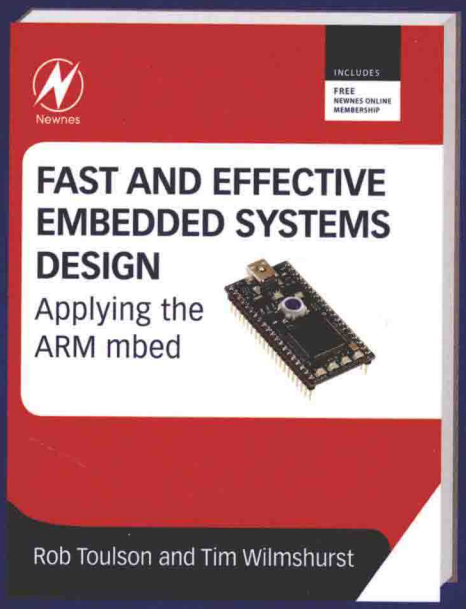
HZ BOOKS
华章科技

ELSEVIER
爱思唯尔

mbed是由嵌入式领域的领导厂商ARM在全球大力推广的开源硬件项目。
本书是国内外针对mbed开源硬件项目的书籍。
理论与实践相结合，对嵌入式系统设计每个环节的讲解都入木三分。
本书是开源硬件、积木式中间件、快速系统原型设计三大核心理念的载体。

电子与嵌入式系统设计译丛



Fast and Effective Embedded Systems Design
Applying the ARM mbed

ARM快速嵌入式系统原型设计

基于开源硬件mbed

［英］ Rob Toulson Tim Wilmshurst 著 韩德强 鲁鹏程 等译

机械工业出版社
China Machine Press

需要整本电子书，联系我QQ：2667271557

本书使用极具创新性的mbed开源硬件，基于Web开发环境，介绍了嵌入式系统设计的各个方面。每个章节引入嵌入式系统设计中的一个主题，并通过边做边学的方式进行一系列的实践验证。介绍完基本内容后，本书逐步讲解了嵌入式设计的热点应用领域，如智能仪器仪表、网络系统、闭环控制和数字信号处理。

本书由嵌入式领域的两位知名专家撰写，通过实验结果，验证了从开发到理论与实践相结合的全过程，并对相关技术的引入及其优缺点进行了评估，探讨了更加广泛的应用范围。

本书特点：

- 实践驱动的嵌入式系统设计教学，重点关注快速原型
- 通过简单而有效的实验，覆盖关键的嵌入式系统设计概念
- 覆盖面广，从简单的数字输入/输出到高级的网络与控制
- 基于嵌入式领域中最易学、最易获取的工具
- 对ARM技术和微控制器架构的深刻见解

作者简介：

Rob Toulson 英国剑桥安格利亚鲁斯金大学研究员，主要致力于推进技术和创意产业相结合方向的研究。博士毕业后的几年里，Rob主要在音频与汽车领域从事数字信号处理和控制系统工程项目。

Tim Wilmshurst 英国德比大学电子专业负责人。在德比大学任职之前，Tim负责剑桥大学工程系的电子开发小组多年，亲历了很多微控制器和嵌入式系统的发展过程。

本书译自原版*Fast and Effective Embedded Systems Design: Applying the ARM mbed*并由Elsevier授权出版

ELSEVIER 爱思唯尔

ELSEVIER

上架指导：集成电路设计

ISBN 978-7-111-46019-0

9 787111 460190 >

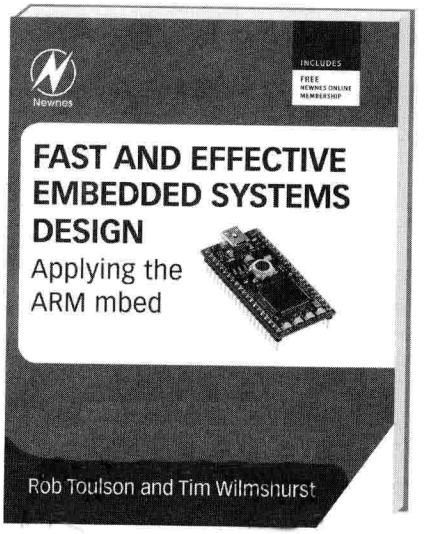
定价：69.00元

投稿热线：(010) 88379604
客服热线：(010) 88378991 88361066
购书热线：(010) 68326294 88379649 68995259

华章网站：www.hzbook.com
网上购书：www.china-pub.com
数字阅读：www.hzmedia.com.cn

需要整本电子书，联系我QQ：2667271557

电子与嵌入式系统
设计译丛



Fast and Effective Embedded Systems Design
Applying the ARM mbed

ARM快速嵌入式系统原型设计

基于开源硬件mbed

［英］Rob Toulson Tim Wilmshurst 著 韩德强 鲁鹏程 等译

机械工业出版社
China Machine Press

图书在版编目（CIP）数据

ARM 快速嵌入式系统原型设计：基于开源硬件 mbed /（英）托尔森（Toulson，R.），（英）威尔姆斯特（Wilmshurst，T.）著；韩德强等译 . —北京：机械工业出版社，2014.3
（电子与嵌入式系统设计译丛）
书名原文：Fast and Effective Embedded Systems Design: Applying the ARM mbed

ISBN 978-7-111-46019-0

I. A… II. ①托… ②威… ③韩… III. 微处理器 – 系统设计 IV. TP332

中国版本图书馆 CIP 数据核字（2014）第 035605 号

本书版权登记号：图字：01-2013-1634

本书旨在通过 mbed 介绍嵌入式系统设计的所有主要议题，便于读者快速掌握嵌入式系统的设计方法。本书共 15 章。第 1~10 章从基本的原理和简单的项目入手，使用 mbed 项目示例提供一套完整的嵌入式系统设计入门课程，旨在揭示如何使用 mbed 快速地设计嵌入式系统。第 11~15 章逐渐深入到更专业的领域，阐述嵌入式系统的设计精髓，为读者进一步阅读或学习更高级的课程打基础。

ARM 快速嵌入式系统原型设计：基于开源硬件 mbed

（英）Rob Toulson　Tim Wilmshurst　著

出版发行：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：谢晓芳　秦　健　刘　涛
印　　刷：北京市荣盛彩色印刷有限公司　　版　　次：2014 年 3 月第 1 版第 1 次印刷
开　　本：186mm×240mm　1/16　　印　　张：19.25
书　　号：ISBN 978-7-111-46019-0　　定　　价：69.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换
客服热线：（010）88378991　88361066　　投稿热线：（010）88379604
购书热线：（010）68326294　88379649　68995259　　读者信箱：hzjsj@hzbook.com

需要整本电子书，联系我QQ：2667271557

译 者 序

近年来，基于ARM处理器的嵌入式系统飞速发展，遍及各种移动设备及工业和家庭控制系统。

介绍基于ARM处理器的嵌入式系统的书籍琳琅满目，这些书籍或以某个或某类处理器为基础，又或者以某个嵌入式系统为基础介绍嵌入式系统的概念及应用开发。而本书的两位作者均是大学教师，具有丰富的嵌入式系统教学和工程实践经验，他们倡导“在实践中学习”的先进理念，在书中给出了大量基于ARM mbed的动手实验教程，在突出实践能力培养的同时，又在每章中针对嵌入式系统的各个功能模块给出了相应基础知识的介绍，免除了读者再去翻阅其他书籍的麻烦。本书的另一个亮点就是不需读者安装软件和配置繁琐的软件开发环境，只需一台能够上网的PC即可，通过特有的Web软件开发环境，即可完成应用程序的开发过程。以译者20多年的嵌入式系统教学与开发经验来看，本书非常适合作为本科、高职高专各专业的嵌入式系统基础课程教材。

本书由北京工业大学计算机学院的部分教师翻译，其中韩德强翻译了前言和第1～2章，鲁鹏程翻译了第3～6章和附录，张丽艳翻译了第7～9章，杨淇善翻译了第10章，王宗侠翻译了第11～12章，邵温翻译了第13～15章，全书的审校由韩德强完成。

在本书的翻译过程中得到了ARM公司中国大学计划经理时昕博士的大力支持与关注，并提供了ARM mbed开发板，在此对时博士表示由衷的感谢！

限于译者的水平，翻译中难免有错误或不妥之处，真诚希望各位读者批评指正。

韩德强

2014年1月于北京工业大学

需要整本电子书，联系我QQ：2667271557

前　言

微处理器无处不在，它为汽车、手机、家居和办公设备、电视和娱乐系统、药品、飞机等无穷无尽的产品提供“智力”。这些司空见惯的产品内部都隐藏着一个微处理器使其智能化，我们称这些产品为嵌入式系统。

不久前，嵌入式系统的设计者必须是电子或软件方面的专家，或者两者都是。现在，无论是专业人员还是初学者，使用界面友好的、复杂的积木式构件，便可以很快设计出嵌入式系统。这种积木式构件便是最近由著名的计算机巨头 ARM 推出的 mbed。本书将围绕 mbed 介绍嵌入式系统设计的所有主要议题，目的在于通过 mbed 的使用教会读者嵌入式系统设计的要素。

本书分为两部分。第 1 ~ 10 章广泛介绍嵌入式系统，使用 mbed 展示如何应用它快速地进行成功的嵌入式系统设计。这几章旨在全力帮助读者掌握一系列精心构建的概念和练习，从基本的原理和简单的项目入手，逐步完成更高级的系统设计。第 11 ~ 15 章在此基础上进入到了许多更高级的系统设计领域。这里讲的速度可能会有点快，你会发现需要进行更多的背景研究。

本书仅要求读者具备基本的电工电子理论知识。本书采用“在实践中学习”的方法，为了更好地使用本书，你需要一块 mbed 开发板，一台连接到因特网的计算机，还需要书中指定的各种额外的电子元器件。如果不做某一个实验或者实践书中某一部分的内容，就不需要准备其中所需的东西。你还会用到数字电压表，不过最好使用示波器，这样能看到更多细节。

全书每章围绕一个嵌入式系统主题展开。每一章或多或少都有一些理论的介绍，其中很多章需要更多的理论基础，然后才能进行一系列实际的实验。准备好将你的 mbed 连接到下一个电路，下载并编译下一个示例程序，然后运行该程序以理解到底是怎么回事。随着你对 mbed 信心的增长，你的创造力和原创性会随之增加，开始将你的想法变成可行的项目吧！

本书会快速地帮助你：

- 理解和应用嵌入式系统的关键环节。
- 理解和应用 ARM mbed 的关键环节。
- 从头学起或提高嵌入式 C/C++ 编程技巧。

- 加深对电子元器件及配置的理解。
- 了解 mbed 如何被应用到一些新兴的、最令人兴奋的、创新的智能产品中。
- 完成你从未想过自己会有能力完成的设计和创新！

如果你遇到问题或者有任何疑问，可以通过本书的网站和 mbed 的网站寻求技术支持，你也可以通过电子邮件与作者讨论。

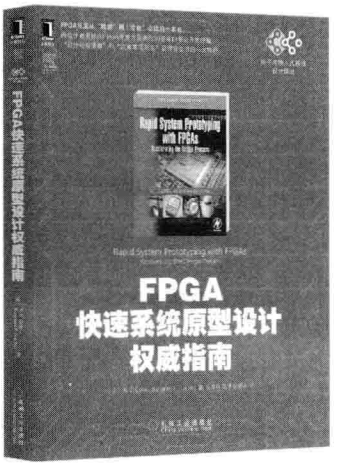
如果你是一位大学教师，本书为你的嵌入式系统课程提供“完整的解决方案”。两位作者都是经验丰富的大学讲师，他们在写书时考虑到了你的学生。本书包含一个实践和理论学习活动的组织顺序。理想状态下，你需要为每个或每对学生配备一套 mbed（一块原型面包板和一个套件），因其高度便携，开发工作不必局限在学校的实验室中进行。在后面的课程中，学生们将会用他们的 mbed 开发板联网。教师可以在本书配套网站（www.embeddedacademic.com）下载每一章的完整 PPT 演示文稿，也可以下载测验题的答案以及练习题和小项目的示例代码。

本书第一部分（第 1 ~ 10 章）使用 mbed 项目示例提供了一个完整的嵌入式系统设计入门课程。第二部分（第 11 ~ 15 章）用于提高课程，为进一步阅读或学习更高级的课程打基础。

本书采用了一种亲身实践的有趣学习方式，适用于任何想要引入嵌入式系统概念的课程。因为仅需要一点电子学理论，所以本书更适用于那些不以使用嵌入式系统为目的的学科。本书原本供工程学、物理学或计算机科学等学科大学一年级本科生使用，但我们希望学生们能更多地在高年级使用它。实践类的专业人员和业余爱好者也会对本书感兴趣。

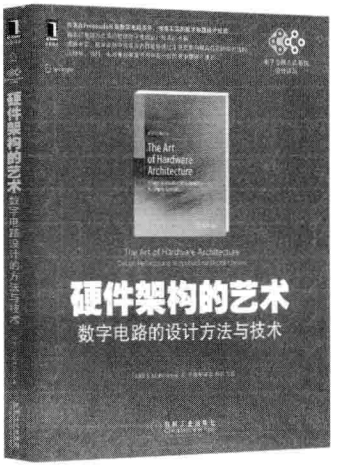
本书由位于英国剑桥的安格利亚斯金大学的研究员 Rob Toulson 和英国德比大学电子专业负责人 Tim Wilmshurst 编写。博士毕业后的几年里，Rob 主要在音频和汽车领域从事数字信号处理与控制系统工程项目。之后，他成为了一名研究员，当前主要致力于推进技术和创意产业的合作研究。到德比大学之前，Tim 负责剑桥大学工程系的电子开发小组多年，他的设计生涯见证了很多微控制器和嵌入式系统的发展过程。除了分享对嵌入式系统的兴趣之外，我们还分享对音乐和音乐技术的兴趣。这本书汇集了我们广泛的经验。规划好书的整体布局，进行了一些最初的准备工作后，我们对章节进行了划分。Tim 负责前几章大部分内容的撰写，并负责电子和计算机硬件相关的章节。Rob 主要负责后面几章的撰写以及 mbed 高级应用。这种工作分工主要是为了方便，而在出版物中我们彼此都对所有章节承担责任。由于 Tim 之前写过一些嵌入式系统方面的书籍，本书中的一些背景知识和图表就取自这些书，这主要是因为在需要解释背景的地方“从零开始”似乎是没有意义的。

推荐阅读



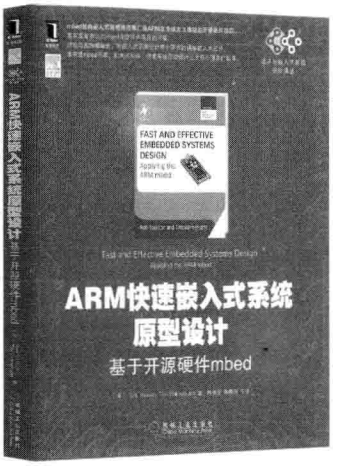
FPGA快速系统原型设计权威指南

作者：R.C. Cofer 等 ISBN：978-7-111-44851-8 定价：69.00元



硬件架构的艺术：数字电路的设计方法与技术

作者：Mohit Arora ISBN：978-7-111-44939-3 定价：59.00元



ARM快速嵌入式系统原型设计：基于开源硬件mbed

作者：Rob Toulson 等 ISBN：978-7-111-46019-0 定价：69.00元

嵌入式软件开发精解

作者：Colin Walls ISBN：978-7-111-44952-2 定价：79.00元

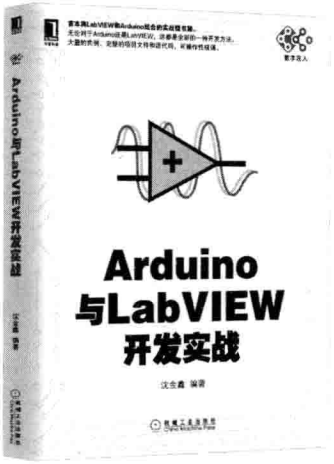
推荐阅读

Arduino高级开发权威指南（原书第2版）

作者：Steven F. Barrett ISBN：978-7-111-45246-1 定价：59.00元

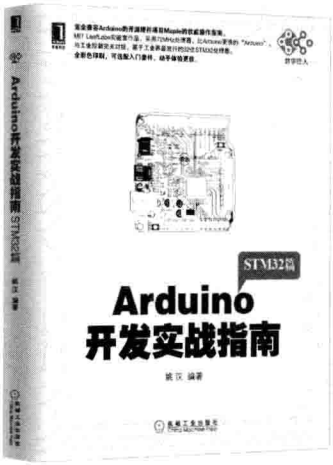
例说XBee无线模块开发

作者：Jonathan A. Titus ISBN：978-7-111-45681-0 定价：59.00元



Arduino与LabVIEW开发实战

作者：沈金鑫 ISBN：978-7-111-45839-5 定价：59.00元



Arduino开发实战指南：STM32篇

作者：姚汉 ISBN：978-7-111-44582-1 定价：59.00元

目　录

第一部分

嵌入式系统概述与玩转 mbed

第 1 章
嵌入式系统、微控制器与 ARM

1.1 嵌入式系统简介

1.1.1 什么是嵌入式系统

我们都对台式机与笔记本电脑非常熟悉，并且惊异于它们的强大功能。这些计算机都是通用计算机，我们可以利用它们在不同的时间做不同的事情，不过这取决于我们在计算机上运行的应用程序。在这些计算机中重要的部分就是微处理器。微处理器是一块微小的、非常复杂的、包含计算机核心细节的电路。其中所有的器件都焊接在一小块称为集成电路（IC）的硅片上。一些非专业的人员也常常把集成电路称为微电路，或是仅仅称为芯片。

为很多人所不熟悉的是，微处理器不仅仅存在于通用计算机中，也可以安置到一些不需要计算的设备内部，比如，洗衣机、烤面包机或者摄像机中。微处理器常常用于控制以上这些产品。微处理器放置在产品内部，在计算机中往往是不可见的，用户可能根本就不知道它在哪儿。此外，计算机往往还关联着如键盘、显示器和鼠标这些附加组件。因为控制这些产品的微控制器镶嵌在产品内部，所以这类产品称作嵌入式系统。在许多情况下，由于部分微处理器更倾向于关注控制，在嵌入式系统中所使用的微处理器相对于通用计算机的中的处理器又逐渐发展出不同的特性，这些称为微控制器。虽然它们在微处理器大家庭中并不起眼，但是它们的影响是巨大的。对于一些电气与系统设计人员来说，这也提供了巨大的商机。

嵌入式系统有很多形式。它们在家庭、电机与工作场合中随处可见。绝大多数家用电器，比如洗衣机、洗碗机、烤箱、中央空调和报警器，都是嵌入式系统。汽车中充满了嵌入式系统，比如引擎管理模块与安全模块（如汽车锁与防盗设备）、空调、刹车、收音机等等。嵌入式系统广泛应用于工业、商业、机械控制、工厂自动化、电子商务与办公设备中。当然，其应用领域远远超过以上范围，并且应用范围正不断扩大。

图 1.1 展示了一个嵌入式系统的简单框图。其中有一组转入来自于被控制系统。嵌入式计算机通常是指一个微控制器，它可以运行一个永久地存储在存储器中的程序。与那些可以

运行许多程序的通用台式机不同，“嵌入式计算机”只能运行一个程序。微控制器根据输入所提供的信息，计算出输出。输出信息往往与系统内部的执行器相连。实际的电子电路与其他与之相连的机电组件通常一同称为硬件。在硬件上运行的程序通常称为软件。除了这些之外，嵌入式系统仍然有很多可以与用户交互的方法，比如通过键盘与显示器。也有很多可以与其他子系统的交互，虽然这些是必不可少的一般性概念。在嵌入式系统中，时间因素是另一个可以影响我们所作所为的变量，在下面的图中表示为一个贯穿的箭头。我们需要测量时间，使得其准确地在我们预先安排的时间内运行，并产生对时间依赖很强的数据流，同时可以及时处理发生的异常。

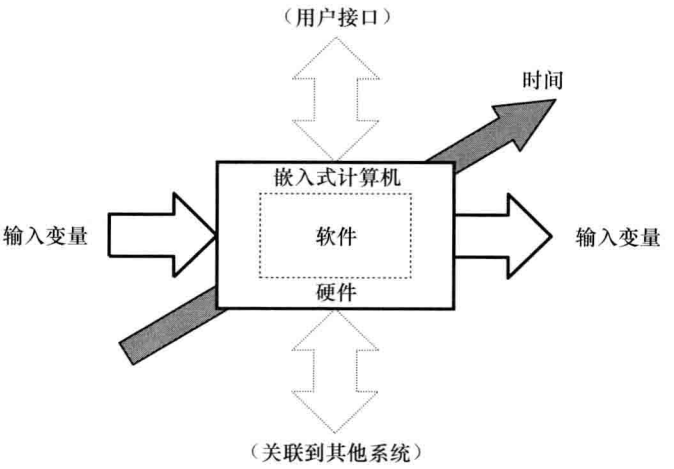


图 1.1　嵌入式系统

这一章阐述并回顾了许多关于计算机、微处理器、微控制器以及嵌入式系统的概念。通过这种回顾的形式，为我们以后学习更深层次的内容提供了一个平台。后面章节描述的内容最终都将回归这些概念，以它们为基础，再加以详细的介绍。如果需要了解更多信息，请参见参考文献 1.1。

1.1.2　嵌入式系统示例

零食自动售货机是一个很好的嵌入式系统例子。在图 1.2 中以框图形式展示了这个例子。图 1.2 的中间是一个微控制器。如图 1.2 所示，微控制器从用户键盘、硬币计数模块以及物品分配模块接受一系列输入信号，并且产生依赖于这些输入的输出。

饥饿的客户们会靠近自动售货机并且猛击按钮或者投入硬币。在第一种按下按钮的情况下，键盘模块会向微控制器发送回馈信号，使得微控制器可以识别出不同的按键，并把这些按键的键值汇总到解码器中以产生更多、更复杂的消息。硬币计数模块也会根据所投入的硬币数量发送信息。微控制器会试图筛选接收到的信息，并输出自身的状态信息至液晶显示屏上。用户是否正确选择了一个产品？是否支付了足够的钱？如果选择了正确的产品并支付

了，微控制器会控制执行器模块分发产品。如果没有，微控制器会在显示屏上显示消息，提示用户投入更多的硬币或者重新输入产品代码。好的自动售货机还应该提供找零功能。同时自动售货机也需要一些感应机制来确保产品使用的可靠性以及分发动作最终完成。以上这些都在图 1.2 中以门位置传感器的形式显示了出来。不好的产品（为什么我们没有遇到呢？）会在显示屏上显示一些没有用的或者让人恼怒的消息，在用户明明给了正确的钱款后却仍然要求付款，或发放给用户错误的产品，甚至不把用户选择的产品送出来。

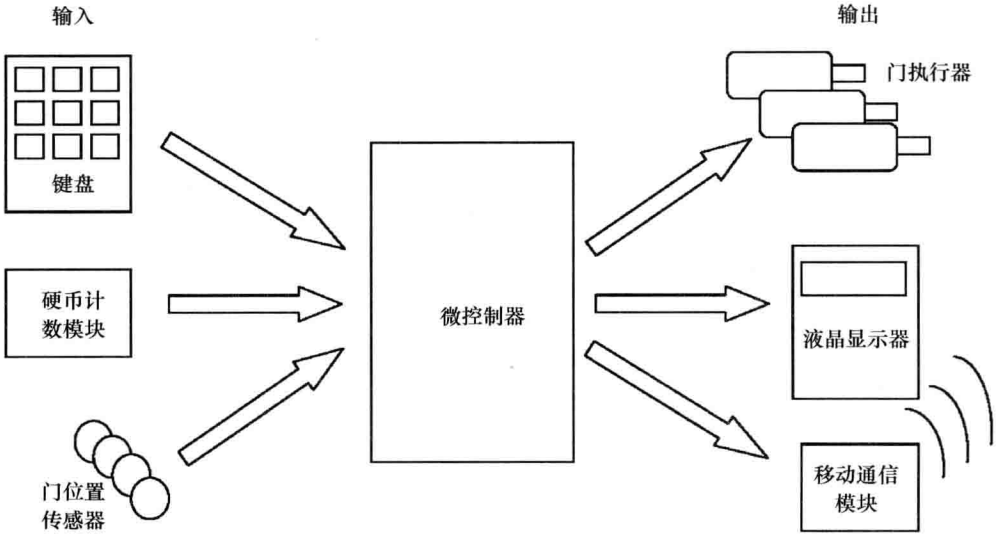


图 1.2 嵌入式自动售货机

以上所描述的都是传统机器的特点，但是我们可以走得更远。在现代的自动售货系统中，可能会植入一些移动通信功能，允许自动售货机直接向维修团队报告产生的故障，这样维修团队很快就能解决这些问题。与此相似，机器也可以通过移动通信或者物联网通信报告库存水平，并提醒服务团队补充库存的不足。

这个简单的例子在图 1.1 中得到准确的反映。微控制器接受输入变量，计算、处理并产生输出。（微处理器的）这些行为其实都包含对时序的正确使用。在上述过程中，一般系统都会提供用户接口。一些现代的机器还提供网络接口。尽管图 1.2 似乎在描述一个硬件系统，但是事实上图 1.2 所述的一切都被设计者编写的软件所控制。那些运行在微控制器上的软件决定了系统实际的功能。

1.2 微处理器与微控制器

回顾之前所述，微控制器是任何嵌入式系统的核心。由于微控制器在本质上是一类计算机，这对我们掌握基本的计算机信息非常重要。为此这里仅作概述，具体功能会在后面的章节中详细介绍。

1.2.1　计算机主要组件

图 1.3 显示了计算机系统中的基本元素。作为其存在的目的，一台计算机可以执行算数或者逻辑运算。这些操作在一个称为算术逻辑单元（ALU）的数字电路中完成。算术逻辑单元被放置在一个称为中央处理单元（CPU）的大型电路中，这个电路能为 ALU 提供细节支持。ALU 可以承担一些简单的算术和逻辑运算工作。代码的反馈称为指令。现在我们更加深入地了解了什么是计算机。如果我们能够通过给 ALU 提供一系列指令以及指令需要的数据来保持 ALU 的忙碌，那么我们实际上就拥有了一个非常有用的机器。

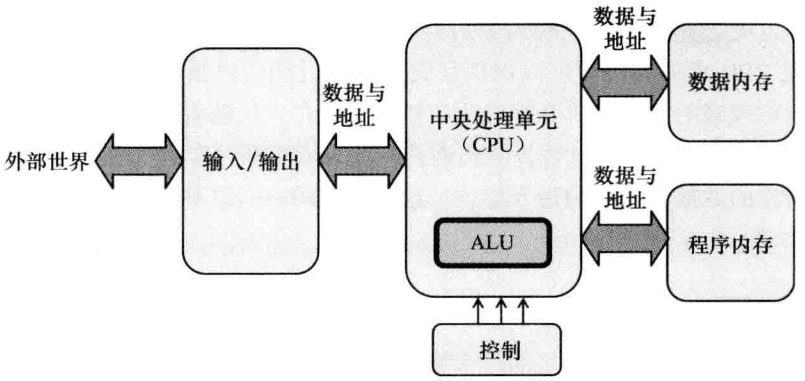


图 1.3　计算机的基本元素

ALU 所需要的指令与数据是由它周边的电路提供的。值得注意的是，这些指令中的任何一条仅仅能完成一个非常简单的功能。但是，由于计算机的运行速度非常快，使得其整体拥有巨大的计算能力。一系列指令集合称为一个程序。程序通常存储在一段内存中，这段内存称为程序存储器。如果这段内存可以永久存放，不论是否供电，程序都会永久地保留在内存中。一旦电源打开，程序就会立刻运行。像这样可以在电源关闭时一直保持内容的内存称为非易失性存储，老式的术语称为只读存储器（ROM）。这个术语一直在使用，尽管对于新的内存技术来说，这一说法已经不再准确。控制电路需要不断访问程序内存以查找下一条指令。ALU 所处理的数据可能来自数据存储器，计算的结果也会存储在那儿。由于通常计算的结果都是一些临时数据，因此存放这些数据的内存不必是永久性的，尽管使用永久性内存也没有什么坏处。这种类型的内存有个过时的定义叫做随机存取存储器（RAM）。尽管我们在这儿使用了过时的术语，但是这也传达了很多有用的信息。

无论是什么用途的电脑都必须可以通过它的输入 / 输出（I/O）与外界沟通。在个人计算机上，这意味着通过键盘、视觉显示单元（Visual Display Unit，VDU）与打印机这类的外设进行人机交互。在最简单的嵌入式系统中，通信更主要是通过传感器和执行器与它周围的物理环境进行的。从外界进入的数据可能会被迅速转移到 ALU 中进行处理，或者存储在数据存储器中。发送到外界的数据也很有可能是 ALU 最近所计算得到的数据。

最后，如图 1.2 中空心箭头所示，在几个主要的框图之间应该有数据通路。数据通路是

携带各个方向数字信息的一系列连线。使用一组导线传递信息的通路称为数据总线，比如，从程序存储器到 CPU 之间的连线就是总线。另一些连线携带地址信息，这些线称为地址总线。地址是指一个包含数据应该存放在内存的何处，或者从哪儿检索的数字。数据和地址连线可以布置在印刷电路板或者在 IC 上互联。

定义计算机的一种方法是根据其 ALU 所能处理数据的大小。简单来说，老式处理器可以处理 8 位数据。在今天，这类处理器仍然扮演着重要的角色。8 位数据意味着可以表示 0 ~ 255 之间的所有数字。(如果你不熟悉二进制数，请查看附录 A)。最新的机器更多是 32 或者 64 位的。这给处理器提供了更大的处理能力，但也增加了其复杂性。ALU 可以处理的数据大小通常也决定着很多其他部件的处理能力，例如，内存位置数据总线宽度等。

可以预见 CPU 也拥有一套可以识别与交互的二进制代码指令集。例如，某些指令需要对两个数进行加或减的操作，或者把这两个数存入内存。伴随着数据与地址的传递，许多指令都会完成。从本质上来说，这些计算机在内存中保存的程序就是一系列从指令集中提取出来的、包含所需的数据及地址的指令集合。这样的代码有时也称为机器代码，以区别于其他版本的程序。

1.2.2 微控制器

除了具有刚刚所描述的计算机的许多重要特质外，微控制器也增加了很多所需的控制功能。我们把它分为三个部分来考虑：内核、内存与外设，如图 1.4 所示。内核是指 CPU 及其控制电路。除了这些，还有程序和数据存储器。最后，还有外围设备（简称外波）区分微控制器与微处理器的一个重要因素是微控制器需要允许与外界环境的广泛交互。外设包含数字或模拟输入 / 输出、串口、定时器、计数器和许多其他有用的子系统。

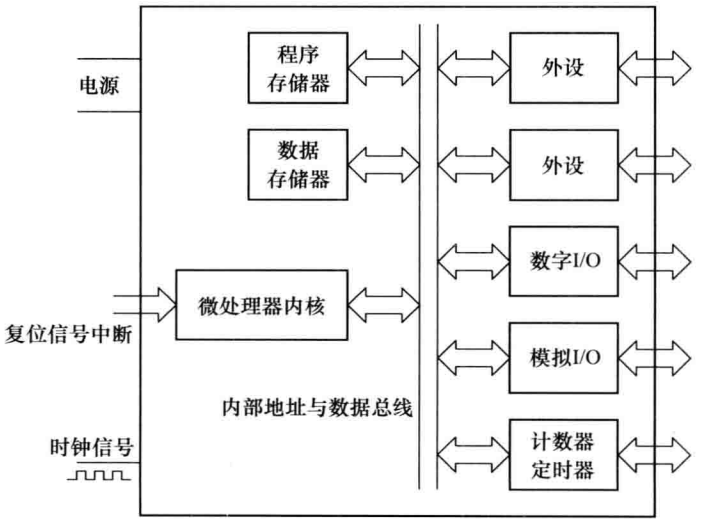


图 1.4 一个微控制器示例：内核 + 内存 + 外设

任何时候都不能忘记，微控制器需要一个稳定的直流（DC）电源。任何个人计算机（PC）都需要一个时钟信号。正是这个不停止的时钟信号使得微控制电路得以完成程序指示的一系列动作。时钟信号通常由石英振荡器产生。在你的手表中就使用了石英振荡器。这就给处理器提供了一个稳定可靠的时钟频率。时钟常常有一个很重要的辅助功能——给微控制器的动作提供时钟信息，例如，启动定时事件或者控制串行数据的时序。

1.3 嵌入式系统的开发流程

1.3.1 程序语言：C/C++ 有什么特别之处

1.2.1 节已经介绍了 CPU 指令集。作为一个程序员，我们的最终目的是编写一个能够使单片机完成我们意愿的程序。这些程序必须是一系列二进制指令集合。这些未经加工的二进制形式代码称为机器代码。因为人类根据二进制代码处理机器代码是极其繁琐的，所以这几乎是不可能完成的。

正确编程的第一步称为汇编程序。在汇编程序中每一条指令都有自己的助记符。指令由一组助记符表示。于是程序可以通过助记符来编写，然后通过计算机进行交叉编译，将助记符转换成实际的二进制代码载入到程序存储器中。使用汇编可以让我们的工作与 CPU 更加紧密。汇编程序也因此变得快速与高效。但是正因如此，我们很容易在汇编程序中犯错，也不容易查找到错误的位置。同时编程的过程也非常耗时。

正确编程的进一步是使用高层语言（HLL）。在这种情况下我们根据 C、Java 或者 Fortran 语言的规则编写程序。紧接着一个称为编译器的计算机程序将会读取我们的程序，并把程序转换（编译）为一系列指令集合。当然，这都是在我们所写的程序没有错误的前提下进行的。接下来这一系列指令集合被转化为我们可以下载到程序存储器中的二进制代码。然而，在嵌入式领域，我们的需求与程序员不同。我们希望能够控制硬件在可以接受的时间内迅速执行完程序。然而，并不是所有的 HLL 都可以满足这些需求。人们至今仍然在不断地争论嵌入式环境中的最佳语言是什么。无论如何，大多数观点都指出使用 C 语言会更有显而易见的优势。这是因为 C 语言本身非常简单而且允许我们与硬件交互。C 语言的上层语言是 C++，它也是一种在嵌入式应用开发中经常使用的高级语言。

1.3.2 开发周期

这本书是关于如何快速地开发可靠的嵌入式系统的，所以我们很有必要对整个开发的过程有一个了解。

在早期的嵌入式系统中，微控制器是非常简单的。它没有片上内存，外设也很少。仅仅设计与搭建硬件都需要花费大量的时间。另外，内存也是非常有限的，所以程序必须非常短。整个开发过程中最主要的部分就是硬件的设计。编程使用的是简单的汇编语言。最近几十年里，微控制器变得越来越复杂，内存也变得越来越充足。至今为止，很多供应商已经开

始销售包含微控制器与所有相关电路的预设计电路板。这些产品在设计时在硬件开发上花费了很少的精力，更多的精力被转移到尽可能利用已有内存编写复杂的程序上。这也是当今嵌入式系统的开发形势。

尽管发生了许多变化，程序的开发周期依然是基于单循环的（见图 1.5）。在本书中，我们主要使用 C 语言编写源代码。图 1.5 显示了我们可能会用到的一些工具。在计算机上源代码需要使用文本编辑器编写，我们把这台计算机叫做主机，比如，使用一台 PC。源代码需要转化为二进制代码下载到微控制器中，放置在它需要控制的电路或者系统中（我们称为目标系统）。代码的转化由主机上的编译器完成。程序的下载需要在主机与目标系统之间建立临时连接。我们不用关心实际写入的程序数据是什么样的。在下载程序前，最好能在主机上进行仿真。这使得待开发的程序在遇到下载故障之前有一个良好的水平。然而，真正的测试还是在目标系统中通过观察运行正确性来进行的。当程序在该阶段出现错误时，这在编程过程中是不可避免的，程序可以再次重写、编译与下载。

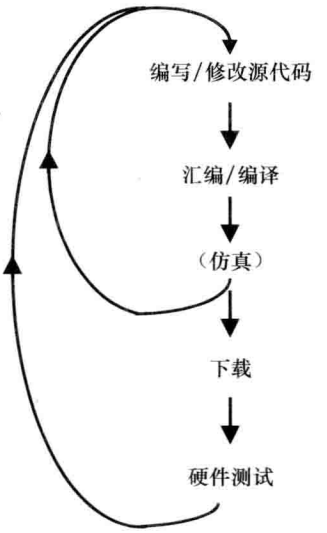


图 1.5 嵌入式程序开发流程

1.4 进入 ARM 世界

计算机和产品处理器的发展曾经一直是在由大公司和无数的小产品创业公司推动：当然也包含有才能的个人或团体。ARM 的发展已经很好地综合了这些因素。正如同最近 30 年来，许多高科技的新兴企业发展一样，整个 ARM 的发展历史非常迷人。接下来是对 ARM 发展史的总结。如果想要全面地了解，请参照专门讨论这个话题的几个网站，然后看这段历史在今后的岁月里继续上演！

1.4.1 关于 ARM 的历史

1981 年英国广播公司（BBC）进行了一项计算机教学计划，要求很多公司通过竞标的方式来提供一台所需的计算机。最终 Acorn 计算机公司中标了。于是这种计算机逐渐成为了广泛流行于英国各个学校的机器。Acorn 计算机公司使用了 6502 微处理器。这个处理器由 Mos 科技公司制作。这是一个 8 位微处理器，虽然现在已经不再广泛使用了，但是这在当时是非常先进的。为了满足人们对个人电脑与台式电脑与日俱增的兴趣，IBM 公司在 1981 年生产出了第一台 PC——一个基于英特尔的更强大的 16 位微处理器，8088。那时有许多公司生产类似的计算机产品，这其中当然包括苹果公司。这些早期的机器几乎彼此都不兼容，并且相当不清楚谁会最终占据主导地位。然而，在整个 20 世纪 80 年代，随着 IBM PC 的影响不断增加，规模较小的竞争对手逐渐没落。尽管英国 Acorn 计算机公司的成功，但是由于出口不

畅，这使得它的未来一片迷茫。

这段时间里，Acorn 计算机公司里面几个聪明的设计师做出了三次比较大的尝试，并成为三个科技转折点。他们希望推出一台新电脑。当然，这次不能继续使用 6502 处理器，但是他们在 6502 系列的升级产品中也无法找到一个合适的替代品。在第一个转折点中他们认识到，他们有能力设计微处理器，而无需从其他地方购买它。作为一个小团队，他们顶着激烈的商业竞争压力，设计了一个小却非常复杂的处理器。在这个处理器上构建新的计算机。这台计算机称为 Archimedes，是当时非常先进的机器。但是因为他们要与 IBM 公司竞争，这家公司逐渐发现他们虽然没有足够的电脑销售，但却拥有一个极其聪明的微处理器的设计。他们意识到自己的未来可能不在销售计算机本身，于是他们做出了第二次尝试。因此，在 1990 年，Acorn 计算机公司与国际电脑共同创办另一个剑桥公司，名为高级 RISC 机器有限公司，简称为 ARM。他们还开始实现第三次尝试：不需要成为一个成功的集成电路制造者，重要的是集成电路里面的设计思路，这些可以作为知识产权（IP）出售。

随着 ARM 的理念逐渐成熟，他们开始销售 IP 给世界上的各大厂商，并且在世界各地拥有越来越多的智能微处理器设计。该公司获得了巨大的成功，目前称为 ARM 控股。一旦有公司购买了 ARM 公司的设计，就可以将 ARM 的设计被纳入自己的产品设计。例如，我们很快就会看到 mbed（本书的主题）采用 ARM Cortex 内核。然而，在 mbed 中也存在 LPC1768 微控制器。该微控制器并非由 ARM 公司生产，而是由恩智浦半导体（NXP Semiconductors）公司生产。ARM 出售给 NXP 包括了 Cortex 内核在内的 LPC1768 微控制器许可，ARM 然后再买回来，并把他们植入 mbed。明白了吗？

1.4.2 技术细节：RISC 的意义

ARM 最初在名字中植入了 RISC 这个概念，因为 RISC 仍然是 ARM 公司设计的一个重要特征，自然我们值得去理解 RISC 到底是指什么。我们已经看到，任何一个微控制器执行的程序都来自由 CPU 硬件本身定义的指令集。在微处理器发展的早期，设计师们试图尽可能使指令集先进和复杂。他们所付出的价格也使得计算机硬件更复杂，更昂贵，效率更低。这样的微处理器称为复杂指令集计算机（CISC）。上述 6502 和 8088 都属于 CISC 时期里的主导产品。CISC 的一个特点是其指令有不同程度的复杂性。CISC 中简单的指令可以用一个字节的数据表示，并可以迅速执行。复杂的指令可能需要几个字节的代码来定义它们，并且需要很长的时间来执行。

随着编译器的不断改进和高级计算机语言的发展，使得关注原始指令集的能力不再那么有用。毕竟，如果你使用高级语言编程，编译器解决大多数编程问题的难度不大。

另一种设计 CPU 的方法是使得 CPU 尽可能简单，并且保持一个有限的指令集。这就出现了 RISC 方法——精简指令集计算机。相对于 CISC，RISC 方法看起来像一个“返璞归真”的举动。一个简单的 RISC CPU 可以快速地执行代码，但它相对于 CISC 可能需要执行更多的指令完成特定任务。随着内存的价格变得越来越便宜，内存密度的不断提高，以及使用更高效的编译器生成程序代码，RISC 的缺点变得越来越少。RISC 方法的一个重要特征是每条

指令都包含在单个二进制字中。这个字包含一切必要的信息，包括指令代码，以及任何需要的地址或数据信息。RISC 方法的另一条特征是每一条指令通常需要相同数量的时间来执行。这样的设计使得很多有用的计算机设计功能得以实现，流水线就是一个很好的例子。当一条指令执行时，下一条指令已经从内存中取出。在 RISC 体系结构中，所有（或大多数）指令很容易在相同数量的时间内完成。

事实证明，RISC 概念的一个有趣的插曲是：正是由于它的简单，往往使得 RISC 设计的功耗很低。这对于任何一个使用电池供电的产品来说都是十分重要的，这样也解释了为什么移动电话中大多使用 ARM 产品。

1.4.3 Cortex 内核

Cortex 微处理内核是一系列杰出的 32 位 ARM 处理器。在图 1.6 中显示了一种 Cortex-M3 内核的简化框图。其中再次展现了一些已知的细节，同时也提供了一些新的想法。在图 1.6 中你可以发现 ALU——计算机的运算核心。指令代码通过取指机制从程序内存中取出。由于使用了管道技术，使得当一条指令在执行时，下一条指令已经开始解码，并且再下一条指令正在从内存中取出。在执行每条指令时，ALU 同时从内存接收数据和（或）将其传送回内存。这些操作都是通过接口模块完成的。内存本身并不是 Cortex 内核的一部分，但是 ALU 中仍然有一组与内存相关的寄存器。有一些微小的本地内存块可以在计算进行时快速访问和用于保存临时数据。Cortex 内核还包括一个中断接口。中断是任何计算机结构的一个重要特征。它们是外部输入，可用于强制 CPU 从正在执行的程序部分跳转到一些其他的代码段。中断控制器管理各种中断输入。不难想象这个微处理器内核在如图 1.4 所示微控制器框图中的位置。

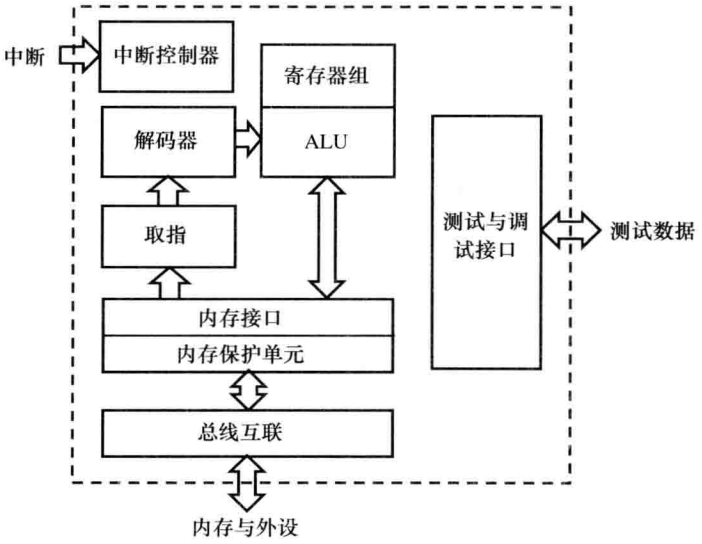


图 1.6 Cortex-M3 内核的简化框图

Cortex 有很多版本，Cortex-M4 是其中最“智能”的版本，它拥有数字信号处理功能。

我们即将使用的 Cortex-M3 内核是专门用于嵌入式应用的，包括汽车及工业。Cortex-M1 是一种小型处理器，可配置在现场可编程门阵列（FPGA）上。FPGA 是一个片上数字电路，当上电时可以配置，并且可以根据需要在操作期间重复配置。Cortex-M0 是 Cortex 系列中最简单的一个，它拥有最小的尺寸和最低的功耗。

因为 mbed 中使用 Cortex-M3 内核，所以我们将要深入了解它。参考文献 1.2 给出了这个内核的一个非常详细的指南。但是，除非你真的很想深入了解，否则，不要尝试阅读它。这是非常复杂的！

本章回顾

- 嵌入式系统包含一个或多个可以被控制的微型智能计算机。
- 嵌入式计算机通常采用微处理器的形式，微处理器包括微处理器内核内存与外设关联起来的微控制器。
- 嵌入式系统设计结合了硬件（电子、电气和机电）和软件（程序）的设计。
- 每一个嵌入式微控制器都含有一套指令集，这也是程序员编写程序的最终目标。
- 大多数编程工作都由高级语言完成，通过编译器转化为可以被微控制器识别的二进制指令集。
- ARM 公司已开发出一系列有效的微处理器和微控制器设计，并广泛应用于嵌入式系统。

习题

1. 解释下列缩写：IC、I/O、ALU、CPU。
2. 用少于 100 个字描述嵌入式系统。
3. 微处理器与微控制器有什么不同？
4. 16 位 ALU 可以表示的数字范围是多少？
5. 在嵌入式系统中“总线”是什么意思？简述嵌入式系统中的两种总线。
6. 简述“指令集”的概念，并解释指令集在高级语言与低级编程语言中使用的不同。
7. 嵌入式程序开发周期中最重要的步骤是什么？
8. 解释 RISC 与 SISC 的优缺点。
9. 什么是管道技术？
10. ARM 作为公司名称的缩写的意义是什么？

参考文献

1.1 Wilmshurst, T. (2001). An Introduction to the Design of Small-Scale Embedded Systems. Palgrave.
1.2 Yiu, J. (2010). The Definitive Guide to the ARM Cortex-M3. 2nd edition. 2010 Newnes.

第 2 章
mbed 开发板

2.1 mbed 简介

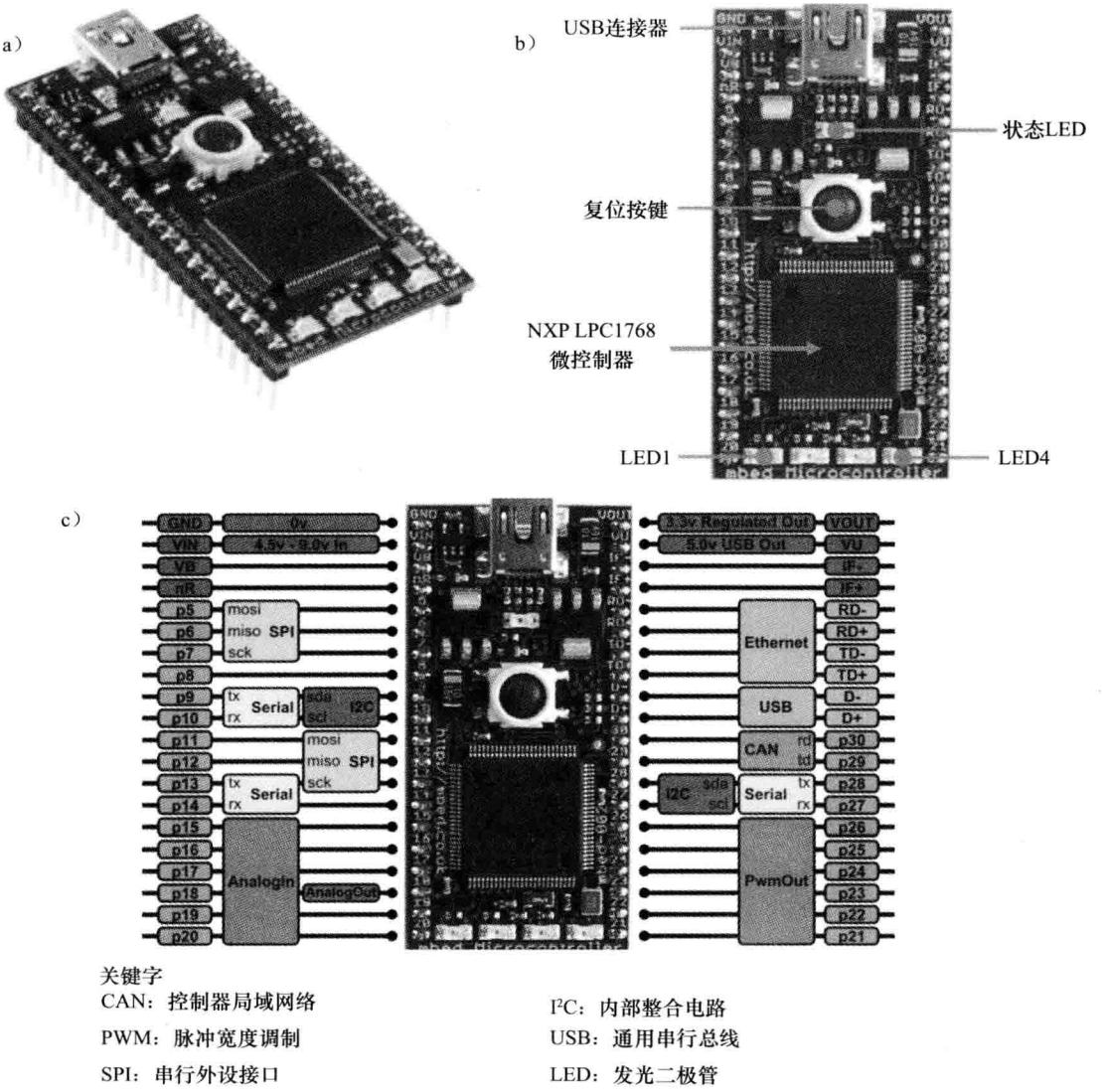
第 1 章回顾了计算机、微处理器和微控制器的一些核心特性。现在，我们要应用这些知识开始本书主要内容 ARM mbed 的学习和研究。

从广义上来说，mbed 有一个如图 1.4 所示的微控制器，在它的周围还有一些非常有用的支持电路。mbed 的微控制器和支持电路全部实现在一块小尺寸的 PCB（Printed Circuit Board，印制电路板）上，并且拥有在线编译器、程序库和开发手册的支持。它提供给用户一个完整的嵌入式系统开发环境，允许用户简单、高效、快速地进行嵌入式系统的开发和原型设计。其中，快速原型设计是 mbed 开发的主要特点之一。

mbed 的 PCB 尺寸为 2 英寸 ×1 英寸（53mm × 26mm），它总共拥有 40 个引脚，分为两排，每排 20 个引脚，引脚间距采用多数电子器件遵循的 0.1 英寸的标准间距。图 2.1 所示为不同视图的 mbed，图 2.1b 中标记出了 mbed 的主要器件，可以看出 mbed 以 LPC1768 微控制器为核心，该微控制器由 NXP 半导体公司生产，包含 ARM Cortex M3 内核。通过 USB（Universal Serial Bus，通用串行总线）接口实现程序下载和供电。电路板上还有 5 个有用的 LED（Light Emitting Diode，发光二极管），其中 1 个用于状态指示，其余 4 个则连接到微控制器的 4 个数字信号输出引脚上。mbed 的这些器件使得它无需在外部连接器件即可进行最小系统的测试。电路板还有一个复位按键，能够强制重启当前的程序。

图 2.1c 中清楚标识了 mbed 每个引脚的功能。在许多情况中为了多种设计选择的需求，引脚能够共享几个功能。电源和地引脚位于左上角，电路板的实际内部运行电压为 3.3V，但是电路板承受的输入电压范围为 4.5 ~ 9.0V，这是因为板载稳压器能够将输入电压降为所需电压。右上角引脚为 3.3V 输出电压，在它之下的引脚是 5V 输出，其余的引脚则用于连接 mbed 外围设备。这些外设都是后面章节介绍的主要内容，可能尽管现在你对其理解有限，我们还是在这里快速概述一下。mbed 有不少于 5 个串行接口类型：I^2C、SPI、CAN、USB

和以太网，还有一组用于读取传感器值的模拟输入和一组用于控制外设的 PWM 输出，例如直流电机。另外，虽然图 2.1 中没有明显的说明，但引脚 5 ~ 30 也可以配置为通用数字输入 / 输出。



关键字

CAN：控制器局域网络

PWM：脉冲宽度调制

SPI：串行外设接口

I²C：内部整合电路

USB：通用串行总线

LED：发光二极管

图 2.1 ARM mbed（图片经 ARM Holdings 公司许可转载）

注：小写形式的信号（如 mosi、miso 或 sck）在相关章节中会介绍。

mbed 构造便于进行原型设计，这当然是其目的。它的 PCB 密度非常高，与外部的互联通过健壮性非常高的传统双列式引脚布局实现。

mbed 的相关信息及其支持工具可以在 mbed 主页找到（参考文献 2.1）。虽然这部书能

够给你使用 mbed 进行开发工作时需要的所有信息，但是你仍然不可避免地需要密切关注网站上的指导文档、手册、博客和论坛，最重要的一点是，该网站提供 mbed 编译器的接入点，只有通过它你才能开发项目。

2.1.1 mbed 体系结构

mbed 体系结构的框图如图 2.2 所示，将这幅图与实际的 mbed 联系起来可能非常有帮助。可以从图 2.1 和图 2.2 清楚地看到，mebed 的核心是 LPC1768 微控制器。图 2.1c 所示的 mbed 信号引脚与微控制器直接连接，因此，根据这一特性，在以后的章节中，当我们使用一个 mbed 数字输入或输出或模拟输入或任何其他的外设接口时，就相当于直接连接到了 mbed 的微控制器上。有趣的一点是 LPC1768 拥有 100 个引脚，但是 mbed 却只有 40 个。当我们深入了解 LPC1768 时会发现，有一些特性对 mbed 用户而言是无法使用的，但是，这不太可能是一个限制使用 mbed 开发的因素。

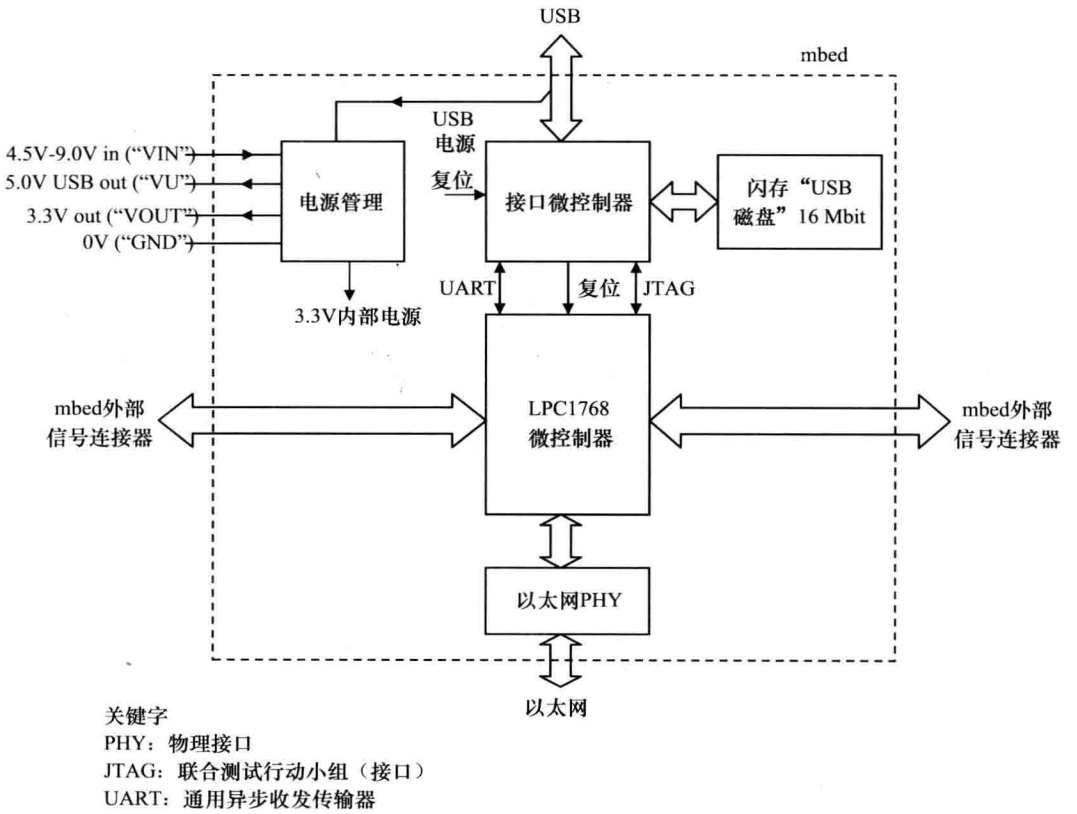


图 2.2 mbed 结构框架图

mbed 上还有一个与 USB 接口的微控制器，它在图 2.2 中称为接口微控制器，是 mbed

PCB 背面上最大的 IC（Integrated Circuit，集成电路）。mbed 硬件设计的明智之处就在于接口控制器设备管理 USB 连接并作为 USB 终端连接至宿主计算机。通常情况下，接口控制器从 USB 接口接收程序代码文件并将程序存放到一个充当“USB 磁盘”的 16Mbit 大小的存储器中。当把一个程序的二进制代码下载到 mbed 时，把代码放置到 USB 磁盘中，按下复位按键后，最近下载的程序将被写入 LPC1768 的闪存中并开始执行。接口微控制器和 LPC1768 之间通过 LPC1768 的 UART 端口（即通用异步接收器 / 发送器——一种串行数据链路，此处不予赘述）传输串行数据实现数据传输。

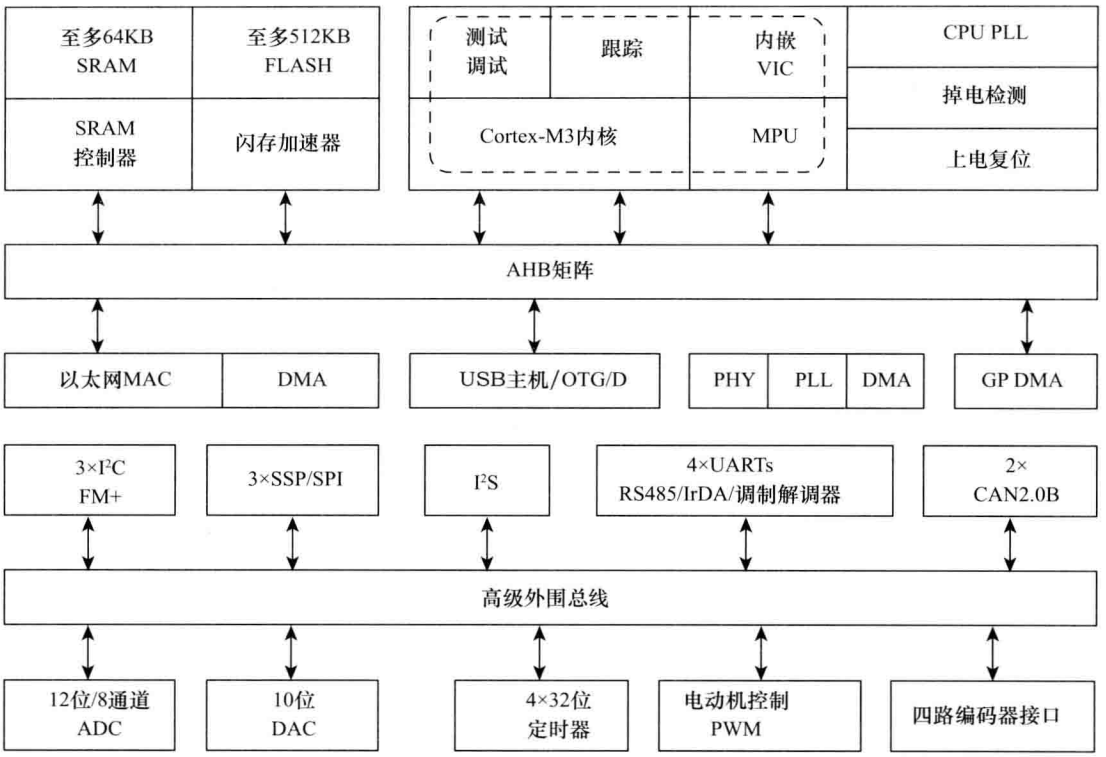
mbed 的电源管理单元由两个稳压器组成，分别位于状态指示灯的两侧。mbed 上还包括一个限流 IC，位于开发板的左上方。mbed 最常使用的方式是从 USB 供电，特别是对简单的应用。对于更耗电或者需要更高电压的应用，mbed 也可以使用一个外部输入的 4.5 ~ 9.0V 电压提供给引脚 2（标记为 VIN）进行供电。mbed 还可以通过引脚 39 和 40（分别标为 VU 和 VOUT）提供电源。VU 提供的 5V 供电连接，因为直接来自 USB 连接，所以只有在 USB 连接方式下可用。VOUT 引脚提供从 USB 或 VIN 输入转换产生的规范 3.3V 供电。

mbed 的电路图可以在 mbed 网站上找到（参考文献 2.2）。

2.1.2 LPC1768 微控制器

LPC1768 微控制器的框图如图 2.3 所示。图 2.3 看上去很复杂，并且我们也不想了解一个极其复杂的数字电路的所有细节。但是这张结构框图某种意义上是这本书的简要概括，它包含了 mbed 的所有功能，所以了解图中 LPC1768 微控制器的主要功能还是有必要的。而如果读者想要完整地了解这款微控制器的详细信息，可以去查阅参考文献 1.2、2.3 和 2.4。虽然本书中会不时提到这些参考文献，但也不必为了完整阅读本书而必须查阅它们。

如图 1.4 所示，微控制器由微处理器内核、存储器和外设接口组成。图 2.3 顶部中间的虚线框中是这个微处理器的内核——ARM Contex-M3，这是图 1.6 中所示 M3 内核框架的简化版。内核的左边是存储器：其中采用 Flash 技术制造的程序存储器用于程序的存储，Flash 存储器的左边是静态 RAM（Random Access Memory，随机存取存储器），用于保存临时数据。图 2.3 中剩下的大部分内容都是外设接口，这些外设接口使得微控制器具有嵌入的能力。位于图 2.3 的中间和下半部分的内容基本上准确反映了 mbed 可以做哪些工作。将这张图中的外设接口同图 2.1c 中 mbed 的输入和输出信号比较一下，你会发现许多有趣的地方。最终，所有模块需要连接到一起，这个任务则由数据总线和地址总线完成，但我们对微控制器这方面的设计几乎没有兴趣，至少这本书没有，只需要知道外设接口通过一个叫做先进外设总线的东西进行连接，再依次通过一个叫做先进高性能总线矩阵的互连总线连接到 CPU（Central Processing Unit，中央处理单元）就足够了，这种互连关系并没有完全在这幅图上展示出来，但我们既不需要也不想进一步去研究它。



关键字

ADC：模/数转换器
AHB：先进高性能总线
CPU：中央处理单元
D：设备（USB）
DAC：数/模转换器
DMA：直接存储器访问
FMþ：快连模式 +（I²C）
GP：通用（DMA）
12C：内部整合电路
12S：内部整合电路音频
IrDA：红外数据协会
MAC：媒体访问控制
MPU：内存保护单元
PHY：物理层
PLL：锍相环
OTG：On-The-Go（USB）
SPI：串行外设接口
SRAM：静态RAM
SSP：同步串行接口
UART：通用异步收发传输器
VIC：矢量中断控制器

图 2.3 LPC1768 结构框图

注：LPC 1768 有 64KB SRAM 和 512KB 闪存。

Cortex 内核由虚线框中的部分组成。

参见图 2.1 中关键字（图片经 mbed NXP LPC 1768 原型设计板的许可转载 .2009.NXP BV. 文档编号 9397 750 16802）

2.2 mbed 入门教程

本教程的内容非常重要，你将第一次连接 mbed，并运行第一个程序。我们将按照 mbed

网站提供的流程，编译示例程序，实现一个简单的 LED 闪烁功能。你需要准备好：

1）mbed 微控制器和 USB 线

2）运行 Windows（XP/Vista/7）、Mac OS X 或 GNU/ Linux 的计算机

3）Web 浏览器，如 Internet Explorer 或 Firefox

本教程旨在介绍在 mbed 上运行程序的主要步骤。第 3 章将介绍编程的细节。现在，我们按照以下步骤开始学习 mbed 入门教程。

2.2.1 步骤 1：连接 mbed 到 PC

使用 USB 线将 mbed 连接到 PC。状态指示灯将亮起，说明 mbed 已上电。几秒钟后，PC 会将 mbed 识别为标准的可移动驱动器，mbed 会出现在所连接的计算机的设备列表中，如图 2.4 所示。

a）Windows XP示例　　b）Mac OS X示例

图 2.4　mbed 在设备列表中的位置

2.2.2 步骤 2：创建 mbed 账户

在 Web 浏览器中打开 mbed 上的 MBED.HTM 文件，单击创建新的 mbed 账户链接。按照说明创建一个 mbed 账户，并引导至 mbed 主页网站，如图 2.5 所示。你可以从这里链接到编译器、库和文档。

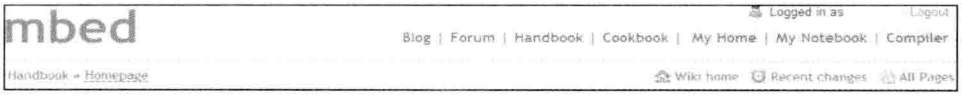


图 2.5　mbed 主页

2.2.3 步骤 3：运行程序

使用网站菜单中的链接（图 2.5 右侧的 Compiler）打开编译器，进入分配的个人编程工

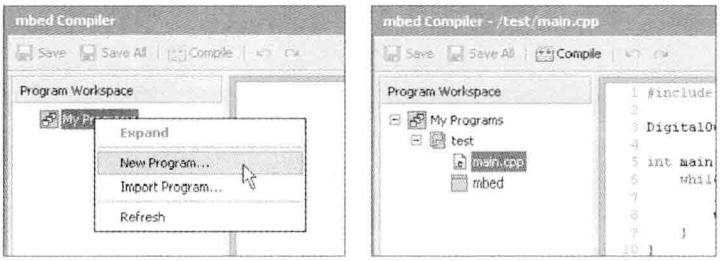
作区，编译器将在新标签页或窗口中打开。请按照下列步骤来创建一个新的程序：

1）如图 2.6a 所示，右击（Mac 用户请按住 Ctrl 键单击）My Programs，然后选择 New Program。

2）选择并输入新程序名（例如 Prog_Ex_2_1），然后单击 OK 按钮。注意，程序名中不能含有空格。

3）新的程序文件夹将创建在 My Programs 目录下。

单击并在文件编辑窗口打开 main.cpp 文件，如图 2.6b 所示。这是程序的主要源代码文件。新创建的程序总是包含相同的简单代码，如程序示例 2.1 所示。该程序将在下一章详细介绍。



a）选择新建程序　　b）打开main源代码文件

图 2.6　打开一个新的程序

程序文件夹中的其他项是 mbed 库文件。库文件提供了用于启动和控制 mbed 所有的功能，如本示例中使用的 DigitalOut 接口。

程序示例 2.1　简单的 LED 闪烁

```
/* 程序示例 2.1: 简单的 LED 闪烁
*/
#include "mbed.h"
DigitalOut myled(LED1);
int main() {
  while(1) {
    myled = 1;
    wait(0.2);
    myled = 0;
    wait(0.2);
  }
}
```

2.2.4　步骤 4：编译程序

单击工具栏中的 Compile 按钮编译程序。这将编译程序文件夹中所有的源代码文件，并创建下载到 mbed 中的二进制机器代码。通常编译的内容是你编写的程序以及调用的库。编译成功后，编译器将输出一条“Success!”的消息，并弹出会一个提示框，提示你将编译产

生的 .bin 文件下载到 mbed 中。

示例程序当然不存在错误，但是你将来编程时难免会遇到错误。尝试在源代码插入一个小错误，例如，删除代码行末尾的分号，并再次编译。注意编译器在屏幕底部输出的错误提示。改正错误，重新编译，然后继续。刚才插入错误的类型通常称为语法错误。该错误涉及 C 语言的代码行编写规则。编译器遇到语法错误时，将无法进行编译，编译器认为代码已超出了既定的语言规则，因此不能正确地编译所写代码。

2.2.5 步骤 5：下载程序二进制代码

在编译完成后，程序代码能够以二进制的形式被下载到 mbed 中，并保存到 mbed 驱动器的位置处。程序下载时，状态指示灯（如图 2.1 所示）将会闪烁。状态指示灯停止闪烁后，按下 mbed 的复位按钮来运行下载的程序。现在，你应该看到 LED1 每隔 0.2 秒闪烁一下。

2.2.6 步骤 6：修改程序代码

在 main.cpp 文件中，更改 DigitalOut 的声明如下：

```
DigitalOut myled(LED4);
```

重新编译代码并下载到 mbed 中。现在你应该看到 LED4 闪烁，而不是 LED1。你也可以通过修改 wait() 命令括号内的值，来改变闪烁的频率。

2.3 开发环境

正如 1.3 节所提到的，嵌入式系统中有很多不同的开发方法。使用 mbed 开发时不需要安装软件，也不需要额外下载程序的硬件。所有软件工具放置在网上，因此，无论在哪里，只要可以访问互联网，就可以编译和下载。值得一提的是，mbed 有一个 C++ 编译器和一套用于驱动外围设备的丰富软件库。因此，没有必要编写代码来配置外围设备，而这在某些系统中是非常耗费时间的。

2.3.1 mbed 编译器和 API

mbed 开发环境使用 ARM RVDS（RealView 开发套件）编译器，目前的版本是 4.1。此编译器与 mbed 相关的所有功能都可通过 mbed 门户网站获得。

让 mbed 与众不同的是它配备了 API（Application Programming Interface，应用程序编程接口）。简单地说，这是一组能够作为 C++ 实用程序的编程构造块，允许快速而可靠地定制程序。因此，我们将按照 API 的功能编写 C 或 C++ 代码。通过这本书，你将会了解 API 的大多数功能。你也可以从 mbed 主页的链接打开 mbed 手册，查看 API 的所有组件。

2.3.2 C/C++ 的使用

正如上文所提到的，mbed 开发环境使用 C++ 编译器。这意味着，所有文件都会携带 .cpp（C plus plus）的后缀名。而 C 是 C++ 的子集，因为它没有使用 C++ 更高级的面向对象方面的概念，所以学习和应用更简单。一般来说，C 代码能够在 C++ 编译器下编译，反之则不行。

C 语言通常是任何复杂度较低或中等的嵌入式程序选择的语言，所以在这本书中，C 语言很适合我们。为了简单起见，我们打算只使用 C 开发程序。但是应该承认，使用 C++ 编写的 mbed API 充分利用了这种语言的特点。我们的目标是当我们想起这些 API 时，我们能概述出它们的基本功能。

C语言语法

本书假设读者没有任何的 C 或 C++ 基础，你如果有，则这是学习本书的一个优势。我们旨在介绍 C 的所有新特性，当出现这些内容时将使用左侧边沿的符号进行标记。当你看到该符号时，如果你是一位 C 语言专家，那意味着你可以略过本部分，如果你不一个专家，你需要参考附录 B 仔细阅读这部分，附录 B 总结了用到的所有 C 功能。即使你是一个 C 语言专家，但你可能没有在嵌入式环境下使用过它。通过这本书，你将看到大量在特定环境中用来优化语言的技巧和技术。

本章回顾

- mbed 是一个结构紧凑的、基于微控制器的硬件平台，有一个程序开发环境。
- 计算机到 mbed 的通信是通过 USB 电缆实现的，也可以通过这条链路供电。
- mbed 有 40 个引脚可以连接到外部电路，电路板上有 4 个用户可编程的 LED，因此可以在引脚没有外部连接的情况下运行一些非常简单的程序。
- mbed 使用了 LPC1768 微控制器，该微控制器包含 ARM Cortex-M3 内核，大多数的 mbed 连接直接连接到微控制器引脚，mbed 的很多特点直接来自微控制器。
- mbed 开发环境托管在 web 上，需要在线进行程序开发，并且程序存储在 mbed 服务器上。

习题

1. ADC、DAC 和 SRAM 分别代表什么？
2. UART CAN I^2C 和 SPI 分别代表什么？这些 mbed 特性有什么共同之处呢？
3. mbed 提供多少路数字输入？
4. mbed 哪些引脚可以用于模拟输入和输出？
5. mbed PCB 上有多少微控制器？它们分别是什么？
6. mbed 编译器软件的独特之处是什么？
7. mbed 是一个 9V 电池供电电路的一部分。编程后 mbed 从 USB 断开连接。mbed 连接的外

部电路一部分需要9V供电电压，另外一部分需要3.3V供电电压。在没有其他电池和电源可以使用的情况下，绘图说明如何建立这些电源连接。

8. mbed连接到一个系统，需要与三个模拟输入，一个SPI、一个模拟输出和两个PWM输出连接。绘制草图显示如何进行连接，并标出mbed引脚号。
9. 一位朋友在mbed编译器中输入如下所示代码，但在编译时提示有多个错误，请找出这些错误并加以改正。

```
#include "mbed"
Digital Out myled(LED1);
int man() {
    white(1) {
        myled = 1;
        wait(0.2)
        myled = 0;
        watt(0.2);
    }
```

10. 在不将LPC1768微控制器所有的引脚都连接到mbed外部引脚的情况下，有一些微控制器外设接口会失去作用。指出下列哪些外设接口如此：ADC、UART、CAN、I^2C、SPI和DAC。

参考文献

2.1 The mbed home site. http://mbed.org/
2.2 MBED Circuit Diagrams. 26/08/2010. http://mbed.org/media/uploads/chris/mbed-005.1.pdf
2.3 NXP B.V. LPC1768/66/65/64 32-bit ARM Cortex-M3 microcontroller. Objective data sheet. Rev. 6.0. August 2010. http://www.nxp.com/
2.4 NXP B.V. LPC17xx User Manual. Rev. 02. August 2010. http://www.nxp.com/

第 3 章
数字输入和输出

3.1 开始编写程序

本章将介绍微控制器的最基本功能：数字信号的输入和输出。除了这些内容之外，还将介绍在程序中如何根据输入值做出相应的控制。图 1.1 已经暗示了在嵌入式系统中时间是非常重要的，因此本章一开始会介绍一些与时间有关的操作。

对于读者来说，可能还没有接触过 C 语言编程或者刚开始学习。本章将介绍有关 C 语言编程方面的一些重要概念。如果是第一次接触 C 语言，建议读者先通读 B.1 ~ B.5 节。

在开始学习本章之前，先声明一点：不要求读者具备详实的数字电路知识，但是有一些电子方面的理论知识还是要好一些，这样理解起来会很容易。如果在阅读过程中需要相关理论支持，建议读者随时翻阅相关的参考书，例如，参考文献 3.1。此外，有关电子方面的知识也有很多不错的网站可供查阅。

3.1.1 思考第一个程序

首先看一下本章的第一个程序，这个程序其实就是之前介绍过的程序示例 2.1。这里为了阅读方面，特意在程序中加入了注释。读者可以与第 2 章未加注释的代码进行对比。如果需要 C 语言方面的相关知识，请读者随时查阅附录 B。

注释是程序员在程序中加入的文本消息，它不会影响程序的执行。在代码中大量引入注释是一个很好的编程习惯，它有助于程序员在编写程序时梳理自己的头绪，以后再次阅读程序时，起到提示程序功能的作用，而且有助于他人阅读和理解程序。最后还需要再强调一点，代码注释在团队开发中是非常重要且必不可少的，尤其是在需要移交代码时，注释带来的好处就能够显现出来。

在代码中有两种加注释的方法，这两种方法在本例中都出现了。第一种方法是采用“/*”和“*/”这对符号进行注释，这种方式可以对一段代码或者多行代码进行注释。另一种方法是使用两个正斜杠“//”进行注释，编译器编译时会忽略掉当前行中“//”之后的所有文本，因此可以在“//”之后添加文字信息，从而起到对当前行注释的作用。

本例一开始有三行注释，用的是第一种注释方法，目的是简要描述该程序的功能。本书中后面出现的程序全部采用这种风格，即一开始先描述程序的功能。请注意，在程序中有些地方加入了一些空行，这样做主要是为了增加程序的可读性。此外，对程序中的关键代码也加了注释，便于读者理解这段程序。

程序示例 2.1　便于理解在程序中加入了注释

```
/* 程序示例 2.1：本程序用来控制 mbed 上 LED1 的亮灭。本程序演示了数字输出和延迟函数的使用。
本程序取自 mbed 官方网站
*/
#include "mbed.h"   // 在程序中包含 mbed 头文件
// 定义程序变量 myled，并绑定到 mbed 的 LED1 上
DigitalOut myled(LED1);
int main() {          // 函数起始处
  while (1) {         // 创建一个连续循环
     myled = 1;       // 输出逻辑 1，led 点亮
     wait(0.2);       // 延迟 0.2 秒
     myled = 0;       // 将 led 熄灭
     wait(0.2);       // 延迟 0.2 秒
   }                  // while 循环结束
}                     // 主函数结束
```

C语言语法

现在我们来学习一下程序示例 2.1 中相关的 C 语言语法。首先要明确一点，任何 C 或 C++ 程序都包含在 main() 函数中，所以始终应该从 main() 函数开始阅读程序。这里特意写成 main() 的形式，即 main 后连接一对小括号，用来告诉读者这是 C 语言的一个函数，本书中所有用到的函数都采用这种形式表示。main() 函数之后紧跟着的是一个左大括号，之后就是函数定义（function definition），也就是函数具体的功能，这部分直到最后一个右大括号才结束。在这对最外层大括号内还嵌套多对大括号。通过这些成对出现的大括号，我们能够有效组织 C 程序的代码结构，当然，还可以使用其他很多方法。

C语言语法

嵌入式系统中的很多程序都是由一个无限循环组成，即运行时不断地重复一段程序。在本例中，循环是通过 while 关键字实现的，该关键字的作用范围为随后大括号内的那段代码。B.7 节介绍 while 循环，该循环在满足特定条件时，会反复执行一段代码。如果写成 while(1) 的形式，就可以利用 while 语句的特点实现无限循环。

C语言语法

本例中 while 循环的实际部分是由 4 行代码构成的，这 4 行代码中有两个库函数调用和两条赋值语句。函数调用 wait() 是 mbed 中的库函数，可以实现延时等待的功能，其他相关函数见表 3.1。此处函数的参数是 0.2，延时单位为秒，因此可实现 0.2 秒的延时（当然，本例中也可设置其他值，实现不同的延时）。代码中的两条赋值语句实现了对变量 myled 值的修改，这是本书第一次使用 C 运算符。该运算符为赋值

（assign）运算符，用常用的等号表示。在 C 语言里赋值运算符的作用是将一个表达式的值赋给一个变量。这样

```
myled = 1;
```

表示无论变量 mled 之前是何值，执行该语句后把变量 mled 设置为 1。而通常意义上的“相等”在 C 语言里用两个等号“==”表示，本章后面会用到这个运算符。在 C 中还有很多运算符，第一次遇到时需要注意并学习如何使用。B.5 节对这些运算符做了一个汇总，请读者随时查阅。

表 3.1 mbed 库中的延时函数

C/C++ 函数	功 能
wait	等待指定的秒数
wait_ms	等待指定的毫秒数
wait_us	等待指定的微秒数

C语言语法

程序开始处有一个非常重要的头文件“mbed.h”，这是用来连接 mbed 所有库函数的。在正式编译前，通过编译器指令 #include，将头文件的内容一字不差地插入程序中。如何使用 #include 编译器指令，可参阅附录 B.2.3 节。该程序使用了 DigitalOut 实用程序库定义数字输出，这个程序库是 mbed 的应用程序编程接口（API），本书之后统一称为 API 函数。本例中数字输出定义为 myled。myled 一经声明，在程序中就可以当做一个变量使用。LED1 在 mbed 的 API 函数中作为保留字用于输出，与 mbed 板上的某个发光二极管（LED）相关联，即图 2.1 上标注的 LED1。

本例中，main() 一开始有两个缩进空格，接着 while 代码块中又缩进了两个空格。请注意，这不会对程序的执行产生任何影响，主要是为了增加程序的可读性，从而减少编程错误。这是一个很好的编程习惯，对于其他一些编程习惯读者可查阅 B.11 节。在公司的项目开发中，编写 C 程序时常采用印刷厂或出版社使用的排版方式，用来保证程序员编写的代码有良好的可读性，而且能够保持代码风格一致。

练习 3.1

通过对程序做以下修改，进一步熟悉程序示例 2.1，以及掌握程序编译的大体过程。请注意观察修改效果。

1. 在给出的示例程序中添加注释，观察程序编译或运行时是否受到影响。
2. 将 wait() 函数中的参数 0.2 修改为其他值。
3. 用 wait_ms() 函数替代 wait() 函数，并设置合适的参数实现相同的等待时间。当然，也可以设置不同的等待时间。
4. 分别使用 LED2、LED3、LED4 替代程序中的 LED1。
5. 使用两个或多个发光二极管并多次调用 wait() 函数，实现一个更复杂的灯光模式。

3.1.2 了解 mbed 的 API 函数

本书中会多次出现对 mbed API 的使用，对于读者来说，熟悉 API 的使用方式非常重要。mbed 的 API 库由一系列实用程序组成，有关库函数的所有内容在 mbed 网站提供的帮助指南上都已经逐项列出。我们接触到的第一个实用程序是 DigitalOut。该 API 函数的相关信息如表 3.2 所示。

表 3.2 mbed 数字输出 API 汇总（引自 www.mbed. org）

函　数	用　途
DigitalOut	创建一个连接到指定引脚的 DigitalOut 对象
write	设置输出，指定为 0 或 1（int）
read	返回输出设置，用 0 或 1（int）表示
operator =	write() 的简写形示
operator int()	read() 的简写形示

读者会很快熟悉 mbed API 函数的使用方式，如 DigitalOut API 函数会生成一个名为 DigitalOut 的 C++ 类。表 3.2 中列出了 DigitalOut 类中的一组成员函数。其中第一个函数是 C++ 中的构造函数，该函数名与类名相同。构造函数用来创建 C++ 对象。通过 DigitalOut 构造函数，可以创建 C++ 对象。程序示例 2.1 中创建了 DigitalOut 类的一个对象 myled。通过函数 write() 和 read() 可以实现对 myled 的读写。DigitalOut 类中有多个成员函数，调用时可以采用 myled.write() 的形式，其他函数使用方式与此相同。一旦创建 myled 对象，类中定义的 API 运算符就可以调用。这里以表 3.2 中提到的赋值运算符“=”为例做一说明。当程序中给出

```
myled = 1;
```

该变量的值，不仅会像标准 C 里那样修改，还被输出到对应的数字输出引脚上。这条语句可以用来代替 myled.write(1)。读者还会发现，在 mbed API 函数中所有与外设相关的函数都有类似的运算符。

3.1.3 分析 while 循环

程序示例 3.1 是基于前面的程序编写的，但它仍然可以被看成是本书中最原始的程序。在 mbed 编译器中创建一个新程序，然后将书中代码复制进去。

C语言语法

首先让我们看一下这个程序的结构，整个程序由三条 while 语句构成。第一条就是之前用到的 while(1)，然后是两个带条件的 while 循环。后面这两条语句的循环条件与变量 i 的值有关。前一个循环条件是：只要 i 小于 10 就重复执行，循环体对 i 进行自增操作，直到 i 等于 10 循环结束。在后一个条件循环中，变量 i 是自减的，只要 i 大于 0 就重复执行。

程序示例 3.1 中出现了一条新的 C 语言语法规则，这条规则非常重要，即变量 i 要

在 main() 开始处声明。对于任何数据类型的变量而言，必须在使用前声明。B.3 节给出了 C 语言中所有的数据类型。在本例中，i 被声明为一个有效数字位为 8 位的字符型变量。在声明的同时，该值被初始化为 0。本例中还引入了 4 个新的运算符：“+”、“-”、“<”和“>”。这些运算符和传统代数中对应运算符的含义是相同的，因此理解起来也很容易。

程序示例 3.1　while 的使用

```
/* 程序示例 3.1: 演示 while 循环的使用。不需要外部连接
*/
#include "mbed.h"
DigitalOut myled(LED1);
DigitalOut yourled(LED4);

int main() {
  char i=0;          // 声明变量 i 并设置为 0
  while(1){          // 无限循环开始
  while(i<10) {      // 第一个条件循环起始处
    myled = 1;
    wait(0.2);
    myled = 0;
    wait(0.2);
    i = i+1;         // i 加 1
  }                  // 第一个条件循环结束
  while(i>0)  {      // 第二个条件循环起始处
    yourled = 1;
    wait(0.2);
    yourled = 0;
    wait(0.2);
    i = i-1;
  }
    }                // 无限循环结束
}                    // 主程序结束
```

编译程序示例 3.1，然后将其下载到 mbed 板上运行。可以看到板子上的 LED1 和 LED4 轮流闪烁 10 次。请读者思考为什么会出现这样的实验结果。

练习 3.2

C语言语法

1. 通过使用递增和递减运算符可以实现变量的递增或递减，这种方式使代码看起来更优雅一些，关于这两个运算符的使用可参阅 B.5 节。读者可以试着用 i++ 和 i-- 分别取替换代码中的 i=i+1 和 i=i-1，然后再运行一次程序。
2. 修改程序，让两个发光二极管每轮仅闪烁 5 次。
3. 用 myled.write(1) 替换代码中的 myled=1，观察执行效果。

3.2　用电压表示逻辑值

计算机处理的二进制数，由大量的二进制位或比特组成，它的每一位都代表逻辑值 0 或 1。现在，我们已经开始正式使用 mbed 了，但有一个问题值得读者思考：在实际工作中，mbed 中的电路以及它的连接引脚是如何表示这些逻辑值的。

在所有数字电路中，逻辑值用电压表示。这种表示方式在数字电路中有一个明显的好处：我们不需要用一个精确的电压来表示逻辑值，而是用一个电压范围表示。这也就意味着，即使一个电压夹带了一些噪音或者信号有些失真，仍然可以认为它是正确的逻辑值。mbed 中使用的微控制器是 LPC1768，它的供电电压是 3.3V。通过查阅 LPC1768 的技术资料，我们可以找到逻辑 0 和逻辑 1 可接受的电压范围。参考文献 2.4（第 2 章）提供了相关数据，附录 C 中已经把该文献中的重要部分罗列出来（现在可以不用去看，需要时可随时查阅）。资料表明，对于大多数数字输入信号，LPC1678 将低于 1.0V（指定为 0.3 × 3.3V）的输入电压视作逻辑 0，高于 2.3V（指定为 0.7 × 3.3V）的输入电压视作逻辑 1。图 3.1 将这一概念用图表的形式表示出来。

如果我们向 mbed 提供一个输入信号，希望该信号所代表的逻辑值能被正确地识别，就要满足图 3.1 中的规定。同样，输出信号时也需要遵从图 3.1 的规定。正常情况下，只要没有电流流动，mbed 用 0V 表示逻辑 0，用 3.3V 表示逻辑 1。假如有电流流过，比如，电流流过一个 LED 时，我们可以预料到输出电压会发生变化。此时前面提到的概念就会非常有用。这里有一个有趣的现象：每输出一个逻辑值，就会到得到一个可预料的电压值。利用该电压，我们就能够点亮 LED（发光二极管），驱动电机或者控制其他一些外围器件。

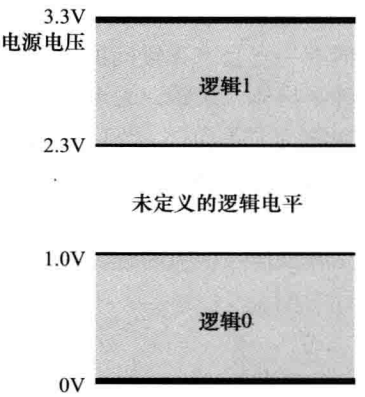


图 3.1　mbed 的输入逻辑电平

只要我们搭建的电路符合图 3.1 中的规定，对于本书中多数应用而言，就无需担心这些电压。但是这里需要提一点：在某些场合有必要关注一下逻辑电压值，以免出现系统不稳定。

3.3　mbed 数字输出

在前面的两个程序中，我们已经对 mbed 板上安装的诊断 LED 做了亮灭控制的实验。mbed 上有 26 个输入 / 输出（I/O）引脚，引脚编号依次为 5 ~ 30，这些引脚可以设置成数字输入或输出，如图 3.2 所示。将该图与 mbed 引脚总体示意图（图 2.1c）做一比较，可以发现这些引脚都是多功能的。除了可以作为简单的数字输入 / 输出外，所有这些引脚还都具备另一个功能：可以用来连接微控制器的外设。怎样配置这些引脚，由程序员在程序中设定。

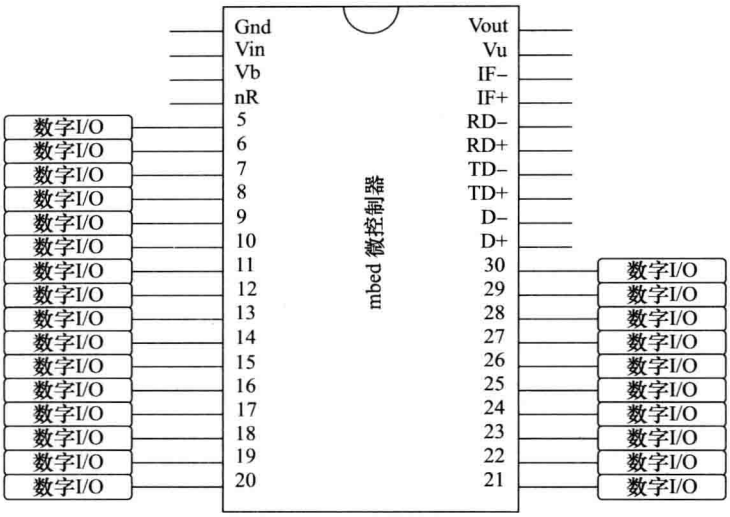
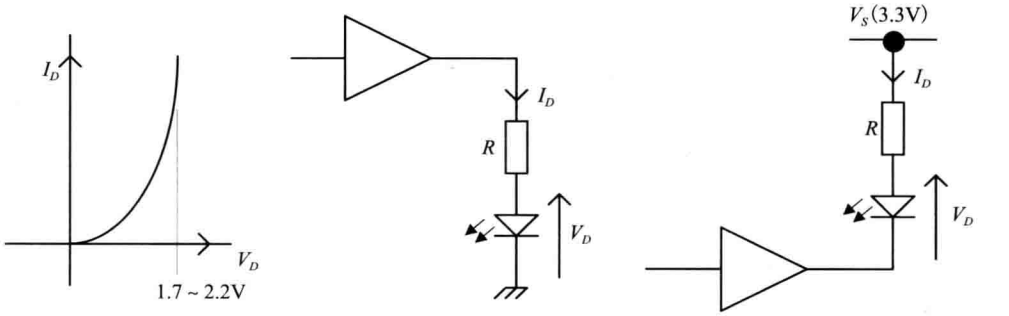


图 3.2 mbed 数字输入 / 输出

3.3.1 发光二极管的使用

现在，我们开始在 mbed 上连接外设了。额外提一句，读者可能不是电子方面的专家，但有一点是毫无疑问的：当把外设连接到 mbed 上，说明读者知道自己在做什么。LED 是一种半导体二极管，表现出与二极管相同的电特性。LED 只允许电流沿一个方向通过，这个方向称为“正向”。当 LED 连接成能够导电时，半导体结上会发出光子，这一现象使得 LED 变得非常有用。LED 的伏安特性如图 3.3a 所示。从图 3.3a 中可以看到，正向电压很小时，会产生非常小的电流。当电压的增加到某一值时，电流突然增大。对于大多数 LED 而言，工作电压在该值附近，典型值为 1.8V 左右。



a）LED伏安特性

关键词

I_D：流过LED的电流

V_S：电源电压

b）输出到LED负载的拉电流

V_D：电流为I_D时，LED两端电压

c）从LED负载输入的灌电流

图 3.3 用逻辑门驱动 LED

可以将一个 mbed 引脚配置成输出，通过把逻辑门电路与 LED 直接相连，如图 3.3b 和图 3.3c 所示。图 3.3 中的门电路（用三角符号表示）可以看成逻辑缓冲。如果采用图 3.3b 中的方式连接，逻辑门输出高电平时 LED 灯亮，电流经门电路流入 LED。若采用图 3.3c 所示连接，门输出为低电平时 LED 点亮，电流流入门电路。通常情况下，为了防止电流过大，在 LED 上需要串接一个限流电阻。当然，若输出电压和门电路内阻共同作用，使电流限制在一个合适的值，此时就不用再接串联电阻了。比如，在下一个程序中推荐使用的 LED，本身就带有串联电阻，因此不再需要连接任何外部电阻。

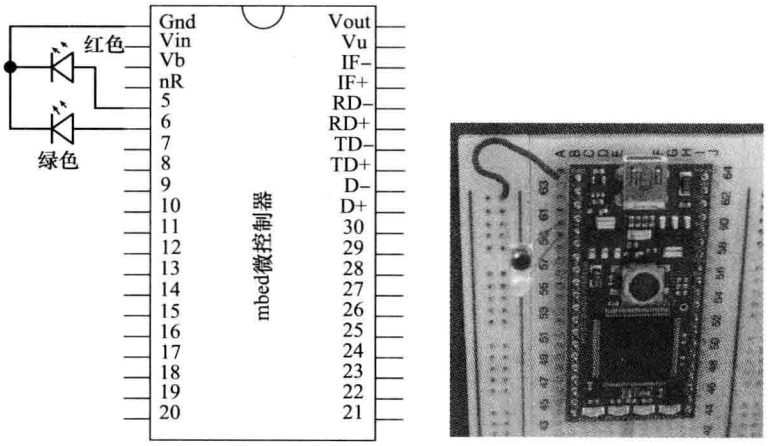
3.3.2　mbed 外部引脚的使用

通过 DigitalOut，数字 I/O 引脚可以命名，而且能够配置为输出状态。就像我们在之前的例子中那样，在程序代码的一开始处定义它们，如

```
DigitalOut myname(p5);
```

通过这种方式可以创建 DigitalOut 对象，该对象可以用来设置引脚的输出状态，并且在需要时能够读取当前状态。

我们按图 3.4a 所示连接电路。附录 D 列出了本例中所有用到的元器件，并给出了产品的型号。当然，也可以选用其他元器件进行替换。在该电路中我们采用的是图 3.3b 中的连接方式。建议采用附录 D 中给出的 LED，该元件内部串联一个约 240Ω 的电阻，所以外部不需要再串接一个电阻。



a）连接图　　b）面包板搭建电路

图 3.4　利用 mbed 实现简单的 LED 闪烁

如图 3.4b 所示，将 mbed 插入面包板，并按图示连接电路。mbed 上引脚 1 是公共地，按照图 3.4b，将该引脚连接到面包板上稍微靠外的插孔中。建议日后搭建类似电路时采用这种方式，并形成一种习惯。需要记清的是，与 mbed 引脚相连的是 LED 的阳极（引脚较长的一侧）。另一侧（阴极）应该连接到地。本例及其他一些电路将采用通用串行总线（USB）供电。

在 mbed 编译器中创建一个新的程序，然后将程序示例 3.2 复制过来。

程序示例 3.2 外部 LED 的闪烁

```
/* 程序示例 3.2: 红色和绿色 LED 按简单的时间模式闪烁
*/
#include "mbed.h"
DigitalOut redled(p5);    // 将引脚 5 定义为数字输出并命名
DigitalOut greenled(p6);  // 将引脚 6 定义为数字输出并命名
int main() {
  while(1) {
    redled = 1;
    greenled = 0;
    wait(0.2);
    redled = 0;
    greenled = 1;
    wait(0.2);
  }
}
```

对代码进行编译、下载后，程序就可以在 mbed 上运行了。我们可以看到面包板上绿灯和红灯交替闪烁，由于这段程序是在程序示例 2.1、3.1 的基础上扩展出来的，因此读者很容易理解工作原理。

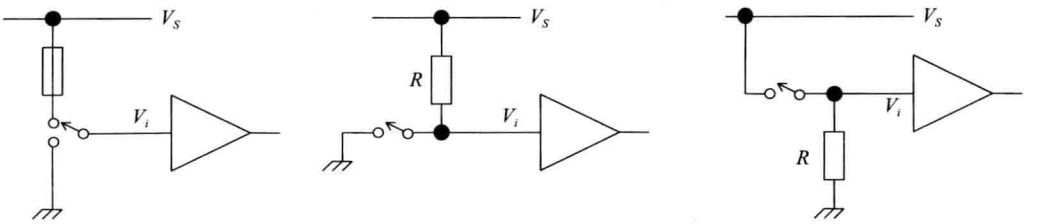
练习 3.3

编写一段程序，使用任何一个输出引脚，通过重复输出逻辑 1 和逻辑 0，输出一个方波。利用 wait() 函数，使输出频率为 100Hz（周期 10ms）。用示波器观察，测出逻辑 0 和逻辑 1 的电压值。思考测量值与图 3.1 有何联系？方波的频率与程序中设定的是否一致？

3.4 mbed 数字输入

3.4.1 开关与数字系统的连接

普通机电开关产生的逻辑电平能够满足图 3.1 的要求。图 3.5 给出了机电开关常见的三种用法。其中最简单的是图 3.5a，使用了一个 SPDT（单刀双掷）开关，下一个例子中就会用到该开关。有些场合中，为了安全起见，在逻辑输入端串接一个电阻。相对于 SPDT 开关而言，SPST（单刀单掷）开关成本更低、尺寸更小，因此得到了广泛应用。在图 3.5b 或 c 中，SPST 开关分别通过上拉电阻或下拉电阻与电源电压相连。当开关打开时，逻辑电平由与之相连的电阻决定。当开关闭合时，跳转为相反逻辑。此时会有电流经电阻消耗掉。如果在此处选择阻值高的电阻器，消耗掉的电流微乎其微。同大多微控制器一样，mbed 内置有上拉和下拉电阻，可以满足外部连接的需要。



a）与单刀双掷开关相连　b）通过上拉电阻与单刀单掷开关相连　c）通过下拉电阻与单刀单掷开关相连

图 3.5　将开关与逻辑输入相连

3.4.2　DigitalIn API

DIgitalIn API 具有数字输入功能，如表 3.3 所示，该表与 DigitalOut（参见表 3.2）的格式完全一致。这部分 API 用来创建名为 DIgitalIn 的类及其成员函数。DIgitalIn 构造函数用来创建数字输入，read() 函数用来读取输入的逻辑值。在实际使用中，读取输入值的简单方法是直接使用数字对象名。在下面的程序中我们就能看到这种用法。与数字输出一样，引脚 5 至引脚 30，这 26 个引脚可以设置成输入。输入电压将根据图 3.1 进行逻辑转换。注意，默认情况下，DIgitalIn 将使能内部下拉电阻，即输入电路按图 3.5c 所示配置电路。通过 mode() 函数，可以禁用下拉电阻或者设置内部上拉电阻使能。具体如何使用，请查阅 mbed 手册。

表 3.3　mbed 逻辑输入 API 汇总（引自 www.mbed. org）

函　数	用　途
DigitalOut	创建一个 DigitalIn 对象，可与指定引脚相连
read	读取输入，用 0 或 1（int）表示
mode	设置输入引脚的模式
operator int()	Read() 的简写形示

3.4.3　用 if 语句响应开关输入

现在将一个开关作为数字输入连入电路，用这个开关来控制 LED 的显示状态。这样做是非常有意义的。我们第一次通过程序来读取外部变量，并确定开关的位置。而这正是嵌入式系统的本质。程序示例 3.3 可实现这一目的。通过 DIgitalIn 构造函数，引脚 7 被配置成输入状态。

C语言语法

为了识别开关状态，程序中使用了语句

```
if(switchinput==1).
```

这条语句中第一次使用了 C 中的相等运算符“==”。阅读 B.6.1 节，复习一下 if 和 else 关键字的用法。如果条件满足，本行代码后面的语句或代码段将执行。在本例中，条件是变量 switchinput 是否等于 1。如果条件不满足，那么 else 后面的代码会执行。分

析程序可以知道，如果开关输入值为 1，绿色 LED 关闭，红色 LED 闪烁。如果开关输入值为 0，else 代码块执行，红色 LED 关闭，绿色 LED 闪烁。此外，通过 while(1) 语句，整段代码实现无限循环，所以 LED 灯连续闪烁。

程序示例 3.3　用 if 和 else 处理开关量输入

```
/* 程序示例 3.3：通过开关状态，实现两个 LED 交替闪烁
*/
#include "mbed.h"
DigitalOut redled(p5);
DigitalOut greenled(p6);
DigitalIn switchinput(p7);
int main() {
 while(1) {
    if (switchinput==1) {    // 测试 switchinput 的值
    // 如果 switchinput 为 1，执行下面的语句
        greenled = 0;        // 绿灯灭
        redled = 1;          // 红灯亮
        wait(0.2);
        redled = 0;
        wait(0.2);
  }                          // if 结束
  else {                     // 如果 switchinput 为 0，跳到这里
   redled = 0;               // 红灯灭
   greenled = 1;             // 绿灯亮
   wait(0.2);
   greenled = 0;
   wait(0.2);
  }                          // else 结束
 }                           // while(1) 结束
}                            // 主程序结束
```

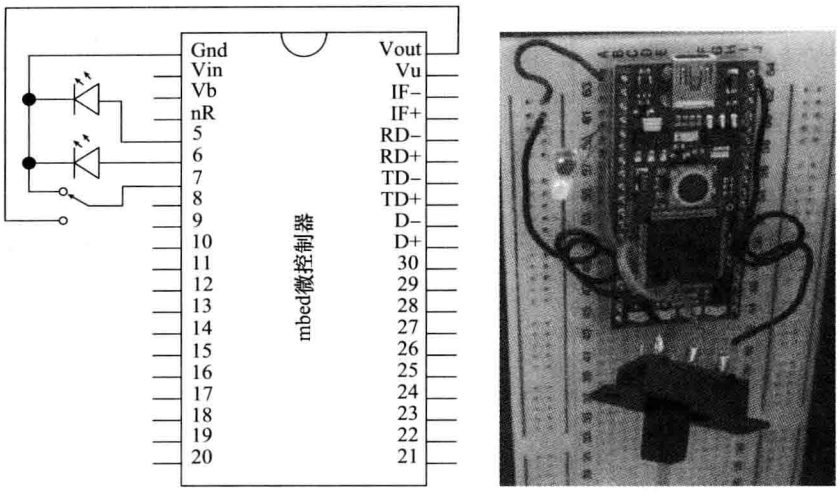
添加一个 SPDT 开关，将图 3.4 中的电路调整为图 3.6a 所示。在程序中将与开关相连的端口配置为数字输入。图 3.6b 为添加了开关的电路实物照片。在本例中，将导线焊接到开关上与面包板相连。当然，也有可能找到能直接插入面包板上的开关。创建一个新的程序，复制整个程序示例 3.3。编译、下载后，在 mbed 上运行程序。

练习 3.4

像练习 3.3 一样，编写程序实现一个方波输出。要求根据输入开关的位置，分别输出频率为 100Hz 和 200Hz 的方波。使用上例中的开关即可，然后在示波器上观察输出波形。

练习 3.5

图 3.6 中的电路使用一个 SPDT 开关，连接方式图 3.5a 所示。mbed 手册告诉我们 DigitalIn 实用程序库实际上已配置了一个片上下拉电阻。使用一个拨动开关或 SPST 按钮，按图 3.5c 重新修改电路。测试程序正确运行。



a）连接图　　b）面包板搭建电路

图 3.6　用开关控制 LED

3.5　简单的光电设备接口

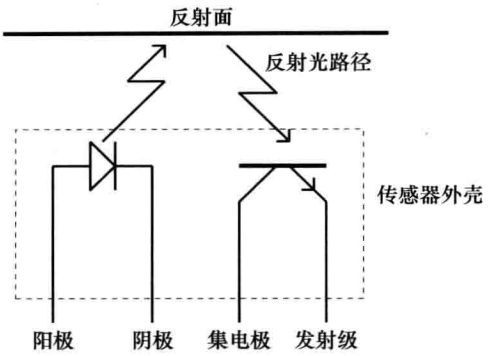
既然我们已经知道了如何在 mbed 上实现一位数据的输入和输出，这样会有越来越多的应用呈现在我们面前。许多简单的传感器可以直接与数字输入相连。还有一些传感器，有自己的内置接口，能产生数字输出。本节侧重于那些简单、传统的传感器和显示器，它们可以直接连接到 mbed 上。而后面的章节会进一步介绍如何与一些非常新且技术含量高的外设进行连接。

3.5.1　光敏反射和透射传感器

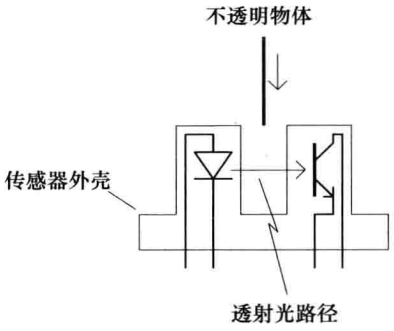
图 3.7a 和 b 中给出的光敏传感器，是较为简单的传感器，“几乎”全部为数字输出。当光照射在光电晶体管的基极上时，处于导通状态；相反，没有光照射时不会导通。在反射式传感器（见图 3.7a）中，红外 LED 同光敏晶体管封装在一起，并在其前面安装反射面。当光照射到反射面上时会反射回来，此时光敏晶体管导通。在透射式传感器（见图 3.7b）中，LED 装配在晶体管的对面。当传播路线上没有物体时，LED 的光会直接照射在光敏晶体管上，使该晶体管导通。若有物体存在，光线被遮挡，晶体管处于截止状态。这种传感器有时也称为槽型光敏传感器或光断续器。每个传感器可以用于检测特定类型的物体。

这两种传感器都可以按照图 3.7c 所示电路进行连接。其中，R_1 用来控制流过 LED 的电流，阻值大小根据传感器数据手册中的要求，经计算后得出。电阻 R_2 阻值的选取要保证能够输出适当的电压幅值，以满足图 3.1 对门限电压的要求。当光线照射到光电晶体管基极上时，电流流过晶体管。R_2 阻值的选取，要保证电流流动时，晶体管的集电极电压 V_C 能够降到几

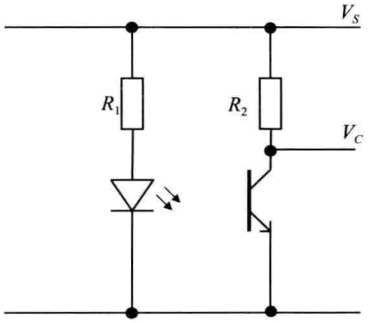
乎为 0V。如果没有电流流过，则 V_C 上升到 V_S。一般情况下，通过减小 R_1 或增加 R_2，可以提高传感器的灵敏度。



a）反射式传感器



b）透射式传感器



c） 用于光敏传感器的简单驱动电路

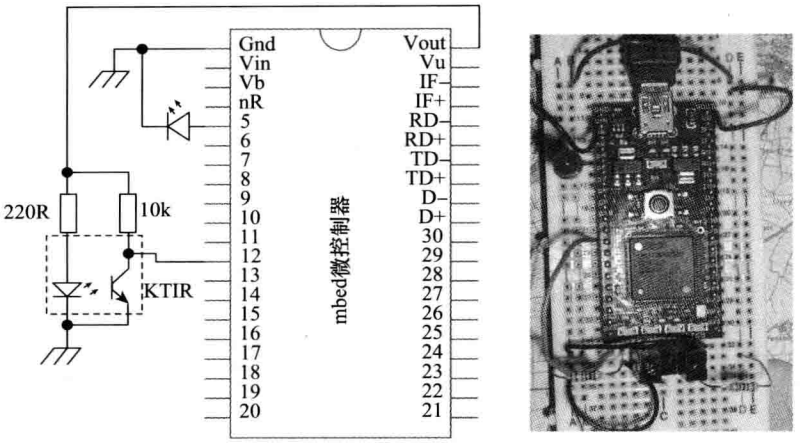
图 3.7 简单光敏传感器

3.5.2 光敏传感器与 mbed 开发板的连接

图 3.8 显示如何将透射式光敏传感器连接到 mbed 上。其中传感器采用的是 KTIR0621DS，由 Kingbright 公司生产，当然，也可以采用其他类似的器件。本例中的传感器的特点是，它的引脚可以直接插入面包板。连接这 4 个引脚时要小心，以免接错。在该传感器的外壳上标有连接图，供连接时查看；或者也可以查阅 Kingbright 公司提供的资料。这些资料可以在参考文献 3.2 给出的 Kingbright 网站上或从你所购买传感器的供应商那里获得。

程序示例 3.4 用来控制这个电路。传感器的输出端连接至引脚 12，因此在程序中，引脚 12 被设置为数字输入。当传感器中没有物体存在时，光线可照射到光敏晶体管上。晶体管导通，传感器输出逻辑 0。当有物体存在时，光束被遮挡，输出逻辑 1。光束被遮挡时，即检测到物体存在，程序点亮 LED。为了实现二选一功能，可以像前面的例子中那样，在程序中使用 if 和

else 关键字。由于本例中的每一分支只有一行代码，因此没有必要用大括号将代码括起来。



a）连接图　　b）面包板搭建电路

图 3.8　mbed 与透射光敏传感器的连接

程序示例 3.4　光断续器的应用

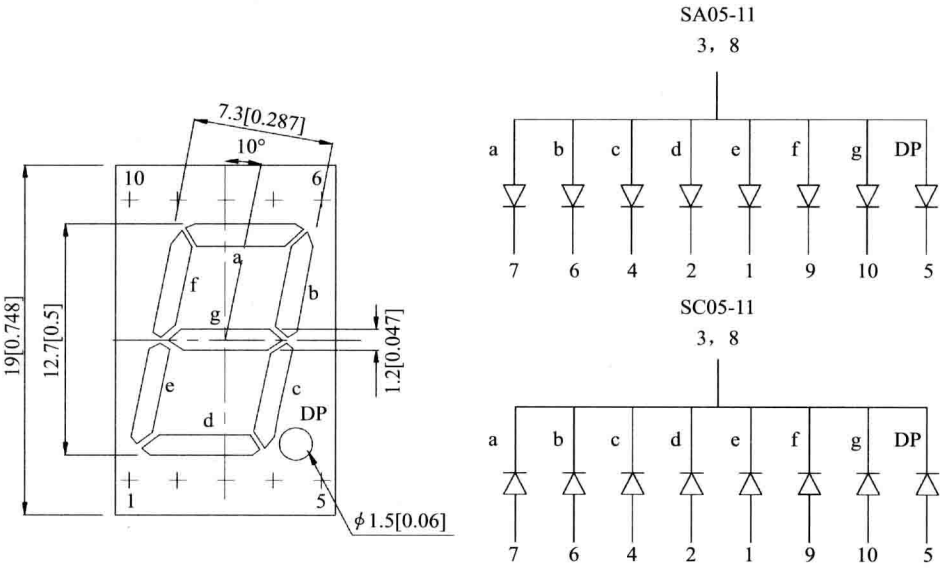
```
/* 程序示例 3.4：一个用来测试 KTIR 公司透射光敏传感器的简单程序。根据传感器状态控制 LED 的亮灭
*/

#include "mbed.h"
DigitalOut redled(p5);
DigitalIn opto_switch(p12);

int main() {
  while(1) {
     if (opto_switch==1)          // 如果光束中断输入为 1
        redled = 1;               // 如果光束中断 led 点亮
     else
        redled = 0;          // 如果光束未中断 led 熄灭
  }                               // while 循环结束
}
```

3.5.3　七段数码管显示

在前面的几个例子中，我们已经多次使用了单个 LED。除此之外，LED 通常还封装在一起，以图案、数字或其他类型的方式显示。许多显示方式已经成为标准配置，在日常生活中得到广泛的应用，这些方式包括条形图、七段数码管显示、点阵和“星爆”式闪灯等。

七段数码管是由 LED 组成的一个独特结构，该结构在显示字符上具有一定的通用性。图 3.9 为 Kingbright（参考文献 3.2）生产的数字型数码管。通过点亮数码管中的不同部分，可以显示所有的数字和绝大多数字符。如图 3.9 中所示，数码管上通常还包括一个小数点。这意味着数码管中有 8 个 LED，因此需要 16 个引脚与之连接。为了简化，可以像图 3.9 所

示的那样，将所有 LED 的阳极或阴极连接在一起。这两种连接模式称为共阳极或共阴极连接。简化连接后不再需要 16 个引脚，只需要 9 个就可以了。这 9 个引脚分别包括每个 LED 需要的一个连接引脚，共有 8 个，还有一个引脚用于公共端。对于本例中用到的数码管，其实际引脚被分成两行，位于待显示数字的顶部和底部。它提供的引脚共有 10 个，其中共阳极或共阴极占用了两个引脚。



a）七段数码管（Kingbright，12.7mm）　　b）电气连接（上：共阳极；下：共阴极）

图 3.9　七段数码管（图片经 Kingbright 电子有限公司许可转载）

图 3.9 中的七段数码管，可直接被微控制器驱动。本例中采用共阴极连接，阴极连至地，每个段连接到微控制器的端口引脚上。如果数码管中的段按以下序列连接，可以形成一个字节。

（最高有效位）DP g f e d c b a（最低有效位）

表 3.4 中列出的数值为数码管的控制字，供使用时查阅。例如，如果要显示 0，那么所有外部段，即 abcdef 必须被点亮，微控制器中与之相对应的位需设置为 1。如果要显示 1，则只有段 b 和段 c 需要被点亮。请注意，如果想让字符显示得更大，每个段需要串联几个 LED。在这种情况下，每个串联组合需要更高的电压来驱动，此时仅靠微控制器的电源电压，可能无法直接驱动该显示器。

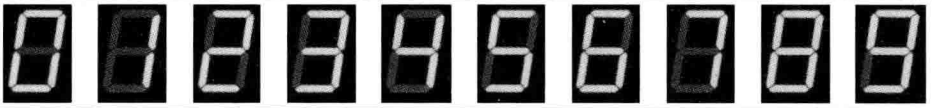
3.5.4　七段数码管与 mbed 开发板的连接

我们知道，mbed 的工作电压为 3.3V。本例中 7 段数码管的数据手册（取自参考文献 3.2）表明，点亮每个 LED 需要的电压为 1.8V 左右，因此 mbed 完全能够驱动它。如果在每个段上有两个 LED 串联，mbed 几乎不能够使这两个 LED 导通。但是我们能否将 mbed 的输出与数码管直接相连呢？或者需不需要像图 3.3b 中那样接一个限流电阻呢？查阅附录 C 中

LPC1768 的相关数据，我们看到当一个端口引脚流过 4mA 电流时，输出电压下降约 0.4V。这表明输出电阻为 100Ω。这里我们不需要去了解内部电路。只要知道该值是一个近似值，仅用在这个工作范围内就足够了。运用欧姆定律，LED 直接连接到引脚时的电流由式 3.1 算出

$$I_D \approx (3.3\text{-}1.8)/100=15\text{mA} \qquad (3.1)$$

表 3.4 七段数码管控制值示例

显示值	0	1	2	3	4	5	6	7	8	9
段驱动（B）										
（MSB）	0011	0000	0101	0100	0110	0110	0111	0000	0111	0110
（LSB）	1111	0110	1011	1111	0110	1101	1101	0111	1111	1111
B（十六进制）	0x3F	0x06	0x5B	0x4F	0x66	0x6D	0x7D	0x07	0x7F	0x6F
实际显示										

15mA 电流会使数码管上各段显示得非常亮，但功耗还是在可接受范围内。对于一些功率敏感的应用，为减小流过的电流，可在数码管各段上串联一个电阻。

按图 3.10 中电路所示，将七段数码管连接到 mbed 上。这是一个简单的数码管应用，采用的是共阴极方式直接连接到地，每段只需连接到 mbed 的一个输出引脚上。

该电路由程序示例 3.5 进行驱动。这里第一次使用了 mbed API 中的 BusOut 类。BusOut 允许将一组数字输出到一条总线上，所以可以直接向 BusOut 写一个字节长度的数字。尽管这里并未使用 BusIn，但读者需要知道，该类与输出相对应，在输入时使用。使用 BusOut 对象时，只需要指定一个变量名，如本例中的变量名为 display，随后小括号内列出的引脚将与总线一一对应。

C语言语法

在这个程序中，第一次使用了 for 循环。对于 while 循环而言，for 循环是另一种实现条件循环的方式。请在 B.7.2 节中查阅它的语法格式。本例中，变量 i 初始时为 0，并在每个循环迭代中加 1。每次循环将使用新的 i 值，直到 i 值为 4，循环结束。然而，由于 for 循环之外是一个由 while 构成的无限循环，且 for 循环是其中的唯一代码，因此 for 循环结束后会重新开始执行。

C语言语法

程序示例 3.5 中还使用了 switch、case 和 break 关键字。这些关键字结合在一起，构成了另一种条件语句，即允许从列表中选择一项来执行，具体语法格式在 B.6.2 节中有详细描述。本例中，变量 i 是递增的，根据 i 的当前值可选择送往数码管的控制字，从而显示出对应的数字。程序中采用的是从列表中选择一个值，这种方法可以用来实现查找表（look-up Table），这是一个很重要的编程技术。

在这个例子中出现了多个代码块的嵌套。main 代码块中嵌套了 while 代码块，while 代码块中嵌套了 for 代码块，for 代码块中又嵌套了 switch 代码块。在给出的程序清单中，可以看到在每一个右大括号处都加了注释。编程时一定要确保每个代码块结束时右大括号在正确

的地方出现，换句话说，要保证左右大括号能够匹配，这在编写更为复杂的 C 程序时会显得非常重要。

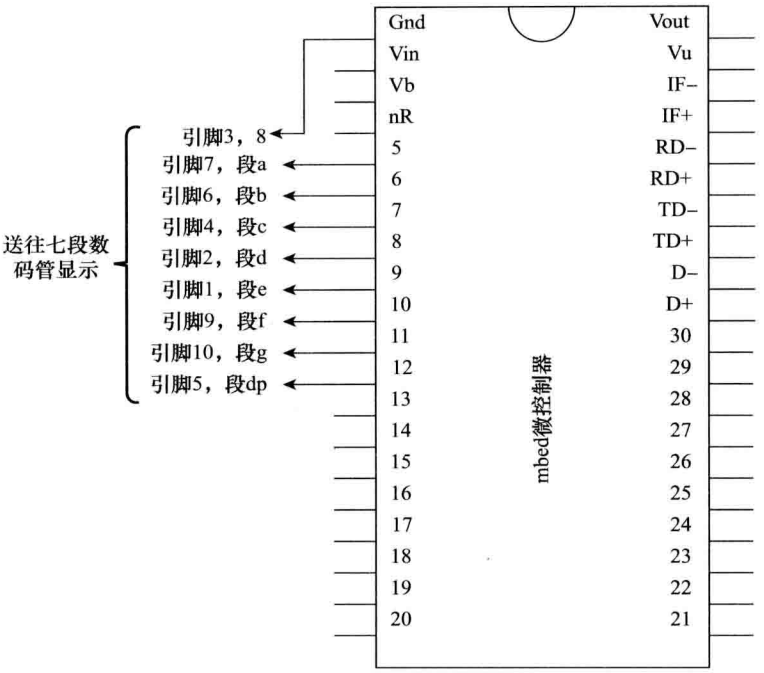


图 3.10 mbed 与共阴极七段数码管的连接

程序示例 3.5 用七段数码管显示一个序列

```
/* 程序示例 3.5：简单演示了 7 段数码管的显示。轮流显示数字 0，1，2，3
*/

#include "mbed.h"
BusOut display(p5,p6,p7,p8,p9,p10,p11,p12);  //a，b，c，d，e，f，g，dp 各段控制引脚

int main() {
  while(1) {
    for(int i=0; i<4; i++) {
      switch (i){
          case 0: display = 0x3F; break;     // 显示 0
          case 1: display = 0x06; break;     // 显示 1
          case 2: display = 0x5B; break;
          case 3: display = 0x4F; break;
       }                                          // switch 结束
    wait(0.2);
  }                                             //for 结束
 }                                              //while 结束
}                                               // 主程序结束
```

编译、下载，然后运行程序。数码管应该会像图 3.11 那样显示。请注意，检查引脚 8 的

连接后发现，这是一个共阳极连接。该电路连接没有完全照搬图 3.10，引脚 3 处于悬空状态。

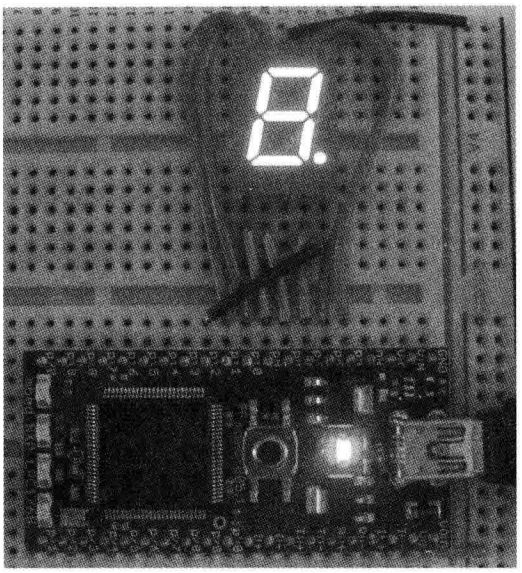


图 3.11 mbed 与共阳极七段数码管的连接

练习 3.6

利用图 3.10 中的电路，编写一段程序，在数码管上交替闪烁字符 H E L P。

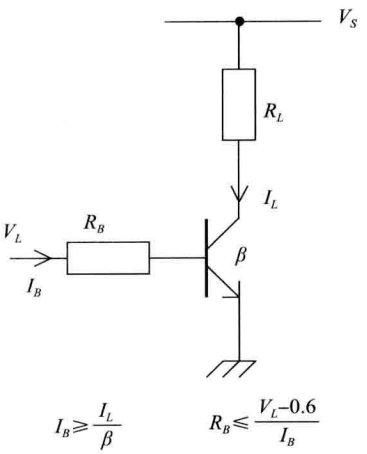
3.6 驱动大型直流负载

3.6.1 使用晶体管驱动

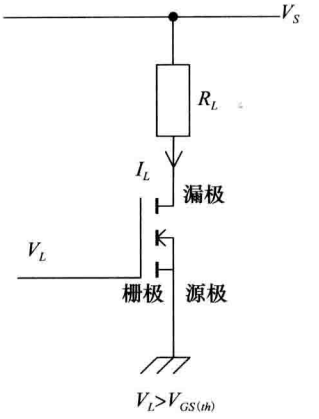
mbed 可以通过自己的数字 I/O 引脚驱动一些简单的直流负载，比如上文提到的发光二极管。mbed 的汇总数据（见附录 C）表明，一个端口引脚可以提供高达约 40mA 的电流。但是，该电流是短路电流，所以我们不可能将一个实际电力负载连接到端口上。

如果必须要驱动一个负载，如一个电机，该负载需要的电流比 mbed 端口引脚提供的电流要大很多，或该负载需要一个更高的电压，这时候就需要接口电路了。图 3.12 给出了能够驱动直流负载的三种情况。其中输入端逻辑电压由端口引脚提供，用 V_L 表示。前两个电路显示了如何驱动一个阻性负载，如电动机或加热器，其中，前一个使用了双极型晶体管，后一个使用了金属 - 氧化物 - 半导体场效应晶体管（MOSFET——尽管这个词读起来很拗口，但它在今天的电子工业中起着非常重要的作用）。采用双极性晶体管作为控制开关时，可用图 3.12 中给出的简单公式计算 R_B，只要知道负载电流和晶体管的增益（β）就可以求出 R_B。采用 MOSFET 时，有一个阈值电压，当栅源电压超过该阈值电压时晶体管导通。这种结构

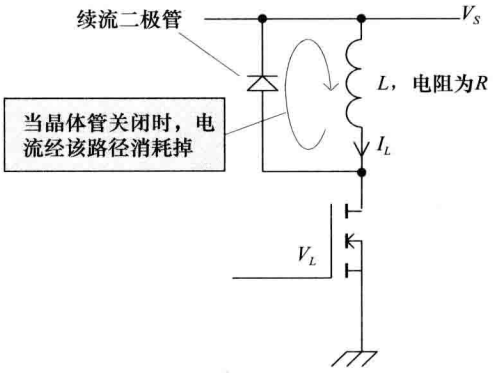
非常有用，因为 MOSFET 的栅极很容易被微控制器输出端口驱动。



a）电阻负载，npn型晶体管



b）电阻负载，n沟道MOSFET



c）感性负载，n沟道MOSFET

图 3.12 用晶体管做开关控制直流负载

图 3.12c 是对一个感性负载进行开关控制，如电磁阀或直流电动机。其中添加了一个重要的元件是续流二极管（freewheeling diode）。因为任何电感有电流流过时，其周围会产生磁场，能量以磁场的形式存储下来。当晶体管处于关闭状态时，电流中断，能量转化为电能返回电路。产生的电流通过续流二极管逐渐消耗掉。如果电路中不放置续流二极管，产生的瞬态高电压，可能会损坏场效应晶体管。

3.6.2 用 mbed 进行电机驱动控制

ZVN4206A 开关晶体管非常适合驱动小型直流负载，其主要的特性见表 3.5。其中一个重要值是 V_{GS} 最大阈值为 3V。这意味着，mbed 中逻辑 1 对应的输出电压 3.3V 正好能够满足

MOSFET 工作要求。

表 3.5　n 沟道 MOSFET ZVN4206A 的特性

特　性	ZVN4206A	特　性	ZVN4206A
最大漏源电压 V_{DS}	60V	最大连续漏电流 I_D	600mA
最大栅源阈值电压 $V_{GS(th)}$	3V	最大功耗	0.7W
导通时最大漏源电阻 $R_{DS(on)}$	1.5Ω	输入电容	100pF

练习 3.7

按图 3.13 电路所示连接一个小的直流电机。电机工作所需要的 6V 电压可以由一个外部电池组或工作台电源提供。当然，这个电压具体是多少取决于你所使用的电机。编写一个程序，使开关连续处于开启和关闭状态，如 1 秒开，1 秒关。然后增大开关频率，直到无法检测到开和关为止。与开关刚刚处于打开时相比，此时电机速度有何变化？

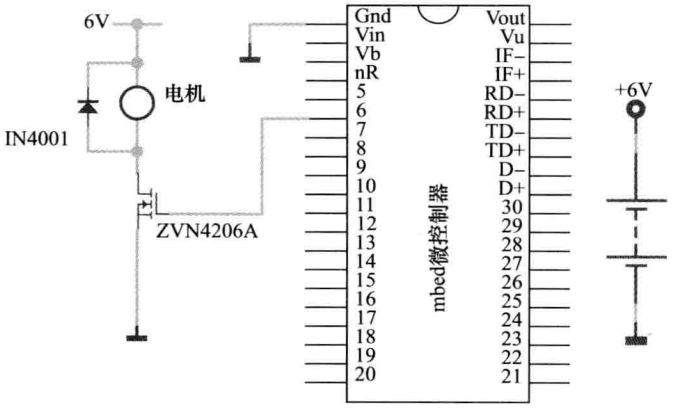
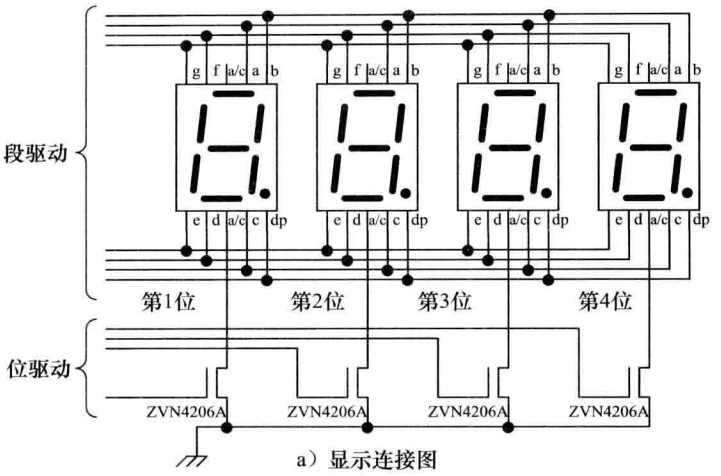


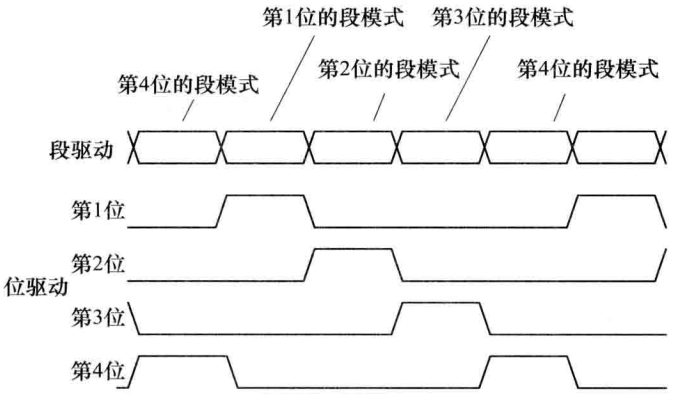
图 3.13　用开关控制一个直流电机

3.6.3　驱动多个七段数码管

在前面章节中，我们知道了如何将一个七段数码管连接到 mbed 上。每个数码管连接需要 8 个引脚，如果想连接多个数码管，那么 I/O 引脚将不够用。图 3.14 中的接法是解决引脚不够的一个有效方法。如图 3.14 所示，将每个数码管的公共端连接在一起，并把与微控制器连接的引脚配置为数字输出。图 3.14 中采用共阴极接法，每个数字的公共端与驱动自己的 MOSFET 相连，然后采用图 3.14 中的时序图进行控制。当数码管 1 的位选信号有效时，根据控制字该数码管的对应的驱动晶体管导通，点亮该数码管。随后，数码管 2 的位选信号有效，对应的驱动晶体管导通。数码管轮流被点亮，并周而复始地持续下去。如果这个时间足够快，那么人眼会感觉到所有数字好像是被同时点亮的。每个数字轮流点亮的时间约 5ms。



a）显示连接图



b）时序图

图 3.14 七段数码管的复用

3.7 小项目：字母计数器

用一个槽式光敏传感器、按钮开关和一个七段数码管设计一个字母计数器。当有字母通过时数码管上的数字加 1。当按钮开关按下时，清除显示。可以增加一个 LED 作为“半位”显示，这时计数范围为 0 ~ 19。

本章回顾

- 逻辑信号在数学中表示为 0 和 1，在数字电路中用电压表示。0 用一个电压范围表示，1 用另一个电压范围表示。
- mbed 有 26 个输入 / 输出引脚，可以配置成输入或输出。

- mbed 的数字输出可以直接驱动 LED。LED 在显示一位逻辑值时非常实用，而且有助于实现简单的人机接口。
- 机电开关可以连接到数字输入上，用来提供一个逻辑值。
- 一些简单的光电传感器几乎都有数字输出，可以直接连接到 mbed 的引脚上，但连接时须谨慎。
- 当 mbed 引脚不能提供足够的电力直接驱动负载时，必须使用接口电路。对于简单的开 / 关，一个晶体管往往是有必要的。

习题

1. 完成表 3.6，实现不同进制的转换。第一行为示例。

表 3.6 不同进制之间的转换

二进制	十六进制	十进制
0101 1110	5E	94
1101		
	77	
		129
	6F2	
1101 1100 1001		
		4096

2. 是否能够在图 3.9 所示的七段数码管上显示大写字母 A、B、C、D、E 和 F。对于能够有效显示的字母，请给出段的驱动值。将答案分别按二进制和十六进制格式给出。
3. 以下是 mbed 的一段循环程序，看上去较为凌乱。循环中以秒、毫秒和微秒表示的时间总共是多少？

```
    while (1)
{redled = 0;
    wait_ms(12);
greenled = 1;
    wait(0.002);
    greenled = 0;
    wait_us(24000);
}
```

4. 将 8 个开关全部按图 3.5b 中电路所示与 mbed 数字输入连接。上拉电阻器为 10kΩ，与 mbed 提供的 3.3V 相连。如果采用这种电路结构，当所有的开关同时闭合时，消耗的电流有多大？如果漏电流必须限制到 0.5mA，上拉电阻必须增大到多少？思考上拉电阻在低功耗电路中可能造成的影响。

5. 图 3.10 中，当显示数字为 3 时，与数码管连接的值是多少？
6. 如果图 3.10 中每个段需要大约 4mA 的电流，需要与多大阻值的电阻串联？
7. 一个学生基于 mbed 搭建了一个系统。其中一个端口连接了两个 LED，连接好后的电路如图 3.15 所示，此处 LED 与图 3.4 中用到的型号完全一致。但是 LED 并没有按意料中的那样被点亮。请解释出现这种情况的原因，并对电路给出修改建议，使 LED 能够点亮。
8. 另一学生打算利用 mbed 控制一个直流电动机，其设计的电路如图 3.15b 所示，其中 V_S 为与之对应的直流电源。为了能够更清楚地指示电机处于运行状态，该学生在端口上直接连接了图 3.3 中用到的一个标准 LED。但是，该学生发现这个电路可靠性差。请说明应该如何修改这个电路。

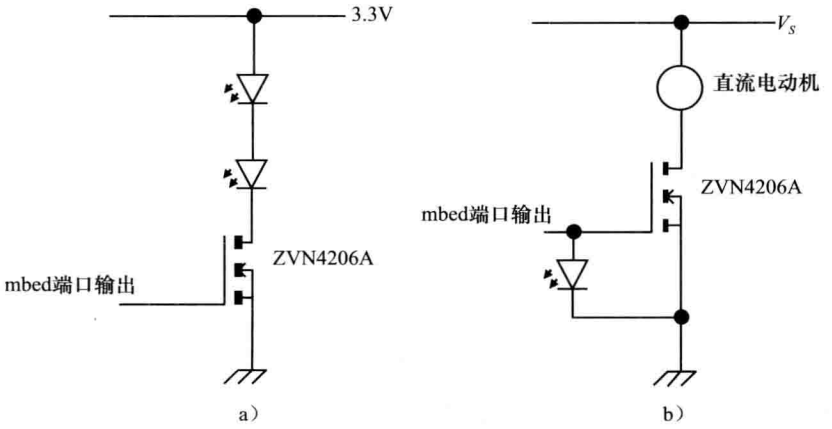


图 3.15 开关电路实验

9. 查阅参考文献 2.2 中的 mbed 电路图，找到板上的 4 个 LED。估算出 LED 点亮时流过的电流大小，假设其正向电压为 1.8V。

参考文献

3.1 Floyd, T. (2008). Digital Fundamentals. 10th edition. Pearson Education.
3.2 The Kingbright home site. http://www.kingbright.com/