中文摘要

摘要：比例-积分-微分(PID)是过程控制中最常用的一种控制算法。算法简单而且容易理解，应用十分广泛。但由于应用领域的不同，功能上差别很大，系统的控制要求及关心的控制对象也不相同。数字 PID 控制比连续 PID 控制更为优越，因为计算机程序的灵活性，很容易克服连续 PID 控制中存在的问题，经修正而得到更完善的数字 PID 算法。本文以三相全控整流桥阻性负载为实际电路，控制主电路电压，旨在提出一种智能数字 PID 控制系统的设计思路，并给出了详细的硬件设计及初步软件设计思路。

PID 控制系统采用高性能、低功耗的 ARM 微处理器 S3C44B0 作为核心处理单元，内部的 10 位 ADC 作为信号采集模块，采用了矩阵键盘和 640*480 的液晶作为人机接口；串口作为通信模块实现了上位机的监控。采用芯片内部自带的 PWM 模块，输出 16M Hz PWM 信号并经过一阶低通滤波器得到 0～5V 的控制信号用于触发主电路控制器，实现 PID 整定。

软件方面，分析和研究了 uC/OS II 的内核源码，实现了其在 32 位微处理器上的移植，作为管理各个子程序执行的系统软件。选用了图形处理软件 uC/GUI 用于完成 LCD 显示及控制。PID 算法采用了增量式数字 PID 算法，采用规一化算法进行参数选取。上位机部分采用了 C#语言进行编写。另外，采用了 RTC（Real Time Clock）作为系统时钟，可以实现系统的定时运行、定时模式切换等。在上位机上也可以方便的控制程序的执行，实现远程监控。

在论文的最后详细的介绍了智能 PID 控制系统在三相全控桥主电路中的具体应用。总结了调试中遇到的问题，对今后工作中需要进一步改善和探索的地方进行了展望。

关键词：PID 控制;ARM; uC/OS

分类号：TP216＋.1; TM934.73; TM932

ABSTRACT

ABSTRACT: The PID control is the most common control algorithm. Because of its relatively simple structure which can easily be understood and implemented in practice ,is widely applied in industry. However,there are many different applications with different performance requirements, and the controlled object .The purpose of this paper is to present the design of a intelligent PID control system, detailed information of both hardware and software is contained .And it is applied in three-phase full bridge circuit and proved excellent performance, and there is a large space of further application.

The hardware of the intelligent PID controller uses the ARM core S3C44B0X as the CPU, I/O module contains 4*4 matrix keyboards and 640*480 LCD and UART, the signal is sampled by the internal 10 bits ADC module of the CPU. The PWM is used as the output function, 0~5v control signal is obtained by low pass filter and it is connected to the trigger module of the main circuit.

The embedded real time operating kernel uC/OS II is adopted as the main part of the system software, the GUI software package is used to manage input and output, the incremental PID algorithm is selected for calculation. The uC/OS II manages the running of each task, and uC/GUI is responsible for LCD display and keyboards scanning. The PC end software is designed using C#. RTC is adopted as system clock, tasks can start at specific time or shift mode, the PC end software can be used to monitor the state of the hardware system.

In the last ,this paper discuss the application of this system in three-phase full bridge circuit is presented with detailed debugging trouble shooting.

KEYWORDS: PID control ; ARM ; uC/OS

CLASSNO: TP216+.1; TM934.73; TM932

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：[signature]　　签字日期：2009 年 6 月 26 日

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：[illegible]　　导师签名：[illegible]

签字日期：2009 年 6 月 26 日　　签字日期：09 年 6 月 26 日

致谢

本论文的工作是在我的导师姜学东教授的悉心指导下完成的，姜学东教授严谨的治学态度和科学的工作方法给了我极大的帮助和影响。在此衷心感谢姜学东老师对我的关心和指导。

姜学东教授悉心指导我们完成了实验室的科研工作，在学习上和生活上都给予了我很大的关心和帮助，对于我的科研工作和论文都提出了许多的宝贵意见。在此向姜学东老师表示衷心的谢意。

在实验室工作及撰写论文期间，邱瑞昌老师，荆龙老师，王爱国、王朝宁等同学对我论文研究工作给予了热情帮助，在此向他们表达我的感激之情。

另外也感谢我的家人，他们的理解和支持使我能够在学校专心完成我的学业。感谢母校给了我学习和成长的机会，

感谢在百忙之中抽出宝贵时间对本论文进行评审的各位专家们，衷心感谢你们对本论文提出的宝贵意见。谢谢！

1 绪论

1.1 课题背景与意义

PID 控制算法是最通用的控制策略，在工业过程控制中 95%以上的控制回路具有 PID 结构，算法简单。传统的 PID 整定方法是在获取对象数学模型的基础上，根据某一整定原则来确定 PID 参数，只要对象数学模型精确，且是非时变的，其整定的参数可以固定不变，控制效果一般能满足要求。然而在实际的过程控制系统中，许多被控过程机理较复杂，具有严重非线性，时变不确定性、大延迟等特点，在噪声，负载扰动等因素的影响下，过程参数，甚至模型结构，均会发生变化，难以建立精确的数学模型。因此，若采用常规 PID 参数整定方法，往往整定不良，性能欠佳，对运行工况的适应性较差，控制效果不理想。因此，针对那些非线性，大时变，大延迟等被控对象，不仅要求 PID 参数的整定不依赖于对象数学模型，而且要求 PID 参数能在线调整，以满足控制要求。所以设计一种智能的 PID 控制系统，以适应复杂工况和高性能的控制要求，具有十分重要的意义。

数字 PID 技术是针对模拟 PID 的缺点而提出的新的控制策略，它主要通过给定值及采集的反馈信号计算出输出控制量的值，以实现闭环控制。与模拟 PID 控制相比，数字 PID 具有参数灵活，工作可靠等优点，但是要实现非常精确的控制，就需要处理器具有较快的速度，强大的计算能力，并且要求反馈信号采集及控制信号输出具有较高的分辨率。

ARM 是一种典型的嵌入式微控制器，性能高、成本低、功耗小，适合于多种领域。ARM 结构是基于精简指令集（RISC）原理而设计的，能实现较高的指令吞吐量，出色的实时中断响应，并且能够支持 Linux、Windows CE 等操作系统。本设计中所采用的 S3C44B0X 是三星公司推出的 ARM7 内核处理器，软件上采用了嵌入式操作系统 uC/OS II，它具有强大的管理功能，并且是一个实时操作系统(RTOS)，可以满足系统高精度和快速响应的要求。移植了 GUI 软件包，实现功能强大的显示。

因此，针对各种工业过程控制的实际需要，寻求适合于实际问题的 PID 参数整定算法，然后利用合适的微处理器实现，提出一种智能 PID 控制系统解决方案，可以应用于多种场合，具用重要的理论意义和使用价值。

本次设计的控制对象为单变量系统，以三相全控桥式整流电路为实际应用，负载为电阻箱，控制主电路电压。智能控制系统采集回路电压，经运算输出 PWM

波，滤波出 0-5v 的直流量控制 6100 驱动板，控制输出脉冲的移向范围从 5 度到 175 度可调，实现三相整流桥输出电压控制。采样时间为 0.5ms，PWM 波频率 16Mhz，主电路电压控制精度 2.5v，设定值不要超出主电路输出最大值即可。超调量 10%—11%，调节时间 8s—15s。控制系统具备计时器和通信功能，可以保存整定参数，LCD 屏幕显示，键盘式操作。

1.2 PID 控制系统设计思路

1.2.1 系统的硬件架构

从 20 世纪 70 年代单片机出现到今天各式各样的嵌入式微处理器、微控制器的大规模应用，嵌入式系统已经有了近 30 年的发展历史。本次设计采用了标准的嵌入式系统的设计模式。嵌入式系统是一种基于微控制器、软件驱动的可靠实时控制系统，以人工的、自治的或者网络的方式进行交互，对各种物理变量进行操作。一般而言，嵌入式系统的构架如图 1-1。

硬件设计主要包括搭建基于 S3C44B0X 的最小系统，另一方面就是实现设计功能所需要的各种功能模块的硬件设计。完成了硬件设计之后，就可以进行编程，实现设计的功能了。

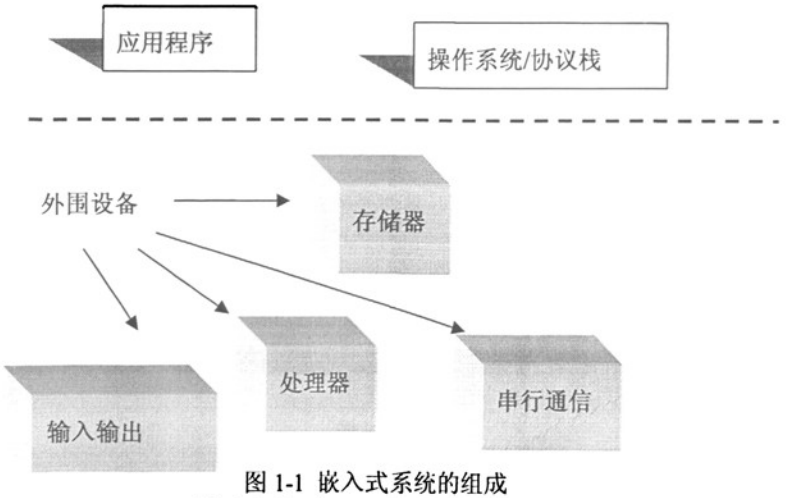


图 1-1 嵌入式系统的组成
Fig 1-1 Consistent of embedded system

1.2.2 系统的软件设计

操作系统将应用分解成多个任务，简化了应用系统软件的设计；每个任务都

有独立的存储器分配，在控制多个设备的同时，执行多个函数的调度，任务之间可以共享存储器。操作系统最主要的功能是任务之间的调度，内部进程（任务）通信和对变量、队列和管道的共享，确保同一瞬间只有一个任务能够进入运行状态。RTOS 使控制系统的实时性得到保证，可以接近理论上能达到的最好水平；良好的多任务设计，有助于提高系统的稳定性和可靠性。较常用的有 Vxworks、WindosCE、uCOS、Linux 等。之所以选择 uCOS，因为对于小型实时操作系统来说，源代码公开的、具有很好的可移植性、可固化可裁剪、高稳定性与可靠性、抢占式多任务的 uCOS 非常适合学习。而且已应用在很多工业及军事领域，值得放心。

uC/GUI 可以极大的简化应用程序的图形设计，uC/GUI 是一种嵌入式应用中的图形支持系统。用于为任何使用 LCD 图形显示的应用提供高效的独立于处理器及 LCD 控制器的图形用户接口,它适用单任务或是多任务系统环境, 并适用于任意 LCD 控制器和 CPU 下任何尺寸的真实显示或虚拟显示。

软件设计主要包括了 uC/OS-II 的移植以及基于 uC/OS-II 实时操作系统内核的程序设计，编程主要采用 C 语言。 uC/OS-II 多任务操作系统加上基于 ARM7 的硬件系统的强大功能，可以使得设计具有非常好的应用效果。设计中，包含了 IIC 读写任务（用于将数据保存在非易失性存储器）、串口通信任务（用于 PC 机管理，实现远程监控）、GUI 任务（用于显示数据以及人机交互），多任务之间的通信采用了 uC/OS-II 中的邮箱机制，方便的实现了数据交换。

控制策略采用了单闭环增量式数字 PID，PID 参数整定方法采用了简易工程法：规一化参数整定法。它是 Roberts P D 在 1974 年提出的一种简化扩充临界比例度整定法，该方法只需整定一个参数，故称其为规一化参数整定法。

1.3 本文的主要研究内容

如前所述，PID 控制在工业过程中得到广泛的应用。然而，实际应用中，许多工业控制回路的整定效果都较差。因此寻求一个适应于多个场合的 PID 控制系统，实现 PID 参数整定具有重要的实际意义。归纳起来，论文做了以下工作。

1. 基于 ARM 的智能控制系统的设计。论文第二章、第三章详细介绍了基于 ARM 的智能控制系统的设计。该系统根据对运算速度、功耗、存储容量等方面的要求，以高性能、低功耗的 ARM 微处理器作为核心处理单元，具有液晶显示、键盘输入等功能，针对被控对象，可以选择不同的整定方法进行 PID 参数的调节。

2. 对实时操作系统的研究以及 uC/OS 系统的移植。论文第三章详细分析了 uC/OS 系统内核及其功能实现，针对智能控制系统模块化、通用化的要求，选择

了实时操作系统 uC/OS，并成功的移植到了以 ARM 为内核的系统中。为实现控制系统的多任务自动调度以及以优先级来判断任务先后顺序的智能化打下了坚实的基础。

3. PID 算法的研究，经过多种算法的比较选定了单闭环增量式数字 PID。并以超调量和稳定时间为参数，提出了自动选定最优系数算法的软件设计思路。使得系统能够自动寻找最优 PID 控制参数。实现智能控制系统自整定控制的初步完成。

2 PID 控制系统算法

2.1 PID 控制原理

2.1.1 原理

在模拟控制系统中，根据偏差的比例（P）、积分（I）、微分（D）进行控制（简称 PID 控制），是控制系统中应用最为广泛的一种控制规律。

系统由模拟控制器和被控对象组成。PID 控制器是一种线性控制器，它根据给定值 r(t)和实际输出值 y(t)构成控制偏差，其控制规律为：

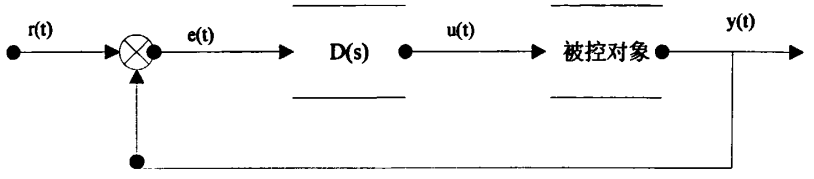


图 2-1 模拟 PID 控制系统
Fig 2-1 Anolog PID control system

$$e(t) = r(t) - y(t) \tag{2-1}$$

$$u(t) = K_p[e(t) + 1/T_I \int_0^t e(t)dt + T_D \frac{de(t)}{dt}] \tag{2-2}$$

或写成传递函数形式：

$$D(s) = \frac{U(s)}{E(s)} = K_P(1 + 1/T_I + T_D s) \qquad (4\text{-}3) \tag{2-3}$$

式中 Kp —比例系数 ； Ti —积分时间常数 ；Td —微分时间常数。

简单来说，PID 控制器各校正环节的作用如下：

1. 比例环节及时成比例的反映控制系统的偏差信号 e(t),代表了变量随时间的变化率，偏差一旦产正，控制器立即产生控制作用，以减少偏差。微分环节有助于减小超调，克服振荡。他加快了系统的动作速度，减小了调整时间。从而改善了系统的动态性能。Kp 的加大会引起系统的不稳定。

2. 积分环节主要用于消除静差，提高系统的无差度。积分作用的强弱取决于积分时间常数 Ti，Ti 越大，积分作用越弱，反之则越强。积分作用太强会使系统超调加大，甚至使系统出现振荡。增大 Ti 会减慢消除静差的过程，但是可以减小超调，提高稳定性。

3. 微分环节能反映偏差信号的变化趋势（变化速率），并能在偏差信号变得太大之前，在系统中引入一个有效的早期修正信号，从而加快系统的动作速度，减少调节时间。能对变量的变化趋势作出响应。有助于减小超调，客服振荡，使系统趋于稳定。他加快了系统的动作速度，减小了调整时间，从而改善了系统的动态性能。过大的 Td 会使控制系统不稳定，一般小于 1。

2.1.2 PID 参数整定概述

PID 控制系统，由于各个控制器参数有明显的物理意义，调整方便，所以其参数整定的方法很多，发展很快。随着控制理论的发展，出现了各种分支，如专家系统、模糊逻辑、神经网络、灰色系统理论等，派生出各种智能控制器，产生了新的 PID 参数整定方法。因此，智能 PID 控制系统以其参数整定方法是目前乃至今后的研究重点 。

PID 参数整定是指在控制器形式已经确定的情况下（P，PI 和 PID），针对一定的控制对象调整控制器参数（Kp，Ti，Td），从而达到控制要求。基本有四类整定方法：第一类是基于过程参数辨识的整定方法，即参数自适应 PID 控制器整定方法；第二类是基于被控过程的特征参数整定方法，又称为非参数自适应 PID 控制器整定方法；第三类是基于最优 PID 控制器设计参数整定方法；第四类是基于控制器本身偏差和偏差变化率的智能控制器参数整定方法。

第一种方法需要通过某种辨识算法获得对象数学模型，然后根据其模型参数进行 PID 控制器参数整定。此方法在得到对象精确数学模型后，可采用解析方法确定 PID 控制器参数，而且控制效果好。但在工业上应用较为困难。

第二种方法是基于被控过程的某些特征参数的整定方法。其设计思路是对于难以通过辨识方法建立精确数学模型的复杂过程，通过实验方法测得其阶跃响应、频率响应等于开环输出特性，从中找到反映对象特性的特征参数，然后根据其特征参数进行 PID 参数整定。这种方法在实际中被广为利用。

第三种方法是基于最优 PID 控制器设计的参数整定方法。在不同准则下，提出 PID 控制最优整定的算法，在这方面，Zhuang 和 Atherton 提出了最优准则的一般形式：

$$J_n(\theta)=\int_0^{\infty}\left[t^n e(\theta,t)\right]^2 dt \qquad (2\text{-}4)$$

第四种方法是基于控制器本身偏差及其变化率的智能 PID 控制器参数整定方法。它是在常规 PID 控制策略中融入智能控制，能模拟人脑的思维，根据专家和操作者的经验对 PID 参数进行在线自整定，根据偏差和偏差变化率的大小，经过推理计算出 PID 参数，从而获得最佳控制策略。

本次设计采用的是常用的 Z-N 法发展而来的，常规 Ziegler-Nichols 整定方法是 Ziegler 和 Nichols 于 1942 年提出的，基于受控过程的开环动态响应中的某些特征参数而进行的 PID 参数整定。其整定经验公式是基于带有延迟的一切惯性模型提出的,这种对象模型可表示为：

$$G(s)=\frac{K}{Ts+1}e^{-Ls}$$

其中 K-放大系数； T-惯性时间； L-延迟时间。

实际控制过程中，将大量的模型近似为上述一阶模型，也可以通过实验获取临界震荡增益 Kc 和临界振荡角频率 Wc，通过经验整定公式得到 PID 参数

控制器 \ 参数	Kp	Ti	Td
P	0.5Kc		
PI	0.4Kc	0.8Tc	
PID	0.6Kc	0.5Tc	0.12Tc

表 2-1 Ziegler-Nichols 整定公式
Fig 2-1 Ziegler-Nichols Modular formula

在连续控制系统中，比较常用的还是简易工程法，这种方法最大的优点在于整定参数时不必依赖被控对象的数字模型。虽然粗糙一点，但是简单易行，适于现场应用。

PID 参数的整定过程可以是离线进行也可以是在线进行。离线整定过程是通过实验方法测出系统的特征参数，然后根据这些参数设计一个合适的 PID 控制器，最后再将此控制器应用到原系统的控制中。当系统的参数发生变化是，则要再重复这一过程。在线整定即自整定的基本思想，是系统中设置两种模态；测试模态和调节模态。测试模态在线测定系统特征参数，并算出 PID 参数；调节模态由 PID 控制器对系统动态进行调节，如果系统测试发生变化，则重新进入测试模态进行测试，测试完成后再回到调节模态进行控制。

2.1.3 数字 PID

在选择数字 PID 参数之前，首先应该确定控制器结构。对允许有静差（或稳态误差）的系统，可以适当选择 P 或 PD 控制器，使稳态误差在允许的范围内。对必须消除稳态误差的系统，应选择包含积分控制的 PI 或 PID 控制器。一般来说，PI、PID 和 P 控制器应用较多。对于有滞后的对象,往往都加入微分控制。本设计主要专注于嵌入式操作系统及 GUI 软件包在电力电子电路控制领域的试探性研究，因此选取了三相全控整流电路为试验平台，负载为阻性负载，基本符合 PID 控制要求。

控制器结构确定后，即可开始选择参数。参数的选择，要根据受控对象的具体特性和对控制系统的性能要求进行。工程上，一般要求整个闭环系统是稳定的，对给定量的变化能迅速响应并平滑跟踪，超调量小；在不同干扰作用下，能保证被控量在给定值；当环境参数发生变化时，整个系统能保持稳定，等等。这些要求，对控制系统自身性能来说，有些是矛盾的。我们必须满足主要的方面的要求，兼顾其他方面，适当地折衷处理。PID 控制器的参数整定，可以不依赖于受控对象的数学模型。工程上，PID 控制器的参数常常是通过实验来确定，通过试凑，或者通过实验经验公式来确定。

本次设计采用了数字 PID 算法，这不是简单的把模拟 PID 控制规律数字化，而是要进一步的与计算机的逻辑判断功能相结合，是 PID 控制更加灵活，更能满足生产过程提出的要求。根据模拟 PID 控制规律，数字 PID 常用的有两种方式：位置型控制算法和增量型控制算法。位置型算法适用于控制调节阀之类表征执行机构的位置的控制对象。如果输出的控制量是相对于上次控制量的增加或减少，则增量式控制算法更为适用。PID 参数的整定本次设计则是选用了在连续控制系统中常用的简易工程法，由经典的频率法简化而来。其中又可以分为扩充临界比例度法和扩充响应曲线法。本设计选用的是一种简化的扩充临界比例度法。实验证明，数字 PID 算法简单，可靠，能够出色的完成任务。

2.2 PID 算法及参数自整定

2.2.1 数字 PID 参数整定

在电子数字计算机直接数字控制系统中，PID 控制器是通过计算机 PID 控制算法程序实现的。计算机直接数字控制系统大多数是采样-数据控制系统。进入计算机的连续-时间信号,必须经过采样和整量化后，变成数字量，方能进入计算机的存贮器和寄存器，而在数字计算机中的计算和处理，不论是积分还是微分，只能用数值计算去逼近。

当采样周期相当短时，用求和代替积分，用差商代替微商，使 PID 算法离散化，将描述连续-时间 PID 算法的微分方程，变为描述离散-时间 PID 算法的差分方程。

(1) 数字 PID 位置型控制算法

为了便于计算机实现，必须把式 2-2 变换成差分方程，为此可作如下近似

$$\int_0^t e(t)dt \approx \sum_{i=0}^{k} Te(i) \qquad (2\text{-}5)$$

$$\frac{de(t)}{dt} \approx \frac{e(k)-e(k-1)}{T} \tag{2-6}$$

式中，T 为采样周期，k 为采样序号。

由以上可得数字 PID 位置型控制算式为

$$u(k) = K_p[e(k) + \frac{T}{T_I}\sum_{i=0}^{k} e(i) + T_D \frac{e(k)-e(k-1)}{T}] \tag{2-7}$$

式 2-7 是数字 PID 算法的非递推形式，称全量算法。算法中，为了求和，必须将系统偏差的全部过去值 e（k）（k=1，2，3，...，k）都存储起来。这种算法得出控制量的全量输出 u（k），是控制量的绝对数值。在控制系统中，这种控制量确定了执行机构的位置，例如在阀门控制中，这种算法的输出对应了阀门的位置（开度）。所以，将这种算法称为“位置算法”。

(2) 数字 PID 增量型控制算法

由式 2-7 可看出，位置型控制算法不够方便，这是因为要累加偏差 e(k)，不仅要占用较多的存储单元，而且不便于编写程序。当执行机构需要的不是控制量的绝对值，而是控制量的增量（例如去驱动步进电动机）时，需要用 PID 的“增量算法”。

为此可对式 2-7 进行改进。

根据式 2-7 可知 u(k-1)的表达式为：

$$u(k-1) = K_p[e(k-1) + \frac{T}{T_I}\sum_{i=0}^{k-1} e(i) + T_D \frac{e(k-1)-e(k-2)}{T}] \tag{2-8}$$

与式 2-7 相减得数字 PID 增量型控制算式为：

$$\begin{aligned}\Delta u(k) &= u(k) - u(k-1) \\ &= K_p[e(k)-e(k-1)] + K_I e(k) + K_D[e(k)-2e(k-1)+e(k-2)]\end{aligned} \tag{2-9}$$

其中

$K_P = \frac{1}{\delta}$称为比例增益；

$K_I = K_P \frac{T}{T_I}$称为积分系数；

$K_D = K_P \frac{T_D}{T}$称为微分系数；

为了编程方便，可将式 2-9 整理成如下形式

$$\Delta u(k) = q_0 e(k) + q_1 e(k-1) + q_2 e(k-2) \tag{2-10}$$

其中：

$$q_0 = K_P(1+\frac{T}{T_I}+\frac{T_D}{T})$$

$$q_1 = -K_P(1+\frac{2T_D}{T})$$

$$q_2 = K_P\frac{T_D}{T}$$

图 2-3 则是本文 PID 算法的大体流程图，从式 2-10 已看不出是 PID 的表达式了，也看不出 P、I、D 作用的直接关系，只表示了各次误差量对控制作用的影响。从式（2-10）看出，数字增量式 PID 算法，只要贮存最近的三个误差采样值 e（k）、e（k-1）、e（k-2）就足够了。

(3) 图 2-2 介绍了自整定控制算法如何进行调整和操作。算法计算控制信号的输出值，具体实现细节参考了控制设计参考文档，即 John F.Dorsey 编著的《Continuous and Discrete Control Systems 》。

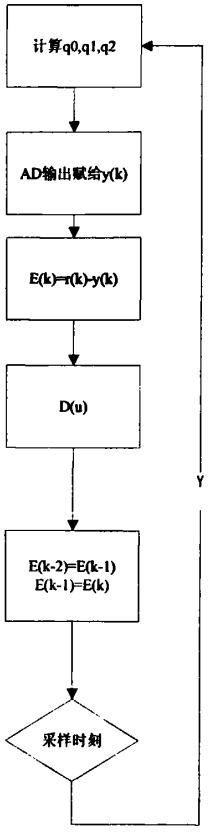


图 2-3 数字 PID 增量型算法的流程图
Fig 2-3 Incremental digital PID flow chart

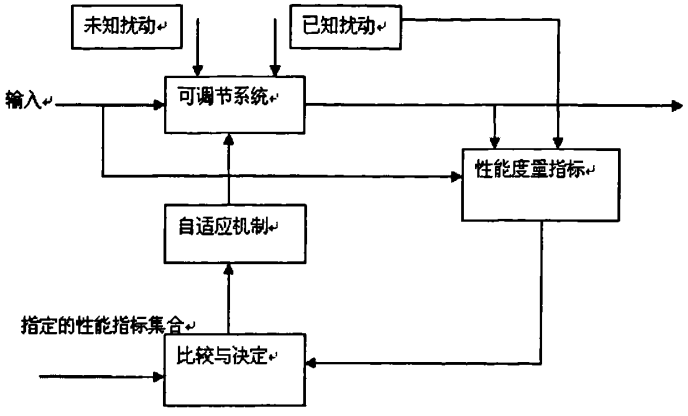


图 2-2 自适应控制算法模型
Fig 2-2 auto self modified control model

2.2.2 数字 PID 的具体实现及改进

1. 归一化参数整定法

本设计采用的参数整定方法为 1974 年 Robert PD 提出的归一化参数整定法，已知增量型 PID 的控制公式为：

$$\Delta u(k) = K_P\{e(k) - e(k-1) + \frac{T}{T_I}e(k) + \frac{T_D}{T}[e(k) - 2e(k-1) + e(k-2)]\}$$

如今令 T=0. 1Tk；TI=0. 5 Tk；TD=0. 125 Tk。式中 Tk 为纯比例作用下的临界振荡周期。则

$$\Delta u(k) = K_P[2.45e(k) - 3.5e(k-1) + 1.25e(k-2)]$$

这样，整个问题便简化为只要整定一个参数 Kp。改变 Kp，观察控制效果，直到满意为止。

2. 采样周期的选择

香农采样定理试图决定了采样周期。根据采样定理，采样周期应满足

$$T \le \frac{\pi}{\omega_{\max}}$$

$\omega_{\max}$ 为被采样信号的上限角频率，$T_{\min}$ 为采样周期的下限也就是计算机执行控制程序和输入输出所耗费的时间，系统的采样周期只能在 $T_{\min}$ 和 $T_{\max}$ 之间。采样周期 T 要合理选取，T 太小增加计算机的负担，无法充分利用计算机的性能；T 太大，两次采样间的偏差变换不够，数字控制器的输出值变换没有明显差异。因此 T 应该在 $T_{\min}$ 和 $T_{\max}$ 之间。

3. 要想在控制性能上超过模拟调节器，要发挥计算机的优势：运算速度，编程灵活，逻辑判断性能好。因此在需要在本文在以下方面做出了改进。

(1) 积分项的改进

在PID控制中，积分的作用是消除残差，为了提高控制性能，对积分项可采取以下4条改进措施

a) 积分分离

当偏差e(k)较大时，选择取消积分作用；当偏差较小时才将积分作用投入，这样做可以避免产生较大的超调和长时间的波动。

b) 抗积分饱和

因长时间出现偏差或偏差较大，计算出的控制量有可能溢出或小于零，而溢出就是指计算机运算得出的控制量超出了D/A转换器所能表示的数值范围。防止积分饱和可以对输出的控制量限幅，同时把积分作用切除掉。

c) 提高运算精度

增加A/D转换位数，运算字长

(2) 微分项的改进

a)不完全微分PID控制算法

标准的PID控制算式，对具有高频扰动的生产过程，微分作用响应过于灵敏，容易引起控制过程的振荡，会降低调节品质。为了克服这一缺点，同时又要使微分作用产生应有的效果，可以在PID控制输出串联一阶惯性环节，这样可使微分作用作用于多个采样周期，且作用变弱，调节作用平缓。

b) 微分先行PID控制算法

微分先行是指只对被控量y(t)微分，不对偏差e(t)微分，也就是说对给定值r(t)无微分作用。这样可以避免给定值升降给控制系统带来的冲击。

而带死区的PID算法是指在计算机控制系统中，某些系统为了避免控制动作过于频繁造成的震荡而采取的一种措施。指定一个范围，当偏差绝对值|e(k)|<=给定小范围是，输出为0，否则按PID算法输出。

4. 自整定算法

(1) 给定希望的设定值r(t),及希望的参数超调量百分比a,稳定时间b以及相应的权重比。

(2) 设定后，kp从0缓慢增加直到输出值达到r(t)的值，kp只有在一系列的输出值被采样后才会增加。当5个连续的输出值达到r(t)时(误差为2%),系统认为达到稳定，记录稳定时间(即相应的横坐标)。

(3) 当产生超调至设定参数时，调节器将以0.05的部长来减少kp值，直到得到平衡态，将输出最大值与设定值之差定为超调量。

(4) 重复以上步骤 50 次，将超调量与调节时间及其相应的权重比相乘得到最后结果，取最佳值并输出。

3 系统的硬件设计

本章主要介绍 PID 控制系统的硬件结构，主要包括最小系统和外围接口电路的设计。选用的控制芯片为三星推出的 32 位控制器 S3C44B0X，采用 ARM7TDMI 内核，具有低功耗、简洁和出色的全静态特色。最高主频可达 66MHz，S3C44B0X 内部提供了丰富的部件，包括：8KB cache，内部 SRAM，LCD 控制器，带握手的 2 通道 UART，4 通道 DMA，外部存储控制器（片选逻辑，FP/EDO/SDRAM 控制器），带有 PWM 功能的 5 通道定时器，I/O 端口，RTC，8 通道 10 位 ADC，I2C 总线接口等。

S3C44B0X 的高效率的内核以及丰富的外设，使其非常适合应用于嵌入式系统，因此本设计中选用了 S3C44B0X 作为主控芯片。

3.1 控制系统的结构及工作原理

本文所设计的 PID 控制系统主要针对 SISO 系统，它的系统机构框图如图 3-1 所示，r(t)是设定值输入，u(t)为控制器输出，也即对象的输入，y(t)是对象输出，PID 控制器为设计的基于 ARM 的智能控制系统。由图看出，系统存在两种固定状态：控制状态和整定状态，可以通过切换开关来转换。当控制效果不理想时，转换到整定状态下进行参数整定，得到新的 PID 参数，将控制器参数重新设置后切换开关到控制状态进行控制。为使参数整定过程对生产过程产生尽可能小的干扰，采用闭环在线整定测试，PID 控制器和被控对象组成一个内部回路，整定时参数整定环节作用于内部回路，避免了开环测试对被控对象平稳运行的影响。

系统主要包括 CPU 核心模块、数据的采集和输出模块、电源管理模块、LCD 显示模块、键盘模块、JTAG 调试模块等。

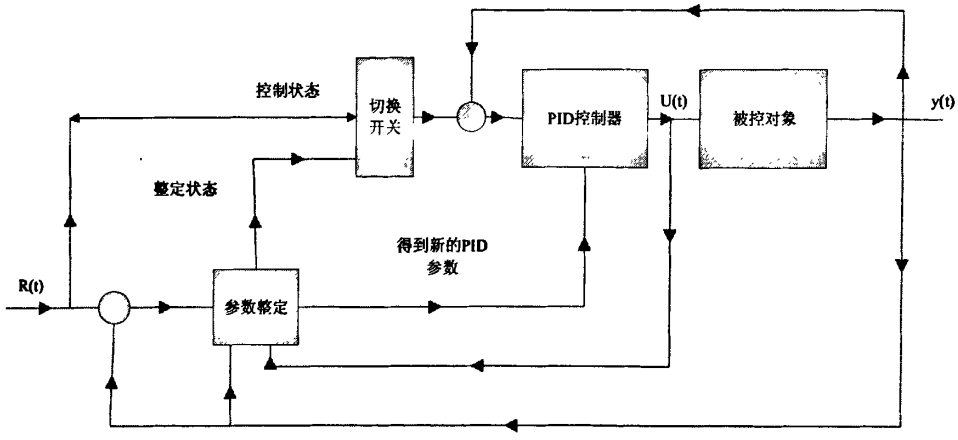


图 3-1 系统结构框图
Fig 3-1 System Architecture Block

3.2 基于 ARM7 内核 S3C44B0X 的最小系统设计

3.2.1 电源电路

最小系统需要提供 5v，3.3v 以及 2.5v 的电压，5v 为驱动电路所需要的电压，3.3v 是 S3C44B0X 的 IO 电压，2.5v 是内核电压。系统外部采用 5V 直流电源供电，3.3V 以及 2.5V 电压由 DC/DC 电路实现，使用低压差线性稳压器 LDO(Low Dropout Regulator)芯片 AMS1117-33 及 AMS1117-25 输出。1117 是一种最简单的电源转换芯片，使用方便、成本低、纹波小、电磁干扰较小，原理图如图 3-2。

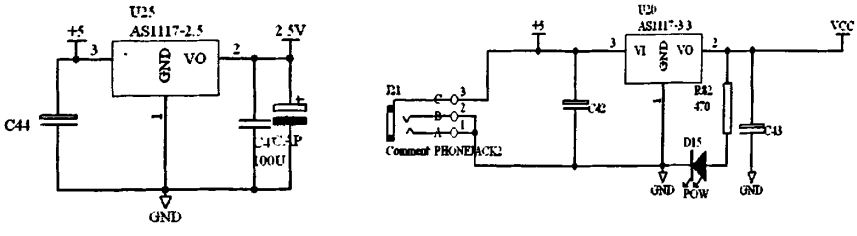


图 3-2 电源部分原理图
Fig 3-2 Schematic diagram of Power Supply

3.2.2 晶振电路及复位电路

时钟控制着 CPU、系统定时器和 CPU 机器周期的各种时钟控制需求。晶振电路用于向 CPU 及其他电路提供工作时钟。本系统中，S3C44B0X 使用 8MHz 无源

晶振，借助于时钟电路才能产生振荡信号，PLL 兼有频率放大和信号提纯的功能，倍频置 64MHz。

复位电路完成系统的上电复位和系统运行时的按键复位功能。系统上电时提供复位信号直至系统电源稳定后撤销复位信号。原理：系统上电时，电容充电，未达到高电平的门限电压时，Reset 输出低电平，复位状态。超过后，正常工作状态。人为按下开关，电容两端电荷释放，Reset 输出变为低电平，复位状态。反相器 7414，实现按钮去抖和波形调整。

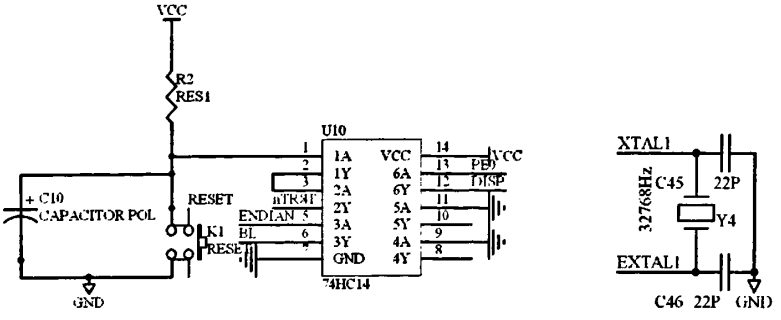


图 3-3　系统的晶振及复位电路
Fig 3-3 System Crystal and Reset Circuit

3.2.3　存储器

1. ROM

芯片采用的 SST39VF160，它是 1M×16b 的 CMOS 多用途 Flash ROM。在系统进行电擦写，掉电后信息不丢失。写入操作前必须先执行擦除（由零变一）再编程。映射在处理器的 Bank0 地址空间，存放系统启动代码，常量表以及一些在系统掉电后需要保存的用户数据，完成初始化工作：设置中断处理程序入口，看门狗，中断控制器，时钟控制器，DMA 控制器。

这里的连接采用了 nCE < nGCS0，即将 Flash ROM 映射到 Bank0，起始地址为 0x00。这样程序便从 Flash 中开始运行。OM 1~0 接 01 选择 16 位总线，而 Arm 的最小处理单元为字节（8 位），因此寻址时处理器地址左移一位。

2. RAM

DRAM 与 Flash 相比具有速度快，写入之前不需要擦除等特点，但是掉电数据会丢失，因此一般作为数据区和堆栈区使用，另外在程序调试的时候也是下载到 SDRAM 中运行。CPU 复位从 0x0 处读取启动代码，调入 SDRAM 中运行，提高系统的运行速度。

本设计采用的是 HYUNGGAU 公司的 HY57V641620HG，容量为 64Mb：1*16*4

Bank。映射到 Bank6，地址范围为 0xC000000~0xC7FFFFF，处理器自动产生读写时序，但是需要定时刷新。在系统进入 C 语言程序之前，44binit.s 程序对 S3C44B0X 的存储器控制器进行了初始化。其中对与 SDRAM 相关的寄存器进行了特殊的设置，以使 SDRAM 能够正常工作，包括器件类型选择、刷新周期设置等。与 Flash 相比具有速度快，写入之前不需要擦除等特点，但是掉电数据会丢失。因此一般作为数据空间使用，在程序调试的时候也是下载到 SDRAM 中运行。

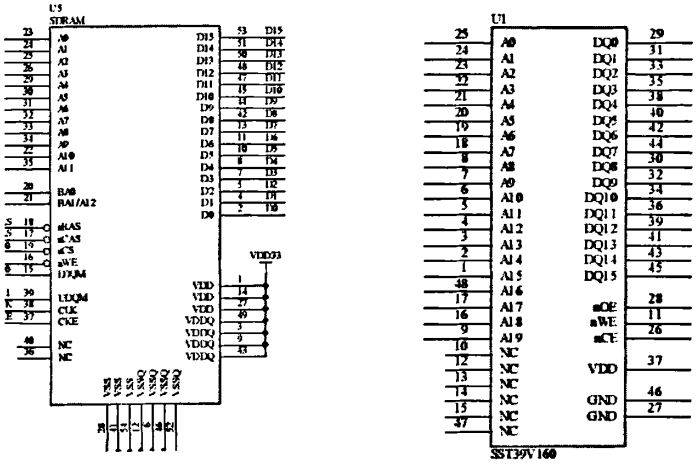


图 3-4　系统的 ROM 和 RAM
Fig 3-4 System's Rom and RAM

3.3 系统外围接口电路设计

3.3.1 采用 UART 的通信接口

要完成最基本的串行通信功能，实际上只需要 RXD、TXD 和 GND 即可，但由于 RS-232-C 标准所定义的高、低电平信号与 S3C44B0X 系统的 LVTTL 电路所定义的高、低电平信号完全不同，LVTTL 的标准逻辑“1”对应 2V～3.3V 电平，标准逻辑“0”对应 0V～0.4V 电平，而 RS-232-C 标准采用负逻辑方式，标准逻辑“1”对应-5V～-15V 电平，标准逻辑“0”对应+5V～+15V 电平，显然，两者间要进行通信必须经过信号电平的转换，图 3-5 为本设计中 MAX232 电路图。

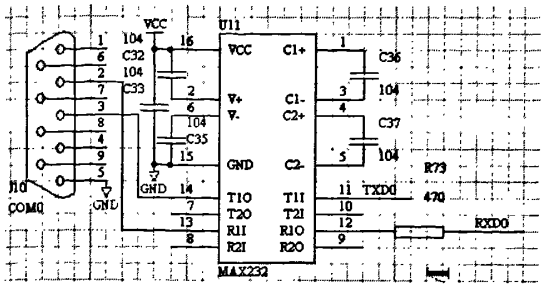


图 3-5 MAX232 电路图
Fig 3-5 MAX232 schematic diagram

3.3.2 LCD 接口设计

本设计所选用的液晶模块为 L78C64，7.8 英寸，640×480 分辨率，256 色 STN 型 LCD 显示屏。LCD 显示缓冲区映射在系统的存储器空间上，程序只需将像素点内容写入从存储器对应地址就可以实现 LCD 屏上像素点颜色的显示。表 3-1 是该液晶模块的接口信号线定义。

序号	符号	描述	备注
1	YD	扫描启动	“H”
2	LP	输入存储信号	“H” →” L”
3	XCK	数据输入时钟信号	“H” →” L”
4	DISP	显示控制信号	“H” 时开显示，“L” 时关显示，用 PE0 接反相器后控制
5	V_{DD}	逻辑电平电源	
6	V_{SS}	逻辑电平地	
7	V_{EE}	LCD 电源	
8~15	D7~D0	显示数据	“H” 时打开，“L” 时关闭

表 3-1 L78C64 模块的接口信号线定义
Table 3-1 L78C64 module interface definition

LCD 控制器外部接口信号的定义及其与 LCD 模块各信号之间的对应关系如下。

VFRAME: LCD 控制器与 LCD 驱动器之间的帧同步信号。该信号表明 LCD

另外一帧开始了。LCD控制器在一个完整帧显示完成后，会马上插入一个VFRAME信号，开始新一帧的显示。该信号与LCD模块的YD信号相对应。

VLINE：　LCD控制器和LCD驱动器之间的线同步脉冲信号。该信号用于LCD驱动器将水平线（行）移位寄存器的内容传送给LCD屏显示。LCD控制器在整个水平线（整行）数据移入LCD控制器后，插入一个VLINE信号。该信号与LCD模块的LP信号相对应。

VCLK：　LCD控制器和LCD驱动器之间的像素时钟信号。由LCD控制器送出的数据在VCLK的上升沿处送出，在VCLK的下降沿被LCD驱动器采样。该信号与LCD模块的XCK信号相对应。

VM：　LCD驱动器的AC信号。VM信号被LCD驱动器用于改变行和列的电压极性，从而控制像素点的显示和熄灭。VM信号可以与每个帧同步，也可以与可变数量的VLINE信号同步。

VD7～0 ：　LCD像素点的数据输入端口。与LCD模块的D7～0相对应。

本设计选用GPE0经反相器输出作为像素点显示开关，采用ENDIAN经反相器输出作为背光控制开关。原理图如图3-6。

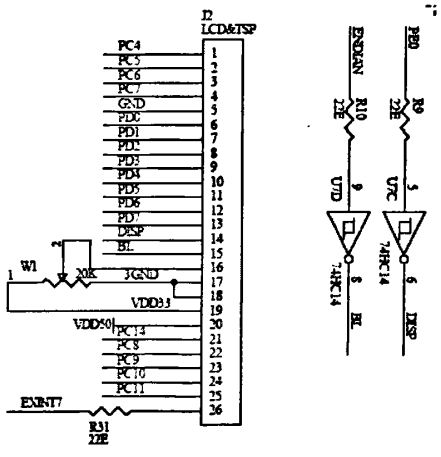


图3-6 LCD接口原理图

Fig 3-6 LCD interface schematic diagram

3.3.3 键盘接口电路设计

矩阵式键盘由行线和列线组成，一般用于按键数量较多的场合。按键位于交叉点上。如图3-7所示，一个4×4的行，列结构可以构成一个有16个按键的键盘。按键设置在行、列交叉点上，行、列分别连接到按键开关的两端。行线通过上拉电阻接到+5V上。平时无按键动作时，行线处于高电平状态；当有按键按下时，

行线电平状态将由通过此按键的列线电平决定；列线电平如果为低，行线电平为低，列线电平如果为高，则行线电平也为高。实际电路采用 GPIOPE4～7 作为输入端口，GPIOPF4～7 作为输出扫描码。

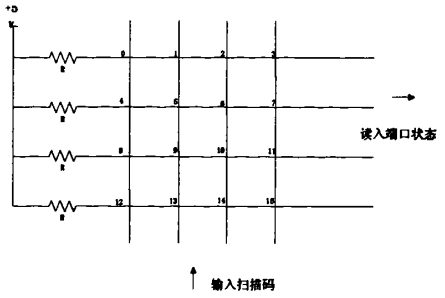


图 3-7 矩阵式键盘结构
Fig 3-7 Matrix Keyboards structure

3.3.4 数据保存接口电路设计

采用 EEPROM 可以将数据永久保存，供下次开机调用，如电压、电流设定值，PID 参数，系统时间，运行状态等，因此采用了 AT24C01 芯片。它提供 128 字节的 EEPROM 存储空间采用 IIC 总线通信。应用电路如图 3-8 所示：

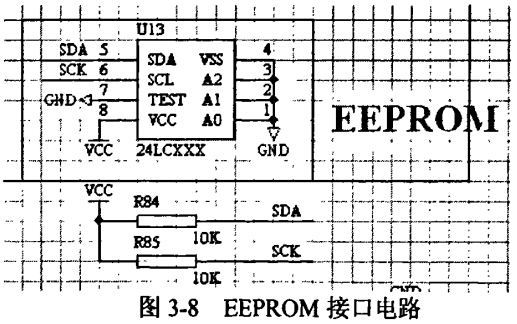


图 3-8 EEPROM 接口电路
Fig 3-8 EEPROM interface

4 系统软件平台的搭建

4.1 uC/OS-II 的移植

本次设计在充分研究了实时内核和硬件平台特性的基础上，将实时内核uC/OS-II 成功移植到 ARM 平台。实现了控制系统软件平台的智能化，模块化。

4.1.1 实时操作系统

实时操作系统(RTOS)的基本功能是任务（线程）调度、存储管理、同步机制、中断管理和 API。和一般操作系统不同的是，RTOS 没有命令行或图形的一个界面，只有一个核心，可以根据不同的应用系统裁剪或扩充。

RTOS 是一段嵌入在目标代码中的程序，系统复位后先初始化各个任务，然后在 RTOS 的调度下，将 CPU 时间、中断、IO、定时器等资源分片的分配给各个任务。从而使系统资源得到很好的利用，并使各任务对事件有更好的响应。RTOS 面对几十个系列的嵌入式处理器 MPU，MCU，DSP,SOC 等提供相同的 API 接口，基于 RTOS 上的 C 语言程序具有极大的可移植性。下面从几个方面分析 RTOS，这将为移植 uC/OS 打下良好的基础。

(1) 前后台系统。一般的小系统都设计成前后台这种模式，应用程序是一个无限的循环，循环中调用函数完成相应的操作，这部分可以看成是后台行为，也称之为任务级。中断服务程序处理异步事件，可以看成前台行为，也称之为中断级。因为中断服务提供的信息一定要等到后台程序运行到该处理这个信息时，才能得到处理。这种系统在处理信息的及时性上，比实际上可以做到的要差。最坏情况下的任务级相应时间取决于整个循环的执行时间。

(2) 任务与任务切换。多任务运行的实现实际上是靠 CPU 在许多任务之间转换和调度，我们可以通过实时操作系统将复杂的应用程序层次化。使用多任务，应用程序将更容易设计和维护。每一个任务都是整个应用的一部分，被赋予一定的优先级，享有自己的 CPU 寄存器和栈空间。每个任务都处在以下 5 种状态之一：休眠态、就绪态、运行态、挂起态和被中断态。休眠态相当于任务驻留在内存中，但并不被多任务内核所调度；就绪态意味着任务已经准备好，可以运行，但由于该任务的优先级比正在运行的任务的优先级低，还暂时不能运行；运行态是指任务掌握了 CPU 的使用权，正在运行中；挂起态，指任务在等待，等待某一事件的

发生，最后发生中断时，CPU 提供相应的中断服务，原来正在运行的任务暂不能运行，就进入了被中断状态。

当多任务内核决定运行另外的任务时，它会先保存正在运行任务的当前状态，即 CPU 寄存器中的全部内容，保存在任务自己的栈区之中。入栈完成之后，就把下一个将要运行的任务的当前状况从该任务的栈中重新装入 CPU 的寄存器，并开始下一个任务的运行。

(3) 可剥夺型内核。在可剥夺性内核中，总是让就绪态且优先级最高的任务先运行，中断服务程序可以抢占 CPU。中断服务完成，内核总是让此时优先级最高的任务运行（不一定是那个被中断的任务），任务级系统响应时间得到了最优化，且是可知的。大多数商业级的 RTOS 都是可剥夺性内核。

(4) 任务间信息的传递有两个途径，通过全称变量，或发消息给另一个任务。用全程变量时，必须保证每个任务或中断服务子程序独享该变量。中断服务中保证独享的唯一办法是关中断。任务只能通过全程变量与中断服务子程序通信。信号量是一种约定机制，在多任务内核中普遍使用，用于控制共享资源的使用权（满足互斥条件），标志某事件的发生，使两个任务的行为同步。只能对信号量实施三种操作，初始化，挂起，发信号。

4.1.2 uC/OS-II 的内核分析

对于小型实时系统来讲，源代码公开、具有很好的可移植性、可固化可裁剪、高稳定性和可靠性、抢占式多任务的 uC/OS-II 非常适合学习。同时 uC/OS-II 已经应用于照相行业、医疗仪器、音响设备、发动机控制、网络接入设备、高速公路电话系统、ATM 机等嵌入式实时系统，其中 uC/OS-IIV2.52 通过了美国航天管理局的安全认证，可以应用于飞机、航天器。这说明 uC/OS-II 不仅是一个很有生命力的，而且是值得放心使用的操作系统。下面对内核代码进行分析。

(1) 主头文件“include.h”,代码中两种类型的源文件都包含“#include”预处理命令。分别为处理器相关的源文件：os_cpu.h ,处理器定义头文件(声明了数据类型，堆栈等，便于编译时分配内存空间) ；os_cfg.h，内核生成配置文件；os_cpu_c.c，用作 ISR（中断服务例程）和 RTOS 定时器。任务切换的汇编代码在 os_cpu_a.s 中。与处理器无关的源文件：ucos_ii.h 和 ucos_ii.c。用作 RTOS 核心、定时器和任务的文件是 os_core.c、os_time.c 和 os_task.c。存储器分区、信号量、队列和邮箱的代码分别在 os_mem.c、os_sem.c、os_q.c 和 os_mbox.c 中。

(2) 当函数或变量的前缀有 OS 和 OS_时，表示它是 uCOS 中的函数或变量。例如，OSTaskCreate()是一个创建任务的函数.OS_NO_ERR 是一个变量，它在函数

没有出错的情况下返回 true。OS_MAX_TASKS 是一个常量，表示用户应用程序中任务的最大数目。

(3) uCOS 是可扩展的 OS。只有必需的 OS 函数才会成为应用程序代码的一部分，这样就减少了对存储器的需求。任务服务，进程间通信等函数必须在配置文件中预先定义。

(4) uCOS 使用抢占式调度程序。有系统级函数，用于 RTOS 初始化和启动、中断初始化以及 ISR 入口和出口函数表。uCOS 在处理变量和数据结构时，使用的是关中断和开中断，提供了两个宏调用，允许在应用程序的 C 代码中关中断，然后再开中断。OS_ENTER_CRITICAL()和 OS_EXIT_CRITICAL（），需要自己编写。

(5) uCOS 有任务服务函数（创建、运行、挂起和恢复执行等），任务延迟函数，进程间通信函数。使用信号量，队列和邮箱。

(6)uCOS 把连续的大块内存按分区来管理，系统中有多个分区，每个分区有多个大小一致的连续的内存块构成。应用程序可以从不同的分区取得所需的内存块，释放时将内存块放回原来所属的内存分区中。使得内存块碎片问题得以解决，相比 ansi C 中的 free()和 malloc()函数。

(7) uCOS 中的任务通常是一个无限的循环，可以管理多达 64 个任务，但目前版本有两个任务为系统任务：taskIdle，taskStat。一旦任务建立，一个任务控制块 OS_TCB 就会被赋值，OS_TCB 是一个数据结构，能保存任务的所有状态。

4.1.3 移植平台 ARM 简介

本小节对 ARM 微处理器的体系结构、寄存器的组织、处理器的工作状态、运行模式等内容进行了描述，这些内容也是 ARM 体系结构的基本内容，是系统软、硬件设计的基础，更是移植操作系统的基本。

(1) 处理器结构

S3C44B0x 是基于 ARM7TDMI 内核的处理器，为低功耗的 32 位 RISC 处理器，最适合用于对价位和功耗要求较高的消费类应用。ARM7 系列微处理器的主要应用领域为：工业控制、Internet 设备、网络和调制解调器设备、移动电话等多种多媒体和嵌入式应用。ARM7 微处理器系列具有如下特点：

- 具有嵌入式 ICE－RT 逻辑，调试开发方便。
- 极低的功耗，适合对功耗要求较高的应用，如便携式产品。
- 能够提供 0.9MIPS/MHz 的三级流水线结构。
- 代码密度高并兼容 16 位的 Thumb 指令集。

－ 对操作系统的支持广泛，包括 Windows CE、Linux、Palm OS 等。

－ 指令系统与 ARM9 系列、ARM9E 系列和 ARM10E 系列兼容，便于用户的产品升级换代。

－ 主频最高可达 130MIPS，高速的运算处理能力能胜任绝大多数的复杂应用。

－ 直接使用 JTAG 下载和调试，可以源代码级跟踪调试。

Arm7 采用 RISC 体系结构，支持两种指令集：ARM 指令集和 Thumb 指令集。其中，ARM 指令为 32 位的长度，Thumb 指令为 16 位长度。Thumb 指令集为 ARM 指令集的功能子集，但与等价的 ARM 代码相比较，可节省 30%～40%以上的存储空间，同时具备 32 位代码的所有优点。在程序的执行过程中，微处理器可以随时在两种工作状态之间切换。

(2) 存储格式及数据类型

ARM体系结构将存储器看作是从零地址开始的字节的线性组合。从零字节到三字节放置第一个存储的字数据，从第四个字节到第七个字节放置第二个存储的字数据，依次排列。作为32位的微处理器，ARM体系结构所支持的最大寻址空间为4GB（232字节）。ARM体系结构可以用大端格式和小端格式两种方法存储字数据。

ARM 微处理器中支持字节（8 位）、半字（16 位）、字（32 位）三种数据类型，其中，字需要 4 字节对齐（地址的低两位为 0）、半字需要 2 字节对齐（地址的最低位为 0）。

(3) 寄存器和工作模式

ARM 处理器共有 37 个寄存器，被分为若干个组（BANK），这些寄存器包括：

－ 31 个通用寄存器，包括程序计数器（PC 指针），均为 32 位的寄存器。

－ 6 个状态寄存器，用以标识 CPU 的工作状态及程序的运行状态，均为 32 位。

同时，ARM 处理器又有 7 种不同的处理器模式，在每一种处理器模式下均有一组相应的寄存器与之对应。即在任意一种处理器模式下，可访问的寄存器包括 15 个通用寄存器（R0～R14）、一至二个状态寄存器和程序计数器。在所有的寄存器中，有些是在 7 种处理器模式下共用的同一个物理寄存器，而有些寄存器则是在不同的处理器模式下有不同的物理寄存器。ARM 微处理器的运行模式可以通过软件改变，也可以通过外部中断或异常处理改变。处理器的中断分成 IRQ 和 FIQ，一个外部中断到底是 FIQ 还是 IRQ，可以通过设置 S3C44B0x 的中断控制器的 INTMOD 端口设定。

大多数的应用程序运行在用户模式下，当处理器运行在用户模式下时，某些

被保护的系统资源是不能被访问的。除用户模式以外，其余的所有6种模式称之为非用户模式，或特权模式（Privileged Modes）；其中除去用户模式和系统模式以外的5种又称为异常模式（Exception Modes），常用于处理中断或异常，以及需要访问受保护的系统资源等情况。

4.1.4 uC/OS-II 的移植（ucos2.83）

uCOS大部分代码是用C语言编写的，但是涉及到对寄存器的直接处理部分就只能用汇编，因此移植性较好。uCOS要求处理器的C编译器能产生可重入代码；可以用C语言打开和关闭中断；处理器支持并能产生定时中断；容纳一定量数据的硬件堆栈；有将堆栈指针和其他CPU寄存器读出和存储到堆栈或内存中的指令。下图表明了uCOS和硬件平台、应用软件的关系，可以看出有的代码与硬件无关，在移植中不用修改，其他与硬件相关的则需要做出相应的修改。下边详细介绍了移植步骤并给出了修改的源文件代码以及说明（灰色部分）。

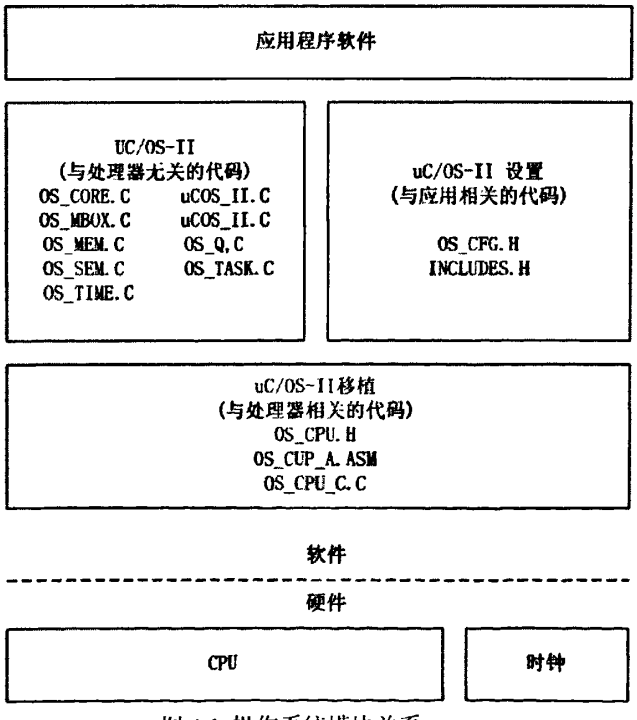


图 4-1 操作系统模块关系

Fig 4-1 relationship beween RTOS modules

(1) Include.h 头文件

所有.C 文件都要包含头文件 include.h ，config.h 文件则进行系统配置的修改和头文件的增减。这样意味着编译时间的增加，但是增强了代码的可移植性。

(2) 设置处理器和编译器相关的代码

OS_CPU.H 文件中主要包含了与处理器、编译器相关的常量，宏和结构体的定义。相关的代码都要在这个头文件中进行修改。

uC/OS-II 不使用 C 语言中的 short、int、long 等数据类型的定义，因为它们与处理器类型有关，隐含着不可移植性。代之以移植性强的整数数据类型，根据 ADS 编译器的特性，不依赖于编译器的代码如下

```
typedef unsigned char BOOLEAN;
typedef unsigned char INT8U;              //8 位无符号整数
typedef signed char INT8S;                //8 位有符号整数
typedef unsigned short INT16U;            //16 位无符号整数
typedef signed short INT16S;              //16 位有符号整数
typedef unsigned long INT32U;             //32 位无符号整数
typedef signed long INT32S;               //32 位有符号整数
typedef float FP32;                       //单精度浮点数
typedef double FP64;                      //双精度浮点数
typedef unsigned int    OS_STK;           //定义堆栈的 宽度 16 位
typedef unsigned int    OS_CPU_SR;        //定义寄存器的 宽度 16 位
```

uC/OS-II 在进入系统临界代码区之前要关闭中断，等到退出临界区后再打开。从而保护核心数据不被多任务环境下的其他任务或中断破坏。uC/OS-II 定义了两个宏用来关闭/打开中断：OS_ENTER_CRITICAL()和 OS_EXIT_CRITICAL()。需要手工修改成 ARM 相关的中断方式。同时设置了堆栈的增长方向，为从上向下生长。原因是因为 ADS 的 C 语言编译器仅支持一种方式，即从上往下长，并且必须是满递减堆栈，所以 OS_STK_GROWTH 的值为 1。

```
#define    OS_ENTER_CRITICAL()
ARMDisableInt() rINTMSK=~(BIT_GLOBAL) //关中断
#define   OS_EXIT_CRITICAL()
ARMEnableInt()   rINTMSK=BIT_GLOBAL   //开中断

#define   OS_STK_GROWTH          1       // 堆栈的增长方向,出高向低
```

为了使底层接口函数与处理器状态无关，同时在任务调用相应的函数不需要知道函数位置，在移植中使用软中断指令 SWI 作为底层接口，使用不同的功能号区分不同的函数。宏 OS_TASK_SW()实际上被定义为 OS_CPU_A.S 中的函数

OSCtxSw()。保存当前任务上下文，装入新任务上下文。

```
#define OS_TASK_SW        OSCtxSw
```

(3) OS_CPU. A. S 文件的编写

uC/OS-II 的移植需要改写 OS_CPU_A.ASM 中的四个函数：OSStartHighRdy(),OSCtxSw(),OSIntCtxSw()和 OSTickISR()。

第一个函数由 OSStart()函数调用，用户必须先调用 OSInit()，并且已经至少创建了一个任务。OSStart()最终调用 OSStartHighRdy(),功能是运行多任务启动前优先级最高的任务。默认指针 OSTCBHighRdy 指向优先级最高就绪任务的任务控制块(OS_TCB),在这之前 OSTCBHighRdy 已由 OSStart()设置好了。

OSTCBHighRdy->OSTCBStkPtr 指向的是任务堆栈的顶端。

```
OSStartHighRdy
    LDR     R0,   OS_TaskSwHook      ; 调用钩子函数
    MOV     LR, PC
    BX      R0

    MSR     CPSR_cxsf, #0xD3         ; 切换到 SVC 模式
    LDR     R4, OS_Running           ; 系统已经运行 OSRunning=true
    MOV     R5, #1
    STRB    R5, [R4]

    LDR     R4, OS_TCBHighRdy        ;  取得新任务的 TCB 指针
    LDR     R4, [R4]                 ;
    LDR     SP, [R4]                 ;  任务切换

    LDR     R4,   [SP], #4           ;   出栈，新任务上下文
    MSR     SPSR_cxsf,R4
    LDMFD   SP!, {R0-R12,LR,PC}^     ;
```

uC/OS-II 是抢占式实时操作系统，得到运行的始终是就绪条件下最高优先级的任务。当处于运行状态的任务因为某种原因进入就绪态，或者有其它更高优先级的任务进入就绪态，操作系统内核就要运行别的就绪任务，这时需要进行任务切换。有两种情况，一种是当前任务主动交出控制权，一种则是发生中断。OSCtxSw()和 OSIntCtxSw()功能相同，OSCtxSw()是任务级的任务切换函数，而在中断程序中调用的 OSIntCtxSw()。

```
OSCtxSw
```

```
STMFD   SP!, {LR}              ; 保存当前任务的寄存器及其他
STMFD   SP!, {R0-R12}          ;

LDR     R4, OS_TCBCur          ; OSTCBCur->OSTCBStkPtr = SP
LDR     R5, [R4]
STR     SP, [R5]               保存当前任务的 TCB 堆栈指针到 TCB 中

LDR     R0, OS_TaskSwHook      ; 调用钩子函数

LDR     R4, OS_PrioCur  ;
LDR     R5, OS_PrioHighRdy
LDRB    R6, [R5]
STRB    R6, [R4]
LDR     R4, OS_TCBCur          ; 得到就绪任务中的最高优先级的任务
LDR     R6, OS_TCBHighRdy
LDR     R6, [R6]
STR     R6, [R4]               ; 使当前任务指针指向最高优先级的任务

LDR     SP, [R6]               ; SP = OSTCBHighRdy->OSTCBStkPtr;
// 用将要运行任务的优先级和 TCB 指针更新 OSPrioHighRdy 和
OSTCBCur
LDMFD   SP!, {R4}              ;恢复新任务的寄存器和数据
MSR     SPSR_cxsf, R4
LDMFD   SP!  {R0-R12,LR,PC}^   ; 运行新任务
```

uCOS Ⅱ要求用户提供一个时钟源来实现时钟节拍的功能。由于本设计中采用Timer0 作为 PID 定时中断时钟源，因此采用了定时器 1 作为时钟节拍发生器，每1ms 产生一个节拍。而 FIQ 和 IRQ 两种中断为了不影响速度，都未做修改。

(4) OS_CPU.C.C 文件的编写

uC/OS-II 的移植需要用户改写 OS_CPU_C.C 中的十个函数：

OSTaskStkInit()：OSTaskCreat()和 OSTaskCreatExt()通过调用本函数，初始化任务的栈结构

OSTaskCreateHook()：每当添加任务时由 OS_TCBInit()函数调用

OSTaskDelHook()：　任务被删除后由 OSTaskDel()调用

OSTaskSwHook()：　任务切换时两种情况均会调用该函数

OSTaskIdleHook()：OSTaskIdle()函数可调用该函数实现 CPU 低功耗模式

OSTimeTickHook()：本函数在每个时钟节拍都会被 OSTimeTick()调用

OSInitHookBegin()：进入 OSInit()函数后本函数会立即被调用

OSInitHookEnd()： OSInit()函数返回之前被调用

OSTCBInitHook()：OS_TCBInit( )在调用 OSTaskCreateHook()之前将先调用本函数

实际需要修改的只有 OSTaskStkInit()函数，其他函数必须声明，但不一定有实际内容。这些函数都是用户定义的，所以 OS_CPU_C.C 中没有给出代码。如果用户需要使用这些函数，请将文件 OS_CFG.H 中的#define constant OS_CPU_HOOKS_EN 设为 1，设为 0 表示不使用这些函数。

这个函数是很重要，由 OSTaskCreate()或 OSTaskCreateExt()调用，用来初始化任务的堆栈。初始状态的堆栈模拟发生一次中断后的堆栈结构。当调用 OSTaskCreate()或 OSTaskCreateExt()创建一个新任务时，需要传递的参数是：任务代码的起始地址，参数指针(pdata)，任务堆栈顶端的地址，任务的优先级。 OSTaskCreateExt()还需要一些其他参数，但与 OSTaskStkInit()没有关系。 OSTaskStkInit()只需要以上提到的 3 个参数（task, pdata,和 ptos)。在这个堆栈初始化函数中要清楚堆栈中都要保存哪些东西，要留多大的空间，这些都很重要，否则会发生很严重的错误。

处理器是在入栈时，先变化 sp，再向当前的 sp 指向的地址写入数据。出栈时是先弹出数据，再变化 sp。状态寄存器，同样也会被自动保存。

```
OS_STK *OSTaskStkInit (void (*task)(void *p_arg), void *p_arg, OS_STK *ptos,
INT16U opt)
{
    OS_STK *stk;
    INT32U   task_addr;                 //定义一个指针
    opt    = opt;                       //这个参数没有用，但是为了防止编译错误
    stk    = (INT32U*)ptos;             //载入堆栈指针
    *(stk)   = (INT32U)task_addr;       // Entry Point
    *(--stk) = (INT32U) 0x14141414L;    //R14(LR)
    *(--stk) = (INT32U) 0x12121212L;    //R12
    *(--stk) = (INT32U) 0x11111111L;    //R11
    *(--stk) = (INT32U) 0x10101010L;    //R10
    *(--stk) = (INT32U) 0x09090909L; //R9
```

```
    *(--stk) = (INT32U) 0x08080808L; //R9
    *(--stk) = (INT32U) 0x07070707L; //R7
    *(--stk) = (INT32U) 0x06060606L; //R6
    *(--stk) = (INT32U) 0x05050505L; //R5
    *(--stk) = (INT32U) 0x04040404L; //R4
    *(--stk) = (INT32U) 0x03030303L; //R3
    *(--stk) = (INT32U) 0x02020202L; //R2
    *(--stk) = (INT32U) 0x01010101L; //R1
    *(--stk) = (INT32U)p_arg;          // R0 : argument
    return ((OS_STK *)stk);            //返回堆栈指针所指向的地址，恢复寄存器时候要用
}
```

4.2 uC/GUI 的移植

4.2.1 uC/GUI 分析

uC/GUI 是一种嵌入式应用中经常采用的图形支持系统。设计的目的在于为任何使用 LCD 图形显示的应用提供高效的独立于处理器及 LCD 控制器的图形用户接口。它适用于单任务或是多任务系统环境，并适用于任意 LCD 控制器的真实显示或虚拟显示.下面首先分析了 uC/GUI 的软件结构和 uC/GUI 移植的详细步骤。

(1) 数据结构类型

uC/GUI 具有类 Windows 窗口风格，内存管理的基本单位为窗口。通过定义结构实现对窗口的管理。每个窗口在创建时根据其结构类型的不同为其在堆空间里分配特定大小的连续内存块，用一个结构体数组中的一个元素标识。块结构体的定义如下：

```
typedef struct
{
tALLOCINT Off;  //在堆中块开始的位置
tALLOCINT Size; //在堆中所占连续内存块的大小
HANDLE Next;  //指向后一窗口指针
HANDLE Prev;  //指向前一窗口指针
}tBlock;
static tBlock aBlock [GUI_MAXBLOCKS];
```

所分配的块结构体数组元素在数组中的序号作为返回值用以标识一个窗口，

即我们经常提到的句柄。这样对于一个窗口而言，记录其各方面属性的窗口类型结构体变量被放在堆空间中并且有一个块标志与之对应。通过定义一个全局结构变 GUI_Alloc 存放相关分配信息，实现对内存的辅助管理。

(2) 工作机制

窗口的创建、显示及删除都离不开消息机制和回调函数。在 uC/GUI 中定义了各种用于各类基本操作的消息宏，通过向其对应的回调函数传送消息参数完成对一个窗口的操作是基本的操作过程。在整个工作过程中，uC/GUI 利用一个全局的结构体变量 GUI_Context 即所谓的上下文变量来记录包括：绘图属性、当前窗口信息、当前 API 列表、字体信息等与当前操作密切相关的信息，以管理整个工作流程。uC/GUI 的开发环境及其流程如图 4-2 所示。

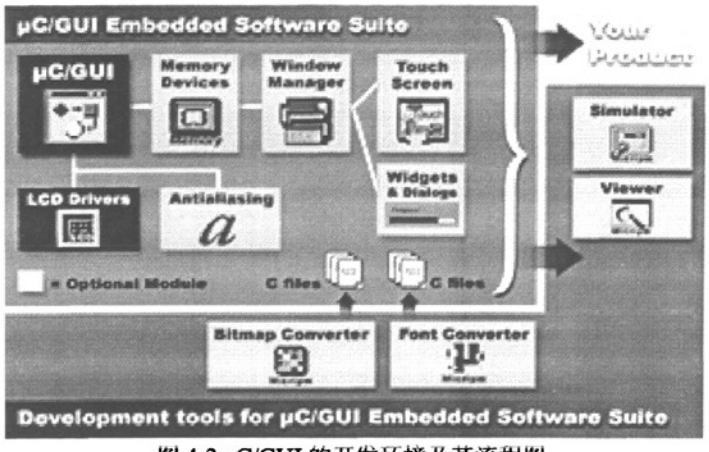


图 4-2 uC/GUI 的开发环境及其流程图
Fig 4-2 uC/GUI development flow chart

4.2.2 移植步骤

(1) 修改 uC/GUI,使之适于移植

首先需要进行 uC/GUI 的目录结构和基本配置，修改 LCDConf.h, GUI_X.C 的内容：

uC/GUI 主要目录如下：

GUI/ConvertMono　使用黑白显示设备时，所要使用的灰度转换函数

GUI/ConvertColor　使用彩色显示设备时，使用的色彩转换函数

GUI/Config　包含了对 uC/GUI 进行配置的一些文件

GUI/Core　uC/GUI 核心代码

GUI/Font　uC/GUI 与字体相关的代码文件

GUI/LCDDriver　LCD 驱动代码文件

GUI/MemDev　　内存设备支持文件代码
GUI/Touch　　输入设备支持的文件代码
GUI/Widget　　uC/GUI 支持的控件代码，包括编辑框、列表框、按钮、选择框
GUI/WM　　uC/GUI 窗口管理部分代码

```
/*LCDConf.h*/
#ifndef LCDCONF_H
#define LCDCONF_H
#define LCD_XSIZE (640) // LCD 水平分辨率
#define LCD_YSIZE (480) // LCD 竖直分辨率
#define LCD_BITSPERPIXEL (8)
#endif // LCDCONF_H
```

因为 GUI_Init()在启动多任务之前调用，而未启动多任务的情况下，OSTCBCur->OSTCBPrio 指向未定值；而 GUI_Init()函数会调用 GUI_LOCK()，进而调用 GUI_X_GetTaskId(),所以此处以布尔型变量 bGUIInitialized 标识,GUI_Init()后，bGUIInitialized＝1,GUI_X_GetTaskId()返回当前任务的优先级作为任务 ID，否则为 bGUIInitialized＝ 0,GUI_X_GetTaskId()返回 100 作为任务 ID。

```
U32 GUI_X_GetTaskId(void)
{
if(bGUIInitialized){
return ((INT32U)(OSTCBCur->OSTCBPrio)); // < 64(任务优先级)
 }
  else{
return 100; // 只要保证>64 即可}}
void GUI_X_InitOS(void)
{   DispSem = OSSemCreate(1);   }
                                          //空函数
void GUI_X_ErrorOut(const char *s){}
void GUI_X_Log(const char *s){}
void GUI_X_Warn(const char *s){}
void GUI_X_Init(void){}
```

(2)　根据硬件平台，编写 LCD 驱动

S3C44B0X 内置 LCD 控制器，相关驱动写在 lcd44b0.c 文件中，主要进行相关的寄存器配置，以及和 GUI 的接口程序。

1）定义显示缓冲区时使用的 char 数据类型,它是 8bit 的:

```
unsigned char Bmp[ARRAY_SIZE_G16];  //液晶显示缓冲数组
```

2）定义读写缓冲区时使用的数据类型,也是 8bit 的 U8:

```
#define LCD_READ_MEM(Off)    *((U8*) (Bmp+(((U32)(Off)))))
#define LCD_WRITE_MEM(Off, data) *((U8*) (Bmp+(((U32)(Off)))))=data
//#define LCD_READ_REG(Off) //这个函数可以不用定义，这里没有用到
#define LCD_WRITE_REG(Off, data) //有些地方用到了，定义为空，避免做大改动
```

3）定义液晶总线宽度定义位 8bit 的

```
#ifndef LCD_BUSWIDTH
#define LCD_BUSWIDTH (8)
#endif
```

4）定义字节顺序

```
#define LCD_SWAP_BYTE_ORDER (0) //8bit 时不需要交换。
```

4.2.3 LCD 驱动程序设计

LCD 驱动是整个 GUI 移植的关键，编写 LCD 的驱动程序后，就可以在 uC/GUI 中调用，用于初始化液晶，建立显示缓冲区。

（1）LCD 液晶屏扫描时序

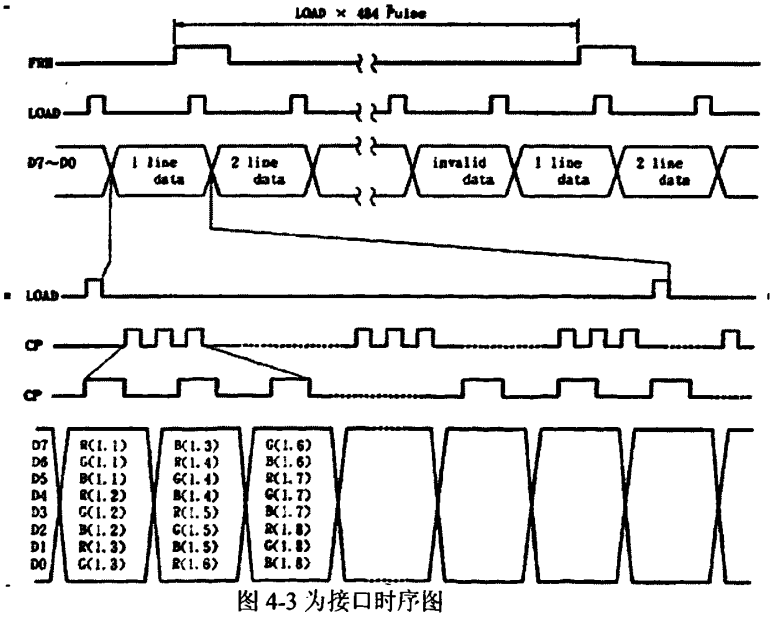


图 4-3 为接口时序图

Fig 4-3 Time sequence

如图 3-3 所示，写满整个屏的数据成为 1 个“帧”数据，YD 是帧同步信号，该信号启动 LCD 屏的新一帧数据。两个 YD 脉冲之间的时间长度称为帧周期。根

据 LCD 模块的特性，刷新时间为 12～14ms，频率为 70～80Hz。每一帧包括 480 个 LP 脉冲。

LP 为行（共 480 行）数据输入锁存信号，也就是行同步脉冲信号。该信号启动 LCD 屏的新一行数据。

XCK 为行数据输入信号，也就是每 1 行中像素点数据传输的时钟信号。每组 8 位的数据在 XCK 的下降沿被输入锁存，因此，每 1 行包括 640×3/8 个 XCK 脉冲信号。

D0-D7 是 8 位的显示数据输入信号。

（2）I/O 口的初始化

设置 PC 口工作在第 3 功能状态和 PD 口工作在第 2 功能状态以使用 S3C44B0X 的 PC 口和 PD 口作为 LCD 驱动接口。

(3) 控制寄存器的设置

S3C44B0X 包括一个 LCD 控制器时序发生器 TIMEGEN，由它来产生 VFRAM，VLINE，VCLK 和 VM 控制时序。这些控制信号由寄存器 LCOCON1 和 LCDCON2 进行配置。通过对寄存器种配置项目的设置，TIMEGEN 就可以产生适应于各种 LCD 屏的控制信号了。

VFRAM 和 VLINE 脉冲的产生是通过对 LCDCON2 寄存器的 HOZVAL 和 LINEVAL 进行配置来完成的。每个域都与 LCD 的尺寸和显示模式有关。

HOZVAL:

HOZVAL=（显示宽度/VD 数据线位数）－1

其中，在彩色模式下

显示宽度＝3×每行的像素点数

对所选的液晶模块，HOZVAL＝（640×3/8）-1。

LINEVAL:

在单扫描显示类型下

LINEVAL＝（显示宽度）－1

在双扫描显示类型下

LINEVAL＝（显示宽度/2）-1

对所选的液晶模块，LINEVAL＝480－1。

VCLK 信号的频率可以通过 LCDCON1 寄存器的 CLKVAL 域来确定，即

VCLK＝MCLK/(CLKVAL×2)

LCD 控制器的最大 VCLK 频率为 16.5MHz，几乎支持所有已有的 LCD 驱动器。由于上述关系，CLKVAL 的值决定了 VCLK 的频率。为了确定 CLKVAL 的值，应该计算一下 LCD 控制器向 VD 端口传输数据的速率，以便使 VCLK 的值大于数

据传输的速率。

数据传输速率的公式为

数据传输速率＝HS×VS×FR×MV

其中，HS：LCD 的行像素值；

VS：LCD 的列像素值；

FR：帧速率；

MV：模式值，这里取 8 位单扫描，彩色。

对于所选用的液晶模块：HS＝640；VS＝480；FR＝70Hz；MV＝3/8。因此

数据传输速率＝640×480×70×3/8=8064000Hz

VCLK 值应该大于 8MHz 而小于 16MHz，因此，CLKVAL 可以取 9～15。

经过以上几个步骤，就完成了 uC/GUI+uC/GUI 在硬件平台上的移植。就可以进行控制器界面设计了。

4.3 系统的界面设计

图形用户界面主要包括开机显示、用户控制面板、实时曲线绘制等，采用 uC/GUI 提供的视窗管理器及对话框来实现。uC/GUI 提供了类似 windows 窗口系统的各种控件，使图形界面变得易于使用。uC/GUI 将视窗界面功能分为 3 个部分：视窗管理器（Windows Manager），视窗控件（Widgets），对话框（Dialog）。

4.3.1 开机界面设计

开机界面包括系统的简单介绍，背景的绘制，用户输入任意键进入控制系统。实现过程主要通过调用 GUI_DispString()和 GUI_SetFont()函数来完成，其中主标题采用 48×55 的宋体字体，副标题采用 29×38 的微软雅黑字体。开机界面的右下角还会显示当前系统的版本。

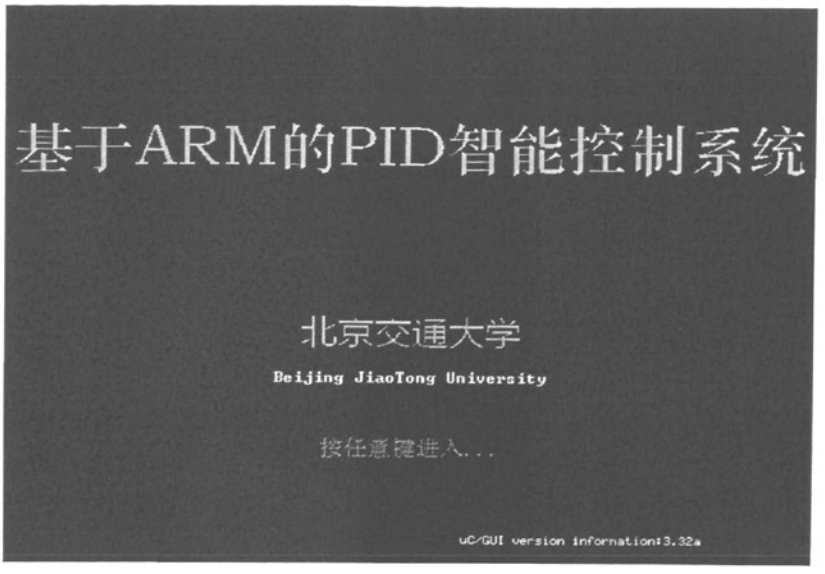


图 4-4 开机界面
Fig 4-4 Start-up interface

4.3.2 控制面板设计

控制面板界面采用对话框（Dialog）进行编程，并且添加了 Radio, Edit 等控件。对话框可以从用户得到输入信息并可以包含多个控件，通过各种选项从用户那里获得信息的一种窗口；它也可以仅仅是一种消息框模式，提供一些简单的信息和一个确认键。

需要设置的参数主要包括系统的工作模式（手动和自动）。手动模式需要输入电压设定值、规一化 PID 参数；而自动整定模式需要输入超调量与整定时间两个最重要的参数。其中电压等输入框定义格式为浮点型，小数点后保留两位。由于 S3C44B0X 的 RTC 模块的时间日期设置值为 BCD 码，所以时间设置框定义格式为 16 进制。波特率、校验等采用 Radio 控件，通过判断 Radio 的返回值，设定串口的工作模式，坐标设置主要包括设置显示区域占实际 LCD 屏的大小，x 轴、y 轴刻度信息，由于实际采用的液晶模块为 640×480 的分辨率，实际 ADC 采样为 10 位（1024），因此默认定义 y 轴刻度为 1024，显示区域大小为 620×440。设置完毕后，按 OK 键，将输入框的值保存在共有变量中，实际显示效果如图 4-5。

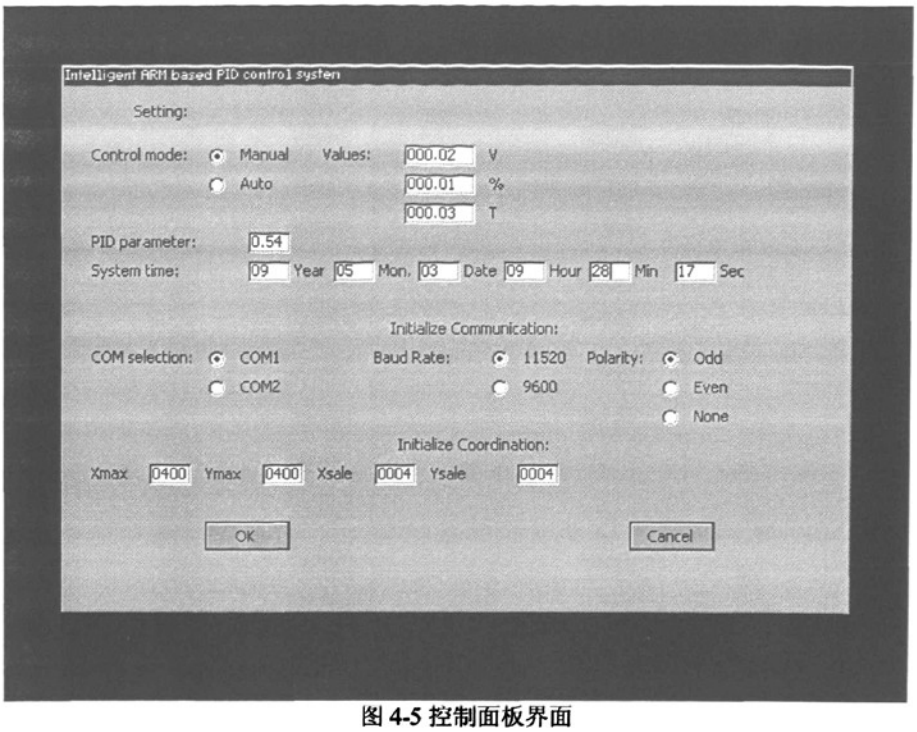


图 4-5 控制面板界面

Fig 4-5 Control panel interface

4.3.3 信号显示设计

显示设计主要包括坐标变换、初始化坐标系统、绘制坐标系，绘制曲线等。绘制曲线主要采用 GUI_DrawLine()方法其函数原型为：

void GUI_DrawLine(int x0, int y0, int x1, int y1);

其中 x0,y0 是曲线起点坐标，x1,y1 是曲线终点坐标

(1) 坐标变换

由于在 LCD 中默认的坐标系 Y 轴是向下的，无法直接调用 GUI_DrawLine 函数用于曲线绘制，需要进行变换。

```
void initP(void)
        {
            p0.X = 20; p0.Y = 460;
            p1.X = 620+xmax; p1.Y = 460-ymax;
            p_0.X = 0; p_0.Y = 0;
            p_1.X = 2000; p_1.Y = 1024;
            xnum = xscale; ynum = yscale;
```

```
            // count = 0;
        }
```

坐标初始化程序函数为 initP，需要定义系统原点及顶点的坐标及显示的最大区域，其中 xmax,ymax 分别指定 x 轴和 y 轴方向的象素数，xscale，yscale 指 x 轴,y 轴的刻度数。函数 initRef()用于坐标变换，通过用户输入的 xmax,ymax 的值，进行方向变换及缩放变换。其原理就是通过同比例变换进行缩放，方向变换则通过改变正负来进行。

```
    void initRef() {
            float sx1,sy1, sx2,sy2, sx_1, sy_1,sx_2,sy_2;//
            float x__1, x__2, y__1, y__2;//
             sx1 = p0.X;     sx2 = p1.X;     sy1 = p0.Y;     sy2 = p1.Y;
            sx_1 = p_0.X;   sx_2 = p_1.X; sy_1 = p_0.Y; sy_2 = p_1.Y;
            xscale = (sx2 - sx1) / (sx_2 - sx_1);
            yscale = -(sy2 - sy1) / (sy_2 - sy_1);/、
        x__1 = sx_1 * xscale; x__2 = sx_2 * xscale; y__1 = sy_1 * yscale; y__2 =
sy_2 * yscale;
            yshift = sy1 + y__1;//   Y=yshift - Y_ * yscale
            xshift = sx1 - x__1;// X=x_ * scale + xshift
            yspace = -(p1.Y - p0.Y) / ynum;
            xspace = (p1.X - p0.X) / xnum;

        }
```

(2) 绘制坐标系

经过坐标变换、刻度和最值确定后就可以直接调用 GUI_DrawLine 进行坐标系的绘制了。如图 3-6 所示

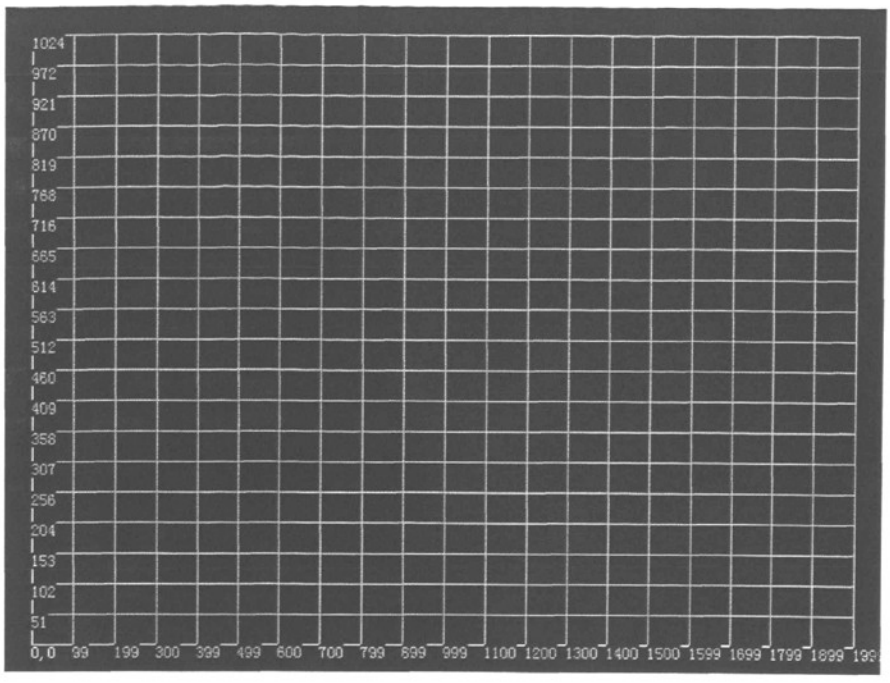


图 4-6 坐标系统
Fig 4-6 coordination system

5 PID 控制系统任务设计及具体实现

5.1 基于任务的软件设计

移植操作系统的目的就在于利用操作系统管理复杂的软件流程，由于在通用 PID 控制器中需要处理 ADC 采样，PID 算法，控制输出，同时需要记录运行时间，故障，使能控制以及完成通信等任务，因此，如果没有操作系统的帮忙，整个程序的设计将变得非常复杂，并且很难达到最优控制，基于操作系统的软件设计的好处就是，可以根据任务的紧急程度，赋予任务不同的优先级，由操作系统自己管理程序的运行，这样使程序的灵活性和鲁棒性大大提高，操作系统的优势也充分发挥出来。下面首先介绍了任务函数及任务函数之间的通信，并分别介绍了本设计中用到的任务函数、流程图及部分代码。

5.1.1 任务函数

在 uC/OS II 中，程序是按照任务函数(Task)来执行的，任务函数是 uC/OS II 中的最小操作对象，一个任务可能处于 4 种不同的状态：休眠状态、就绪状态、挂起状态和运行状态。在运行状态下，还可能由于发生中断而转向中断服务子程序运行。任务管理函数的调用将引起各任务状态间的转换，如图 5-1 所示。

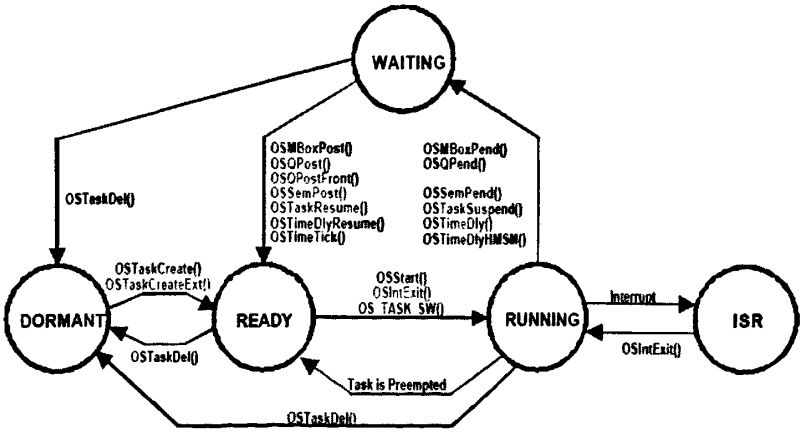


图 5-1 任务状态转换与任务管理函数
Fig 5-1 Status change of tasks and tasks management functions

uCOS/II的任务管理函数主要包括：

1. 建立任务函数 OSTaskCreate()

若想让 uCOS/II管理用户任务，用户必须先建立任务。用户可以通过将任务地址和其它参数传递给一下两个任务之一来建立任务：OSTaskCreate()或OSTaskCreateExt()。OSTaskCreateExt()是 OSTaskCreate()的扩展版本，提供了一些附加功能。在任务调度（即调用 OSStart()）前，用户必须建立至少一个任务。任务不能由中断服务程序（ISR）来建立。

```
INT8U OSTaskCreate(void(* task)(void * pd),void * pdata,OS_STK * ptos,INT8U prio);
```

可知，OSTaskCreate()需要 4 个参数：task 是任务代码的指针，pdata 是当任务开始执行时传递给任务的参数指针，ptos 是分配给任务的堆栈的栈顶指针，prio 是分配给任务的优先级。

2. 删除任务函数 OSTaskDel()

删除任务是指任务将返回并处于休眠状态，并不是说任务的代码被删除，只是任务的代码不再被 uCOS/II 调用。通过调用 OSTaskDel()就可以完全删除任务。

```
INT8U OSTaskDel(INT8U prio);
```

3. 请求删除任务函数 OSTaskDelReq()

```
INT8U OSTaskDelReq(INT8U prio);
```

可见，删除任务是通过任务唯一的标志——优先级来进行的。

4. 改变任务的优先级 OSTaskChangePrio()

```
INT8U OSTaskChangePrio(INT8U oldprio,INT8U newprio);
```

5. 挂起任务 OSTaskSuspend()

```
INT8U OSTaskSuspend(INT8U prio):
```

6. 恢复任务函数 OSTaskResume()

```
INT8U OSTaskResume(INT8U prio);
```

5.1.2 任务通信

在 uCOS/II中，采用多种方法保护任务之间的共享数据和提供任务之间的通信。例如提供 OS_ENTER_CRITICAL 和 OS_EXIT_CRITICAL 来对临界资源进行保护。另外一种是利用函数 OSSchedLock( )禁止调度保护任务级的共享资源。uCOS/II中同样提供了经典操作系统任务间通信方法：信号量、邮箱、消息队列，事件标志。在本设计中主要采用邮箱的方式进行通信。

邮箱可以使一个任务或者中断服务子程序向另一个任务发送一个指针型的变

量。该指针指向一个包含了特定“消息”的数据结构。为了在 uCOS/II 中使用邮箱，必须将 OS_CFG.H 中的 OS_MBOX_EN 常数置 1。

使用邮箱之前，必须先通过 OSMboxCreate()建立邮箱，并且指定指针的初始值。一般情况下，这个初始值是 NULL，但也可以初始化一个邮箱，使其在最开始就包含一条消息。如果使用邮箱的目的是用来通知一个事件的发生（发送一条消息），那么就要初始化该邮箱为 NULL，因为在开始时，事件还没有发生；如果用户用邮箱来共享某些资源，那么就要初始化该邮箱为一个非 NULL 的指针。在这种情况下，邮箱被当作一个二值信号量使用，关系如图。

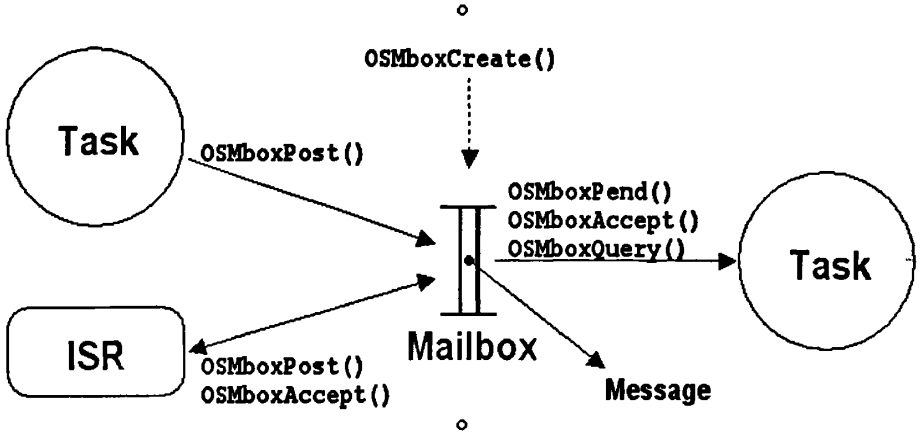


图 5-2 任务和邮箱间关系
Fig 5-2 Relationship of tasks and mailbox

5.2 PID 控制系统任务

5.2.1 主任务

主任务的作用是在操作系统开始运行后，创建各个子任务，然后进入消息队列等待状态，根据不同的消息，进行不同的处理。本设计中主任务首先读取 EEPROM 中的配置信息，然后建立各项子任务：PID 任务，用于 PID 运算；键盘扫描任务，用于检测用户输入；串行通讯任务，用于上位机监控；输入输出控制量任务，用于进行 AD 采样和 PWM 输出；GUI 任务，用于显示输出；实时时钟任务 RTC，用于提供系统时间基准。在 uC/OS II 中，每个任务都有唯一的特定的优先级，用户可以创建任务的优先级为 1～OS_LOWEST_PRIO-2 。Maintask 的设计流程如图 5-3，主任务创建邮箱，用于接收其他任务发来的消息，根据消息管理、控制其他任务的运行。

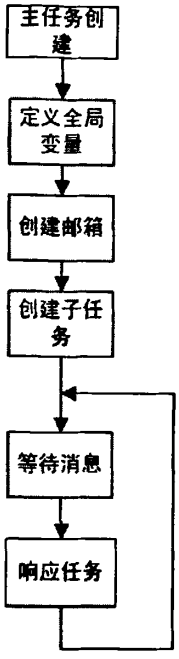


图 5-3 主任务流程图
Fig 5-3 flow chart of Maintask

5.2.2 GUI 任务

GUI 任务包括开机界面、控制界面和数据显示界面。

控制面板任务主要采用了对话框（Dialog）编程，用户可以输入的控制量包括：控制模式,包括在线整定模式及用户自定义模式。采用 radio 控件接受用户的输入，radio 控件是自上而下的小圆圈，依次对应 0 和 1 两个值。设定值，用户可以输入期望的信号量的值，PID 的整定参数以及自整定模式下的超调量与整定时间。采用 EDIT 控件，通过 EDIT_GetValue（）来返回输入框里的值。时间信息，采用 EDIT 控件，因为 S3C44B0 内部的 RTC 时钟寄存器默认是按照 BCD 码写入和读取的，因此，为了编程方便，在这里设置成十六进制显示。通信模式，主要包括端口选择，波特率，校验等。坐标设置，主要用于初始化 X，Y 轴要显示的数据最大值，及 X，Y 轴要显示的刻度等。

当用户输入信息完成后，点击 button"OK"后，触发回调函数的 GUI_KEY_ENTER 消息处理函数，该消息处理函数将用户的输入值读入到公共变量中，实现输入功能。完成后控制面板自我 delete，并向主任务发送包含包含输入量的系统信息，定义为一个数据结构 sysCFG。主任务再根据 sysCFG，创建与显

示相关的 displayCFG 结构体参数发送给 GUI 任务进行信号量的绘制：绘制坐标系，绘制曲线，显示状态信息等，然后创建邮箱，等待主任务发送的刷新数据，然后继续绘制曲线以实现显示的更新。

5.2.3 串口通信任务

S3C44B0 的 UART 单元可以在中断和 DMA 两种模式下工作，最高波特率为 115.2kbps。每个 UART 通道包含了 2 个 16 位 FIFO 分别提供接收了发送使用。

串口通信任务主要包括串口发送和接收。发送是指将系统的运行状态：控制模式、当前电压值、电压给定值、系统运行时间等信息定时发送给上位机软件；接收是指接收上位机的控制信息。本设计采用 UART0，波特率 115200，8 位数据位，1 位停止位，无校验。为了通信方便，定义了一组通信协议，如表 5-1 所示，

起始位	Flag	V1	V2	V3	V4	校验位

表 5-1 数据帧结构

Table 5-1 Data frame diagram structure

其中，起始位：0x02，表示数据帧头；Flag 位，用于表示数据帧里包含的信息；V1-V4：包括 4 个字节的数据信息；校验位：前六个字节的异或

比如 Flag=0 表示：设置电压，其中 V1~V4 是一个 Int32 的数据结构，按照字节顺序依次发送，上位机收到信息后解码得到设置电压信息。

UART 发送任务创建时获得一个指向 uartFrame 结构体的指针，其结构与数据帧结构类似：

```
typedef struct
{
int tasknum;//与任务优先级相同，用于主任务判断消息来源
int flag;//控制信息
byte[]  dataByte[4];//数据信息
}uartFrame;
```

UART 接收任务与 UART 发送任务相似，只不过 UART 接收任务收到上位机发送过来的数据后，首先进行信息校验，然后提取信息，向 maintask 邮箱发送。

UART 接收和发送任务的流程图如图 5-4。

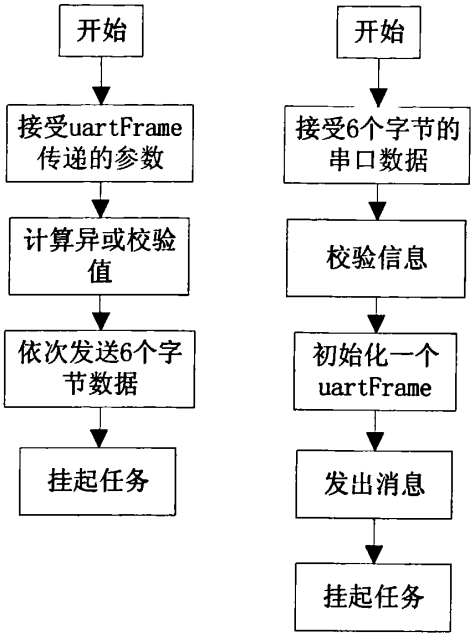


图 5-4 UART 发送和接收任务流程图
Fig 5-4 UART send and receive tasks flow charts

5.2.4 实时时钟任务

RTC 任务主要用于记录运行时间。S3C44B0X 内部有一个实时日历时钟(RTC)单元，需要外接一个 32.78kHz 的晶振。实时时钟可以提供系统运行的日期和时间(包括年、月、日、星期、时、分、秒)。在读写程序中，通过设置 RTCON 寄存器的第 0 位来表示要“读”还是“写”RTC 模块中的寄存器。分别从 BCDSEC，BCDMIN，BCDHOUR，BCDDAY，BCDDATE、BCDMON、BCDYEAR 寄存器中读取数据来显示秒、分、小时、日期、星期、月和年。由于是对多个寄存器同时读取，因此很可能产生 1s 的偏离。例如，如果用户读取寄存器 BCDYEAR 到 BCDMIN，假设结果为 1959 年、12 月、31 日、23 点、59 分。在用户读取 BCDSEC 寄存器时，如果结果是 1~59，肯定没有问题；但如果结果为 0，那么很有可能年、月、日、时、分已经变成了 1960 年 1 月 1 日 0 时 0 分。解决的方法是，当读取到的 BCDSEC 等于 0 时，用户应该再读取一次 BCDYEAR 到 BCDSEC 的值。编写完 RTC 任务后，就可以与上位机进行同步，记录系统运行的状态。RTC 任务比较简单，向主任务发送一个 osTime 的结构体指针，然后将自身挂起 1 秒钟。

5.2.5 键盘任务

uC/GUI 软件中可以检测键盘输入是因为包含了对于键盘消息的处理函数，位于 gui/core/GUI_exec.c，函数原型为 void Key_Exec(void)。

键盘任务需要修改键盘的硬件驱动部分，然后判断按键信息，发出对应的 GUI 按键信息。由于 uC/GUI 加入了对操作系统的支持，因此只需建立一个单独的任务，该任务只有一个函数调用，即调用 GUI_Exec(void)来完成的。GUI 的窗口管理器 WM 收到键盘消息后，调用相应的回调函数来相应用户的输入。这里主要介绍一下矩阵键盘扫描程序的设计。

键盘扫描程序流程图如图 5-5。按键被按下时，与此键相连的行线电平将由与此键相连的列线电平决定；而行线电平在无按键下时处于高电平状态。如果让所有的列线处于高电平，那么键按下与否不会引起行线电平的状态变化，始终是高电平，所以，让所有列线处于高电平是没法识别出按键的。让所有的列线处于低电平，这样，按键所在的行电平将被拉成低电平，根据此行电平的变化，便能判断此行一定有键被按下，为了进一步判定到底是该行的哪个按键被按下，应该在某一时刻只让一条列线处于低电平，而其余所有列线处于高电平。

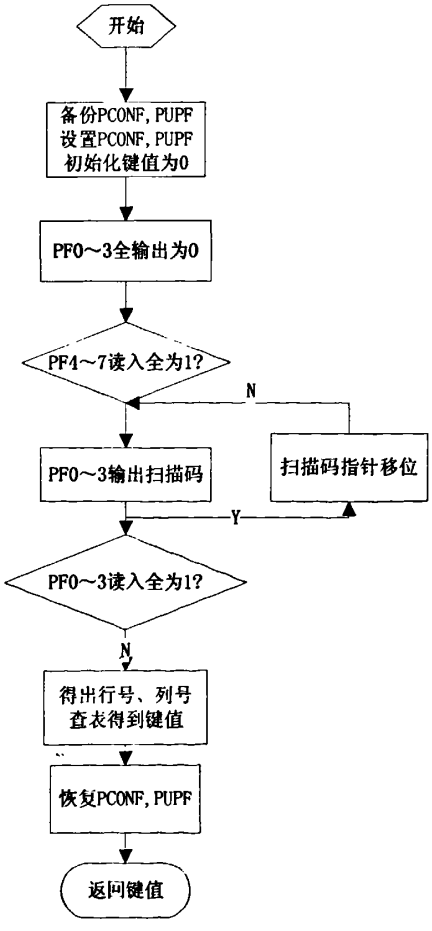


图 5-5 键盘扫描驱动程序流程图
Fig 5-5 Keyboards scan flow chart

5.2.6 基于 IIC 总线的数据保存任务

数据读取与保存主要是指将用户数据以及控制变量保存在非易失性存储器 EEPROM 中，数据保存任务优先级最低，挂起时间最长，因为它主要是为了防止突然断电等突发事件用于保存用户的配置信息。

需要保存的变量有：

osTime：系统运行时间；

pidInfo：PID 配置信息；

displayCFG:当前显示配置信息；

数据保存的流程图如图 5-8。

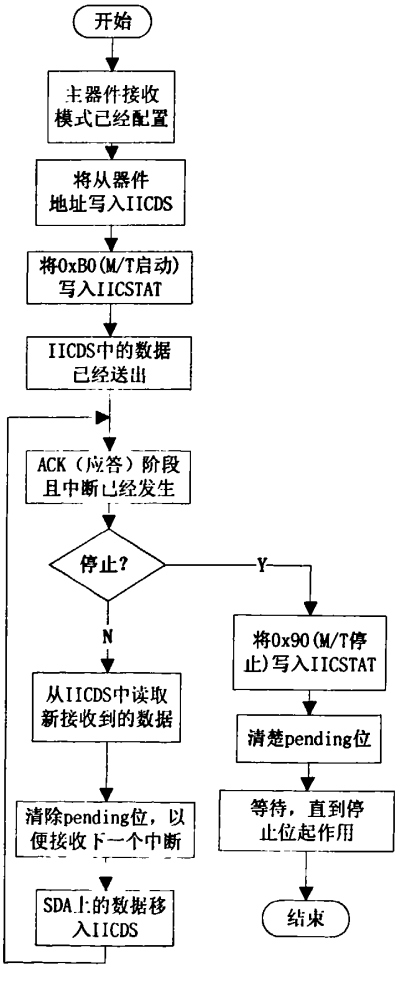


图 5-8 EEPROM 接收数据流程图
Fig 5-8 EEPROM receiving data flow chart

5.2.7 AD 转换任务

S3C44B0X 内部具有一个逐次逼近型 8 路模拟信号输入的 10 位 ADC，内部结构包括模拟输入多路复用器（AMUX）、自动调零比较器（COMP）、时钟产生器（PSR）、10 位逐次逼近寄存器（SAR）和输出寄存器（ADCDAT）。A/D 转换的设置比较简单，主要通过配置 ADCCON 寄存器。A/D 转换流程图如图 5-9。采集到的数据供 PID 任务调用。

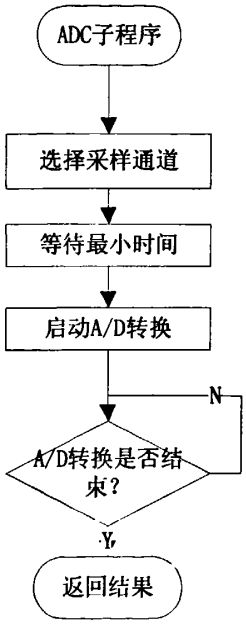


图 5-9 A/D 采样流程图
Fig 5-9 A/D sampling flow chart

5.2.8 PWM 定时器任务

S3C44B0X 具有 6 个 16 位定时器，其中定时器 0，1，2，3 都具有 PWM 输出功能，每个定时器都具有一个倒计时器，实际上就是一个通过定时器时钟源驱动的 16 位倒计时寄存器 TCNTn。当倒计时值减少到 0，定时器中断请求就产生了，这个中断通知 CPU 定时器定时已经完成。另外 S3C44B0X 内部具有双缓冲器以及自动重载功能，双缓冲器是指定时器计数缓冲区寄存器 TCNTBn 和定时器比较缓

冲区寄存器 TCMPBn，它们都分别具有一个初始值，用来载入到定时器计数寄存器 TCNTn 以及定时器比较寄存器 TCMPn 中与倒计时值相比较。

PWM 脉冲频率由 TCNTBn 决定。PWM 脉冲宽度值则由 TCMPBn 的值来决定，PWM

输出的流程图如图 5-10。程序中设定 PWM 输出频率为 16kHz，调节范围为 5%~95%。

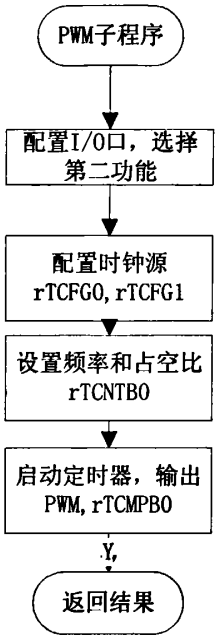


图 5-10 PWM 输出流程图
Fig 5-10 PWM flow chart

5.3 上位机

5.3.1 基于 C#的实现方式

控制系统需要有必要的通信功能，实现用户在电脑上对硬件的管理和实时监控。本设计中上位机软件的设计主要包括串行通信、曲线绘制等。Net Framework 是 Microsoft 为开发应用程序而创建的一个新平台。采用了 C#语言。C#编程语言是由微软公司的 Anders Hejlsberg 和 Scott Willamette 领导的开发小组专门为.NET 平台设计的语言，从 C，C++和 Java 发展而来，它采用了这三种语言最优秀的特点，并加入了它自己的特性。C#是事件的驱动的，完全面向对象的可视化编程语

言，程序员可以方便的建立，运行，测试和调试 C#程序，这就将开发一个可用程序的时间减少到不用 IDE 开发时所用时间的一小部分。

NET Framework 是微软的几个开发团队一起努力发展的成果，最主要用来产生一个可以用来快速开发、部署网站服务及应用程序的开发平台。这个架构是两个项目的结果：第一个项目的目的是用来改善 Windows 作业平台上的程序开发，特别是改善 COM（Component Object Model，组件对象模块。一种微软所制定的软件技术；让对象的功能可以被其它软件所叫用，可以让组件重复使用、容易更新及维护）；第二个项目则是制作一个以发展服务（Service）软件为目标的开发平台。

5.3.2 串口通信

在.Net Framework 里，主要通过 serialport 类来进行串行通信，实现步骤为：

1. 引入这个 reference:

```
using System.IO.ports;
```

2. 建立新的对象：

```
SerialPort myPort = new SerialPort();
```

3. 初始化端口：

```
myPort.PortName = "com1"; //端口名
myPort.BaudRate = 115200; //速率
myPort.DataBits = 8; //数据位
myPort.StopBits = StopBits.One; //停止位
myPort.Parity = Parity.None;  //奇偶校验
myPort.RtsEnable = true; //  打开 RTS ，
```

4. 串口接收数据

串口接收数据主要通过调用 serialPortserialPort1_DataReceived 事件，接收到有效数据后，根据表 5-1 的数据帧结构进行数据解码，得到和硬件系统相同的配置、状态和控制信息。另外需要注意的是：由于串行接口是按照字节发送和接收的，因此，上位机接收到的 V1～V3 还不能够直接调用，需要将它们转换成两个 int16 型的变量 Vh，Vl，然后再进行处理。

5. 串口发送数据

串口发送数据主要包括将上位机设置的控制信息按照表 5-1 的格式发送给硬件系统，包括电压设定、PID 参数、坐标系配置、时间等信息。主要调用 serialPort1.Write(myByte, i, 1)方法，另外在发送之前也需要计算校验位的值。

5.3.3 曲线绘制

GUI+是.Net Framework 的绘图技术，使用三个坐标空间：世界、页面和设备。世界坐标是用于建立特殊图形世界模型的坐标系，也是在 .NET Framework 中传递给方法的坐标系。页面坐标系是指绘图图面（如窗体或控件）使用的坐标系。设备坐标系是在其上进行绘制的物理设备（如屏幕或纸张）所使用的坐标系。当调用 myGraphics.DrawLine(myPen, 0, 0, 160, 80) 时，传递给 DrawLine 方法的点（(0, 0) 和 (160, 80)）位于世界坐标空间内。在 GDI+ 可以在屏幕上绘制线条之前，坐标先要经过一系列变换。一种称为“世界变换”的变换可将世界坐标转换为页面坐标，而另一种称为“页面变换”的变换可将页面坐标转换为设备坐标。

如前所述，GDI+绘图主要包括坐标变换，然后调用 GDI+绘图函数即可。坐标变换主要包括尺度变换、平移和翻转。系统定义的显示区域 picturebox 为 640×480，这样上位机显示的效果与硬件系统基本相同。坐标系显示效果如图 5-9。

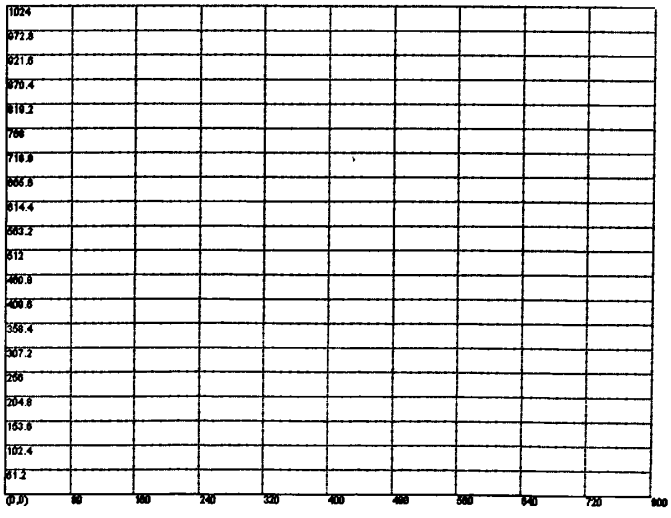


图 5-9 实际坐标效果图
Fig 5-9 Coordination system

5.4 开发流程

ARM 的调试与开发与一般嵌入式开发一样，需要经过编译-连接-调试-下载阶段，在此不再细述。本次开发所用软件为 ARM Developer Suite 1.2 和 CodeWarrior for ARM。

6 PID 控制系统的实际应用

6.1 主电路及触发电路

本章主要介绍智能PID控制系统在三相全控桥式整流电路的实际应用。图6-1，6-2 为主电路及触发电路。主电路通过三相调压器输入三相交流信号，输入电压范围为 0~380v，经过全桥整流电路，输出一路直流信号，负载为电阻箱，阻抗可调。

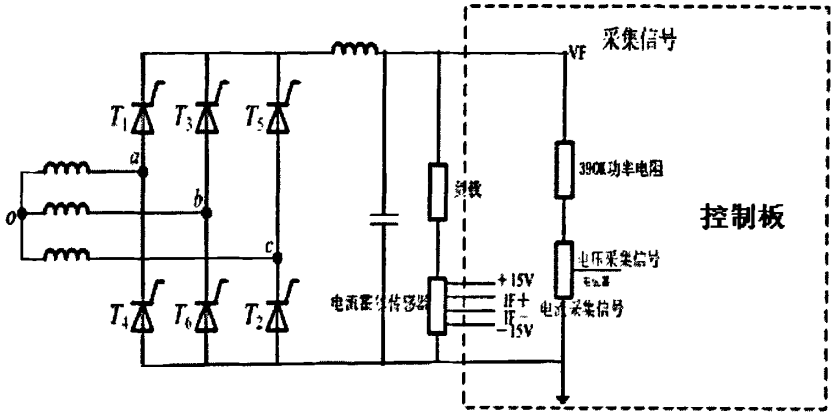


图 6-1 主电路示意图
Fig 6-1 Main circuit diagram

触发电路 CA6100 可控硅触发板如图 6-2 所示，是以 40 芯 CMOS 大规模集成电路为核心，利用锁相环技术和多芯片合成技术，根据压控振荡器锁定的三相同步信号间的逻辑关系设计出的一种可控硅触发系统。用于控制可控硅的门极延迟触发角，从而实现移向控制。0-5v 的直流输入电压信号，可以控制输出脉冲的移向范围从 5 度到 175 度可调。其中 J1、J2 为触发脉冲输出端。J1 的 1 和 2 、4 和 5、7 和 8 分别接到主回路的+A、+B、+C 三个可控硅的门极和阴极；J2 的 1 和 2 、4 和 5、7 和 8 分别接到主回路的-A、-B、-C 三个可控硅的门极和阴极。J5 的 1 和 5 端引入 220V 交流电源。J3 的 1、2、4 端分别和通用 PID 控制器的使能信号、地和控制输出信号相连。三个指示灯分别代表脉冲禁止、交流缺相、电源显示。

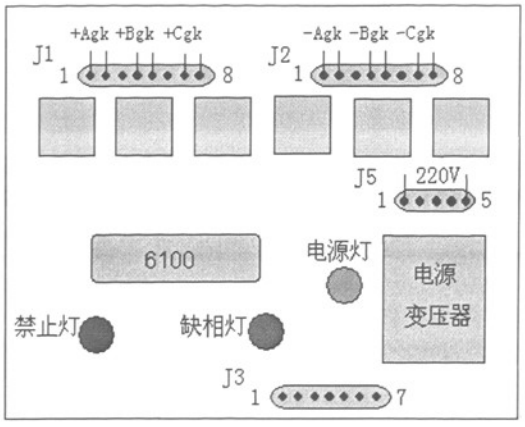


图 6-2 CA6100 触发板示意图
Fig 6-2 CA6100 Trigger circuit

6.2 接口电路

主电路直流电压信号采用直接分压的方法反馈到控制系统的 ADC0 端（0~2.5v），输出直流电流信号经过霍尔电流传感器得到 0~2.5v 的电流信号，接控制器的 ADC1 端。系统控制信号输出采用 PWM（16kHz），经一阶低通滤波得到基波直流信号，作为主电路触发板的控制信号输出。触发板需要 1 路使能信号用于使能/禁止系统运行。主电路及控制电路与通用 PID 控制器之间的硬件接线原理如图所示。

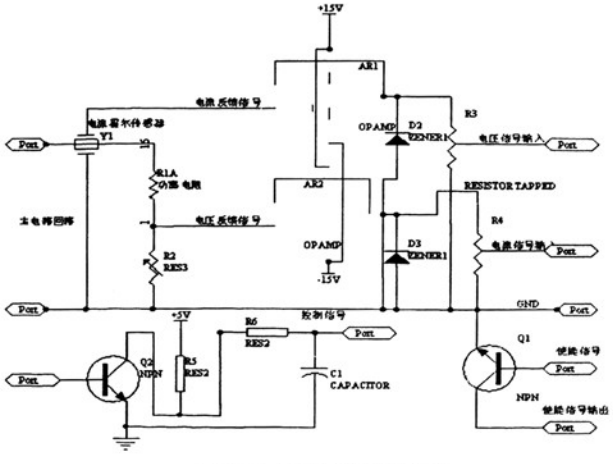
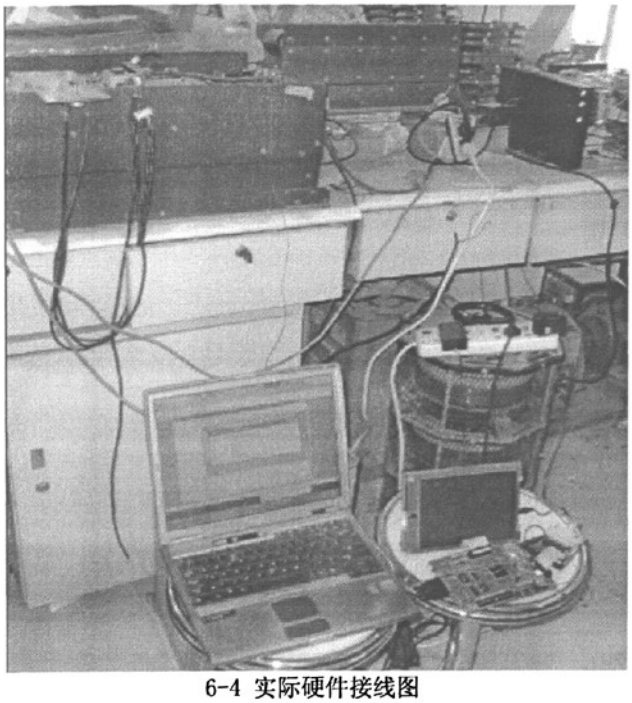


图 6-3 硬件接线原理图
Fig 6-3 Hardware connection schematic diagram

其中，主电路电压信号经过功率电阻与小电阻分压后，经过运方输出 0~2.5v 的反馈信号，接到 PID 控制系统的 AD 输入。分压比为 400：1。这样采样信号输

出的 2.5v 对应于主电路输出 1000v，与显示系统 y 轴最大值对应，便于测量。

PID 控制器与触发电路的连接主要有：使能信号，对应于触发电路板 J3 的第 1 个管脚，PID 控制器输出一路使能信号，高电平信号经过三极管射击输出将关闭触发板的触发信号；共地端，对应于触发电路板 J3 的第 1 个管脚；0~5v 模拟控制信号，由 PID 控制器的 PWM 输出在经一阶阻容滤波，得到基波分量，作为触发板的控制信号。实际硬件连接如图 6-4。



6-4 实际硬件接线图

Fig 6-4 Hardware connection

6.3 调试过程

6.3.1 反馈回路调节

在进行闭环调节之前，首先要保证反馈电压信号的正确采集。采用电位器输出 0 到 5v 模拟信号到触发电路 J3 的控制信号，调节分压器输出 2.5v 的信号。使主电路正常工作，此时测量主电路的电压信号，调节电压分压电路的阻值，使得主电路输出电压与采集回路输出电压之比为 400：1。主电路电压输出波形如图 6-5 所示，调节分压电位器使得输出到控制器的电压信号为如图 6-6 所示。

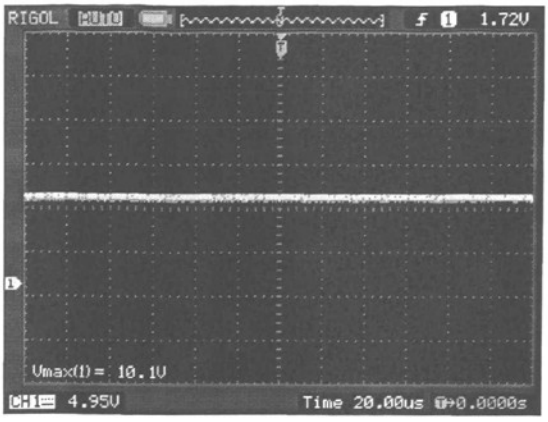


图 6-5 主电路电压信号波形
Fig 6-5 Voltage wave of main circuit

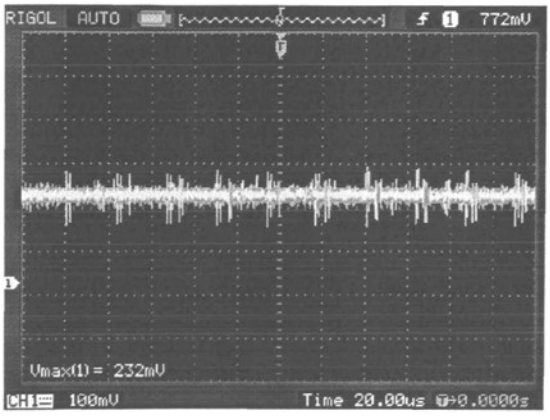


图 6-6 反馈电压信号波形
Fig 6-6 Voltage wave of voltage signal

其中，示波器探头已经将信号缩小了 10 倍，因此，实际电压为 100v。采样电压为 232 mv。分压比为 100/0.232=430。满刻度 2.5v 对应主电路电压为 2.5*430=1075v。逐步调节电阻，使得满刻度所对应电压为 1000v。与 LCD 显示的最大值相等。

6.3.2 调试系统

控制系统上电后，按任意键，进入控制面板。如图 6-7，设置对应的参数如图所示。设置当前工作模式为：稳压限流，电压设定值为 200v，电流限定值为 25A，规一化 PID 参数为 0.55。通信模式采用 COM，115200，无校验。曲线绘制:y 轴最大值 1000,x 轴点数 2000；x 轴刻度数 40，y 轴刻度数 20；设置完毕后按确认键。

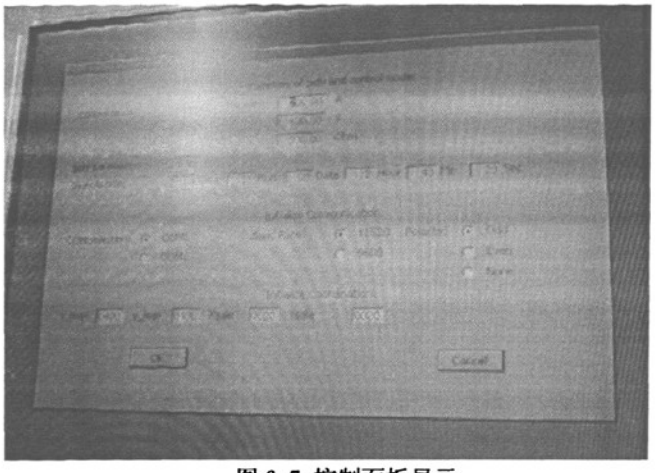


图 6-7 控制面板显示
Fig 6-7 Start up interface

图 6-8 系统显示效果图
Fig 6-8 System display

运行系统，LCD 及上位机显示效果如图显示如图 6-8 。电脑上所绘制的是串行接口发送过来的电压值信息，LCD 液晶屏显示主电路电压曲线，实际设定电压值为 200v。如图，系统超调量约为 20v。系统输出 PWM 波形如图 6-9。可见 PID 控制器已经开始了调节作用。调节作用良好。当主电路电压改变时，PWM 改变迅速。图 6-10，图 6-11 为将 6100 板作用取消时 PWM 的输出值，可见当设定值和反馈值都不变时，积分作用会将误差累加，PWM 输出极值，符合 PID 的理论分析。

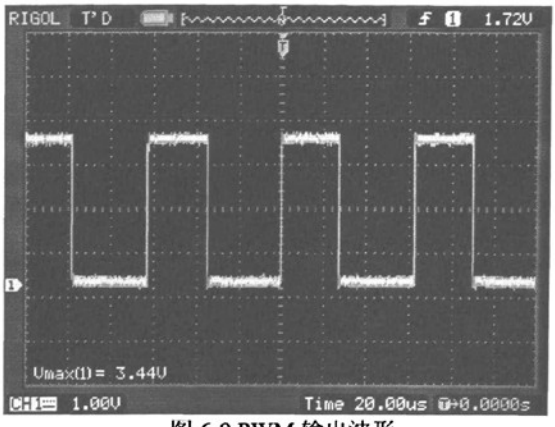


图 6-9 PWM 输出波形
Fig 6-9 PWM output

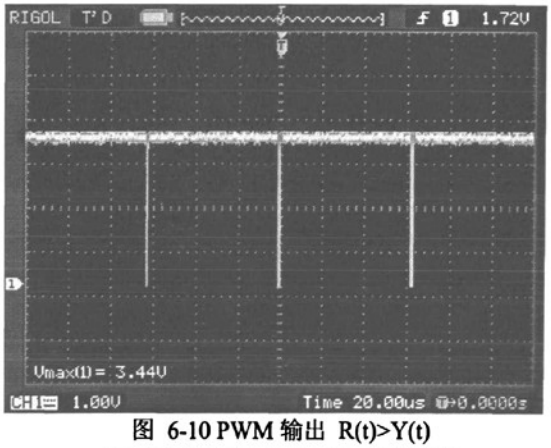


图 6-10 PWM 输出 R(t)>Y(t)
Fig 6-9 PWM output when R(t)>Y(t)

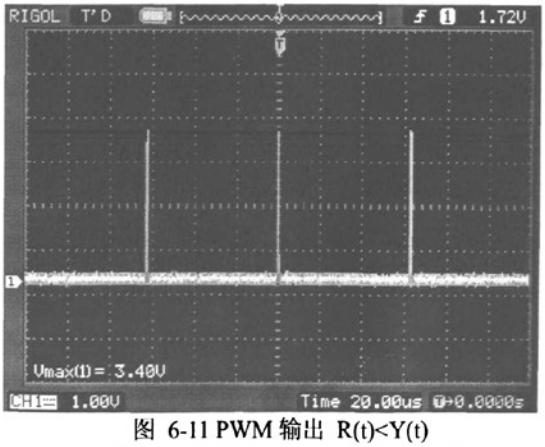


图 6-11 PWM 输出 R(t)<Y(t)
Fig 6-11 PWM output when R(t)<Y(t)

6.3.3 系统参数选择

(1) 采样周期 T

系统采样时间应满足香农定理，根据采样定理：

$$T \le \frac{1}{2f_{\max}}$$

其中 $f_{\max}$ 为被采样信号的最大波动频率。采样周期也要受计算机执行控制程序和输入输出所耗费的时间限制，系统的采样周期只能在 $T_{\min}$ 和 $T_{\max}$ 之间选择。三相桥式电路进行整流，输出直流电压的最大波动频率为 300Hz，采样周期的上限：

$$T_{\max} = \frac{1}{2f_{\max}} = 1.67ms$$

采样周期的下限由 S3C44B0X 系统软件执行时间决定，在 uC/OS II 中主要由任务挂起的最小时间决定，OS_Delay+算法执行时间。通过反复调试，最终选择采样周期为 $T = 0.5ms$，能使系统工作在稳定状态。

(2) PID 参数 Kp

控制系统采用规一化参数整定法，实际调试的时候采用工程法，先让 Kp 为一个比较小的值，0.1，然后观察主电路阶越响应情况，此时 LCD 显示的阶越响应波形如图 6-11。当 Kp=0.55 时，超调量较小，但是调节时间比较长。

由于输入信号存在高频干扰，因此采用了平均值滤波的方法，即 AD 采样一定次数，求其平均值，然后再送 PID 算法运算，目前采用的采样数为 10 次。

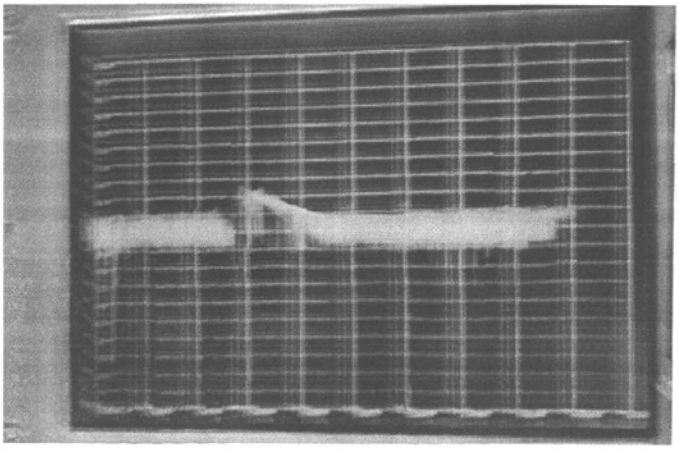


图 6-12 Kp=0.55 时的阶跃响应
Fig 6-12 Step response,Kp=0.55

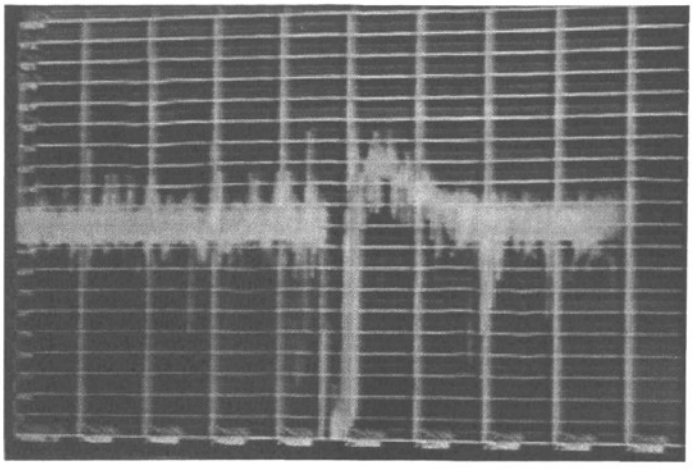


图 6-13 Kp＝0.1 时的阶跃响应
Fig 6-13 Step resonose,Kp=0.1

现场经过反复测量，设定 Kp 值为 0.55，此时系统的超调量，响应时间都比较合适。设计的控制系统可以根据不同的主电路调节 Kp 的值得到满意的调节效果。从图中可以看出系统的调节非常迅速，但是振荡现象比较严重。经分析，这是由于主电路与控制电路干扰较大，AD 采样芯片为 ARM 自带芯片，性能没有专用芯片采样能力强。编写了一个专门的程序去采集 AD 转换的数字量，发现 ARM 的 AD 采集的数字量中会出现离均值相差很远的值，就是这些值干扰了控制量，再加上 LCD 上一个格子代表 2.5mv，基本达到了预期的效果。

7 总结

PID 控制算法是迄今为止比较通用的控制策略，但 PID 参数的整定一般需要经验丰富的技术人员来实现，且阻容参数采用经验法选取，实际控制效果不是太理想。本文的研究目的就在于解决上述问题。提供一个可操作性强，人机界面比较亲切的 PID 控制系统，而且该系统采用了 RTOS，可以将现有功能模块化，任务分配更加迅速合理，达到智能化、通用化的要求。

本设计是在实验室已有项目：PIC 单片机与模拟电路和数码管组合设计的一款 PID 控制器的基础上设计的，同样用于控制三相全控桥的触发板。原项目功能单一，很难具有通用性，另外采用大循环的模式，系统的采样时间很难修改，数码管显示的信息也较为单一。本设计与模拟 PID 调节器与单片机的组合相比，基于 ARM7 内核及 RTOS 内核和 GUI 软件包的智能控制系统具有明显的优势，特别是采用了 640×480 的 STN 液晶显示，使得用户可以方便的观察系统的运行状态。另外 PID 参数可以灵活调节，对于有经验的用户来说，他可以很容易的调节 PID 参数以适应不同的控制性能要求，具有良好的控制效果。由于 PID 采用单独的任务函数，可以方便的设置 PID 调节的采样时间（及任务挂起的时间）。另外，该设计还具有很强的延伸性能，比如，可以用触摸屏代替矩阵键盘，用以太网代替串行接口，以实现更强大的功能，另外，本设计的不足之处是 S3C44B0X 自带的 ADC 模块采样精度不够，使得控制性能有所下降，不过由于采用了 uC/OS II 的实时操作系统，可以很方便的配置 PID 的结构，选择合适的输入输出通道，实现更强大的功能。

在系统调试的时候，由于采用了开发板，信号采集和输出通过搭建外部电路实现，使得系统的抗干扰能力较差，但是由于本设计主要专注于嵌入式操作系统及 GUI 软件包在电力电子电路控制领域的试探性研究，硬件的表现还是比较令人满意的。在操作系统的移植上，需要注意的是采用汇编程序打开中断与关闭中断的部分以及堆栈的生长方向。另外在进行任务函数设计的时候，特别需要注意合理分配任务的优先级机挂起的时间，确保最重要的任务优先运行，这里就是 PID 任务，特别需要保证采用的周期性。另外 GUI 刷新任务挂起时间要短，否则就会造成 LCD 屏的闪烁。另外就是要保证系统的逻辑正确，在出现控制问题后，及时切断使能信号。另外系统需要改进的地方就是采用精度高、采样率高的专用 AD、DA 芯片，这样，系统的性能将会有很大的提高。在进行数据显示的时候，GUI 软件包本身存在 bug，调试花费了很多时间。在 GUI 软件包正常运行后，进行的曲线绘制和基于 WM 的对话框程序设计，曲线绘制主要调用 GUI_DrawLine()命令，这与 C++程序设计比较类似，需要注意的就是系统默认坐标系和用户显示坐标系的不同，因此需要进行尺度变换和旋转变换。另外在进行基于 WM 的设计的时候，

需要注意的是在 PC 机上 Visual C++开发环境下编写的程序，不一定能在 ADS1.2 上，或者说在硬件系统上正常运行，因为 PC 机可以有模拟鼠标按键的功能，而在实际硬件中，由于没有采用鼠标作为输入设备，因此，必须将消息处理主体定义的按键处理程序上。

下一步要做的工作需要进一步研究 PID 自整定控制技术，提高控制系统性能。实现 PID 参数的自动整定。同时采用更高精度的 AD 芯片，将信号采集电路和控制电路集成，减小不必要的干扰。还要将 GUI 界面做的更加完美，便于用户使用。

参考文献

[1] [美]Jean J.Labrosse 著,邵贝贝等译.嵌入式操作系统 uC/OS-II(第二版).北京航空航天大学出版社.2003.93-98

[2] 于海生等著.微型计算机控制技术.清华大学出版社.1999.84-86

[3] 胥静.嵌入式系统设计与开发实例详解.北京航空航天大学出版社.2005.111-113

[4] 任哲.嵌入式实时操作系统 UC/OS-II 原理及应用.北航空航天大学出版社.2005.12-15

[5] uC/GUI user manual v3.32

[6] uC/OS II user manual v2.52

[7] 谭浩强.C 语言程序设计.清华大学出版社.1999

[8] 任哲.嵌入式实时操作系统 UC/OS-II 原理及应用.北航空航天大学出版社.2005.44-45

[9] 刘滨,刘兵,赵艳华.基于μC/GUI 的嵌入式图形界面设计.液晶与显示 2005.10

[10] Koivo H N,Tanttu J T.Tuning of PID controllers:survey of SISO and MIMO techniques,In;Preprints of IFAC international Symp . on Intelligent Tuning and Adaptive Control Session, Singapore ,1991

[11] 郝德宁. 富士通微控制器的开发应用. 北京：仪表仪器用户.08.12

[12] 三星公司.S3C44B0X Data Sheet 数据手册

[13] 王伟等，先进 PID 整定方法，自动化学报，2000.5.85-87

[14] 李向阳,曾旖,奚大顺.在μC/GUI中实现汉字显示.单片机与嵌入式系统应用.2005.5.76-77

[15] 李丽,郝继飞,杨莹,陈章祥. ARM嵌入式系统Bootloader的设计与分析.自动化与信息工程.2007年第2期.44-46

[16] 郭光名.数字PID的设计与调试.机电一体化.2001年第4期. 69-71

[17] 历风满.数字PID控制算法的研究.辽宁大学学报.2005年第32卷第4期. 367-370

[18] 叶彦斌.基于ARM7的电路检测平台数据采集系统.自动化与仪表.2007年第4期.71-75

[19] 刘广路,郝红旗. uCOS-Ⅱ在ARM7上的移植. 电脑知识与技术.2007年第3期.1316-1317

[20] 张泰山，计算机控制系统.北京：冶金工业出版社，1986

[22] 胡庆武,崔贤玉. 基于ARM的嵌入式系统BootLoader的编译与启动分析.科学技术与工程.2007年第14期. 3571-3574

[24] Astrom K J.Toward intelligent control.IEEE control systems Magazine,1989(April)

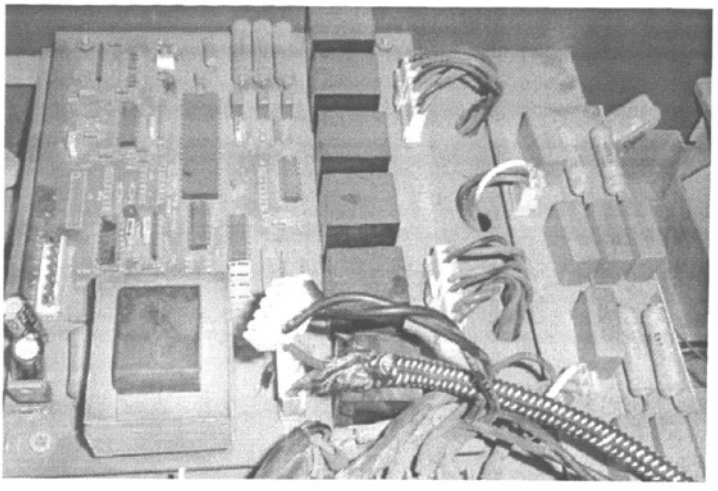
[25] 江敏. 基于S3C44B0X的嵌入式系统BootLoader的设计与实现.工矿自动化. 2007年2期.66-68

[26] 白雪峰,李沛. 三相全控桥式整流电路实验装置的研制. 现代电子技术.2006年第15期.83-84

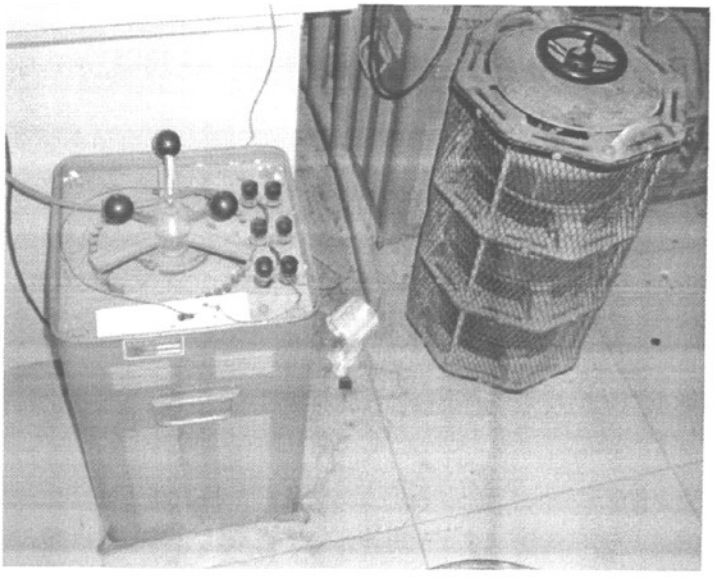
[27] 陈洁. 基于单片机的三相全控桥整流电路触发电路设计.机械工程与自动化.2005年第4期.82-83

附录 A 实物图

触发电路/6100

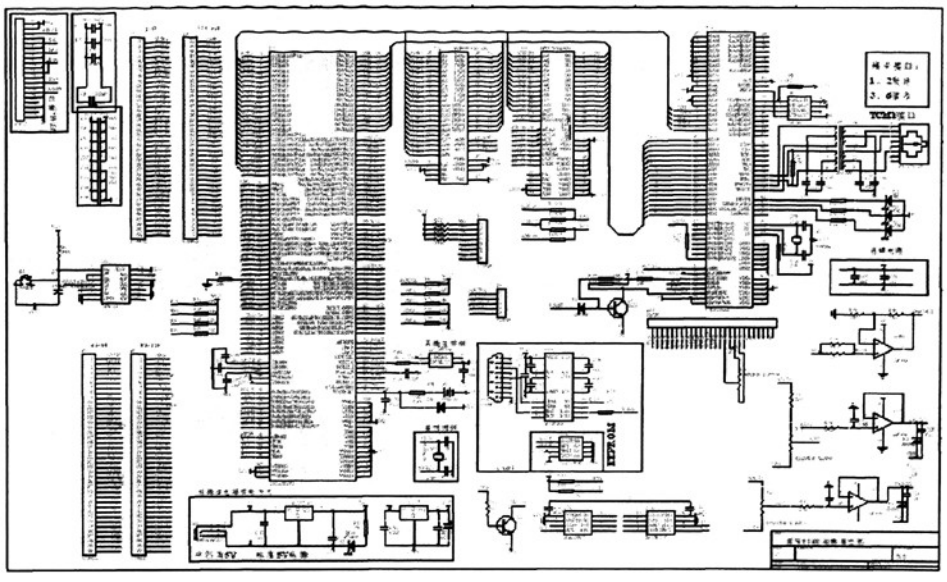


实验平台（三相调压器及负载）

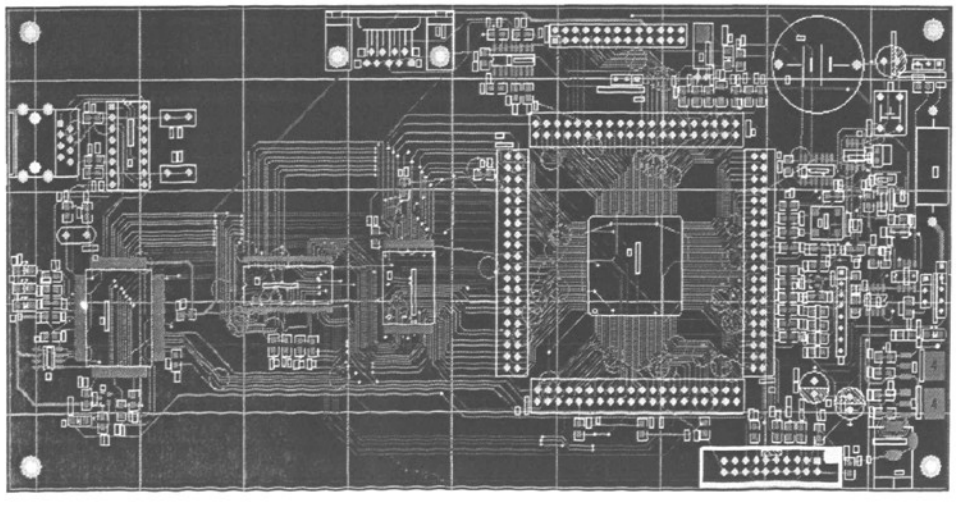


附录 B 控制系统原理图及 PCB

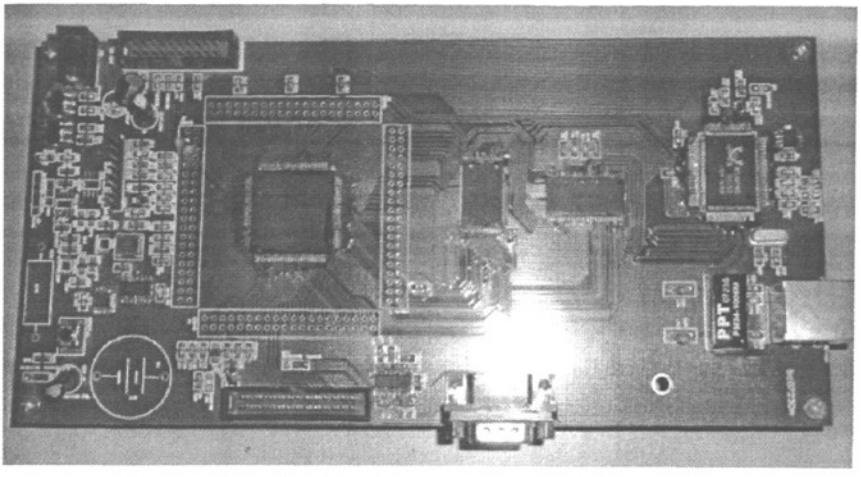
原理图



PCB



附录 C 智能 PID 控制系统板



作者简历

郑飞，男，生于1984年11月15日，河北邯郸人，汉族。

2003年9月考入北京交通大学电气工程学院攻读电气信息类学士学位，期间曾获得三等学习奖学金。2007年7月毕业，获工学学士学位，同年考入北京交通大学电气工程学院攻读电力电子与电力传动专业硕士研究生。主要从事嵌入式系统方面的研究。

研究生期间发表论文有

《嵌入式PID控制器及显示系统的设计》，发表于《微计算机信息》2010年1月