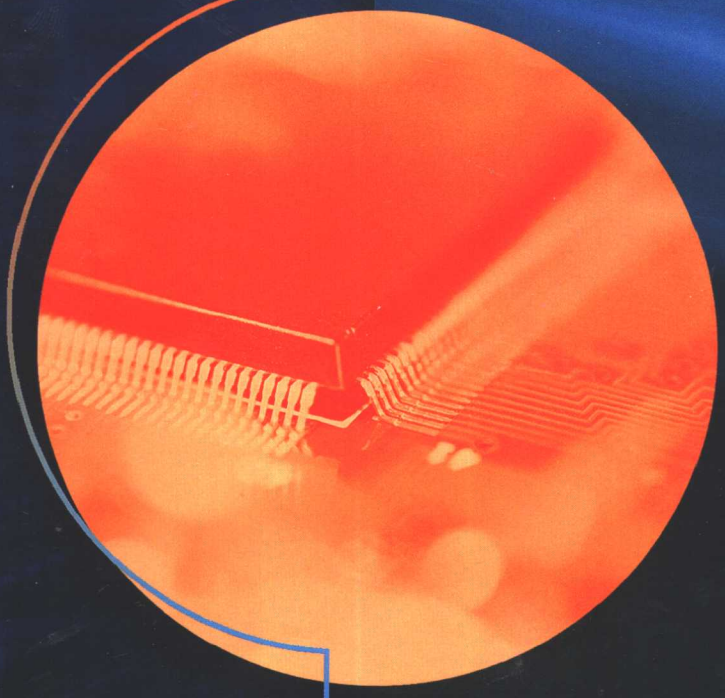
HZ BOOKS

嵌入式系统技术丛书

嵌入式系统的C程序设计

C Programming for Embedded Systems



(加) Kirk Zurell 著

艾克武 张剑波 艾克文 等译

附赠 CD-ROM

机械工业出版社
China Machine Press

RD BOOKS

嵌入式系统技术丛书

嵌入式系统的C程序设计

（加） Kirk Zurell 著

艾克武 张剑波 艾克文 等译

机械工业出版社
China Machine Press

当今嵌入式处理器的生产已远远超过了为桌面系统所设计的处理器。本书通过介绍8位微控制器（MCU）的开发原理和C语言的相关知识，向读者展示了如何用C语言编写8位嵌入式MCU程序，内容涉及数据类型和变量、函数库、优化和测试嵌入式C语言程序等。利用贯穿全书的样例工程，揭示了特定设备硬件开发的详细细节，以及用C语言进行嵌入式程序设计所带来的快速开发、增强的可移植性和再利用等优越性能。

本书适合于从事嵌入式研究的相关技术人员使用。

本书版权登记号：图字：01-2001-1109

图书在版编目（CIP）数据

嵌入式系统的C程序设计/（加）泽雷尔（Zurell, K.）著；艾克武等译. -北京：机械工业出版社，2001.10
（嵌入式系统技术丛书）
书名原文：C Programming for Embedded Systems
ISBN 7-111-09309-7

Ⅰ.嵌… Ⅱ.①泽… ②艾… Ⅲ.C语言-程序设计 Ⅳ.TP312

中国版本图书馆CIP数据核字（2001）第067257号

机械工业出版社（北京市西城区百万庄大街22号 邮政编码 100037）
责任编辑：马 珂
北京第二外国语学院印刷厂印刷 · 新华书店北京发行所发行
2002年1月第1版第1次印刷
787mm × 1092mm1/16 · 7.5印张
印数：0001-5 000册
定价：29.00元（附光盘）

译 者 序

当今工业界包括家电领域的自动化、信息化已经发展为一种浪潮，对运行于微处理器的嵌入式系统的编程人员的需求显得十分迫切。嵌入式系统编程已成为软件业当前的新热之一，但目前国内介绍嵌入式系统开发的书籍并不多。本书作者Kirk Zurell是一位从事十多年嵌入式系统编程工作经验丰富的技术作者，他的书由浅入深地介绍了8位嵌入式系统C语言的开发方法。本书是一本在嵌 入式系统编程方面很有影响的、很值得一读的好书。

本书有两个特点：第一，行文言简意赅，作者将大量的技术和知识融入到尽可能少的描述中进行说明，而读者依然能深刻地理解其含义；第二，本书实用性很强 ，附有应用样例工程的详细介绍和大量代码。本书可作为大专院校的基础教材，对从事嵌入式系统开发、科研的工程技术人员也有重要参考价值。

参加本书翻译工作的人员有：艾克武、张剑波、艾克文、蒋心晓和李首幸。限于译者水平，译文难免有不妥之处，恳请广大读者朋友批评指正。

艾克武

2001年7月

目　录

第1章 简 介

1.1 本书的作用

本书提供了使用C编程语言进行微控制器程序设计的一个完整的中级讨论，覆盖了设计嵌入式环境所需对C的改编，以及一个成功开发工程的通用组成部分的全部内容。

C是编写基于32位内核的较大微控制器（MCU）所选择的语言。这些微控制器通常由它们的通用微控制器衍生而来，并且同通用微控制器一样，既复杂又功能丰富。因此，对于这些MCU，C（和C++）编译器是必需的，也是很容易得到的。

相反，选择采用8位控制器的设计者经常求助于汇编语言的手工编码。虽然用于精确控制的手工汇编程序设计从来都不会过时，但也不会推动降低成本。因此，即使在8位MCU的有限资源里，编译高级C语言仍然有许多优势。

- 对如16位或更长的数据类型的算法之类的重复编码任务能自动生成代码。
- 硬件特殊性的直观处理。对一个串行闪存设备的读或写能用C语言表达为一个简单的赋值语句，尽管存储操作需要一些编码。
- 平台独立性。C带给桌面计算的跨平台能力对目前市场上的8位微控制器领域也是同样适用的。

本书将展示怎样用C语言编写8位嵌入式MCU程序。我们希望您不仅熟悉C，同时还具备有关微控制器程序设计更深层次的知识。

本书的主要样例工程是计算机控制的自动调温器。从一个最初规范开始，我们用与其他任何消费品或控制产品相同的方式逐步求精和增加设备。软件开发是我们关注的焦点，我们将做出任何设计者将要做出的选择和权衡。

1.2 嵌入式系统中使用C语言的好处

在嵌入式系统中使用C语言的好处如下：

不会困于细节。8位微控制器不仅仅是外形小；微控制器只包括执行其限定的任务所必需的逻辑，而程序员的付出是“轻松”的。通过一个C编译器与这些有限的资源协调工作，有助于抽象体系结构和避免坠入操作代码序列和硅片故障的云里雾里。

学习可移植性的原理。嵌入式应用程序是成本敏感的。为减少每个单元的成本，改变部件（或者甚至体系结构）可能是最大的激励。然而，修改汇编代码使得针对一种微处理器编写的程序能够在不同的处理器上运行，这样降低成本可能打消做出这种改变的任何念头。

通过传统编程降低成本。本书强调C语言代码能够概括微控制器特征。与特定硬件实施相关的细节能够被放置在不同的库函数和头文件中。使用C库函数和头文件可保证应用程序源代码能

够针对不同微控制器目标重新编译。

花费在算法设计上的时间更多而在实现上的时间更少。C是一种高级语言，使用它能够快速而轻松地编写应用程序。C的表达范围是简明而强大的。因此，用C语言编写的每行代码能够代替多行汇编语言代码。调试和维护C语言代码将比汇编代码容易得多。

1.3 本书概览

确定软件开发目标是第1步，将在第2章中论述。它包括有关对高效软件开发至关重要的预设计文档规则的嵌入式注解。

第3章为以前没有涉及过8位微控制器的读者提供一个由浅入深的介绍。

有了一个好的计划和关于中央控制器的深入知识后，设计过程（在第4章论述）把以前的评估都最后确定下来。与实现自动调温器有关的处理器细节也在第4章介绍。

第5章详细描述了硬件的C语言表达。它汇集了编写程序源码所必需的所有设置。

第6章提供对嵌入式数据的深刻剖析。变量存储修饰符near和far在运行微软视窗的Intel PC上和在运行您的代码的嵌入式处理器上将代表不同的事物。

第7章讲述C语句，提供关于嵌入式的函数、语句和操作符的信息。

第8章介绍函数库。即使在只有很少ROM和有极其特定工作要做的环境里，预先编写的函数库也会带来很大的帮助。

第9章提供关于代码优化方面的深入知识，并帮助您彻底测试您创造的产品。

第10章总结了样例工程的更多信息。尽管某些信息已经在本章前面出现过，但它包括以前没有讨论的内容。

1.4 修改和补充信息

如果需要查找关于自动调温器工程的更多信息，请通过web咨询我们的补充信息：http://www.bytecraft.com/embedded_C/。

第2章 问题规范

问题规范是设备和软件将要解决的问题的最初文档，它不应当包括任何具体设计难题或者产品解决方案，其主要的目的是详细解释程序将要做什么。

当然，正如在这个星球上有许多工作场所一样，处理工程计划的方法也有许多种。即使在那些最标准化了的阶段，也体现出不同的风格或者不同的次序。下面各节之所以被包含进来是因为它们增添了嵌入式领域的信息，有的则特别地适合样例工程。

2.1 产品需求

通常，这个文档是从用户的视角来写的，由一系列的用户需求组成。就针对一单任务设计的嵌入式系统的情况而言，应相当明确和肯定产品预期的功能范围。

有关硬件的总体决策是问题规范的组成部分，尤其在硬件将要受到充分控制的嵌入式工程中。

结果

- 程序将测量和显示当前的温度。
- 程序将按12或24小时制计数实际时间，并在一个数字显示屏上显示小时和分钟。
- 程序将接受时间设置并设置时钟。
- 程序将为三个日常使用周期接受和存储时间设置。
- 程序将控制制冷和制热之间的切换。注意，某些HVAC（heating ventilating and air conditioning，供热通风及空调）专家可能会想到偶尔在同一时间里既操作制冷又操作制热的需要，但这里的需求更接近于传统的自动调温器操作。
- 程序将比较当前温度与当前时间周期的设置，进而按需要打开或关闭外部的制冷或制热单元。
- 程序将抑制在一个短周期时间里两次改变外部单元的状态，这是为了使HVAC设备运行良好。
- 程序将在任何时间接受人工干预，进而立即关闭制热或制冷单元。

2.2 硬件管理

除了样例工程，本书并不直接涉及硬件。尽管如此，目标平台影响到产品的每个方面。它决定由编译器产生代码的容易程度，并且决定某些整体软件的设计决策。

如果软件开发者非常幸运，已经参与了硬件开发过程，那么影响设计的机会就太重要了且不要错过。请求的意向清单项目如下所示：

内置调试接口 现场可编程（field-programmability）的另一种方法也能满足需要。当一个设备必须在现场安装、定制或修补时，选用Flash-ROM芯片将比选用EEPROM或ROM设备更明智。

ROM代码保护 嵌入式处理器往往提供ROM代码保护，以防止偶然的检查。一个配置位禁止通过程序设计接口读ROM。尽管存在几个破坏这项保护的方法，但只有某个致命的对手才能够成功地读到您的程序。

合理的外设接口 电子线路设计时因贪图方便而影响I/O组织时，可能会迅速地降低软件性能。首选处理器有独立改变端口比特的位操作指令吗？多路复用接口将需要很多数据方向切换吗？

通过采用通用I/O端口线路和驱动器软件，某些外设能够被（软）复制。这节省了钱但增加了程序设计任务的复杂性。为模仿专用外设电路的逻辑信号，软件必须快速而重复地写比特序列到端口输出线路，这被典型地描述为比特风暴（bit-banging）。

标准函数库也许不会产生特别优化的硬件方案，但能够用减少的软件成本来补偿增加的硬件成本。

硬件设计的中心决策是处理器选择。处理器的选定是一个协商的决定，权衡的因素有：预期应用所必需的资源、芯片供应的价格和可用性以及可得到的开发工具等。针对微控制器的深入探讨请参见下一章。内存估计是问题规范的组成部分，因此RAM和ROM大小的估计在2.3.5节的资源管理中讨论。

结果

本书不涉及硬件管理，但本书包括了实现信息设置的硬件的某些样例产品规范信息。

表2-1 初始硬件规范

管理因素	评 估
操作环境	• 局部环境
	• 中等功耗、中等噪声电子线路
	• 偶尔断电
接口	• 一个针对切换HVAC的多位端口：也许只有3个针脚是必需的
	• 一个针对显示屏的多位I/O接口
	• 一个针对键盘的多位I/O接口
	• 一个针对温度传感的A/D设备
	• 实时时钟源：1秒粒度
内存大小	（参见下文）
特殊功能	• 时钟/计数器或者实时时钟
	• NVRAM的使用依赖于处理器是否和怎样睡眠
	• 监视定时器也许是有用的
开发工具	• C编译器
	• 模拟器或仿真器
	• 开发板

2.3 软件计划

软件计划应该对程序设计语言的选择做某些说明。对于嵌入式系统，通常的开发语言有三种选择：机器语言、C语言或者像BASIC那样的某种高级语言。三种选择当中，C平衡了两个对抗的需求。

- 比起像BASIC语言那样的解释系统，C有接近手工编码的机器语言的性能。如果一个BASIC系统通过暴露指针或通过预编译源代码变得更加复杂，则在测试时的困难将与C不相上下。
- C提供了机器语言没有提供的设备独立性。如果用汇编语言手工编码一个程序，则会遇到随微控制器的改变而废弃全部代码的风险。用C编程在改变处理器时只需要付出很小的努力，在软件模块里修改一个头文件即可。

软件计划中的第1步是选择解决在问题规范中所描述问题的算法。各种不同的算法应当被考虑到，并且依照它们的代码大小、速度、难度以及维护成本作出比较。

一旦一个基本的算法被选定，整个问题应该被分解为多个更小的问题。家庭自动调温器工程很自然地细分为对应于下面每个设备的模块以及每个设备的每一种函数：

- HVAC接口
- 键盘
- LCD
- 温度传感器

从模块开始工作，可以写传统的伪代码。这有助于产生将要在代码中实现的标识符和逻辑段落。

从流程图着手把自然语言伪代码翻译为实际代码。在流程图里可以开始重点考虑函数将接受和提供的数据。最重要的是可以开始计划函数库的使用了。即使这里没有预写的外设或数据转换库存在，我们也能够以库的形式编写原始代码并在将来更加易于重用。

把不同的状态引入到计划中是可能的。状态框图映射了这种状态转换，它可以作为流程图的一种补充工具。

从伪代码开始，能够建立一个变量清单并对RAM和ROM的需要作出评估。内存资源的限制将对某些程序员是一个打击。用现代桌面环境工作的程序员在巨大的内存空间里很舒适，大量的RAM可用来创建可能永远也不会被初始化或用到的大数据结构或数组。

相反，微控制器只能在特定类目标应用所需要的那么多RAM和ROM里运转。制造商力求提供一系列类似的部件，每个变种的贡献只是增加少量的单片资源。

结果

2.3.1 软件体系结构

自动调温器设备的程序设计语言是C。主要体系结构的难以决断之处涉及采用中断还是轮询（或查询）。决断当中部分将随芯片选择而自动决定：一些处理器变种根本就不包括中断。其他选择包括对中断驱动的键盘的显式支持，或者在超时时产生中断的定时器。

一个基于中断方案的重要方面是在中断和主程序代码之间的通信协议。因为中断和主程序是尽可能独立的（一个中断可能在任何主程序指令期间出现），所以竞争条件是出现的一个结果。

我们已经选定几个替代算法中最简单的那个：一个时钟/计数器中断将计算时间，请求显示屏修改和设置目标温度。主程序将循环查询键盘，采样环境温度，修改显示屏和切换HVAC机械。这一切只需要一个精确的计时中断，它必须24小时不间断工作。

2.3.2 伪代码

伪代码用自然语言表达程序的必要步骤。它在嵌入式程序设计里显得特别有用，因为执行的每个方面能够一起规划，而没有必要考虑操作系统的古怪特性。

在下面的示例里，假定时间是用计数器和软件跟踪的。

1. *初始化*

1) 设置时钟计数器为0。

2) 设置时间和温度目标变量为默认值。

3) 启用时间中断。

2. *时钟/计数器每秒触发中断一次。*

1) 增加时钟计数器。

2) 请求显示屏修改。

3) 循环遍历预置周期。如果时钟正好在或过了指明的周期时间，设置目标温度为那个周期的温度。

3. *主循环*

1) 采样环境温度。

(a) 如果环境温度在目标温度范围之外，打开制冷或制热开关。

(b) 如果环境温度在目标温度范围之内，关闭制冷或制热开关。

2) 写时间、环境温度和状态到LCD。

3) 等待按键

如果按键被按下，等待弹起周期并再次检查。

4) 分析按键。

(a) 如果关闭命令被发送，则立即关闭操作单元。

(b) 如果周期选择命令被发送，则改换到下一周期记录。

(c) 如果时间设置命令被发送，则调整在当前周期记录里的时间。

(d) 如果温度设置命令被发送，则调整在当前周期记录里的温度。

2.3.3 流程图

图2-1基本上是嵌入式软件里主要和次要任务之间关系的一种表达。此流程图帮助决定：

- 什么功能出现在哪个逻辑模块里。
- 希望由函数库来支持哪些功能

也能够对流程图的重要结构给出标识符。

2.3.4 状态图

软件可能用来表达在处理外部交互或内部事件后在状态间转换的不同状态。图2-2显示了这些状态和使它们变化的相应激励条件。

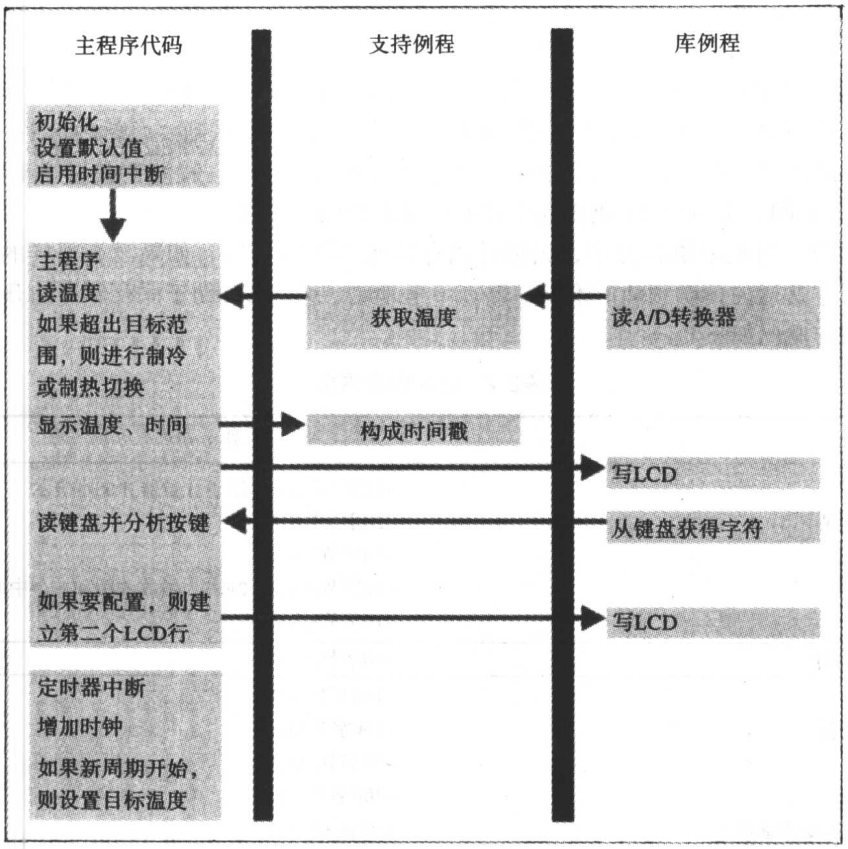


图2-1 关于算法的数据流程

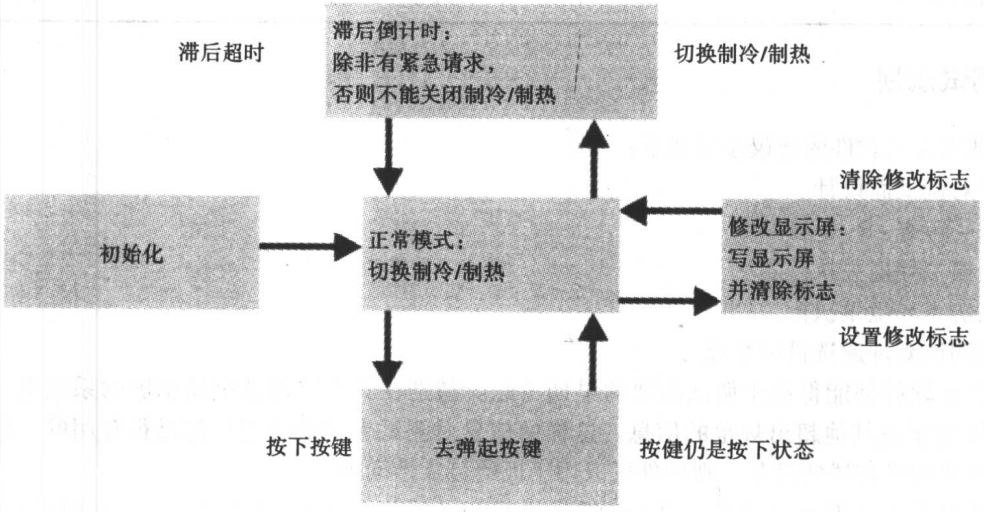


图2-2 关于算法的状态图

2.3.5 资源管理

在微控制器的受限环境里，一个多余的变量或常量能够改变被选择芯片的内存需求，进而影响到价格。像多语言支持这样的特征能够迅速地使资源需求膨胀到一个新的水平。

显式地规划出必需的资源是明智之举。这不是极其幼稚的——我们还在这里谈论通用变量，而不是像页面0访问、串行ROM 或者其他技术选择那样的具体细节。

如果您以前写过汇编语言程序，则估计内存需求会更容易些。如果没有那样的经验，写样本代码并编译它是通向精确预估的惟一道路。幸运地是，采用C有助于保存所有的开发成果。

一个粗略的概况如下。

表2-2 估计内存需求

变量/模块	资 源
实时时钟	~10字节RAM，用于计数器和文本表示
日常周期记录	~20字节RAM
用户设置	~10字节RAM
栈	~10字节RAM：2或3个函数调用和一个中断
局部变量	~10字节RAM
RAM估计合计	~60字节RAM
常量	~100字节ROM
中断服务例程	~100字节ROM
初始化	~50字节ROM
主程序	~300字节ROM
A/D转换（温度传感器）	~50字节ROM
LCD	~300字节ROM，依赖于接口类型而具有较大变化
键盘解码	~100字节ROM
ROM估计合计	~1000字节ROM

2.4 测试规划

调试嵌入式软件的建议步骤如下：

- 针对调试而设计。
- 代码审查。
- 在模拟环境里执行。
- 在仿真环境里执行。
- 按测试套件遴选目标系统。

硬件和软件都能得益于调试需要的早期考虑。特别在具有字母数字显示屏的系统里，软件能够通知错误或其他超出规范的信息。这样的信息对测试者和最终用户都是很有用的，但是如果市场不能容忍有错的设备，则软件可以用于证明可靠性问题。

在没有显示面板的情况下，LED能够发送有意义的状态或者事件信号。运行时诊断反馈信息、应该在伪代码和资源管理中体现。

调试的第1步，需要检查由编译器产生的汇编代码。在8位CPU上的嵌入式控制应用程序是足够小的、体系结构也足够简单，开发者能够很容易地审查产生的全部汇编语言代码。按行排列C源代码片段与它们产生的汇编代码的清单文件提供了最容易浏览的形式。

然而，经过这一步之后，测试变成一个挑战：当正测试中的代码实现了机器的最基本行为时，深入系统里的调试变得更加困难。一个故障可能彻底阻止来自嵌入式系统的任何有意义的响应，而桌面操作系统却能够提供内核卸载或其他的诊断帮助。

为使深入系统里的调试成为可能，要将模拟器和仿真器与嵌入式系统相匹配。每种工具尝试模拟目标环境的不同部分，从而可以彻底和容易地检测软件性能。在不需要或不关心硬件的条件下，纯软件的模拟器最适合用于检查算法性能和正确性。在真实世界里，仿真器更加集中于I/O和内部的外围设备操作。您将需要访问至少一个仿真器。因为工具选择是与硬件设计过程和处理器选择紧密联系的，所以对它做了讨论。

最后，在测试套件里放入一个原型设备能提供工作软件的最准确证明。

结果

我们的设计将有一个LCD面板。由于这项能力，系统能够写调试消息到显示屏。这些消息可能包括打开电源时的一个回应按键或者显示状态信息的“耀眼屏幕”。

编译器必须支持调试。产生的汇编代码必须是可用于检查的。

选择的产品应该为模拟器执行源代码级的调试提供方便，能匹配当前正执行的机器代码与原始C代码。对于自动调温器而言，模拟速度不是一个关键因素，惟一的时间依赖的功能是实时时钟。

测试套件由一个灯泡和风扇组成，它们由控制器切换开关并指向热敏电阻。这是一个最简单有效的方案。

第3章 微控制器

本章回顾微控制器特征并概述在8位微控制器市场里的可能选择。某些特征可能在中央处理器里看到过的，如图形增强或浮点支持等，在嵌入式系统里是不存在的。

计算机硬件设计最令人注意和具有超凡魅力的部分是中央处理单元的选择。在桌面世界里，处理器选择围绕着同Intel x86产品线的兼容性展开：与Intel兼容的处理器，接近兼容的处理器和完全不兼容的处理器。

在嵌入式世界里，没有这样的连续性，特别在谈论一个新的设计时。8位控制器市场竞争非常激烈，主要是因为它在容量上的焦点。它通常没有商标名称识别，且消费品生产厂商不愿意让用户知道技术细节。如果用户关心驱动他们产品的芯片，他们则可能要去了解超越产品预期应用的新应用领域。

8位微控制器不像32位处理器那样是程序员友好的。32位处理器高度优化的体系结构的后天增强，如额外的ROM地址空间，能够快速超出8位体系结构的限制。这反过来推动处理器设计者在组装机里增加如存储体切换或者寻址限制等技术以作为补偿。

最后，如体系结构的预期寿命等因素也应该考虑到。首选处理器达到它的产品生命周期的末期的时候，采用产生设备程序的C编译器可以减少不断改换处理器的成本。

8位微控制器具有计算机的所有传统功能部件。

中央处理单元（CPU） 微控制器的算法和逻辑单元针对存在于这么小的体系结构里的有限资源受到限制和优化。多路复用和分路操作非常罕见，并且浮点也不存在。寻址模式有时受到令人恼怒的限制。

ROM和RAM 8位微控制器很少有ROM和RAM寻址超过16线的（64 Kb）。如果某个芯片封装暴露所有地址或数据总线，则它们提供的寻址空间只有几千字节。MCU（微控制器单元）包含少量的内部RAM和ROM阵列。因为程序化单个芯片的需要，ROM往往像电子可编程（或电子可擦写）内存一样常见。

定时器 有两种常见类型：计数器和监视定时器。简单的计数器能够对一个时钟周期或者输入信号作出响应。在到达一个零点或者一预置的阈值时，它们能触发一个中断。

中断电路 在通用微处理器有多个通用化的中断输入或者级别的场合，一个微控制器有多个专用于特殊任务的中断信号：计数器超时，或者在某个输入针脚上的信号变化。

那就是说，上述情况是在控制器根本有中断的条件下；如果预期的应用足够简单而不需要它们，就不能保证设计者将一定包含它们。

输入与输出 大多数芯片提供能够切换外围设备的一些I/O线。偶然地，这些针脚能够减弱大电流以减少外部组件。某些变种提供A/D和D/A转换器或者驱动某些设备（如红外LED）的特殊逻辑。

外设总线 并行外设总线将减少“单片机”的优势，因此它们的应用是受到妨碍的。因为速

度在嵌入式系统设计里不是位于前列的功能要素，几个有竞争力的串行外设总线标准已经开发出来。仅仅使用1~3根线，这些总线可使外围设备芯片，例如ROM，与微控制器交互而不独占现有的接口线路。

微控制器的微小尺寸的主要后果是，它的资源相对于桌面个人计算机按比例限制了。尽管具备了计算机的所有特征——RAM、ROM、I/O以及微处理器——但是，开发者不能指望具有其上有8个并行位的I/O端口。

在决定首选处理器之前，必须考虑针对目标可得到的外部开发工具。一个嵌入式系统不像一台PC机那样是自运行的。为开发嵌入式软件，开发工具必须运行在一个桌面计算机上，并至少使用某些非常特殊的硬件。

3.1 中央处理单元

各种不同微控制器中，寄存器的数量和名称可能不同。它们有时出现在某个内存地址空间里，有时又完全各自独立。尽管名称可能不同，但是某些寄存器对大部分微控制器是通用的。

- 累加器
- 变址寄存器
- 栈指针
- 程序计数器
- 处理器状态寄存器

用C直接访问累加器和变址寄存器只能偶尔令人满意。C的register数据类型修饰符等于向一个寄存器发出的要直接访问的“请求”：如果编译器不能令人满意地做到这点的话，实际上则不使用寄存器。

然而，当使用寄存器是最合适的或者是必须的时候，另一类型声明能够链接变量名和一个寄存器本身。Byte Craft编译器提供了registera类型（以及其他寄存器的相应类型）。给一个registera变量赋值产生进入到累加器寄存器的一个数据装载，但是不产生到内存的一个存储。对该标识符的求值返回在寄存器里的值，而不是来自内存的值。

```
registera important_variable = 0x55 ;
```

直接访问栈指针或程序计数器是不太明智的。采用C语言的目的是从直接机器语言引用中抽象程序逻辑。函数调用和循环将使设备依赖的栈操作和分支操作平衡，是结构化编程的最好方法。如果有必要，可以对带标号的目标使用C关键词goto：编译器将插入合适的跳转指令，并且，最重要地是，自动地考虑任何分页或设置问题。

3.1.1 指令集

期待出现在通用微处理器单元MPU（microprocessor unit）里的机器指令有：乘法、除法、表查询或者乘法-加法，而在8位处理器里的相应指令并不总是出现在控制器家族的每个变种里。

一个#pragma语句能够通知编译器目标芯片确实有一个可选指令功能，因此编译器能够利用该指令提供的便利条件而优化代码。这样的例子出现在MC68HC05C8的头文件中。