一款基于计算机模式的标准 12 导联心电图仪。

图 1.2 通用医疗器械 MAC5500 心电图仪

综上所述，针对便携式和计算机模式的特点，本课题希望实现一款“小型化标准 12 导联心电图机”。它的主要目标不仅仅是要实现便携式和计算机模式所具有的常用功能，而且具有以下的特点：

(1) 12导联同步采集、显示；

(2) 支持通过无线网络同步打印12导联信号；

(3) 使用嵌入式操作系统提供存储管理、扩展外设等功能；

(4) 高分辨率、多色彩的显示屏幕；

1.3 本文主要内容

本文专注于小型化标准 12 导联心电图仪的操作系统构建与心电图噪声滤波算法的实现。因此不妨认为其前期采集/远程传送模块都已经完成并工作正常，能够同处理模块很好的通讯和传送数据。系统构建方面的研究主要包括了以下几点：

(1) Windows CE的系统结构

(2) Windows CE Boot Loader 设计

(3) 设备驱动开发

在 Windows CE 操作系统的构建过程中，主要开发和移植了 Nboot，Eboot；液晶显示驱动程序，触摸屏驱动，串口驱动程序。在滤波算法方面的研究主要包含了以下几点：

(1) 心电噪声的来源及分类

(2) 常见的心电图噪声滤波算法的比较

(3) 自适应模板滤波算法的实现

由于考虑到系统本身的资源消耗，另外还要有实时的图像显示，而且要实时的处理来自 A/D 的大数据量的采集，所以必须采用高效率的滤波算法。在本课题中采用了改进的的自适应模板滤波器，该滤波器性能优良，而且有很好的幅频特性。经过试验分析该滤波器具有运算量少，存储空间小的特点。完全能满足实时滤波的需要。

本文集中论述了 Windows CE 系统原理，系统构建的过程，以及滤波算法的比较与实现过程，其各章内容如下：

第一章，综述，介绍了心电图仪的意义，讨论了课题的发展现状，提出了课题的研究方向及需要实现目标；

第二章，系统设计方案，介绍了本课题中所采用的硬件系统以及软件系统的组成以及硬件选型的基本原则；

第三章，Windows CE 系统构建，介绍 CE 系统的系统特性以及从裸机到运行 Windows CE 操作系统过程中的设计与实现；

第四章，心电图滤波算法设计，根据心电图中存在的两种主要噪声，提出两种相应的滤波算法，着重介绍了几种可行的滤波算法的性能与结果比较，提出了一种改进的滤波算法。

第五章，结论，总结课题收获，并对下一步目标做出展望。

2 系统设计方案

2.1 嵌入式硬件电路设计的基本原则

嵌入式系统的硬件电路设计包含两部分内容。一是系统扩展，即微处理器内部的功能单元，如ROM、RAM、I/O接口、定时器/计数器、中断系统等不能满足应用系统的要求时，必须在片外进行扩展，此时要求选择适当的芯片并设计相应的电路。二是系统的I/O设备的配置，即按照系统功能要求配置外围电路，如键盘、显示器、打印机、ADC、DAC等，有时需要设计合适的接口电路。设计嵌入式系统的硬件电路应遵循以下原则[3]。

(1) 根据系统功能尽可能选择合适的处理器，尽量朝片上系统(SoC: System On Chip)方向设计硬件系统。系统器件越多，器件之间相互干扰也越强，功耗也越大，同时不可避免的降低了系统的稳定性。另外，所用的器件越多，系统的可靠性越低，系统总成本也会越高。

(2) 尽可能选择典型电路，并符合处理器常规用法。为硬件系统的标准化、模块化打下良好的基础。

(3) 系统扩展与I/O的配置应充分满足应用系统的功能要求，并留有适当余地，以便进行二次开发。

(4) 硬件结构应结合应用软件方案一并考虑。硬件结构与软件方案会产生相互影响考虑的原则是：软件能实现的功能尽可能由软件实现，以简化硬件结构，但必须注意，由软件实现的硬件功能，一般响应时间比硬件时间长，且占用处理器时间。

(5) 处理器外围电路较多时，必须考虑其驱动能力。系统驱动能力不足时会引起系统工作不稳定。

(6) 可靠性及抗干扰设计是硬件设计必不可少的一部分，它包括：芯片、器件选择、去耦滤波、印刷电路板布线、通道隔离等。

2.2 本系统的硬件设计原则

通过对心电监护系统的主要功能和使用场合的分析，确定在进行本系统的硬件电路设计时应注意以下原则。

(1) 尽可能的减小系统体积，方便携带是本心电图仪的最基本特点。在硬件设计时，可以用软件实现的功能尽可能用软件实现，可以不外扩芯片的尽可能使用片上外设，以尽可能的减少使用芯片的数量；必须外扩芯片的也要

尽可能使用体积比较小的芯片，从而尽可能的减小整个系统的体积。

(2) 尽可能的降低系统功耗，延长系统的连续工作时间。在进行硬件设计时，应该尽可能选择低功耗的芯片，并根据不同的功能允许一些器件进入休眠（低功耗）状态，采取积极措施降低系统功耗。

(3) 尽可能的降低系统成本。

2.3 硬件系统

本系统主要分为主控、采集、处理和显示三大模块。核心微处理器采用S3C2410A ARM 芯片，该芯片片上资源丰富包括了 LCD 控制器、UART 控制器、NAND Flash 控制器等等。数据的采集由外部模块提供，主要使用 C8051 系列单片机进行数据采集。而液晶显示模块使用的 Sharp LQ035Q7DB02 该液晶附带有触摸屏功能。而数据的存储主要依托于 K9F1208U0M Nand Flash，该 Flash 具有速度快，价格低廉的特点，适用于存储大量的心电图数据，系统总体结构如图 2.1 所示。

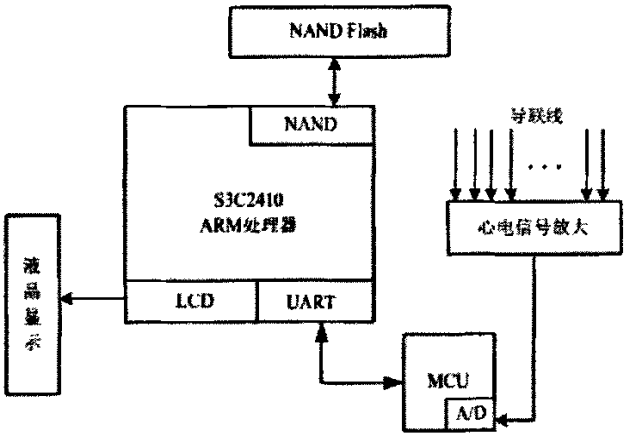

图 2.1 系统总体结构图

2.3.1 系统实现硬件方案

本课题采用 OURS 2410A[4] 开发板作为开发操作系统的实验测试平台，它具有和目标系统类似的功能和接口。该开发板是一款基于三星 ARM9 系列嵌入式处理器 S3C2410A 的一款开发平台，系统运行在 200M 的主频下发挥出色的性能。可以完成 MP3、MPG、VOIP 等工作。该开发板主要分为核心板和底板两个部分。

(1) 核心板的资源如下：

CPU:　S3C2410A ARM920T 内核，最高主频 200M;

SDRAM:　16M×32bit 共 64M PC100;

Flash:　64M NAND flash ;

Ethernet: 10/100M 网络控制器;

LED:　4 个 SMD LED;

RESET:　一个 RESET 按键;

I/O 资源:　一个 RJ45 接口，一个 USB host，一个 USB device，电源接口 5VDC，200PIN 1.27 双排引出插座，一个电源开关

(2) 底板的资源如下:

LCD:　SHARP 3.5 寸 TFT 含触摸屏;

Key:　5 个顷触按键;

CompactFlash port: CF 卡扩展口;

音频接口:　一个耳机接口和一个 MIC 接口;

一个多功能扩展口:　内含总线，中断源，SPI 等信号;

2.3.2 处理器模块

S3C2410A[5]是三星公司的 32 位 RISC 嵌入式处理器，它专为手持设备和一般应用而设计，能满足嵌入式系统中的低成本、低功耗、高性能、小体积的要求。为了尽可能减少系统的整体成本，S3C2410A 芯片集成了多个功能外设，S3C2410A 其结构如图 2.2 所示。它主要包括了以下组件:

(1) 分离的16KB指令缓存和16KB数据缓存

(2) MMU处理虚拟内存管理

(3) LCD控制器(支持STN&TFT)

(4) NAND FLASH引导

(5) 系统管理(逻辑片选和SDRAM控制器)

(6) 3通道UART

(7) 4通道DMA

(8) 4个具有PWM的定时器和1个内部定时器

(9) 看门狗定时器

(10) 117个GPIO端口和24个外部中断源

(11) 具有日历功能的RTC(实时时钟)

(12) 8通道10位ADC

(13) 触摸屏接口

(14) 1个IIC总线接口

(15) 1个IIS总线接口

(16) 2个USB Host和1个USB设备

(17) 多媒体卡接口

(18) SD卡 接口

(19) PLL片上时钟发生器

(20) 电源控制:普通、缓慢、空闲和关闭模式

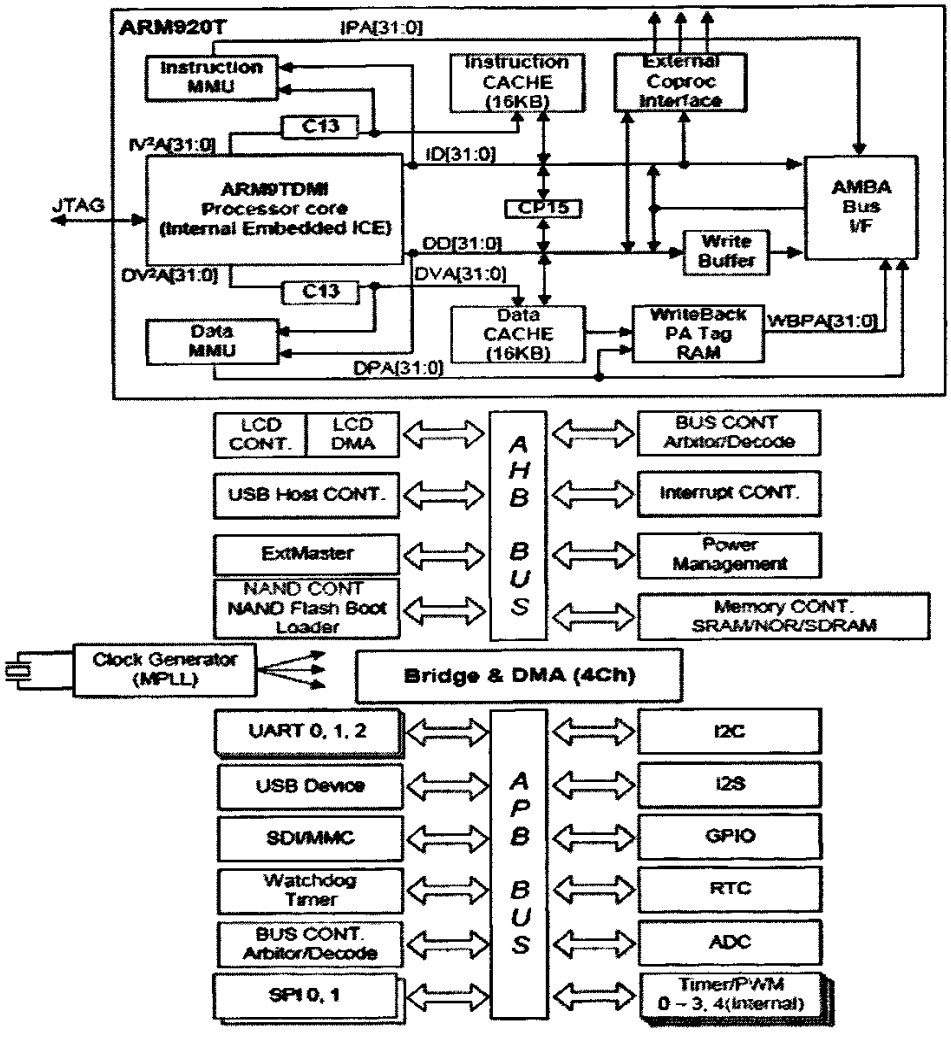

图 2.2 S3C2410A 结构图

S3C2410A使用ARM920T内核，0.18微米的CMOS标准单元和一个附属内存。它低功耗、简单、完美、全静态的设计特别适合对成本和功耗敏感的应用场合。

它采用了全新的总线体系结构：先进微控制器总线体系(AMBA)。由于采用了 ARM 公司的 16/32 位 ARM920T 内核，S3C2410A 具有显著特性。ARM920T 内核具有 MMU、AMBA 总线、哈佛(Harvard)体系结构、分离的 16KB 指令缓存和 16KB 数据缓存由于提供了一套完整的通用系统外设，S3C2410A 使得整个系统成本最小化，而且避免了一些额外的配置。

2.3.3 存储模块

本系统的存储主要分为两个部分：内部存储器和外部存储器。其结构如图 2.3 所示：

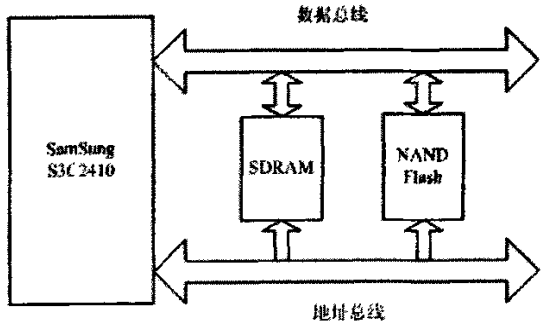

图 2.3 存储模块结构图

本系统的内部存储器使用的是 SamSung K4S561632C[6]的 CMOS SDRAM 作为系统的主存储器，该存储器与 S3C2410A 的连接如图 2.4 所示。每片 SDRAM 的大小为 32M，共使用了两片，它具有如下的特性：

(1) JEDEC 标准，3.3V 供电
(2) LVTTL 兼容多元地址
(3) 四个 Bank 可供使用
(4) 所有的输入在系统时钟的正沿被采样
(5) 突发读单位写操作
(6) 自动自我更新
(7) 64ms 更新时间(8K 周期)

本系统使用 K9F1208U0M[7] NAND Flash，作为系统的外部存储器，它用于存放系统的数据和 Windows CE 操作系统的内核镜像，从而保证掉电后系统依然能正常工作。K9F1208U0M 是 Samsung 公司生产的采用 NAND 技术的大容量、高可靠 Flash 存储器。该器件采用三星公司的 CMOS 浮置门技术和与非存储结构，存储容量为 64M

×8bit，除此之外还有 2048K×8bit 的空闲存储区。K9F1208U0M 有 YCB、GCB、VCB、JCB 和 FCB 等多种封装形式，其中本系统中用到的 K9F1208U0M－YCB0 为 48 针 TSOP1 封装，K9F1209U0M 与 ARM 芯片的连接示意图如图 2.5 所示：

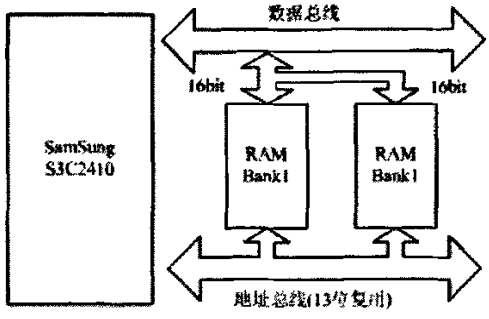

图 2.4 CPU 与内存连接示意图

K9F1208U0M Nand Flash 的主要特性如下：

(1) 工作电压：2.7V~3.6V
(2) 自动编程和擦除
(3) 528 字节页面读取操作
(4) 快速写周期
(5) 命令/地址/数据复合 I/O 端口
(6) 可靠的 CMOS 浮动们技术
(7) 命令寄存器操作
(8) 只能回拷贝操作
(9) 48 脚 TSOP1 封装

2.3.4 显示与输入模块

本系统中使用 Sharp LQ035Q7DB02[8] 作为系统的显示器，它是一个 3.5 英寸 TFT 显示器，它的分辨率是 320x240，同时它还是一款支持触摸输入的显示器，其主要特性有：

(1) Transflective 类型的 TFT-LCD
(2) 6LED 背光
(3) 使用清晰类型触摸屏面板
(4) 提供了 262,144 种色彩
(5) 低功耗(小于 365mW)

它与 ARM 处理器间的连接是通过 EPM7128STC100 的芯片完成的。该 LCD

与CPU间的连接如图2.5所示。S3C2410A的GPC8~GPC15，GPD0~GPD15这24个端口用于向触摸屏发送数据，当EPM7128STC100接收到色彩信息后，将其转换为RGB三原色，然后传送到LCD面板的相应的接口。另外触摸板的水平/垂直同步信号的控制，由GPC0~GPC7端口控制。

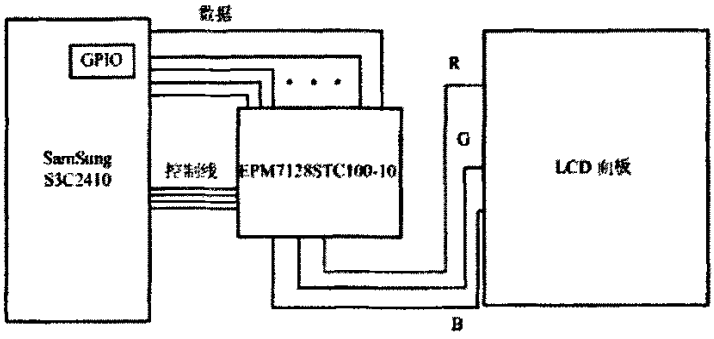

图 2.5　与CPU与LCD连接示意图

2.4 软件系统

本课题的软件系统，主要由2个部分组成：Windows CE嵌入式实时操作系统、操作系统上运行的心电图数据采集和显示软件。在操作系统的选择上，经过比较VxWorks、QNX、Linux、Windows CE等嵌入式操作系统。最终选定了Windows CE作为本课题的操作系统。选择它主要考虑了以下的几个因素：

(1) 与实时系统VxWorks、QNX相比，Windows CE授权的价格低廉

(2) Linux内核的在调度方面主要采用时间轮片方式，无法做到快速中断

(3) Linux系统开发的工具功能简单，调试难度大

(4) Linux系统对的设备驱动库相对较少

(5) Linux系统在嵌入式系统方面，对图形化界面的支持不如Windows CE

正是基于成本、性能、开发工具等方面的综合考虑，本课题采用了Windows CE 5.0作为系统平台，并且以Platform Builder作为主要的开发环境。Windows CE嵌入式系统，它不仅有类似于桌面Windows系统的操作界面，而且作为一款面向实时的嵌入式系统它有自己一些优秀特点：

(1) 已被证明的可靠性

Windows CE具有比任何桌面Windows都可靠的稳定性，从Windows CE 3.0以来的实践已经充分证明了这一点

(2) 多CPU支持和丰富的驱动程序支持

Windows CE 5.0 支持 X86、ARM/Strong ARM、MIPS、SHx 四种架构的 CPU，并且提供了丰富的 BSP 和驱动程序支持。

(3) 企业级的连接性

Windows CE 不仅仅可以用作终端设备的操作系统，也可以用作 Web Server、File Server、Printer Server 等企业服务器。

(4) 实时多任务处理

从 CE4.0 开始，Windows CE 已经成为一个硬实时系统，在 5.0 中实时能力得到了进一步的加强。Windows CE 这种实时多任务处理能力，使它可用于处理工业控制、航空航天等许多时间关键的任务。

(5) 高级电源管理

电源管理用于管理系统设备的电源并提高整个操作系统的效率，电源管理用于设置每个设备的电源状态以及实现不同电源状态间的切换

(6) 可定制的用户接口

在 CE 4.0 后续版本中，Windows CE 允许用户为自己的 CE 设备产生特定的用户界面，允许定制控件和其它用户界面元素的外表等等。

本系统的 Windows CE 操作系统，是根据课题的实际情况由 Platform Builder 5.0 所定制的。仅选取了 ActiveSync、Internet Explorer、QVGA 显示资源。放弃了其他的不必要的功能，如基于 Windows CE 的 Email、微软的文档浏览器、Windows 媒体/MP3 播放器、基于 Windows CE 的 windows 信件客户端、基于 Windows CE 的文字处理器。另外添加了 USB 存储器、FAT 文件系统、USB 鼠标/键盘等功能的支持。

3 Windows CE 系统构建

3.1 Windows CE 系统介绍与分析

在本课题中涉及到驱动程序的开发，Boot Loader 的移植等多个与系统内核相关的内容，因此在此章节介绍一些 CE 内核的相关内容。Windows CE 属于比较典型的微内核操作系统。在内核中仅仅实现进程、线程、调度以及内存管理等最基本的模块，而把图形系统、文件系统及设备驱动程序等等作为单独的用户进程来实现。这样做显著地增加了系统的稳定性和灵活性。

3.1.1 层次体系结构

层次化的设计方法在软件体系结构中非常的普遍。每一个层次都有自己的关注要点和要实现的功能。层次与层次之间构成单项依赖，原则上每一个层次都只与它的临近层次打交道，利用它下一层提供的服务构建自身，同时向更高层次提供一些服务。Windows CE 体系结构可以被分为若干个层次，如图 3.1 所示。

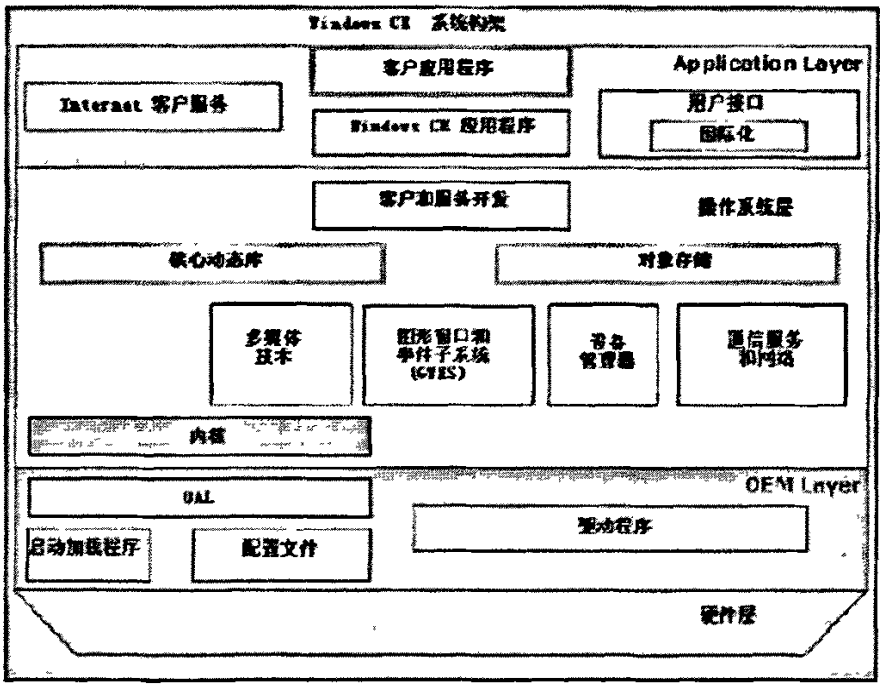

图 3.1 Windows CE 层次体系结构

1. OEM 层

OEM 层是逻辑上位于硬件和 Windows CE 操作系统之间的一层硬件相关的代码。它的主要作用是对具体的硬件进行抽象，抽象出统一的接口然后 Windows CE 内核可以使用这些接口与硬件进行通信。OEM 层包括 OEM 抽象层、引导程序(Boot Loader)、配置文件和驱动程序共 4 个模块：

(1) OEM 抽象层(OEM Abstraction Layer，OAL)是整个 OEM 层的主体。它包含了高度硬件相关的代码。OAL 主要负责 Windows CE 内核与硬件的通信。当引导程序引导操作系统后，由 OAL 负责硬件平台初始化，包括中断服务例程(Interrupt Service Routines，ISR)、实时时钟、计时器、内核调试、开关中断和内核性能监视等工作。OAL 的代码在物理上是内核的一部分，最终经过编译链接，OAL 会称为内核的一部分。

(2) 引导程序(Boot Loader)是硬件开发板上执行的一段代码，它的主要功能是初始化硬件，加载操作系统映像，例如从串口、USB、以太网下载； Boot Loader 也可以从本地的存储设备，例如 CF 卡和硬盘中读取操作系统映像。当 Boot Loder 读取操作系统映像之后，它可把操作系统映像放在内存中或本地的存储设备中以备以后使用。

(3) 配置文件是一些包含配置信息的文本文件。这些配置信息通常与操作系统映像或源代码有关。例如编译系统如何编译某些源代码，或告诉编译系统如何配置最终的操作系统映像文件。

(4) 设备驱动是 OEM 层的另外一个重点。在实际中，设备驱动程序的种类非常多，几乎每一种驱动都有不同的接口，例如声卡驱动、电池驱动、显卡驱动以及 USB 驱动等等。驱动程序直接与硬件打交道，是应用程序或操作系统与硬件外设交互的桥梁。

由于有了 OEM 层，Windows CE 操作系统才可以运行在不同的硬件平台上，实现了广泛的硬件支持。

2. 操作系统层

操作系统层实现了 Windows CE 作为一个操作系统的主要功能。Windows CE 的进程管理、线程管理、处理机管理与调度、物理内存和虚拟内存管理、文件系统及设备管理等功能的实现都位于这一层。由于 Windows CE 是一个微内核操作系统，操作系统的基本功能被放在多个独立的进程(EXE)里面实现。在运行的时候这些进程主要是：内核 NK.EXE，图形系统 GWES.EXE，对象存储 FILESYS.EXE，设备管理系统 DEVICE.EXE，服务 SERVICES.EXE。

由于 Windows CE 是一个可裁剪的嵌入式系统，所以未必所有的 Windows CE

系统中都具有上述的几个进程，如果有的 Windows CE 不包含图形界面，那么这个系统中就不会有 GWES.EXE。其实只有 NK.EXE 和 FILESYS.EXE 是所有 Windows CE 中都必须有的。

(1) 系统调用与 CoreDLL.dll

CoreDLL.DLL 不是一个单独的进程，它是一个会被所有用户进程都加载的动态链接库。所有的应用程序不能直接与操作系统或硬件打交道，如果应用程序希望访问 Windows CE 所提供的服务，那么只能通过 CoreDLL.DLL 进行。CoreDLL.DLL 的主要功能是负责应用程序与 Windows CE 间通信及完成 Windows CE 的系统调用。图 3.2 为 Windows CE 的系统调用示意图。

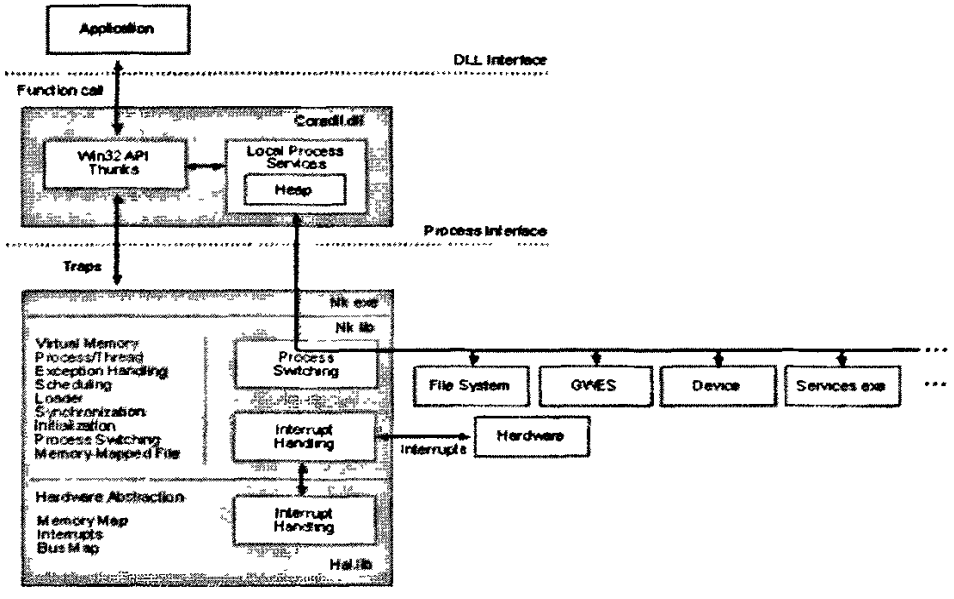

图 3.2 系统调用示意图

(2) 内核 NK.EXE

当系统运行时，Windows CE 的内核表现为 NK.EXE 进程。NK.EXE 是所有基于 Windows CE 的系统中都存在的核心进程，它实现了 Win32API 核心中进程创建加载、线程调度、中断处理和内存管理等核心功能。NK.EXE 由 NK.LIB 和 OAL.LIB 组成。NK.LIB 是由微软提供，它的代码与 CPU 的指令体系结构相关而与具体的外设无关，此种设计使得 OAL 尽可能小；OAL.LIB 是 OEM 层中的 OAL 代码编译后的输出。

(3) 图形系统 GWES.EXE

图形窗口事件系统(Graphical Windowing and Event System，GWES)负责操作系统的图形界面相关的部分。它提供基本的绘图功能和窗口管理。所有的用户输入

相关的驱动程序都由 GWES.EXE 来管理。

(4) 对象存储 FILESYS.EXE

在 Windows CE 中，文件系统与数据存储被统称为对象存储。FILESYS.EXE 就是在运行时负责对象存储的进程。对象存储包括文件系统、Windows CE 数据库和系统注册表 3 部分。而文件系统又可分为 RAM 文件系统和 ROM 系统文件，它们可以为应用程序提供永久存储服务；Windows CE 数据库是对流式文件的更高级抽象，它提供了结构化的数据存储。Windows CE 数据库中的数据以记录的形式存在；系统注册表负责存放系统和应用程序的配置信息。

(5) 服务 SERVICE.EXE

SERVICES.EXE 是负责加载系统服务的进程。系统服务与驱动程序非常类似，但是它们没有管理真实的硬件。它们提供了一些后台的处理或者为应用程序提供高级的功能。在 Windows CE 中，FTP、HTTP、TLENET 都是以系统服务的形式实现的。Windows CE 提供了单独的 API 来启动、停止和操作服务。第三方开发者可以自由地向系统中添加新服务。

(6) 应用程序层

应用层位于 Windows CE 层次结构的最顶层。从系统的角度看，每个应用程序都是 Windows CE 中的一个单独的进程。通常，应用程序运行在权限较低的处理机状态下。它使用操作系统提供的系统调用 API 与操作系统交互。

3.1.2 内存管理

内存管理是操作系统最为重要的模块之一。嵌入式设备拥有的物理内存通常比较小，所幸的是，尽管基于 Windows CE 设备的可用内存通常很小，但是 Windows CE 的内存管理的功能却十分完善。Windows CE 的内存管理几乎实现了 Windows XP/ME 等桌面操作系统内存管理的所有功能，并且针对嵌入式系统的特点对内存管理作了优化和改进。Windows CE 系统是一个 32 位的嵌入式操作系统，因此具有 32 位的寻址能力。

1. 物理内存

在 Windows CE 中 RAM、ROM 和 Flash Memory 都被看成物理内存，而不是仅仅传统意义上的只有 RAM 被认为是物理内存。RAM 在 Windows CE 中像传统的物理内存一样，为操作系统和应用程序提供运行和缓冲空间。ROM 在 Windows CE 中通常用来存储程序。其内容可断电永久保存，就像桌面 PC 中的硬盘一样，它通常存放包括操作系统映像本身和组成操作系统的一些其他文件。ROM 中的内

容通常不能改动的，这样也防止了操作系统的内容被无意改写。ROM 中的程序可被复制到内存中然后再执行，Windows CE 也支持本地执行(eXecute In Place，XIP)，本地执行使代码无需复制到内存中，而直接在 ROM 中执行。

ROM 的一个替代品是 Flash Memory，也通常称为 Flash 或闪存。Flash Memory 和 ROM 相比最大的优点在于可擦写。这样它既可以用来存放操作系统本身，也可以用来存储后来安装的程序。从结构上分类，Flash Memory 主要有 AND、NAND、NOR 或 DiNOR 等，其中主流的是 NAND 和 NOR。NOR 支持线性访问和随机读写，具有较高的速度，但是缺点是容量小，价格偏贵，它可支持 XIP。NAND 的存储和传输是以页和块为单位的，相对适合大数据的连续传输。NAND Flash 的优点是容量大，不对相对的速度较慢，且不支持 XIP。

2. 虚拟内存

相对桌面 Windows 的内存管理，Windows CE 的内存管理最为独特。Windows CE 是 32 位的操作系统所以虚拟寻址能力可达 4GB。与 Windows XP 的每个进程独享 4GB 虚拟地址空间不同，Windows CE 中所有的进程共享一个 4GB 的虚拟地址空间。如图 3.3 所示，Windows CE 的虚拟地址空间又被分成了两个 2GB 区域。低地址 2GB 是用户空间供应用程序使用， 而高地址的 2GB 供 Windows CE 操作系统本身使用。Windows CE 把 4GB 虚拟地址空间分成若干个 Slot，每个 Slot 占用 32MB，Slot 的编号从 0 开始。

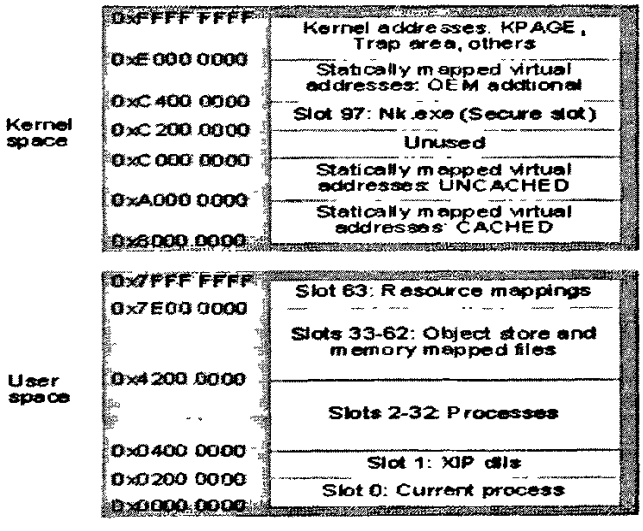

图 3.3 虚拟内存空间

Slot0~Slot32 对应的虚拟地址是 0x00000000~0x41FFFFFF，它们用于存放进程

的虚拟地址空间。Slot0 和 Slot1 基本是一起的。其中 Slot0 用于映射当前在处理器上执行的进程(确切的说是当前在处理器上执行的线程属于哪个进程); Slot1 由 XIP 的 DLL 代码使用。Slot2~Slot32 对应 Windows CE 中每个进程的 32MB 虚拟地址空间，其中 Slot2 通常被 FILESYS.EXE 占用。也就是说理论上在 Windows CE 中最多可以有 30 个用户进程，对应 Slot3~Slot32。但实际上通常的 WinCE 除了 NK.EXE 和 FILESYS.EXE 会产生很多的进程，因为用户实际可用的进程数大大少于 30 个。

Slot33～Slot63 对应的虚拟地址是 0x42000000～0x7FFFFFFF。这块虚拟内存是由所有进程共享的，由于每个进程只有 32MB 的虚拟地址空间，如果应用程序需要使用更多的内存，就可以在这个范围内申请。这个范围包括对象存储和内存映射文件。此范围内的最后一个 Slot 从 0x7e000000～0x7FFFFFFF，用来存放纯资源 DLL。如果某个 DLL 里面只有资源信息，这个 DLL 就会被加载到该空间内。

从 0x80000000 开始是 Windows CE 内核使用的虚拟地址空间。虚拟地址 0x80000000~0x9FFFFFFF 一段用来映射所有的物理地址，这一段共有 512MB。也就是说 WinCE 支持最大为 512MB 的物理地址。虚拟地址 xA0000000~0xBFFFFFFF 会重复映射所有的物理内存，这一段对物理内存地址的映射与 0x80000000 一段最大的不同在于从 0x80000000 开始的一段物理内存是有缓冲的，而从 0xA0000000 开始的一段是没有缓冲的。通常缓冲可以提高系统的 I/O 效率，但是对于一些 OAL 或者 Boot Loader 中的设备驱动来说， 使用缓冲可能带来灾难性的后果，因为缓冲可能改写我们对设备的写操作顺序。因此在驱动程序中如果直接访问 I/O 或寄存器，那么通常使用 0xA0000000 一段的物理地址。

在 ARM 平台上，Windows CE 的静态内存映射是由 OAL 层 OEMAddressTable 数组决定的。物理内存被影射到内核空间之后，Windows CE 内核访问物理内存就变得简单了，如果访问某个物理地址，只需要将该物理地址加上 0x80000000 或 0xA0000000 就可以了。当然这种简便的方式只能由内核使用，通常的应用程序是没有权限这么做的。

地址 0xC2000000~0xC3FFFFFF 是 Slot97，此 Slot 是 Windows CE 的核心进程 NK.EXE 专用的。0xE0000000~0xFFFFFFFF 一段最高的地址是供内核使用的地址空间，对于不同的处理器体系结构，这里保存着不同的东西。通常会放置一些供虚拟内存使用的页表和供内核使用的中断向量表等数据结构。

除了系统启动时 MMU 被启动之前的一小段代码之外，Windows CE 内核和应用程序都是工作在虚拟内存模式下的。Windows CE 提供了以下几个访问函数来访问虚拟内存：

(1) VirtualAlloc 申请虚拟内存

(2) VirtualFree 释放虚拟内存

(3) VirtualProtect 改变虚拟内存的访问权限

(4) VirutalQuery 查询虚拟内存页的属性

3.1.3 文件系统

在 Windows CE 中，存储管理与文件系统都是在 FILESYS.EXE 中实现，它不仅实现了操作系统中最基本的文件系统功能，而且还实现了更多的附加功能。FILESYS.EXE 的结构如图 3.4 所示。文件系统 Windows CE 中提供了 3 种文件系统，分别是：

(1) RAM 文件系统

(2) ROM 文件系统

(3) 可安装文件系统

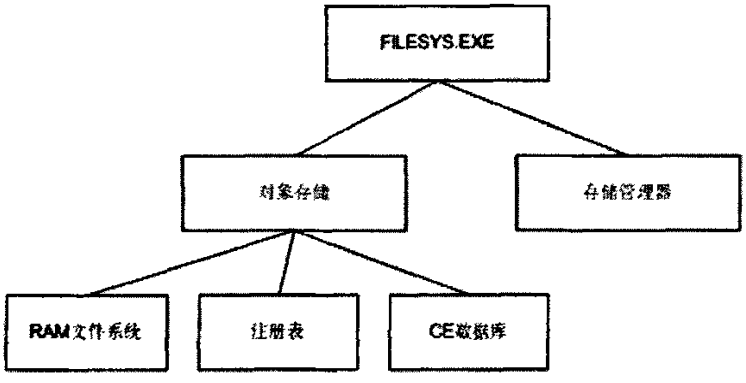

图 3.4 Filesys 的体系结构

Windows CE 中使用的文件系统类似于传统 UNIX 的单根属性目录。它的根目录是“\”，其他所有的可安装文件系统和目录都挂载在根目录之下。无论文件系统的类型是什么， 都可以通过 Win32 API 中的文件系统函数对它们进行访问。

RAM 文件系统通常被直接挂载在根目录下，也就是说根目录下除了挂载的外存目录（例如，CF 卡有可能会被挂载到\Storage Card 目录）和“\Windows”目录之外的所有文件都位于 RAM 文件系统中。例如，在根目录下创建文件 Hello.txt，其实 Hello.txt 位于 RAM 中，对系统进行冷启动后，Hello.txt 就不复存在了。

ROM 文件系统通常挂载到“\Windows”目录。Windows 目录里的内容通常就是 ROMImage.exe 最终生成的 Windows CE 运行时镜像(NK.bin 或者 NK.nb0)中的内容。此外，用户根据外设的需要,挂载自己的文件系统。例如，CF 卡通常被挂载在“\Sotrage Card”目录， CD 驱动器通常挂载在“\CD Driver”目录等等。

Windows CE 与桌面 Windows 一样也使用注册表来保存应用程序、驱动程序

和用户的设定以及一些其他配置信息。Windows CE 注册表也采用树形结构来管理配置信息。注册表相当于整个系统级的配置文件中心数据库。

由于嵌入式系统的特点，一些嵌入式设备是没有外存的，因此 Windows CE 的注册表提供了两种实现方式：基于 RAM 的注册表和基于 Hive 的注册表。

(1) 基于RAM的注册表

正如其名，基于 RAM 的注册表把整个注册表作为一个对象存储堆放在系统的内存中。这意味着如果对系统进行冷启动或者系统断电，那么对注册表的所有改动都会丢失。但是使用 RAM 的注册表，对注册表的读/写访问操作会变得非常高效。因此基于 RAM 的这册表比较适用于没有外部存储器，而且有电池保存内存（Batter-backed RAM）数据的设备。

(2)基于Hive的注册表

基于 Hive 的注册表把注册表数据放在文件系统的文件上。这就意味着无须在系统断电和启动时进行保存/恢复注册表操作。Hive 是注册表中一组键及子键的统称，Hive 在文件系统上表现为单个文件。Windows CE 中有 3 种 Hive， 如表 3.1 所示。

表 3.1 3种Hive

类型	文件	描述
Boot Hive	ROM 中的 Boot.hv	HKEY_LOCAL_MACHINE，HKEY_CLASSES_ROOT，HKEY_USERS 中的所有数据。只在启动时使用。
System Hive	有OEM 决定(通常为 System.hv	HKEY_LOCAL_MACHINE，HKEY_CLASSES_ROOT，HEKY_USERS 中的所有数据，包含设备范围内不随着用户改变而改变的数据。
User Hive	User.hv	HKEY_CURRENT_USER 下面的所有数据。包含用户特有的设置，每个用户都有一个单独的 User.hv

基于 Hive 的注册表适用于有永久存储而且需经常启动的设备。也可以看到，基于 Hive 的注册表把系统数据和用户数据分开存储，这也就意味这基于 Hive 的注册表还提供多用户支持。对于每一个用户，可提供不同的 User.hv，当用户登录时加载相应的 User.hv，从而达到多用户的目的。

3.2 Boot Loader 设计

Boot Loader 是在操作系统内核运行前的一段小程序。通过这段小程序，可以初始化硬件、建立系统的内存空间映射，从而将系统的软硬件环境带到一个已知

的状态，为最终调用操作系统内核准备好环境。最终，Boot Loader 将操作系统内核加载到 RAM 中，并将控制权交给系统内核。Boot Loader 程序对硬件的依赖性非常强，不同的处理器体系结构需要不同的 Boot Loader 支持。同时，Boot Loader 也依赖于具体的嵌入式板级设备的配置，即使同一型号处理器，其使用的外设不同，Boot Loader 代码也有所不同。虽然实现在不同硬件系统环境中通用的 Boot Loader 程序在现阶段是不可能的，但是现有的一些著名的开源 Boot Loader 程序，包括 u-boot、RedBoot、armboot 以及 blob 等，仍致力于归纳出 Boot Loader 程序一些通用的概念和规则，达到程序的最大可重用化。

本课题的启动代码实现也借鉴了相关 Boot Loader 开源程序的源代码。由于本系统中采用 NAND Flash 作为存储系统，所以不能直接从 Flash 上读取 Eboot 并且运行操作系统。由于 S3C2410A 芯片支持将 4K 的 NAND Flash 内容直接拷贝到 RAM 中去，所以为了实现开机就从 NAND Flash 上读取 Eboot 运行操作系统，首先要实现一个 Nboot，其目的是将 NAND Flash 中的 Eboot 加载到内存中，并且调用内存中的 Eboot，通过 Eboot 再加载操作系统。

3.2.1 Image 文件格式分析

Windows CE 源代码、库文件及其它资源经过编译以后，最终生成一个操作系统 Image 文件(称为映像文件)。该文件中包含定制的操作系统的全部内容，它被加载到目标机的内存中执行。在默认情况下，Image 文件名为 NK.bin，位于定制平台的发布目录中，用 Platform Builder 提供的用于查看 Image 文件的命令行工具 viewbin.exe，打开文件，可以列出.bin 文件的内部格式和内容。本课题最终生成的 NK.bin 的查看结果如下：

```
D:\WINCE500\PBWorkspaces>viewbin -r nk.bin
ViewBin... nk.bin
Image Start = 0x80201000, length = 0x00C31154
Record [ 0] : Start = 0x80201000, Length = 0x00000004, Chksum = 0x00000176
Record [ 1] : Start = 0x80201040, Length = 0x00000008, Chksum = 0x000002B5
Record [ 2] : Start = 0x80201048, Length = 0x00000004, Chksum = 0x000001F4
Record [ 3] : Start = 0x80202000, Length = 0x0004AC04, Chksum = 0x01D3493F
Record [ 4] : Start = 0x8024CC04, Length = 0x000033F8, Chksum = 0x0010F5FF
Record [ 5] : Start = 0x80250000, Length = 0x00078298, Chksum = 0x030D05D9
Record [ 6] : Start = 0x802C9000, Length = 0x000062A4, Chksum = 0x000FDF10
Record [ 7] : Start = 0x802D0000, Length = 0x000340A4, Chksum = 0x01455A16
```

Record [8] : Start = 0x80305000, Length = 0x0009D350, Chksum = 0x041089CC

Record [9] : Start = 0x803A3000, Length = 0x000090F4, Chksum = 0x003DFFD6

....

Record [126] : Start = 0x00000000, Length = 0x802099B0, Chksum = 0x00000000

Start address = 0x802099B0

Checking record #124 for potential TOC (ROMOFFSET = 0x00000000)

Found pTOC = 0x80e3063c

ROMOFFSET = 0x00000000

Done.

这是一个二进制编码文件，以记录形式存放数据。开头有一个文件头，其后是 2 个字段的 Image 信息，后面主题部分被划分为 N 个记录(N 的大小视平台配置的不同而不同)，其中前 N-1 个记录由 3 个字段的记录头和记录正文组成。最后一条记录只有记录头，没有记录正文。首先文件开始 7 字节为文件格式标识，它标记了该文件格式。接下来的 2 个字段的内容分别为 Image 开始地址 Image Start 和 Image 大小 Length，以 16 进制数表示，各占 4 个字节。每个记录的记录头的 3 个字段内容分别为记录开始地址 Start、记录大小 Length、校验和 Checksum，以 16 进制数表示， 各占用 4 个字节，后面跟着记录正文。需要强调的是，记录正文长度为 Length，故每个记录在 NK.bin 文件中占用的空间是 3×4 + length 字节。最后一条记录的 Start 和 Chksum 总是 0，该记录无正文，它的作用只是在 Length 字段存储 Start Address。StartAddress 是 Image 文件开始执行的入口点。这个地址一般位于 Record[2]内。

Boot Loader 把 Image 文件 NK.bin 通过网络(串口、或本地网络)装入目标机内存之中。而写入内存中的哪个地址，装入位置与 Image 文件中每条记录的关系如何在微软的相关文档上并未给出相应的内容。

表 3.2 Image 记录头格式

记录	内容
Record[0],Record[1]	系统内容
Record[2]	Nk.exe的text段
Record[3]	ROMHDR
Record[4]	Table of Contents
Record[5]	Nk.exe的data段
Record[6]	Image文件跳转地址

通过分析 BLCOMMON 中的源代码 blcommon.c 可知，如果通过 Ethernet 加载内核，首先读入 usReadSize=文件头(7 字节)+Image 开始地址+Image 大小，比较前面的 7 个字节是否为 B000FF 以确认.bin 文件(参考前面提到的 Image 格式分析)。当读取映像的头 7 个字节之后，检测到是 BIN 文件后，继续读取映像文件的内容，源码如下：

```
while (OEMReadData (sizeof (DWORD), (LPBYTE) &dwRecAddr) &&
    OEMReadData (sizeof (DWORD), (LPBYTE) &dwRecLen) &&
    OEMReadData (sizeof (DWORD), (LPBYTE) &dwRecChk))
{
....
}
```

从源码中可以看出，Boot Loader 按照 Image 中每条记录的地址，把这个记录的正文直接装入目标及内存中，即记录地址就是该记录在目标机内存中实际存储的物理地址。最后跳转地址存入 pulEntryPoint 中，Boot Loader 执行完毕后，程序跳转至 pulEntryPoint，Image 开始执行。

3.2.2 Boot Loader 启动流程

本课题中的的启动分为 stage1 和 stage2 两个过程，依赖于 CPU 体系结构的代码在 stage1 中由汇编语言实现，以达到短小高效的目的。而 stage2 则通常使用 C 语言以实现更加复杂的功能，而且加强了代码的可读性和可移植性。本系统的 Boot Loader 流程如图 3.5 所示。其中，stage1 和 stage2 分别包括如下步骤：

(1) 硬件设备初始化；
(2) 为加载 stage2 准备 RAM 空间；
(3) 复制 stage2 到 RAM 空间；
(4) 设置堆栈；
(5) 搬运 Stage2 代码到 RAM 中
(6) 跳转到 stage2 的 C 入口

而硬件设备的初始化又包括以下几个方面：

(1) 屏蔽所有的中断
(2) 设置 CPU 的速度和时钟频率
(3) RAM 初始化
(4) 初始化 LED

(5) 关闭 CPU 内部指令/数据 Cache

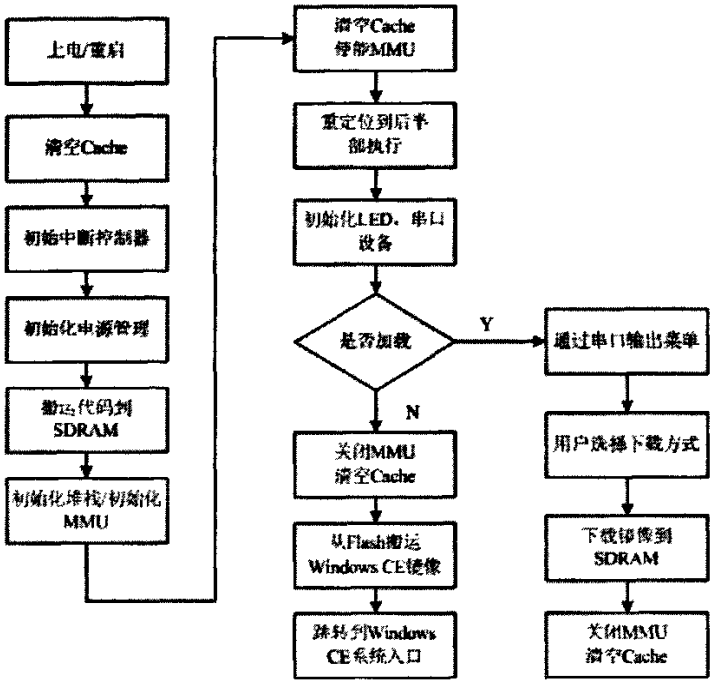

图 3.5 Boot Loader 启动流程

当执行完第一阶段的各项工作后进入第二阶段，Stage2 的过程主要如下：

(1) 初始化本阶段要使用到的硬件设备；

(2) 检测系统内存映射；

(3) 将 Eboot 从 FLASH 读到 RAM 空间中；

(4) 设置 Eboot 的参数；

(5) 执行 Eboot

3.2.3 Nboot

Nboot 是专门用于 NAND Flash 的 Boot Loader。由于本系统所采用的开发板中并没有自带 NOR Flash，只带了 64MB 的 NAND Flash，这也就意味着开机后无法直接在 NAND Flash 运行 Windows CE 内核（因为 NAND Flash 不支持本地执行）。由于 S3C2410A 为了支持 NAND Flash 的系统引导，其具备了一个内部 SRAM 缓冲器，叫做“SteppingStong”。当系统启动时，NAND Flash 存储器前面的 4KB 字节将被自动载入到 SteppingStone 中，然后系统自动执行这些载入的引导代码。而本系统正是利用了 S3C2410A 的这个特性，将存放于 NAND Flash 中的 EBoot

映像拷贝到内存中。NBoot 的实现是根据前面所述的 Boot Loader 的第一、第二阶段的概念完成的，在完成第一阶段的硬件相关初始化工作以后，跳转到用 C 语言编写的第二阶段中，跳转由以下代码完成：

```
;Copy and paste RW data/zero initialized data
ldr r0, =|Image$$RO$$Limit| ; r0 = pointer to ROM data (0x000009bc)
ldr r1, =|Image$$RW$$Base| ; r1 = RAM copy (0x31ff0000)
ldr r3, =|Image$$ZI$$Base| ; r3 = globals (0x31ff0004)
;Zero init base => top of initialised data
cmp r0, r1 ; Check that they are different

beq %F2
1
cmp r1, r3 ; Copy init data
ldrcc r2, [r0], #4 ;--> LDRCC r2, [r0] + ADD r0, r0, #4
strcc r2, [r1], #4 ;--> STRCC r2, [r1] + ADD r1, r1, #4
bcc %B1
2
ldr r1, =|Image$$ZI$$Limit| ; Top of zero init segment
mov r2, #0
3
cmp r3, r1 ; Zero init
strcc r2, [r3], #4
bcc %B3
b Main
```

在第二阶段中，主要的功能在于如何将 Eboot 从 FLASH 中读取出来，并且加载到内存空间中去，然后执行 Eboot。本课题使用 K9F1208U0M 存放操作系统的映像。K9F1208U0M 由 X 地址锁存解码器、Y 地址所存解码器、命令寄存器、NAND Flash 存储矩阵、IO 锁存器等几部分组成。其结构模块如图 3.6 所示。

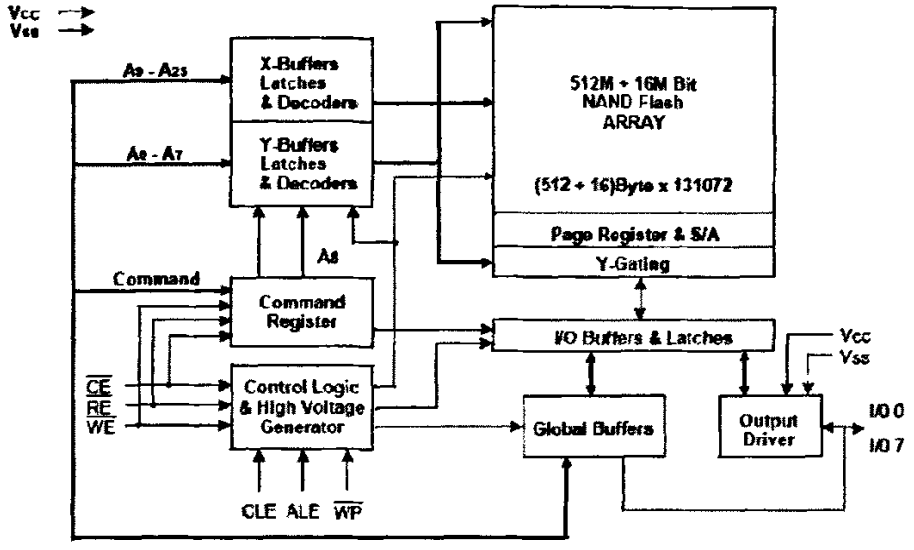

图 3.6 K9F1208U0M 功能模块图

1. K9F1208U0M 结构

正如硬盘的盘片被分为磁道，每个磁道又被分为若干扇区，一块 NandFlash 被分为若干 Block，每个 Block 又被分为若干 Page。由图 3.7 我们可以知道 flash 中 Byte(字节)，Page(页)，Block（块）3 个单位之间的关系为：

1 Page =512 Bytes(Data Field)+16 Bytes(Spare Field)=528Bytes

1 Blcok=32 Pages

K9F1208U0B 总共有 4096 个 Blocks，故我们可以知道这块 flash 的容量为 4096 ×(32 ×528)= 69206016 Bytes = 66 MB 但事实上每个 Page 上的最后 16Bytes 是用于存放检验码用的，并不能存放实际的数据，所以实际上我们可以操作的芯片容量为：

4096 ×(32 ×512) = 67108864 Bytes = 64 MB

1 Blcok=32 Pages

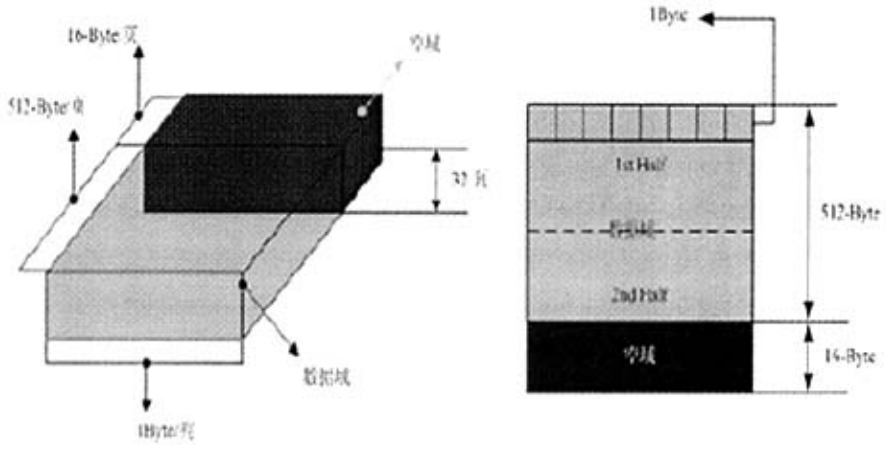

图 3.7 K9F1208U0M 存储结构

2. Nand Flash 寻址

Nand Flash 的地址寄存器把一个完整的 Nand Flash 地址分解成 ColumnAddress 与 Page Address 进行寻址。Column Address 其实就是指定 Page 上的某个 Byte，指定这个 Byte 其实也就是指定此页的读写起始地址。Page Address 总是以 512Bytes 对齐的，所以它的低 9 位总是 0。确定读写操作是在 Flash 上的哪个页进行的。当我们得到一个 Nand Flash 地址 srcaddr 时我们可以这样分解出 Column Address 和 Page Address：

```
column addr=srcaddr%512;    // column address
pageaddress=(srcaddr >>9);    // page address
```

3. Nand Flash 的读取

K9F1208U0M 的读写流程如图 3.8 所示。一般而言可以认为一个 Nand Flash 地址的 A0～A7 是它的 Column Address，A9～A25 是它的 Page Address。Read1 命令的操作分为 4 个 Cycle，发送完读命令 00h 或 01h 之后将分 4 个 Cycle 发送参数，1st.Cycle 是发送 Column Address。2nd.Cycle ，3rd.Cycle 和 4th.Cycle 则是指定 Page Address（每次向地址寄存器发送的数据只能是 8 位，所以 17 位的 Page Address 必须分成 3 次进行发送）。由于 A0～A7 是它的 ColumnAddressr 所以能表示的为 0～255Bytes，但是每个 Page 有 512 Bytes，显然用 A0～A7 是无法表示大于 255 的地址。K9F1208U0M 通过使用不同的读取命令来表示读取前 256 Bytes 还是后 256 Bytes。正因为如此 Data Field 分为两个半区，当要读取的起始地址（Column Address）在 0～255 内时我们用 00h 命令，当读取的起始地址是在 256～511 时，则使用 01h 命令。

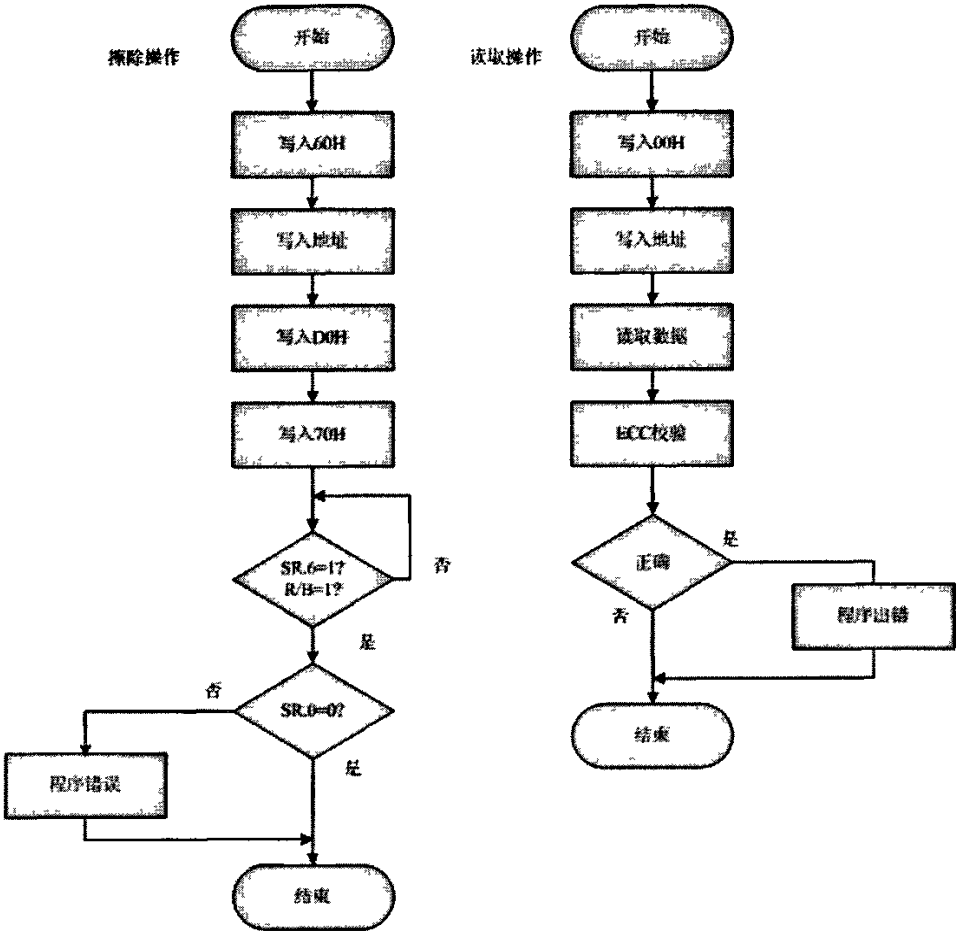

图 3.8 K9F1208U0M 擦除和读取操作

假设现在从第 256 个 byte 开始读取此页则可以如下操作：

```
column_addr=256;
NF_CMD=0x01; //从2nd half开始读取
NF_ADDR=column_addr&0xff; //1st Cycle
NF_ADDR=page_address&0xff; //2nd.Cycle
NF_ADDR=(page_address>>8)&0xff; //3rd.Cycle
NF_ADDR=(page_address>>16)&0xff; //4th.Cycle
```

4. Nand Flash 擦除

对于闪存的独缺和擦除同样是以块为单位的。擦除的时候写入擦出命令(60H)，然后写入要擦除的块地址，写入擦出确认指令 70H，接卸来根据状态寄存器和 R/B 来判断是否成功。如果要对 00B 空间进行操作，需要使用指令(00H)，然后写入要

读取的地址，从寄存器读取数据，接着进行校验。

5. Nand Flash 的写入

NAND 芯片的擦写操作是以块为单位的(块清零操作)，首先要向 I/0 写入串行数据输入指令(80h)，然后使用 3 个时钟周期写入目的地址，接下来向 I/0 写入数据。上面操作完成以后写入页面擦除确认指令(10h)这时由 NAND 片内的逻辑电路完成页面擦除和写入的工作。此后 CPU 需要写入状态寄存器读取指令(70h)读取状态寄存器 SR 对应的状态位(第 6 位)。写入的流程如图 3.9 所示。

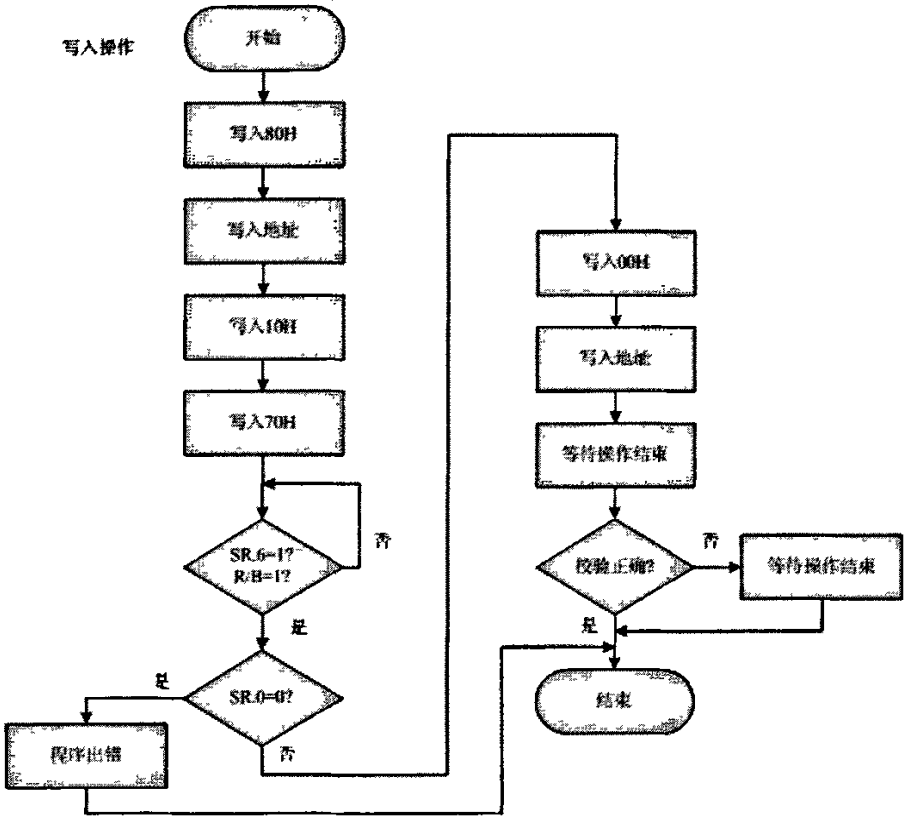

图 3.9 NAND Flash 写入操作

CPU 通过检测 SR 和 R/B 引脚可以确认写操作是否完成。如果操作结束，CPU 检查 SR 的第 0 位来判断擦写是否成功，如果成功程序将正常结束。写操作中如果没有使用 ECC，那么还需要对写操作进行校验。校验过程如图 3.9 所示，首先写入 00h，然后输入要读的页面(就是刚写的页面)，接下来由 NAND 片内的逻辑电路完成写操作的校验工作。

如果在擦写操作和校验的过程中出现问题，则处理比较复杂。如果只是数据

传输过程出现问题可以重新写一次；如果是芯片对应的页面失效，就需要将目的页面映射到其他的页面重新写入，并将对应的页面标记为失效。失效页面通常包括一个或多个失效的比特，可以将其内部数据擦除，但是无法写入。

为了防止对失效页面的操作，通常将失效的页面号写入块的第一页和第二页。对失效的页面管理和检测是非常重要的。通常 NAND 芯片出厂的时候都会被格式化（写入 FF），但是那些有问题的页面不会被格式化。所以在第一次使用芯片的时候可以通过检测芯片内的数据是否为 FF 来判断页面是否有效，从而建立失效页面表。

利用以上的对 NAND Flash 操作的函数，就可以读取存放在 NAND Flash 上的操作系统内核，而在 NBboot 将 Eboot 加载到内存空间中去之前，首先需要知道如下的一些信息：Eboot 读取后放在在 RAM 中的哪个位置、Eboot 在 NAND FLASH 中的位置、Eboot 镜像的校验码、Eboot 镜像的大小等等。因此定义了以下的程序结构：

```
typedef struct _IMAGE_DESCRIPTOR {
DWORD dwVersion; // e.g: build number
DWORD dwSignature; // e.g: "EBOT", "CFSH", etc
UCHAR ucString[IMAGE_STRING_LEN]; // e.g: "PocketPC_2002"
DWORD dwImageType; // IMAGE_TYPE_ flags
DWORD dwTtlSectors; // Total image size in sectors.
DWORD dwLoadAddress; // Virtual address to load image (ImageStart)
DWORD dwJumpAddress; // Virtual address to jump (StartAddress/LaunchAddr)
SG_SECTOR sgList[MAX_SG_SECTORS];
} IMAGE_DESCRIPTOR, *PIMAGE_DESCRIPTOR;

#define IMAGE_EBOOT_SIG 0x45424F54 // "EBOT", nk.nb0
#define IMAGE_RAM_SIG 0x43465348 // "CFSH", nk.nb0
#define IMAGE_BINFS_SIG (IMAGE_RAM_SIG + 1)

typedef struct _CHAININFO {
DWORD dwLoadAddress; // Load address in SDRAM
DWORD dwFlashAddress; // Start location on the NAND
DWORD dwLength; // The length of the image
} CHAININFO, *PCHAININFO;
//
```

```
// TOC: Table Of Contents, OEM on disk structure.
// sizeof(TOC) = SECTOR_SIZE.
typedef struct _TOC {
DWORD dwSignature;
BYTE udid[8];
BOOT_CFG BootCfg;
IMAGE_DESCRIPTOR id[MAX_TOC_DESCRIPTORS];

CHAININFO chainInfo;
} TOC, *PTOC; // 512 bytes
```

在读取了 TOC 块的内容之后，就开始根据 TOC 的内容来读取 NAND Flash，在读取完 TOC 之后，将 Eboot 复制并且加载到 RAM 中，最后跳转到 Eboot 的入口函数，开始运行 Eboot。完整的第二阶段由以下的函数实现：

```
MMU_EnableCache(); //打开MMU缓存
Uart_Init(); //串口的初始化
Uart_SendString(SIGN_ON);
NF_INIT(); //NAND Flash初始化
LcdDisplayPic(); //显示启动界面
ReadImageFromNand(0,0); //从NAND Flash读取镜像
Launch(JumpAddr); //跳转到目标地址
```

3.2.4 Eboot

Eboot 又称为以太网 Boot Loader，是由 BLCommon、OEM 代码、Eboot 和网络驱动程序构成的，如图 3.10 所示。BLCommon 是通用的 Boot Loader 框架；OEM 代码是与目标硬件相关的初始化和扩展功能；Eboot 是以太网功能程序，如 UDP、DHCP、TFTP 程序等等；EDBG 驱动是调试以太网卡的驱动程序；BootPart 是用于存储设备的分区管理程序、FMD 是用于 NAND Flash 或 NOR Flash 的 Flash 管理程序。

当打开电源或者复位的时候，CPU 首先执行 Startup 函数，Startup 函数通常由汇编语言编写，主要用来建立存储器访问和初始化缓存。为了解释物理内存和虚拟内存间的映射关系，首先来看一下 S3C2410A 芯片的物理内存虚拟映射。根据图 3.11 所示，真正的物理内存是放在 SLOT6，SLOT7 中，而且是成对出现的， 也

就是说如果要支持 64MB 的物理内存， 那么必须在 SLOT6、SLOT7 各放置一个 32MB 的物理内存。而且相应的 Slot 的起始地址分别为 0x30000000 和 0x32000000。而且需要注意的一点就是特殊功能寄存器是从 0x48000000 开始的。

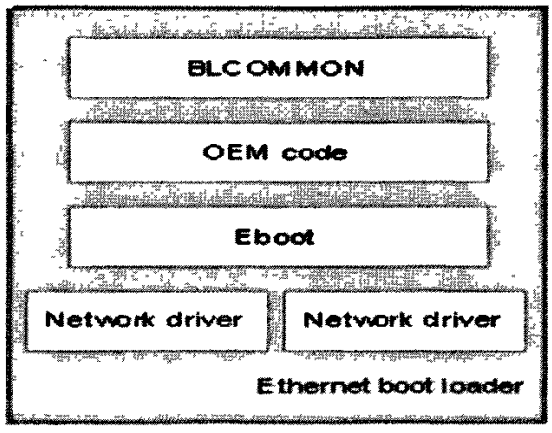

图 3.10 以太网 Boot Loader 的构架

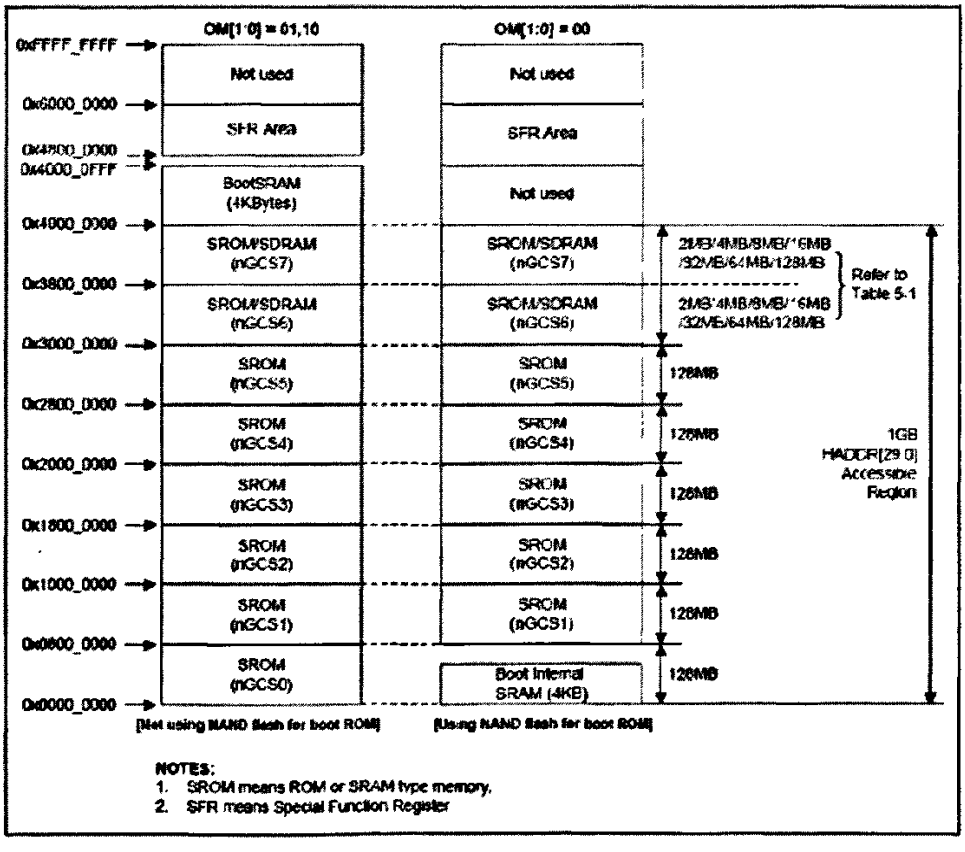

图 3.11 S3C2410A 物理存储器分布

Windows CE 系统和设备进行交互不是通过直接的寄存器或物理地址进行访问的，而是通过虚拟地址和物理地址的转换，对设备寄存器进行访问的。Windows CE 物理内存和虚拟内存的映射表如下定义：

```
g_oalAddressTable
DCD 0x80000000, 0x32000000, 32 ; 32 MB DRAM BANK 7
DCD 0x82000000, 0x08000000, 32 ; 32 MB SROM(SRAM/ROM) BANK 1
DCD 0x84000000, 0x10000000, 32 ; nGCS2: PCMCIA/PCCARD
DCD 0x86000000, 0x18000000, 32 ; 32 MB SROM(SRAM/ROM) BANK 3
DCD 0x88000000, 0x20000000, 32 ; 32 MB SROM(SRAM/ROM) BANK 4
DCD 0x8A000000, 0x28000000, 32 ; 32 MB SROM(SRAM/ROM) BANK 5
DCD 0x8C000000, 0x30000000, 32 ; 32 MB DRAM BANK 6
DCD 0x90800000, 0x48000000, 1 ; Memory control register
DCD 0x90900000, 0x49000000, 1 ; USB Host register
DCD 0x90A00000, 0x4A000000, 1 ; Interrupt Control register
DCD 0x90B00000, 0x4B000000, 1 ; DMA control register
DCD 0x90C00000, 0x4C000000, 1 ; Clock & Power register
DCD 0x90D00000, 0x4D000000, 1 ; LCD control register
DCD 0x90E00000, 0x4E000000, 1 ; NAND flash control register
DCD 0x91000000, 0x50000000, 1 ; UART control register
DCD 0x91100000, 0x51000000, 1 ; PWM timer register
DCD 0x91200000, 0x52000000, 1 ; USB device register
DCD 0x91300000, 0x53000000, 1 ; Watchdog Timer register
DCD 0x91400000, 0x54000000, 1 ; IIC control register
DCD 0x91500000, 0x55000000, 1 ; IIS control register
DCD 0x91600000, 0x56000000, 1 ; I/O Port register
DCD 0x91700000, 0x57000000, 1 ; RTC control register
DCD 0x91800000, 0x58000000, 1 ; A/D convert register
DCD 0x91900000, 0x59000000, 1 ; SPI register
DCD 0x91A00000, 0x5A000000, 1 ; SD Interface register
DCD 0x92000000, 0x00000000, 32 ; 32 MB SROM(SRAM/ROM) BANK 0
DCD 0x00000000, 0x00000000, 0 ; end of table
```

然后 Startup 函数跳转到 BLCommon 框架的 BootLoaderMain 函数。BLCommon 框架的实现为 BLcommon.lib 库，并且与平台特定的 Boot Loader 代码进行连接。

其主框架代码如下所示：

```
void BootloaderMain(void)
{
  DWORD dwAction;
  DWORD dwpToc=0;
  DWORD dwImageStart=0,dwImageLength=0,dwLaunchAddr=0;
  BOOL bDownloaded=FALSE;
  //重定位全局变量到 RAM
  if(!KernelRelocate(pToc))
{
  HALT(BLERR_KERNELRELOCATE);
}
//初始化调试端口(通常用串口)
if(!OEMDebugInit())
{
  HALT(BLERR_DEBUGINIT);
}
EdbgOutputDEbugString(NKSignon,
CURRENT_VERSION_MAJOR,
CURRENT_VERSION_MINOR);
//初始化硬件平台(时钟、驱动器和端口等等)
if(!OEMPlatformInit())
{
  HALT(BLERR_PLATINIT);
}
EdbgOutputDebugString("System...");
//调用 OEM 特定的预下载函数，选择直接跳转到本地 Flash 映像
switch(dwAction=OEMPreDownload())
{
  case BL_DOWNLOAD: //下载内核镜像
  ....
  case BL_JUMP: //直接跳转到本地 Flash 地址
  ....
  OEMLaunch(dwImageStart,dwImageLength,dwLaunchAddr,(const ROMHDR *)dwpToc);
```

}

由以上的代码过程可以看出来，其实 Windows CE 已经将 Boot Loader 框架制定好了，真正需要我们去开发的主要是 OEMPlatformInit 函数的代码，OEMDebugInit 函数的代码，OEMPreDownload 函数的代码等等。由于 OEMDebugInit 等函数涉及到平台的驱动，所讲这一部分的内容放在 OAL 设计部分介绍。

最后通过编译器生成的是 EBoot.exe 文件，但是最终写入 Flash 的是二进制的映像文件，即 bin 文件或者 nb0 文件。转换需要工具 RomImage.exe 程序，RomImage.exe 将 Eboot.exe 转换成相应的二进制文件，转换的过程中需要 Eboot 的.bib 文件。.bib 文件详细描述了 Boot Loader 所使用的内存布局。bib 文件内容如下：

```
MEMORY
; Name Start Size Type
; ------- ------- -------- ----
ARGS 80020800 00000800 RESERVED
RAM 80026000 00006000 RAM
STACK 8002c000 00004000 RESERVED
EBOOT 80038000 00040000 RAMIMAGE
SCRATCH 80078000 00012000 RESERVED
; Area used to cache nk.bin while programming flash
DISPLAY 80100000 00100000 RESERVED
FLSCACHE 80200000 01E00000 RESERVED
CONFIG
COMPRESSION=OFF
PROFILE=OFF
KERNELFIXUPS=ON
ROMOFFSET=31FC8000
SRE=ON
ROMSTART=80038000
ROMWIDTH=32
ROMSIZE=40000
MODULES
; Name Path Memory Type
```

nk.exe $(_TARGETPLATROOT)\target\$(_TGTCPU)\$(WINCEDEBUG)\eboot.exe
EBOOT

启动 Eboot 后，默认情况下，一直等待用户输入选择启动方式，可以是直接从 NAND Flash 读取内核到内存然后运行操作系统，也可以是通过网络下载到内存然后运行操作系统，具体的启动画面及参数设置见附录 A。

3.3 OAL 层开发

在 Windows CE 操作系统中，Boot Loader 和 OAL 共享部分硬件初始化代码，其差别是 Boot Loader 初始化系统 Cache，而 OAL 不初始化 Cache。一般情况下，当基于 Windows CE 的目标设备加电或复位时，系统首先加载并且运行 Boot Loader，Boot Loader 进行必要的硬件初始化，然后通过 Boot Loader 再加载启动操作系统内核映像。启动操作系统内核的过程实际上就是首先加载 OAL 的一个过程。

OAL 并不能被单独加载并运行，最终它是被编译进操作系统内核，从而通过操作系统内核存在并发挥作用，操作系统的启动过程就是一个加载并运行 OAL 程序的过程。OAL 是位于 Windows CE 内核与目标设备硬件之间的一个代码层，用于实现 Windows CE 内核与目标硬件之间的通信。在 OAL 的开发过程中，为了实现内核和硬件之间的最基本的功能，OEM 必须实现一些必要的功能，同时为了适合不同的硬件配置和操作系统功能，OEM 有必要有选择地实现一些其他的功能。以下是必要的 OAL 功能：

(1) Startup 函数
(2) 调试串口
(3) OEMInit 函数
(4) 系统计时器
(5) 中断处理
(6) 内核的输入输出
(7) KITL

由于 Windows CE 系统已经实现了 S3C2410A 处理器相关的中断处理，内核的输入输出，KITL，OEMInit 等部分，所以只需要实现 Startup 函数和调试串口即可。

3.3.1 Startup 函数

Startup 函数是内核启动期间调用的第一个函数，它负责将 CPU 初始化到一个

已知的状态，并且调用内核的初始化函数。在本系统中，由于Boot Loader已经执行了硬件相关的初始化工作，所以这里的Startup函数主要完成以下功能：

(1) 完成最小的CPU和硬件初始化

(2) 关闭终端、缓存(Cache)和内存管理单元(MMU)

源码如下所示：

```
INCLUDE kxarm.h
IMPORT KernelStart
TEXTAREA
;g_oalAddressTable数组包含了内存配置文件
INCLUDE oemaddrtab_cfg.inc
LEAF_ENTRY StartUp
; 计算OEMAddressTable的物理地址并将它加载到寄存器r0
; 这是的MMU还没有打开，知道完全进入kernelStart后才打开
add r0, pc, #g_oalAddressTable - (. + 8)
bl KernelStart
ENTRY_END
END
```

一旦Startup完成了初始化CPU，它会调用函数KernelStart。KernelStart函数主要完成以下任务：

(1) 初始化基于OEMAddressTable内容的第一级存储器页表

(2) 打开MMU和Cache

(3) 为每一种模式初始化堆栈

KernelStart完成了以上的任务后，接着调用主要的内核初始化函数，对于ARM平台KernelStart将调用ARMInit作为主要的内核初始化函数。KernelStart在系统启动时会被自动调用。

3.3.2 调试串口

调试串口使开发人员能够对OAL进行测试，本系统硬件平台有2个串口，其中串口1作为调试串口。为了支持调试串口，需要实现如下几个调试串口函数：

(1) OEMInitDebugSerial()

(2) OEMRealDebugByte()

(3) OEMWriteDebugByte()

(4) OEMWriteDebugString()

这部分代码是 Boot Loader 与 OAL 共用的，所以在前面的 Boot Loader 部分就没有详细描述了。在 OAL 中实现对串口的操作，其实和对普通的单片机中串口进行操作是类似的，主要的区别在于 OAL 层中对串口寄存器的操作是通过物理地址转换为虚拟地址，然后对寄存器变量进行操作的。具体实现如下：

```
//OEMInitDebugSerial ,初始化调试串口
VOID OEMInitDebugSerial()
{
//寄存器变量
S3C2410X_IOPORT_REG *pIOPortReg;
// Configure port H for UART1.
//物理地址到虚拟地址的转换
    pIOPortReg =
(S3C2410X_IOPORT_REG*)OALPAtoVA(S3C2410X_BASE_REG_PA_IOPORT,
                                                    FALSE);
// GPH2 and GHP3 are UART1 Tx and Rx, respectively.
CLRREG32(&pIOPortReg->GPHCON, (3 << 8)|(3 << 10));
SETREG32(&pIOPortReg->GPHCON, (2 << 8)|(2 << 10));
// Disable pull-up on TXD1 and RXD1.
//
SETREG32(&pIOPortReg->GPHUP, (1 << 4)|(1 << 5));
// UART1 (TXD1 & RXD1) used for debug serial.
//
g_pUARTReg = (S3C2410X_UART_REG *)\
OALPAtoVA(S3C2410X_BASE_REG_PA_UART1, FALSE);
// Configure the UART.
//
OUTREG32(&g_pUARTReg->UFCON, BSP_UART1_UFCON);
OUTREG32(&g_pUARTReg->UMCON, BSP_UART1_UMCON);
OUTREG32(&g_pUARTReg->ULCON, BSP_UART1_ULCON);
OUTREG32(&g_pUARTReg->UCON, BSP_UART1_UCON);
OUTREG32(&g_pUARTReg->UBRDIV, BSP_UART1_UBRDIV);
}
//OEMWriteDebugByte ，输出一个字符到调试串口
```

```
VOID OEMWriteDebugByte(UINT8 ch)
{
//等待清空传输缓冲
while((INREG32(&g_pUARTReg->UTRSTAT)&0x02==0);
OUTREG32(&g_pUARTReg->UTXH,ch);
}
//OEMReadDebugByte() 读取一个字符，如果没有读到字符则返回
//OEM_DEBUG_READ_NODATA
int OEMReadDebugByte()
{
UINT32 status,ch;
status=INREG32(&g_pUARTReg->UTRSTAT);
if((status&0x01)!=0){
ch=INREG32(&g_pUARTReg->URTX);
if((status&UART_LINESTAT_RF)!=0)
ch=OEM_DEBUG_COM_ERROR;
else
ch=OEM_DEBUG_READ_NODATA;
}
return (int)ch;
}
```

3.4 驱动程序设计

驱动程序是一个抽象物理设备或虚拟设备的功能软件，驱动程序管理这些设备的操作。物理设备包括网络适配器、计时器和 UART 等，而文件系统是逻辑设备的一个典型例子。实现一个设备驱动程序允许一个设备的功能导出给应用程序和操作系统的其他部分。

根据驱动程序导出的接口不同，驱动程序可以分为本地设备驱动和流接口驱动程序。一些类型的设备，如键盘和显示，对操作系统有一个定制的接口。由于它们使用的接口是 Windows CE 特定的，所以这些类型设备的驱动被称为本地设备驱动。本地设备驱动为特定类型的所有设备都给出了一组标准的功能，这使得 Windows CE 操作系统以相同的方式对待一个特定设备类的所有实例，而忽略它们在物理上的差别。Platform Builder 提供了下列本地设备驱动：

(1) 显示

(2) 键盘

(3) 触摸屏

(4) 指示LED

不管由驱动程序控制的设备是什么类型，凡是导出流接口函数的驱动都是流接口驱动。流接口适用于任何在逻辑上被认为是一个数据源或数据存储的 I/O 设备。即任何以产生或者获取数据流作为主要功能的外围设备都是导出流接口的最好选择。串口驱动就是流接口驱动的一个典型例子。

流接口函数被设计为用来与通常的文件系统 API(如 ReadFile、WriteFile 和 IOControl 等)紧密匹配，即由流接口驱动管理的设备向应用程序表现为一个文件系统，应用程序通过对文件系统的特殊文件进行操作从而完成对设备的操作。如图 3.12 所示，操作系统启动时由设备管理器加载的流接口内建设备驱动程序的构架，应用程序通过文件 API 使用流接口驱动和设备管理器与硬件进行通信。

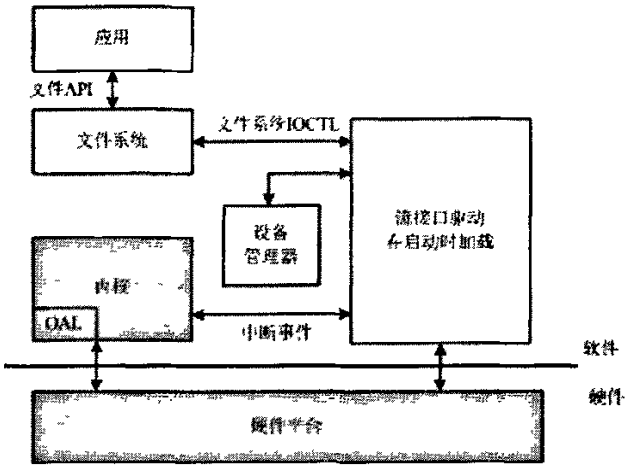

图 3.12 流驱动结构

流接口驱动借助于文件系统调用从设备管理器和应用程序接收命令， 驱动程序封装了将这些命令转换为它所控制设备的相应操作的所有必要信息。除了以上的本地驱动和流驱动的两种驱动外，Windows CE 还提出了相应的分层驱动的概念，如图 3.13 所示。分层的驱动程序将驱动程序代码分为：模型设备驱动(Model Device Driver，MDD)的上层和平台相关驱动(PlatformDependent Driver，PDD)的下层。MDD 层包含了给定类型所有驱动都共用的代码， 而 PDD 层是由特定硬件设备或平台的代码组成。

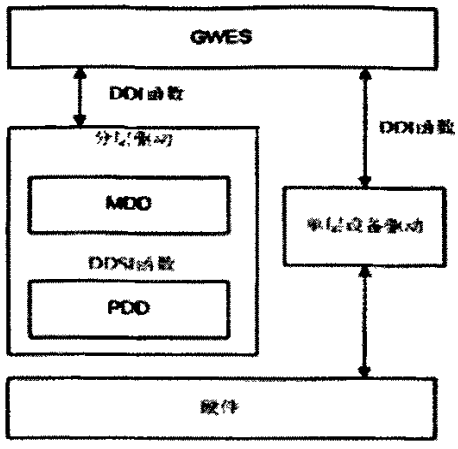

图 3.13 Windows CE 分层驱动模型

3.4.1 触摸屏

S3C2410A 片上内嵌了一个触摸屏控制器，用于控制四线电阻式触摸屏，其结构如图 3.14 所示。在 8 路 A/D(Analogy to Digital Converter ACD)通道中，A[5]，A[7]是用于触摸屏坐标采集的，同时芯片提供 nYPON，YMONn，XPON，nXPON，XMON 四根扫描线对 X 和 Y 方向进行扫描。触摸屏控制器能够以 500KPS 的最大转换速率，把输入的模拟信号转换成 10 位二进制数字同步输出。。S3C2410A 触摸屏控制器有以下几种处理模式：

(1) X/Y 标准转换模式；

(2) X/Y位置分别转换模式；

(3) X/Y为孩子自动转换模式。触摸屏控制器会自动转换X和Y坐标；

(4) 中断等待触发模式，不过要和前面的3种模式之一结合；

ADC 控制器和触摸屏接口的处理流程如下：

(1) 通过外部晶体管连接触摸屏面板和S3C2410A

(2) 选择触摸屏的接口模式

(3) 设置触摸屏接口为中断等待模式

(4) 如果中断发生，合适的坐标转换机制被激活

(5) 得到合适的X/Y坐标值之后，返回到中断等待模式

Windows CE 的设备驱动程序实现有两种方法：轮询和中断。为了减少系统的功耗和触摸屏占用 CPU 资源，系统采用中断的方式接受输入，只有在落笔时触摸屏控制器才启动扫描，通过 A/D 转换得到相应的坐标值。由于触摸屏控制器的 X，

Y 值仅仅是对当前触摸点的电压值的 A/D 转换值，它不具有实用价值，这个值的大小不但和触摸屏的分辨率相关，而且也和触摸屏与 LCD 贴合的情况有关。

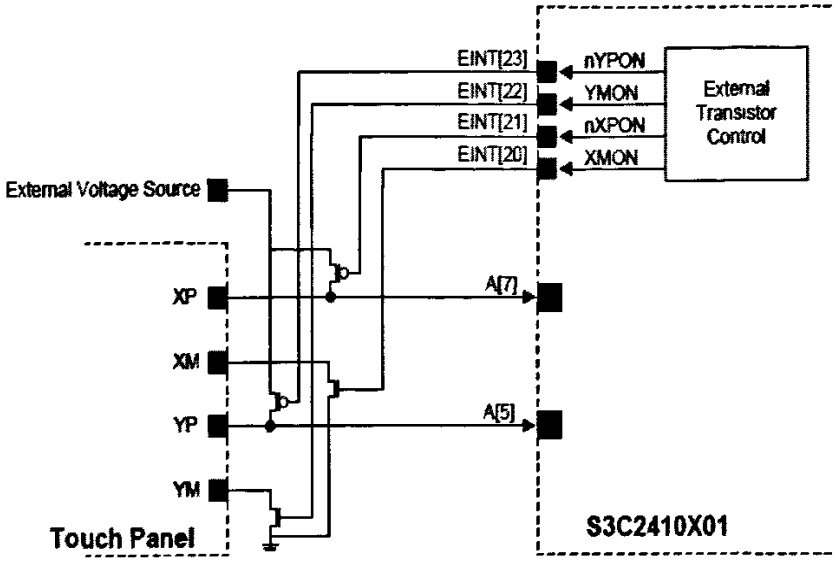

图 3.14 触摸屏连接示意图

为了想得到体现 LCD 坐标的触摸屏位置，还需要在程序进行转换。转换公式如下：

x=(x-TchSrcXMin)×LCDWIDTH/(TchSrcXMax-TchSrcXMin)

y=(y-TchSrcYMin)×LCDHEIGHT/(TchSrcYMax-TchSrcYMin)

其中 TchScrXMax，TchScrXMin，TchScrYMin，TchScrYMax 是触摸屏返回电压值 x，y 轴的范围，LCDWIDTH、LCDHEIGHT 分别为液晶的宽度和高度。

由于触摸屏控制器启动后能自动完成扫描，并且将物理坐标的 A/D 值存入特定寄存器然后产生触摸屏扫描 A/D 中断，因此可以在扫描 A/D 中断服务程序中采集物理坐标并将其转换为显示坐标。整个触摸屏中断流程图如图 3.15 所示。

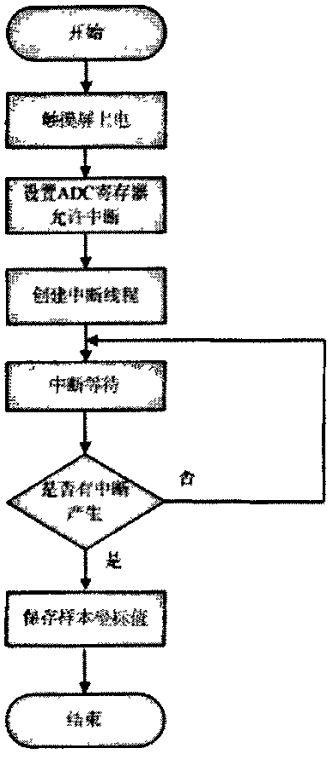

图 3.15 触摸屏中断流程

本驱动程序是在微软提供的样本驱动程序的基础上开发，触摸屏样本驱动程序是一个分层结构：上层是模块设备驱动程序(MDD)，下层是依赖的驱动(PDD)。MDD 层是不需要修改的，而 PDD 层与 MDD 和硬件都有接口，是与具体平台相关的。以下是触摸屏驱动程序的 DDSI 函数集(PDD 层)：

(1) DdsiTouchPanelGetDeviceCaps()
查询触摸屏的参数，比如采样平率，校正点数等等

(2) DdsiTouchPanelSetModel()
设置触摸屏的信息，主要是设置采样频率的高低

(3) DdsiTouchPanelEnable()
给触摸屏供电，并且初始化

(4) DdsiTouchPanelDisalbe()
关闭触摸屏

(5) DdsiTouchPanelAttach()
当MDD的DLL入口函数收到一个DLL PROCESS ATTACH消息时将执行该函数

(6) DdsiTouchPanelDetach()

当MDD的DLL入口函数收到一个DLL PROCESS DEATTACH消息时将执行该函数

(7) DdsiTouchPanelGetPoint()

返回当前的坐标点

(8) DdsiTouchPanelPowerHandler()

用于告知驱动程序，系统处于休眠或非休眠状态，仅由MDD调用

以下是简化的实现代码：

```
//由于本系统不用手写输入所以不用
//实现Attach和Detach功能
LONG   DdsiTouchPanelAttach(VOID)
{
  return(0);
}
//
LONG   DdsiTouchPanelDetach(VOID)
{
return(0);
}
PUBLIC VOID DdsiTouchPanelPowerHandler(BOOL bOff)
{
    RETAILMSG(0, (TEXT("::: DdsiTouchPanelPowerHandler()\r\n")));
    if (bOff)
    {
       v_pINTregs->INTSUBMSK |= (1<<IRQ_SUB_TC);
    }
    else
    {
   TSP_PowerOn(VOID); //打开电源并且进行初始化
    }
}
//
PUBLIC VOID DdsiTouchPanelGetPoint()
{
    TSP_SampleStart(); //开始坐标采样
```

```
        ...
        TSP_SampleStop(); //
        TSP_GetXY(); //估计样本返回X/Y坐标的最佳值
         ...
  }
  //
  PUBLIC BOOL DdsiTouchPanelGetDeviceCaps(INT iIndex, LPVOID lpOutput)
  {
    if ( lpOutput == NULL )
    {
        ERRORMSG(1, (__TEXT("TouchPanelGetDeviceCaps: invalid parameter.\r\n")));
        SetLastError(ERROR_INVALID_PARAMETER);
        DebugBreak();
        return (FALSE);
    }
   switch ( iIndex )
   {
     case TPDC_SAMPLE_RATE_ID:
     {
      .... //返回采样频率
     }
     break;
     case TPDC_CALIBRATION_POINT_COUNT_ID:
     {
        .... //返回坐标校准的点数
     }
     break;
case TPDC_CALIBRATION_POINT_ID:
.... //返回对应某个点的校准后的数据
default:
....
}
return (TRUE);
  }
```

最后需要在 Platform.reg 中配置与触摸屏驱动有关的信息：

```
[HKEY_LOCAL_MACHINE\ControlPanel]
InputConfig=dword:3:3;
```

3.4.2 LCD

Windows CE 下的显示驱动与 Windows NT 下的显示驱动在实现上是类似的，都是基于用户模式的动态链接库。显示驱动被 GWES(图像、窗口事件子系统)加载并且调用。显示驱动也是分层的结构，因为有许多的硬件独立的操作。

GPE(Graphics Primitive Engine)库处理默认的画图方面的功能，它是作为显示驱动的 MDD 层。而我们需要开发的就是依赖于设备的 PDD 层。在 WindowsCE 中提供了 GPE 类，该类定义了基本的数据显示的操作，而将与设备相关的一些初始化、开关等操作作为虚拟函数。在开发显示驱动的过程中，只需要实现这些导出的接口函数即可。

Windows CE 针对屏幕旋转/不旋转提供了 GEP 类和 GPERotate 类，本系统中不需要屏幕旋转，所以直接从 GPE 类继承，实现的类为 S3C2410DISP。由于微软已经提供了部分的显示代码，所以主要的工作，集中在如何初始化 LCD 寄存器，和初始化显示缓冲区。

S3C2410A 的 LCD 控制器的结构如图 3.16 所示。REGBANK 是 LCD 控制器的寄存器组，用来对 LCD 控制器的各项参数进行设置。LCDCDMA 则是 LCD 控制器专用的 DMA 信道，负责将视频资料从系统总线（System Bus）上取来，通过 VIDPRCS 从 VD[23:0]发送给 LCD 屏。同时 TIMEGEN 和 LPC3600 负责产生 LCD 屏所需要的控制时序，例如 VSYNC、HSYNC、VCLK、VDEN，然后从 VIDEO MUX 送给 LCD 屏。该 LCD 控制器有以下特点：

(1) 支持单色、4级灰度、256色的调色板显示模式

(2) 支持64K和16M色非调色板显示模式

(3) 支持分辨率为640*480，320*240及其它多种规格的LCD

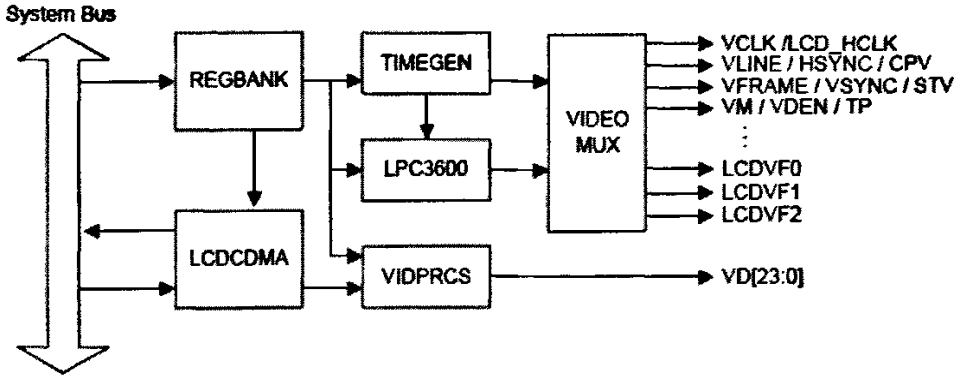

图 3.16 LCD 功能模块图

对于控制 TFT 屏来说，除了要给它送显示数据（VD[23:0]）以外，还有以下一些信号是必不可少的：

(1) VSYNC（VFRAME）　帧同步信号

(2) HSYNC（VLINE）　行同步信号

(3) VCLK　像数时钟信号

(4) VDEN（VM）　数据有效标志信号

具体的寄存器设置如下：

```
s2410IOP->GPGUP &= ~(1 << 4);
s2410IOP->GPGCON &= ~(3 << 8);
/* VCLK = HCLK / ((CLKVAL + 1) * 2) -> About 7 Mhz */
s2410LCD->LCDCON1 = (7 << 8) |
(LCD_MVAL_USED << 7) | /* 0 : Each Frame */
(3 << 5) | /* TFT LCD Pannel */
(12 << 1) | /* 16bpp Mode */
(0 << 0) ; /* Disable LCD Output */
s2410LCD->LCDCON2 = (LCD_VBPD << 24) | /* VBPD: 1 */
(LCD_LINEVAL_TFT << 14) | /* Vertical Size : 320-1 */
(LCD_VFPD << 6) | /* VFPD : 2 */
(LCD_VSPW << 0) ; /* VSPW : 1 */
s2410LCD->LCDCON3 = (LCD_HBPD << 19) | /* HBPD: 6 */
(LCD_HOZVAL_TFT << 8) | /* HOZVAL_TFT : 240-1*/
(LCD_HFPD << 0) ; /* HFPD : 2 */
s2410LCD->LCDCON4 = (LCD_MVAL << 8) | /* MVAL : 13*/
(LCD_HSPW << 0) ; /* HSPW : 4 */
```

```
s2410LCD->LCDCON5 = (0 << 12) | /* BPP24BL : LSB valid */
                    (1 << 11) | /* FRM565 MODE : 5:6:5 Format */
                    (0 << 10) | /* INVVCLK : VCLK Falling Edge */
                    (0 << 9) | /* INVVLINE : Inverted Polarity */
                    (0 << 8) | /* INVVFRAME : Inverted Polarity */
                    (0 << 7) | /* INVVD : Normal */
                    (0 << 6) | /* INVVDEN : Normal */
                    (0 << 5) | /* INVPWREN : Normal */
                    (0 << 4) | /* INVENDLINE : Normal */
                    (1 << 3) | /* PWREN : Disable PWREN */
                    (0 << 2) | /* ENLEND : Disable LEND signal */
                    (0 << 1) | /* BSWP : Swap Disable */
                    (1 << 0) ; /* HWSWP : Swap Enable */
    s2410LCD->LCDSADDR1 = ((dwPhysicalFrameBase >> 22) << 21) |
                    ((M5D(dwPhysicalFrameBase >> 1)) << 0);
    s2410LCD->LCDSADDR2 =
M5D((dwPhysicalFrameBase+(LCD_XSIZE_TFT*LCD_YSIZE_TFT*2))>>1);
    s2410LCD->LCDSADDR3 =
(((LCD_XSIZE_TFT-LCD_XSIZE_TFT)/1)<<11)|(LCD_XSIZE_TFT/1);
                    s2410LCD->LPCSEL |= 0x0;
                    s2410LCD->TPAL = 0x0;
                    s2410LCD->LCDCON1 |= 1;
```

以上部分代码主要是在 S3C2410DISP 类中的 InitializeHardware 类中实现，但是真正的驱动要加载到系统中去，并且要判断驱动程序是否存在，需要注册表的支持，注册表的实现代码如下：

```
IF BSP_NODISPLAY !
[HKEY_LOCAL_MACHINE\Drivers\Display\S3C2410\CONFIG]
"DisplayDll"="s3c2410x_lcd.dll"
"LCDVirtualFrameBase"=dword:a0100000
"LCDPhysicalFrameBase"=dword:30100000
[HKEY_LOCAL_MACHINE\System\GDI\Drivers]
"Display"="s3c2410x_lcd.dll
```

4 心电图滤波算法设计

心电图(Electrocardiogram，ECG)是心脏活动相关的电位变化图，它是医生诊断疾病的一种现代技术，尤其在确诊和鉴别各种心律失常方面，心电图诊断法较其他方法都可靠。由于ECG和绝大多数的生物数字信号一样，都是信噪比很低的微弱信号，往往混有很强的背景噪声，主要的噪声为：工频干扰、肌电干扰、基线漂移等。噪声严重时可完全淹没ECG信号或使基线漂移剧烈。因此必须从硬件设备和分析软件上消除噪声的影响。如何利用数字滤波器过滤工频干扰和基线漂移是本课题解决的重点。

4.1 心电图数据特征

正常人体内，每一个心动周期中，心脏各部分兴奋过程中出现的电位变化的方向、途径、次序和时间都有一定的规律，这种生物电变化通过心脏周围的导电组织和体液反映到身体表面上来，使身体各部位在每一心动周期中也都发生有规律的电变化。将测量电极放置在人体表面的一定部位，记录出来的心肌细胞生物电变化曲线即为临床常规心电图，它反映心脏兴奋的产生、传导和恢复过程中的生物电变化。

与其他生物电信号相比，心电信号更易于检测并具有较直观的规律性。心电信号主要是由P、QRS、T和静息期组成，各个特征波形幅值及频率特性都不尽相同，表示了不同的心脏活动。作为一种非平稳信号，它在采样过程中常常掺杂各种噪声和干扰。一个典型的心电信号周期如图4.1所示。该图清晰的表明了P、Q、R、S等子波及各个间期的区间，它们的具体意义及参数分别为：

(1) P波：由心房的激动所产生，反映心房肌除极过程的电位变化。前半部分主要由右心房所产生，后半部分则主要由左心房所产生。正常P波的宽度不超过0.11s，典型值为0.2mV，在同一导联内P波的心态应该是一致的；

(2) P-R间期：代表从心房肌开始除极到心室肌开始除极的时限。正常P-R间期为0.10~0.12s，不同导联测量的P-R间期可略有差别。P-R间期的长短与年龄、心率有关；

(3) QRS波群：由左、右心室的激动产生，反映心室肌除极过程的电位变化。QRS波群的宽度称为QRS时限，代表该过程所需要的时间，正常情况下该间期最高不超过0.10s，Q波：0.1mV；R波：0.5~1.5mV；S波：0.2mV；

(4) T波：反映心室复极时的电位变化。范围为0.1~0.5mV，在以R波为主的心电图上，T波不应低于R波1/10；

(5) Q-T间期：自QRS波群开始至T波结束称为Q-T间期，反映心室除极和复极时间的总和(电收缩时间)。心率愈慢，Q-T间期愈长，心率愈快，Q-T间期愈短；

(6) U波：位于T波之后，可能是反映心肌激动后电位与时间的变化，人们对它的认识仍在探讨之中

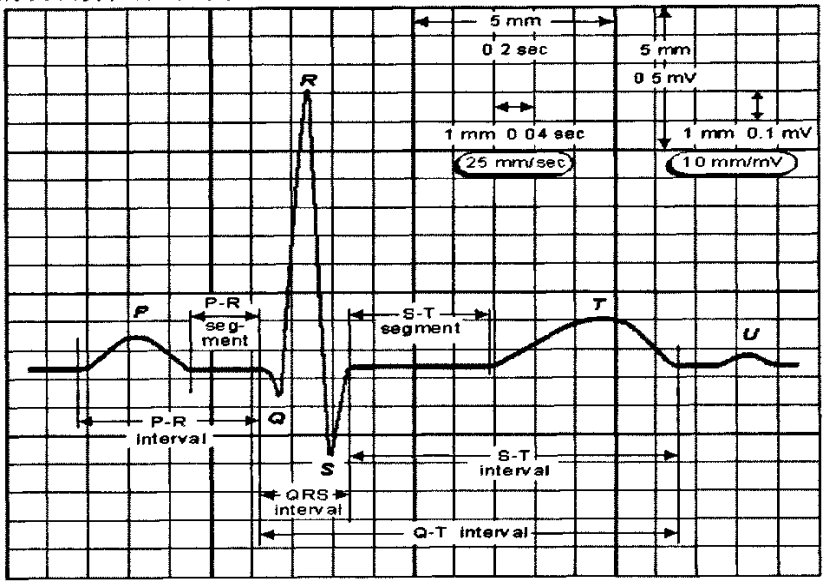

图 4.1 典型的心电信号波形

在心电图仪的采样数据采集过程中，需要尽可能准确的获得这些信息。同时，在数据的处理，如滤波和显示的过程中，需要在滤除噪声的同时，保留特征波形的细节，以便医生做出准确的诊断。这一要求也是衡量滤波效果，甚至是衡量整个心电图设备质量好坏的重要标准。

4.2 滤波器算法设计

由于心电信号存在多种噪声，在实际应用中为解决这一问题通常需要从软硬件两个方面进行滤波。从硬件上，通过合理屏蔽和接地的措施，使用性能更优良的器件，可以明显的减弱噪声，但仅仅依靠硬件并不能完全解决这一问题，而且，更多的器件会增加设备功耗和成本。 从软件上，针对不同的噪声类型有不同的滤波器设计方法可用，但是在性能上存在较大的差异。

在本课题中，对多种滤波器进行了分析，并且进行了比较，实验中的数据样本均为 8 位，采样频率为 256。

4.2.1 去除基线漂移

在人体做心电图检查时(ECG)，由于体位的移动，在记录到的心电图中往往伴有基线漂移现象，严重时会使信号跑出记录纸或屏幕可显示的范围以外，图 4.1 就是一个明显带有基线漂移的心电图数据。基线漂移现象是由很低频率的信号引起的，类似这样的低频信号有时又被称为信号中的趋势项。基线漂移的频率范围在 0.05~1.0Hz 左右表现为一个缓慢变化的信号[6]。

常见的去除基线漂移的主要方法有：

(1) 滑动平均滤波

(2) 简单整系数滤波

(3) 自适应滤波

(4) 小波变换法

由于在本课题中，如何提高算法的效率，以及减少滤波算法本身的时间复杂度从而来实现实时滤波，是本课题的主要方向。下面介绍本课题中所采用的几种实验过的滤波算法，来去除基线漂移以及最终采用的滤波算法。

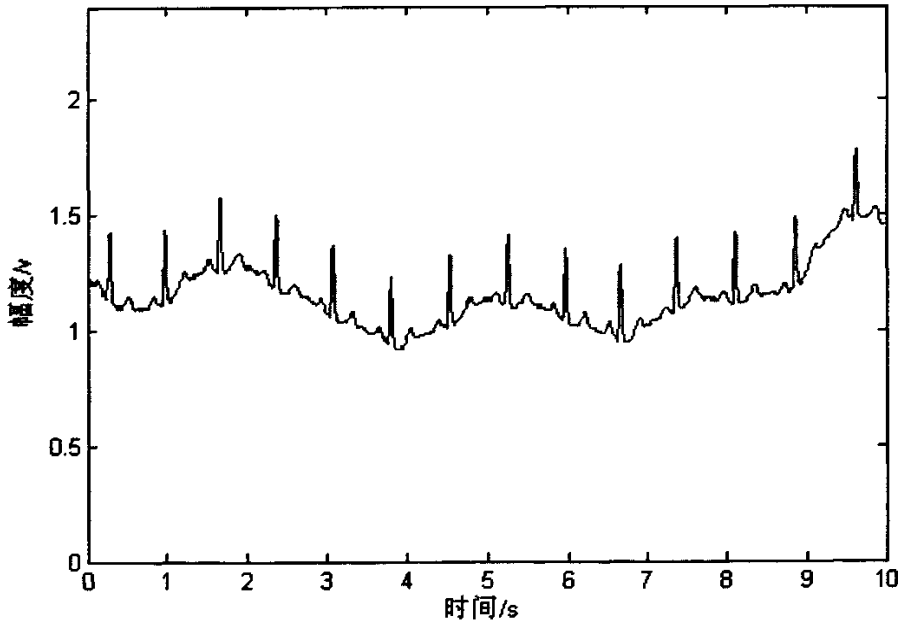

图 4.2 带有基线漂移的心电图

1. 滑动平均滤波

滑动平均滤波[7]实现比较简单。当心点信号的采样频率为 256Hz 时，采用平均点数为 256 点的平均进行滤波：

$$y(n)=((x(n)+x(n-1)+...+x(n-254)+(n-255))/256 \quad (4.1)$$

其中x(n)为原始带有干扰的新店信号，y(n)为要滤除的干扰数据。然后用原始数据减去该干扰数据就可以得到滤波后的数据，该滤波器原理简单，但是由于受心率的影响，滤波器阶数较高，计算量大。其算法的时间估计为

$$\sum_{i=0}^{n}(x(i)+x(i-1)+...+x(i-n+1)+x(i-n))=O(n^2) \quad (4.2)$$

另外，它有很大时延，因为每一个滤波的数据点，都需要等待下一次完整的n个数据采集到才行。这在实时滤波中，显然是不可行的。

2. 滑动窗口中值滤波

滑动中值滤波是滑动平均滤波的一种改进算法，它主要是对ECG信号中已经漂移了的基线进行估计和提取，然后通过减法运算，去掉信号中的漂移成分，从而达到滤波的目的，算法过程如下图所示：

图 4.3 滑动窗口中值滤波过程

假设待处理的ECG信号为ecg1，信号长度为L，算法流程图如图4.3所示。

(1) 选择合适的窗宽W，为了便于处理和加快算法的运算速度，可以取W为32、64等2的整数幂；

(2) 对ecg1信号加窗，对窗口内的信号进行中值滤波，即对窗口内的信号进行排序，然后用中指来取代窗口中心点的值，然后移动该窗口，遍历ecg1的信号，拟合出漂移了的基线BL，BL与ecg1间的关系如下公式所示：

$$BL(i)=median[ecg1(i):ecg1(i+W)],0\le i\le L-1 \quad (4.3)$$

其中median()函数表示去中值的操作，另外通过以上公式可知，由于每一个点需要W个点的排序运算，然后取其中的中值；

(3) 从原始ECG信号ecg1中减去BL，得到消除极限漂移后的ECG信号ecg2；

(4) 输出ecg2信号；

该算法相对于于移动平均算法，主要的改进的地方在于它缩小了窗口的大小，但是它的弱点在于中值算法。由于中值算法主要的操作在于排序，目前最好的常用的排序算法的时间复杂度为O(nlgn)，假设W选为64，那么整个算法的时间复杂度为：

$$\sum_{i=1}^{n} 64\lg 64 = n64\lg 64 = O(n) \qquad (4.4)$$

显然，从时间复杂度来看该算法有了很大的改进，但是窗口的大小严重的影响着滤波的效果，如图 4.3~4.6 所示，列出了不同窗口 W 大小滤除极限漂移的效果。从各图的滤波结果来看，当 W=32 时，基线基本上拉回到水平位置，但是可以发现，QRS 波形并未被完整的保留下来，而 QRS 波形是医生诊断病人病情的主要依据，显然 W=32 的滤波结果是无法满足应用需求的。当 W=64 时，可以看

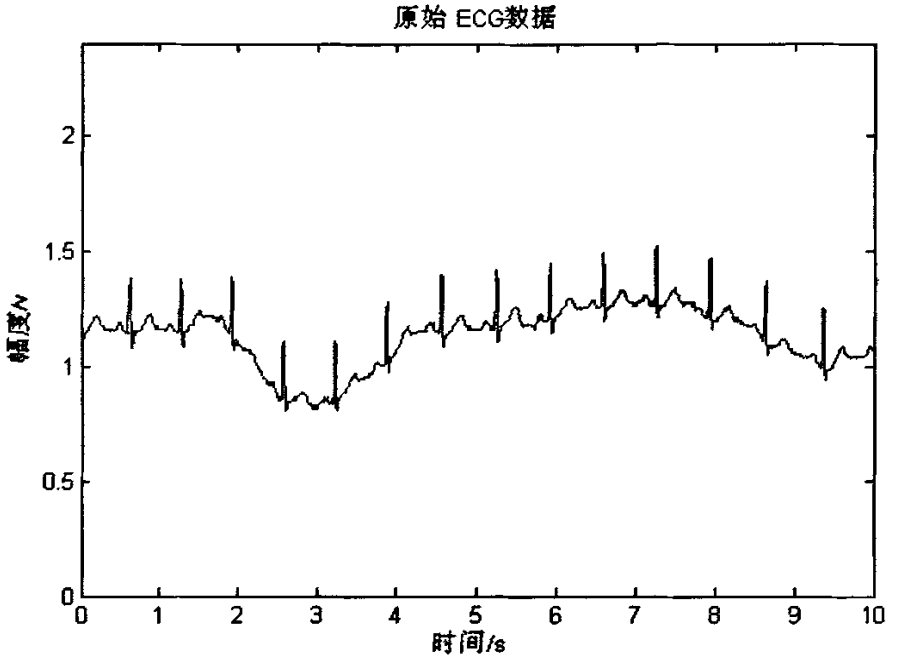

图 4.4 带有基线的原始测试数据

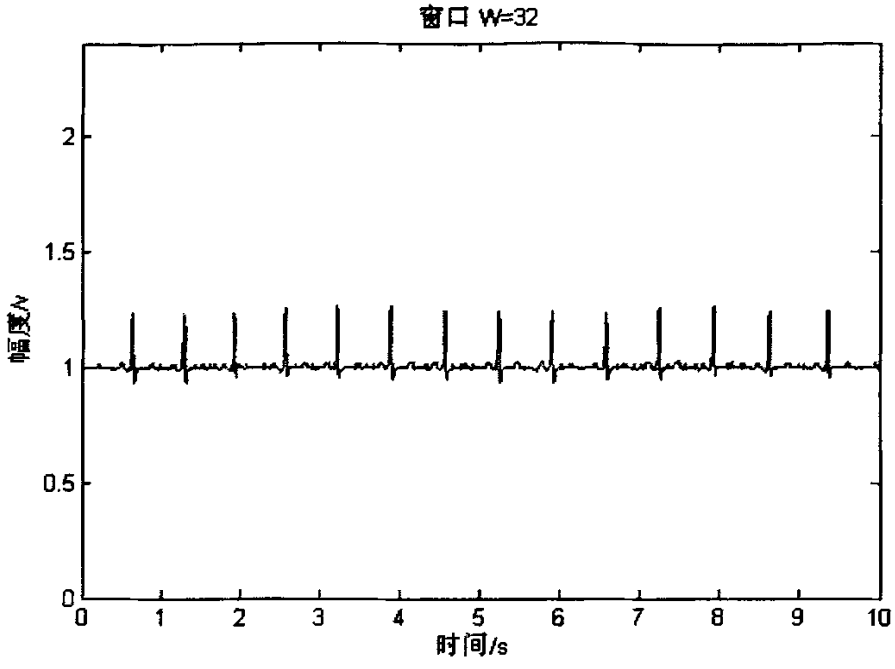

图 4.5 窗口 w=32 的滤波结果

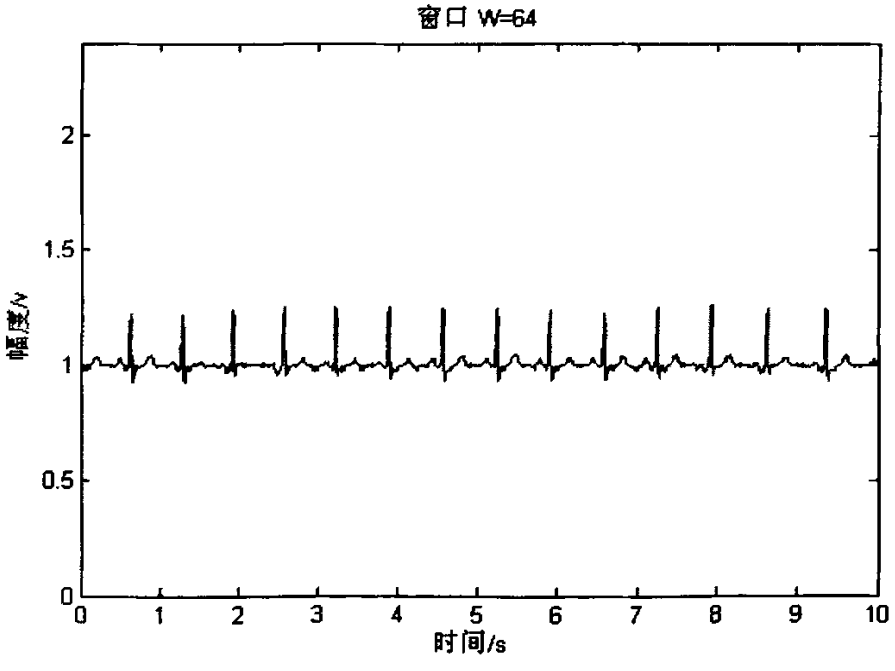

图 4.6 窗口 w=64 的滤波结果

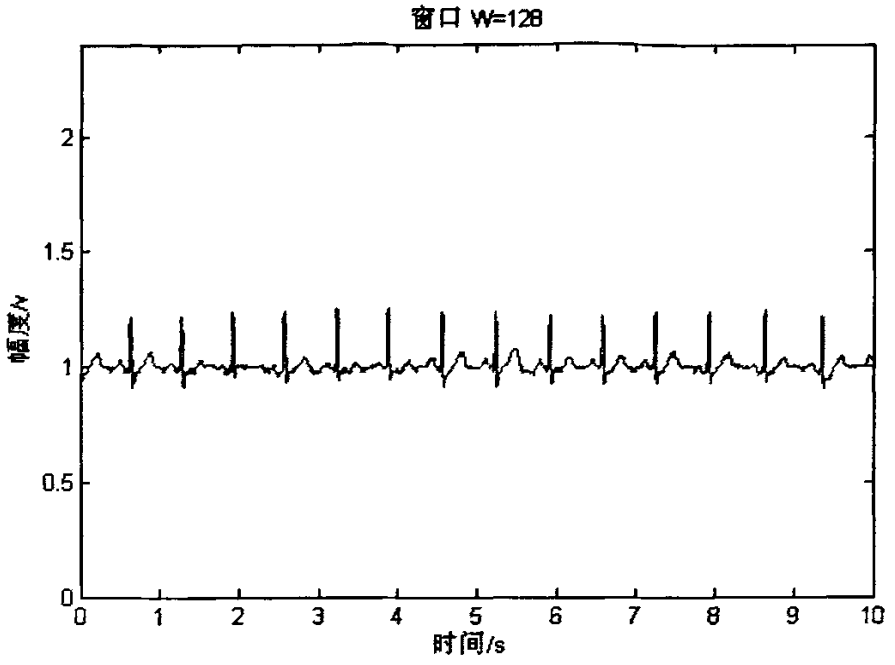

图 4.7 窗口 w=128 的滤波结果

出，QRS 波形大致被保留了，但是还是有相当部分被削弱了，最后调整 W 到 128 时，QRS 基本完整的保留了，效果非常令人满意。

综上所述，可以看出中值滤波器不论从处理效果上，还是时间复杂度上都克服了前一种方法的缺点，因为中值滤波器只是对序列的统计和排序，具有算法简单，计算速度较快的特点，而且中值滤波器对于某些输入信号的不变性能够保证信号的不失真。但是同时我们也可以看到，随着 W 的增大，该滤波器的效果越好，计算量也越大，假如 W 最后等于采样频率，那么算法的时间复杂度会陡然增加至 $O(n^2 lgn)$。

3. 自适应模板滤波器

文献[12]提出了自适应模板法，把一个干扰信号周期内所采的信号作为一个 L 维向量。假设 $\vec{X}(n)$ 为观察信号，$\vec{S}(n)$ 为有用信号，$\vec{N}(n)$ 为干扰信号，则

$$\vec{X}(n)=\vec{S}(n)+\vec{N}(n) \tag{4.5}$$

定义模板信号为：

$$\vec{M}(n)=\frac{1}{M}\sum_{i=0}^{M-1}\vec{X}(n-i)=\frac{1}{M}\sum_{i=0}^{M-1}\vec{S}(n-i)+\frac{1}{M}\sum_{i=0}^{M-1}\vec{N}(n-i) \tag{4.6}$$

由于$\overline{N}(n)$未周期信号，若$\overline{S}(n)$为零均值信号(实际上，在提取大多数生理电信号时，都采用交流放大器，如心电图机中的时间常数电路，所以这些信号都可以视为零均值信号)，当M足够大时

$$\frac{1}{M}\sum_{i=0}^{M-1}\overline{S}(n-i)=\overline{O} \tag{4.7}$$

而

$$\frac{1}{M}\sum_{i=0}^{M-1}\overline{N}(n-i)=\overline{N}(n) \tag{4.8}$$

所以，只要从原始输入信号中减去模板信号，而无需像常规的自适应滤波器那样需要参考信号和一大套复杂的算法就能达到滤除工频干扰的目的，即：

$$\overline{S}(n)=\overline{X}(n)-\frac{1}{M}\sum_{i=0}^{M-1}\overline{X}(n-i) \tag{4.9}$$

对上式两端取Z变换，得到：

$$H(z)=1-\frac{1-z^{-M}}{1-z^{-1}}=1-\frac{z^{M}-1}{z^{M}-z^{M-1}} \tag{4.10}$$

定义

$$H'(z)=\frac{1-z^{-M}}{1-z^{-1}} \tag{4.11}$$

则式4.10可写为

$$H(z)=1-H'(z) \tag{4.12}$$

为了进一步简化问题，我们首先考虑$H'(z)$的频率特性，其相应的网络结构如下所示：

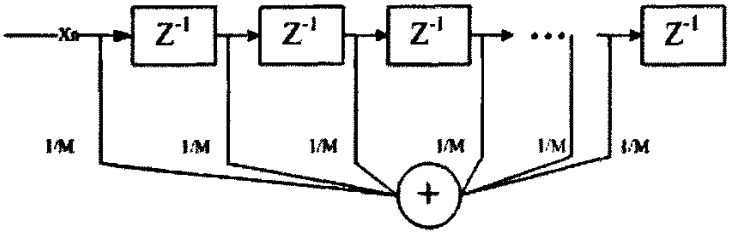

图 4.8 滤波网络结构图

$H'(z)$的零点为

$$z_k=e^{j\frac{2\pi k}{M}},k=0,1,\cdots,M-1$$

这些零点分布在|Z|=1 的圆周上，并对圆周进行 M 等分，特别是它的第一个零点 k=0，这正好与式 4.10 的分母上的(z-1)的极点相抵消，因此整个 $H'(z)$ 共有(M-1)个零点，$z_k = e^{j\frac{2\pi k}{M}}$；(M-1)个极点，都集中在原点处。

M=8 时的零极点分布如图 4.8 所示，M=8，M=64 的幅频特性如图 4.9 所示。由以上的推导可以看出，虽然 $H'(z)$ 有 M-1 个零点，但无论 M 取值如何，$H'(z)$ 的幅频特性相当于一个低通，在 $w=0, w=2\pi$ 处总是通带，前面我们以一个干扰信号周期内的采样点作为一个 L 维向量，对应此 L 维向量的采样频率为 50HZ，所以

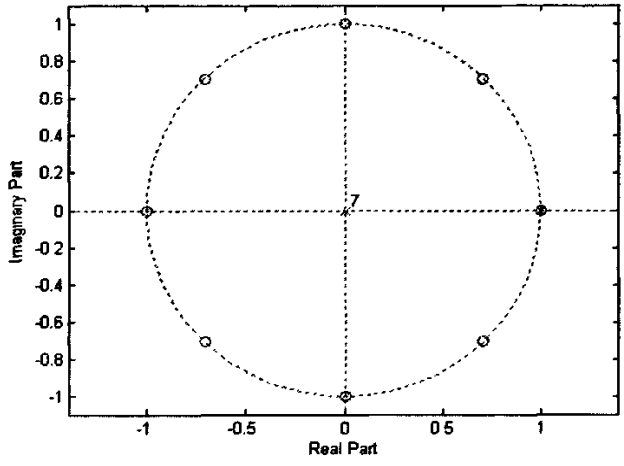

图 4.9　H'(z)零极点图

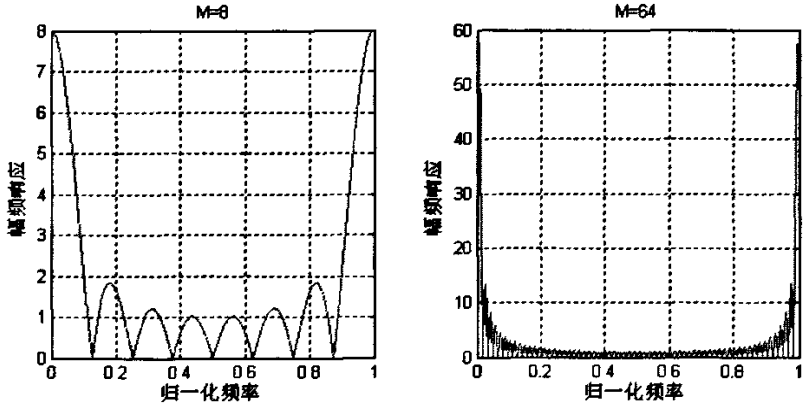

图 4.10　M=8 与 M=64 时的 H'(z)幅频特性

$w=0, w=2\pi$ 两处分别对应 0Hz，50Hz 的实际频率，无论 M 取值如何，都可以抑制 50HZ 和 0Hz 的频率成分。

根据 $H'(z)$ 的零极点可以得出，当 M 越大时，其零点在圆周上分布越紧密，$H'(z)$ 的幅频特性的次级峰值越矮，当 M 特性趋于无穷时，仅仅通过 50HZ 与直流

分量，相当于理想低通，而相应的 $H(z)$ 此时为高通，仅仅滤除 50Hz 与直流分量。文献中[15]中将该算法应用于抑制 50HZ 的工频干扰，并把 M 取为 128、256、512，因此，其滤波器特性接近于理想高通，对于滤除 50Hz 有较好的使用效果，但是在实际应用中，但是该算法仅仅对 50HZ 的采样频率是有效的，当采样频率大于 50Hz 就无法过滤掉工频干扰。根据以上的推导，基线漂移的频率，根据 $H'(z)$ 的零极点，可以非常容易地估算其截至频率，取第一个零点，所对应的频率为截至频率，则

$$f_0 = \frac{50}{M}$$

对于 0.5Hz 的极限漂移，选择 M 为 100 就可以满足抑制基线漂移的要求。总之，选择适当的 M 值，既可以节省存储空间，提高响应速度，而且能够同时消除基线漂移。在实验中发现当 M=50 时，有较好的滤波的效果，也就是说在我们的实验数据中基线漂移主要是 0~1.0hz，又因为基线漂移在 0.05~1.2Hz 范围内，所以这是可能的。当 M=50 时的滤波效果如如 4.11 所示。可以发现滤波后的基线基本拉直了，虽然没有达到前面的滑动窗口中值滤波的效果，但是这种滤波算法在执行效率上去远远优于前面的两种算法。

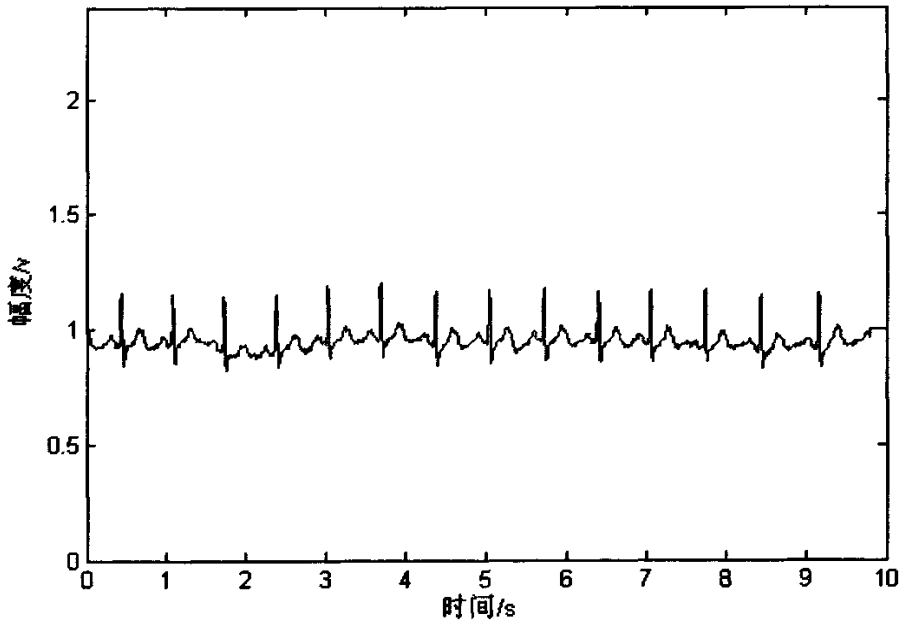

图 4.11 自适应模板滤波算法结果 M=50

4.2.2 去除 50Hz 工频干扰

本课题中，去除工频干扰的滤波算法采用平滑滤波器。下面给出滤波前后的效果：

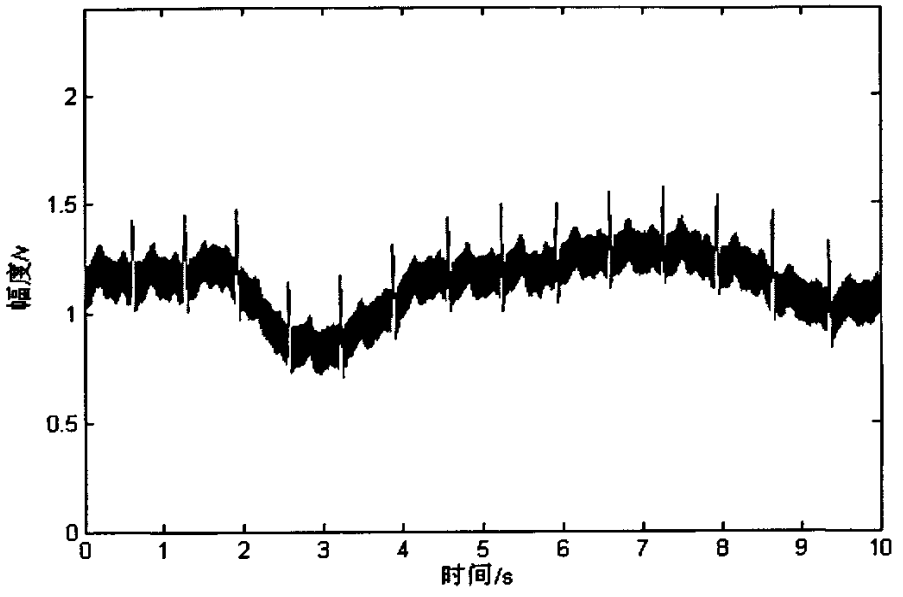

图 4.12 带有 50HZ 工频干扰的心电波形

100 图 4.13 平滑滤波结果

当采样频率为 250Hz 时，使用如下的频率响应函数：

$$H(z)=\frac{1}{5}(1+z^{-1}+z^{-2}+z^{-3}+z^{-4}) \tag{4.13}$$

其对应的差分方程为：

$$y(n)=\frac{1}{5}(x(n)+x(n-1)+x(n-2)+x(n-3)+x(n-4)) \tag{4.14}$$

其频率特性如下所示：

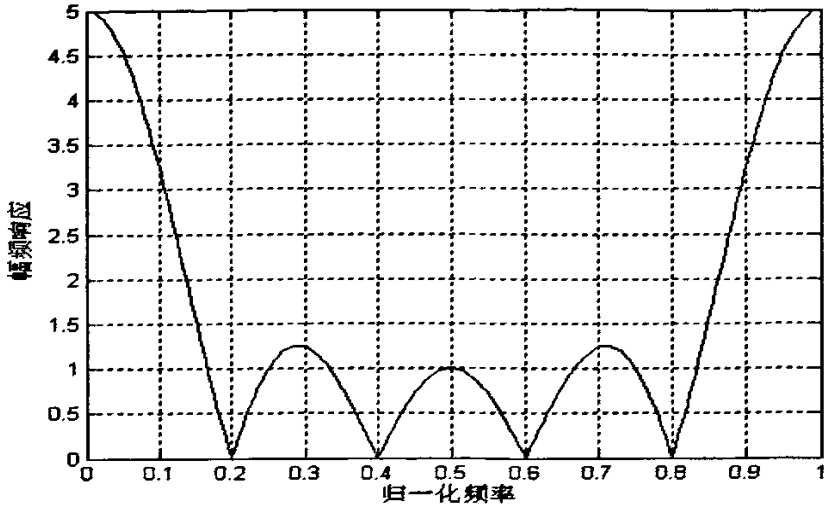

图 4.14 平滑滤波器频率特性

该滤波算法简单，而且处理速度快，但是从上面的幅频特性可以看出，这个滤波器其实具有低通特性，通频带很窄，对心电图的 QRS 有较大的削峰，信号衰减比较大，但是从实际的实验效果来看，该算法对于 50Hz 的正弦波工频干扰还是很有效的，但是对于非线性的 50Hz 正弦波干扰，仍然需要进一步的探索。

虽然该算法有以上的几点不足，但是考虑到其他的算法，比如简单整系数带阻滤波器存在很大的延时、FIR 滤波器需要很高的阶数等等；另外，由于本课题中的数据采集硬件上，已经对工频干扰进行了部分隔离；所以综合考虑，还是使用了这种简单的平滑滤波器。

5 总结

本文研究的是一种小型化的标准 12 导联心电图仪，它采用了 Windows CE 5.0 操作系统和优化设计的心电图滤波算法，具有体积小，功耗低，实时处理能力强等优点。论文中分别对该心电图仪的操作系统构建和实时滤波算法的设计做了详细的论述，其中重点分析了系统中 Boot Loader 设计与开发，驱动程序开发和 OAL 层的开发以及算法的优化设计。

本课题所研究的小型化心电图仪的主要目标是克服传统的基于计算机模式的心电图仪体积大，功耗高，价格昂贵等缺点并且具有便携式心电图仪的小型化，功耗低的特点。本系统采用了 S3C2410A 这款高性能、低功耗的 ARM 微处理器作为系统的核心，配以 Windows CE 操作系统和高效率的实时心电图滤波算法，完全能够应付 12 导联心电图数据实时处理的需要。而且系统具有很好的可扩展性，能够很容易的实现蓝牙无线打印，局域网数据传输等功能。

由于时间和和精力的关系，本课题未能实现一个完整的心电图仪。本课题目前已经完成了心电图仪的核心部分的开发，为后续的工作打下了基础，但是还需要后续的一些研究与开发工作，展望如下：

(1) 开发数据显示的应用程序，并且移植到 CE 系统

(2) 开发手写输入驱动，便于用户输入信息

(3) 从计算机语言的角度，对算法进行进一步的优化

(4) 研究心电图信号的其他滤波算法，增加心电图仪数据分析的功能

(5) 完善和扩展已有的驱动程序，增强系统稳定性和扩展系统功能

总之，目前的工作已经为整个课题的进一步深入发展打下了良好的基础，但是离真正成熟的，可商业化的小型化心电图仪还有一定的距离，这些将在以后的研究中需要进一步探讨和分析，并且最终实现目标。

参考文献

[1] 郭继鸿，张萍. 动态心电图学[M]. 北京：人民卫生出版社，2003

[2] 宁新宝. 生物医学电子学[M]. 长沙：湖南科学技术出版社，1985

[3] 陈连坤. 嵌入式系统的设计与开发[M]. 北京：清华大学出版社，北京交通大学出版社，2005

[4] 奥尔斯电子科技有限公司. OURS2410 产品说明. www.ourselec.com

[5] SAMSUNG Electronics. S3C2410X User's Manual.rev1.2 . www.samsungsemi.com

[6] SAMSUNG Electronics . K4S561632C Data Sheet rev04 . www.samsungsemi.com

[7] SAMSUNG Electronics . K9F1208x0b Data Sheet rev03 . www.samsungsemi.com

[8] SHARP Microelectronics. LQ035Q7DB02 TFT LCD Module. www.sharpsma.com

[9] Willis J. TompKins . 林家瑞 徐邦荃 等译.武汉，华中科技大学出版社， 2001.11

[10] 胡广书. 数字信号处理——理论、算法与实现(第二版).北京，清华大学出版社，2003.8

[11] Richard G. Lyons . 数字信号处理.北京，机械工业出版社，2005.1

[12] 何宗健. Windows CE 嵌入式系统. 北京，北京航天航空大学出版社，2006.9

[13] Sanjit K.Mitra. 数字信号处理—基于计算机的方案.北京，清华大学出版社，2006.10

[14] 张冬泉，谭南林，王雪梅等.Windows CE 实用开发技术. 北京，电子工业出版社，2006.4

[15] 李刚，刘巍，虞启琏，马建英，于学敏.抑制工频干扰及基线漂移的快速算法. 中国生物医学工程学报.2000 年 3 月.第 19 卷第 1 期.

[16] 王林泓，杨浩. 心电信号中滤波器设计的研究. 北京生物医学工程.2002 年 9 月.第 21 卷第 3 期.

[17] 蔡坤，陆尧胜. 基于中值滤波的心电基线矫正方法的研究.医疗设备信息.2004 年 2 月.第 19 卷第 2 期.

[18] 邢国泉. 自适应滤波器在 ECG 信号除噪中的应用. 北京生物医学工程. 2005 年 6 月. 第 24 卷第 3 期.

[19] 张跃，周炳坤，刘文煌. 医用 PDA 心电实时监护软件系统的设计与实现. 计算机应用. 2006 年 12 月.第 26 卷.

[20] 王镇，蔡萍. 用于去除心电信号中工频干扰数字滤波技术. 电子测量技术.2000 年.第 2 期

附录 A

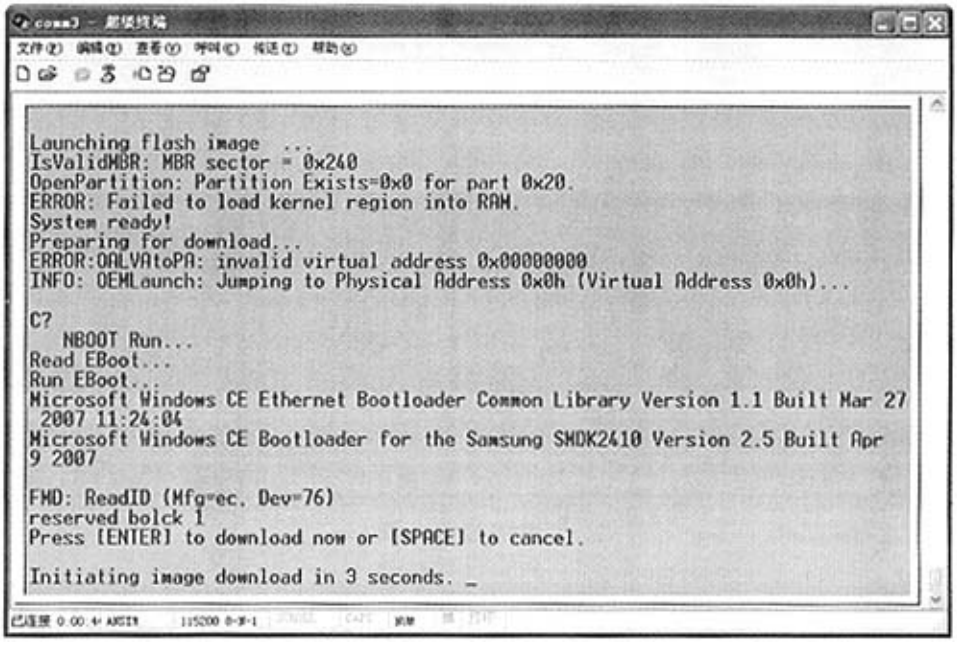

图 1 等待下载内核映像

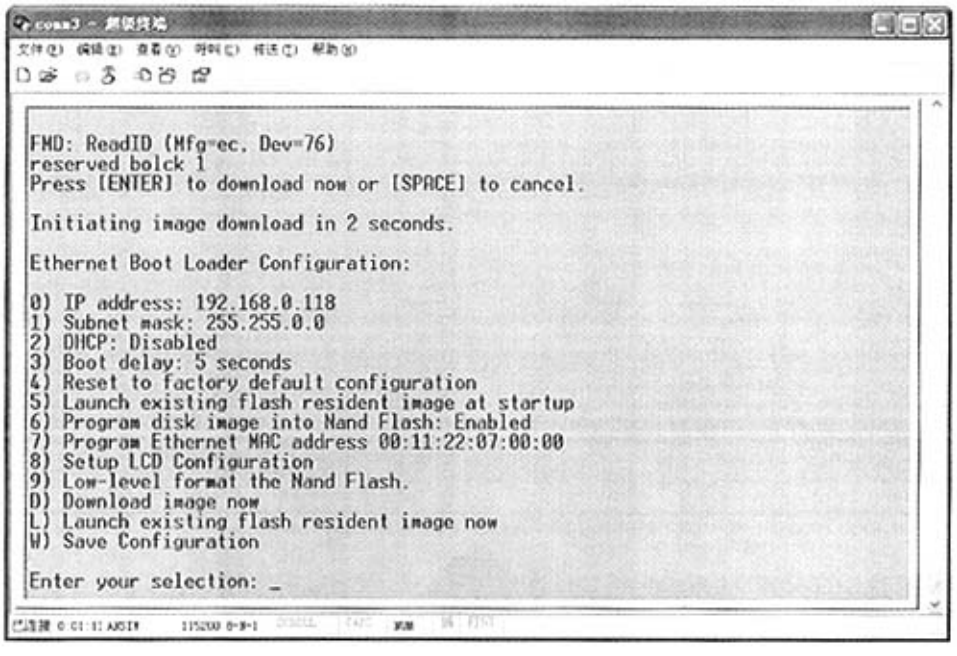

图 2 Eboot 参数设置选项

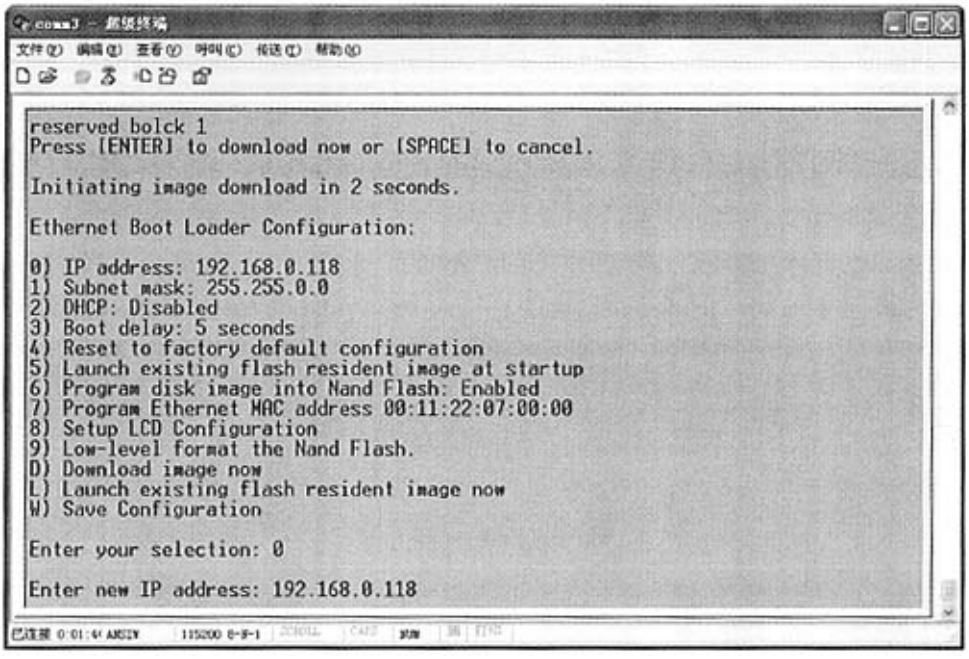

图 3 设置 IP 地址

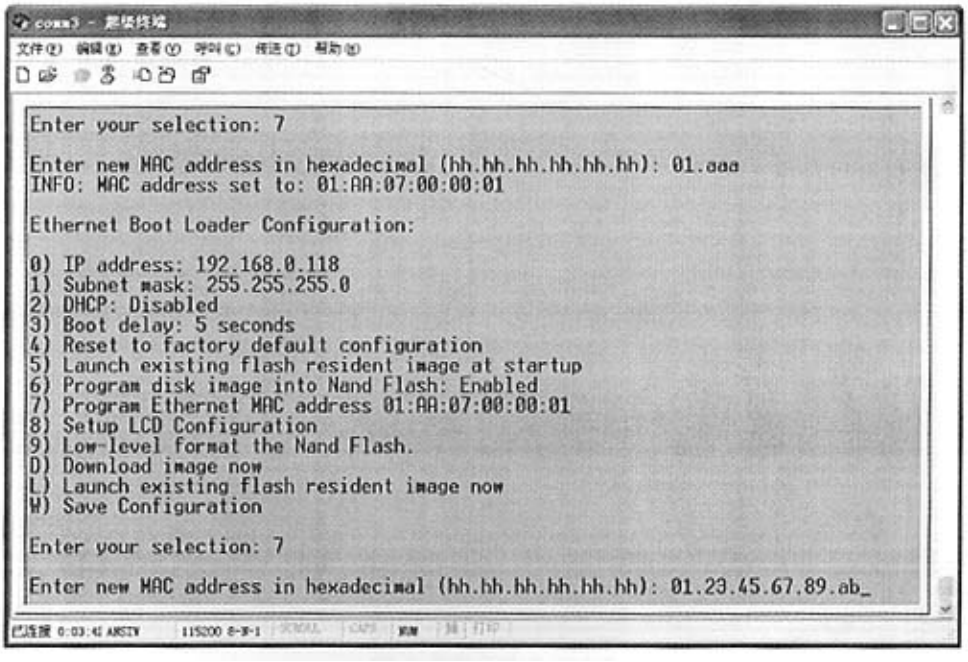

图 4 设置 MAC 地址

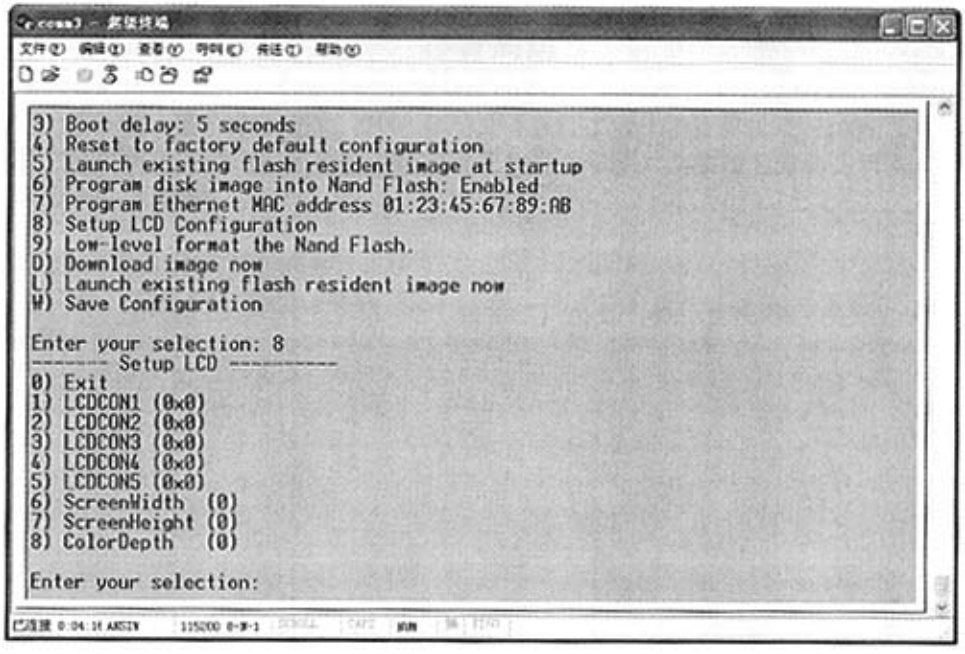

图 5 设置显示器参数

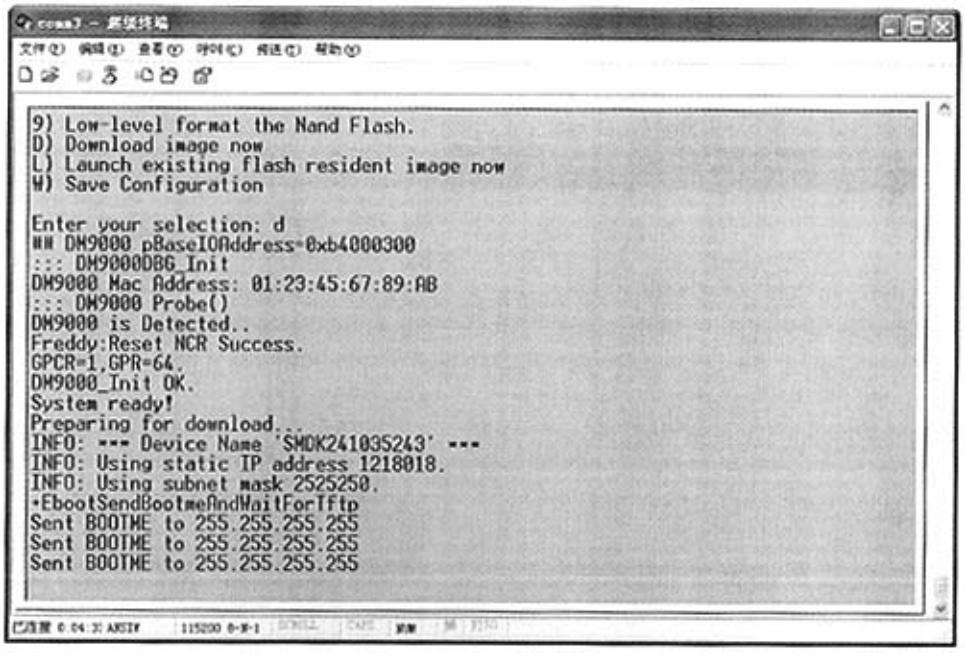

图 6 等待下载内核映像

作者简历

张小栋，男，1981 年生，2004 年毕业于浙江工业大学计算机系，2005 考入北京交通大学计算机与信息技术学院攻读工学硕士学位，主要研究方向为嵌入式系统的应用与开发。

攻读硕士学位期间发表的论文：
张小栋， 周洪利. 《ARM 平台 Boot Loader 的设计》现代计算机(专业版) 2007 年第 5 期