- 兼顾基础，重在提高与技巧
- 解惑答疑，提升系统开发水平

嵌入式系统开发

陈　卓　王　田　梁新元　等编著

電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

代码的力量——

嵌入式系统开发

陈 卓 王 田 梁新元 等编著

電子工業出版社

Publishing House of Electronics Industry

北京·BEIJING

内 容 简 介

本书介绍了嵌入式开发多个方面的内容，涵盖面较为广泛。整本书分为 3 个部分:第 1 部分介绍嵌入式系统开发必须具备的软硬件基础。这一部分虽然是相对基础的内容，但在介绍重要内容的时候着重从应用的角度加以描述，在简单说明原理之后，回答了这些基础内容到底是怎么用的问题。由于涉及了不少实际系统，所以第 1 部分的内容是有一定深度的。第 2 部分讨论基于μC/OSII 的嵌入式系统开发，由于μC/OSII 系统的内核相对简单，所以这部分重点是说明μC/OSII 的程序设计框架及它的移植应用。在第 3 部分中，全面讨论了嵌入式 Linux 开发的各个重要方面，其中包括嵌入式 Linux 的驱动程序设计，这部分介绍了多种外设的驱动设计实例。另外，本书还讨论了 Qt 的程序设计和核心机制，以及 Linux 的网络实现的实例。最后还讨论了对一个嵌入式 Linux 系统移植所需要做的工作，分别介绍了 BootLoader、内核及驱动程序的移植方法。

本书的读者对象包括：刚接触嵌入式开发的人员，以及有一定嵌入式开发基础，希望参考各种开发项目的工程人员。本书在编写的时候尽量做到了按知识需求组织，以方便不同的读者按需阅读。

图书在版编目（CIP）数据

嵌入式系统开发 / 陈卓等编著.—北京：电子工业出版社，2009.4
(代码的力量)
ISBN 978-7-121-08576-5

I. 嵌… II.陈… III.微型计算机－系统开发 IV.TP360.21

中国版本图书馆 CIP 数据核字（2009）第 044869 号

策划编辑：祁玉芹
责任编辑：段春荣
印　　刷：北京市天竺颖华印刷厂
装　　订：三河市鑫金马印装有限公司
出版发行：电子工业出版社
　　北京市海淀区万寿路 173 信箱　邮编 100036
开　　本：787×1092　1/16　印张：25　字数：640 千字
印　　次：2009 年 4 月第 1 次印刷
定　　价：38.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

服务热线：（010）88258888。

前言

本书是讲解嵌入式系统开发的一本著作，完整地讨论了嵌入式开发必备的软、硬件基础知识，并且重点讲解了嵌入式开发的两个主要应用:驱动程序开发和嵌入式 GUI 程序开发。由于嵌入式系统开发有别于“一台 PC 就可以搞定”的纯软件开发，因此需要开发者有良好的软件开发技能，如：C/C++、汇编语言、操作系统等，同时还需要具备一定的硬件方面的知识，如：数字电路、计算机组成原理、接口电路等。而往往进入嵌入式开发领域的开发者要么以前是做软件开发的，对电路图、硬件驱动的原理等知识知之甚少，要么以前是一名硬件工程师，不太了解软件开发方面的基本要领，这些原因导致目前合格的嵌入式开发人员比较少。虽然已有不少关于嵌入式开发方面的优秀著作，但要么是针对水平较高的开发者编写，要么就是对嵌入式开发的某一个方面做深入的分析。这对刚涉及嵌入式开发的读者来说，会总觉得自己欠缺点什么，无法透彻理解其中的知识。

本书为了解决这个问题，特地提供给希望涉足嵌入式开发的朋友们以必要的、系统的知识，这也是本书的一个特色。在本书的第 1 部分中，比较详细地讨论了软件方面和硬件方面的必需了解和掌握的基本内容。第 1 章讨论嵌入式开发的最基本最重要的开发语言——C 语言的基本技能，特别是在嵌入式环境下的一些使用方法。第 2 章讨论嵌入式开发的硬件基础，包括对必要的数字电路方面知识的介绍及以 ARM 为核心的 S3C2410/S3C2440 处理器的介绍。阅读完本章后，读者首先应该能够分析一个电路的基本作用，并且能够对以 ARM 为核心的系列处理器有个比较深刻的认识。对于只有一定程序设计经历的读者，建议深入阅读整个第 1 部分；对硬件方面已经比较熟悉的读者则可以只阅读第 1 章。本书的第 2 部分讨论如何构建一个比较简单的嵌入式软件，并以实时嵌入式操作系统 μC/OSII 为背景，首先分析其基本原理，然后结合具体的实例讨论其移植和相关应用。第 3 部分讨论比较复杂的嵌入式 Linux 系统，第 7 章着重讨论做嵌入式 Linux 的基础知识，包括重要的命令、开发环境的建立、交叉编译工具的建立与使用等。第 8 章在具备第 1 部分的软硬件基础的情况下讲解 Linux 驱动编写的基本方法，并且以几个典型的示例从基本的实现分析到驱动编写完整的向读者展示嵌

入式 Linux 的驱动软件开发的基本流程及实现方法。第 9 章讨论采用 Qt 为嵌入式 GUI 的嵌入式软件开发，由于 Qt 的开发涉及的内容很广不可能全面去叙述，所以本书先讨论 Qt 程序设计的基本方法，然后重点讨论了 Qt 实现的一些核心机制，这些机制的学习对于深入理解 Qt 开发是很有必要的。由于通信和网络领域为嵌入式系统应用最广泛和成熟的领域之一，所以本书在第 8 章详细地分析了 Linux 平台下的部分重要的协议及网络控制机制，如 TCP、UDP、IP、QoS、Congestion Control 等，这也算本书的另一个特色。

总的说来，本书有几个特点：一是“有浅有深”，既有为进一步深入实践必须具备的基础知识，也有对一些相对比较深入话题的探讨。二是“新”，本书讨论的技术都是目前该领域最新的内容。三是“全”，本书在给出基础知识以后，就话题展开，在第 3 部分讨论了嵌入式开发的诸多方面，包括：驱动开发、嵌入式 GUI 程序设计、嵌入式系统的网络协议等，其目的是适应更多读者的需要，并且能给读者一些提示。

读者朋友如有任何疑问和建议，可以到如下网站的本书讨论区同笔者一起探讨：http://www.pubeta.com。

编著者

2009 年 3 月

目录

CONTENTS

第 1 部分　嵌入式开发必备基础

第1部分

嵌入式开发必备基础

嵌入式系统开发是一个“门槛”比较高的技术领域，因为该领域不但需要开发者能编写应用程序，还需具备对硬件的基本分析和调试的能力。单纯只了解软件的开发者由于对系统底层硬件很多原理缺乏基本认识，可能会发现自己始终浮于表面，深入不下去。因此，做嵌入式开发需要“软硬兼施”。

第1章 软件开发基础

要成为一名合格的嵌入式开发工程师，对C语言的熟练掌握是必须的，可以说C语言是嵌入式开发最重要和最通用的语言。目前很多成熟的嵌入式软件产品自身就是采用C语言为主开发的，如：μC/OSII、嵌入式Linux、Qt、MiniGUI等。所以，熟练掌握C语言不仅可以开发很多底层系统程序，并且能更加深刻的了解软件是如何操作硬件资源的。

1.1 嵌入式环境下的C语言使用技巧

本节主要结合嵌入式开发的特点，讲解C语言的应用要点。随着进一步的学习和实践，读者将会发现本章总结的几个基本技巧是很有用的。

1.1.1 重要的位（bit）操作

位（bit）是程序设计中可以操作的最小数据单位，理论上可以用“位运算”来完成所有的运算和操作。在PC程序设计中涉及到对位操作的机会比较少。但如上面提到的，在嵌入式程序开发中常常操作的是以位为单位的数据，因此首先讨论C语言位操作的两个重要作用。

位操作的作用之一是可以减少除法和取模的运算。灵活的位操作可以有效地提高程序运行的效率，这对于处理器能力相对较弱的嵌入式开发来讲是非常有必要的。举例如下：

```
/* 方法1 */
unsigned int i,j;
i = 156 / 16;
```

```
/* 方法2 */
unsigned int i,j;
i = 156 >> 4;
```

```
j = 562 % 32;                    j = 562 - (562 >> 5 << 5);
```

对于以 2 的指数次方为“*”、“/”或“%”因子的数学运算，右边的方法 2 采用移位运算“<<”及“>>”通常可以提高算法效率。这是因为乘除运算的处理器指令周期通常比移位运算大，在读一些嵌入式程序代码时经常会看到这样的移位运算，实际作用就是降低代码的运行时间，提高效率。

移位操作有两种，即左移位和右移位。左移位操作比较简单，左移几位就在右边空出的位置补上几个零，即对于一个数 NUM 的位表示[Xn-1,Xn-2,…,X0]，如果 NUM << k 过后，NUM 的位表示应该为[Xn-k-1,Xn-k-2,…,X0，0,…,0]，一共在右边填充了 k 个零。而相比之下右移位操作要复杂一些，有两种情况即：算数右移和逻辑右移，逻辑右移在左边补 k 个零，对于 NUM>>k 后得到的结果为[0,…,0,Xn-1,Xn-2,…,Xk]，而算数右移 NUM>>k 后得到的结果为[Xn-1,…,Xn-1,Xn-1,Xn-2,…,Xk]，结果是在左边补充 k 个最高位 Xn-1。一般而言，编译程序对于有符号数的右移都是算数右移，而对于无符号数的右移都是逻辑右移，在做嵌入式程序设计中常定义一个无符号数，如上面的例子，使用 unsigned int 类型，而不直接使用 int 类型，其目的就是为使用逻辑右移。

C 语言位运算除了可以提高运算效率外，在嵌入式程序设计中，它的第二个重要作用是实现位间的与（&）、或（|）、非（~）操作。这跟嵌入式系统的程序设计特点有很大关系。因为对底层硬件（处理器，内存，外围器件）的操作实际上都是通过读写相关寄存器进行，而对寄存器的读写，实际上就是按照硬件的器件手册（Data Sheet）的描述对该寄存器的某个或某些位进行与、或、非操作后置位。这一点，会在第 2 章中进一步解释。例如，通过将 AM186ER 型 80186 处理器的中断屏蔽控制寄存器的低 6 位设置为 0（开中断 2），最通用的做法是：

```
#define INT_I2_MASK 0x0040
wTemp = inword(INT_MASK);
outword(INT_MASK, wTemp & INT_I2_MASK);
```

而将该位设置为 1 的做法是：

```
#define INT_I2_MASK 0x0040
wTemp = inword(INT_MASK);
outword(INT_MASK, wTemp | ~INT_I2_MASK);
```

判断该位是否为 1 的做法是：

```
#define INT_I2_MASK 0x0040
wTemp = inword(INT_MASK);
if(wTemp & ~INT_I2_MASK)
{
… /* 该位为1 */
}
```

1.1.2 正确使用数据指针

在嵌入式系统的程序设计中，常常要求在特定的存储单元读写内容或者直接访问嵌入式处理器的某个经过地址编址的寄存器（Register），通常汇编有对应的 MOV 指令，而除 C/C++

以外的其他程序设计语言基本没有直接访问绝对地址的能力。在嵌入式系统的实际调试中，多借助 C 语言指针所具有的对绝对地址单元内容的读写能力。以指针直接操作内存多发生在如下几种情况：①某 I/O 芯片被定位在 CPU 的存储空间而非 I/O 空间，而且寄存器对应于某特定地址；②两个 CPU 之间以双埠 RAM 通信，CPU 需要在双埠口 RAM 的特定单元（称为 mail box）书写内容以在对方 CPU 产生中断；③在利用字符迭加做自字符及简单图形显示时，需要读取在 ROM 或 Flash 的特定单元所刻录的汉字和英文字母。例如：

```
unsigned char *p = (unsigned char *)0xF000FF00;
*p=11;
```

以上程序的意义为在绝对地址 0xF000+0xFF00（这里假设处理器为 16 位的）写入 11。在使用绝对地址指标时，要注意指标自增自减操作的结果取决于指针指向的数据类别。上例中 p++后的结果是 p=0xF000FF01，若 p 指向 int，即：int *p = （int *）0xF000FF00;p++（或++p）的结果等同于：p = p+sizeof（int），而 p-（或-p）的结果是 p = p-sizeof（int）。同理，若执行：long int *p = （long int *）0xF000FF00; 则 p++（或++p）的结果等同于：p = p+sizeof（long int），而 p-（或-p）的结果是 p = p-sizeof（long int）。

通过上面的解释，可以看到处理器是以字节为单位编址，而 C 语言指针以指向的数据类型长度做自增和自减。理解这一点对于以指标直接操作存储空间或硬件的地址空间是相当重要的。

1.1.3　函数等价于指令的集合

首先要理解以下 3 个问题：

① 语言中函数名直接对应于函数生成的指令代码在内存中的地址，因此函数名可以直接赋给指向函数的指针。

② 调用函数实际上等同于“调转指令＋参数传递处理＋回归位置入栈”，本质上最核心的操作是将函数生成的目标代码的首地址赋给 CPU 的 PC 寄存器。

③ 因为函数调用的本质是跳转到某一个地址单元的 code 去执行，所以可以“调用”一个根本就不存在的函数实体。大学里的《计算机组成原理》课程中曾经讲到，186 CPU 启动后跳转至绝对地址 0xFFFF0（对应 C 语言指针是 0xF000FFF0，0xF000 为段地址，0xFFF0 为段内偏移）执行，请看下面的代码：

```
typedef void (*lpFunction) ( ); /* 定义一个无参数、无返回类型的 */
/* 函数指针类型 */
lpFunction lpReset = (lpFunction)0xF000FFF0; /* 定义一个函数指针，指向*/
                                              /*CPU启动后所执行第一条指令的位置 */
lpReset(); /* 调用函数 */
```

在以上的程序中，根本没有看到任何一个函数实体，但是却执行了这样的函数调用 lpReset()，它实际上起到了“软重启”的作用，跳转到 CPU 启动后第一条要执行的指令的位置。所以应该理解函数的本质，实际上函数就是一个指令集合；你可以调用一个没有函数体的函数，本质上只是换一个地址开始执行指令。

1.1.4 操作有限的存储空间

在嵌入式系统中易失存储器申请比一般系统程序设计时有更严格的要求，这是因为嵌入式系统的存储空间往往十分有限，不经意的存储空间泄露可能会很快导致整个系统崩溃。所以如果要采用动态分配策略操作存储空间，一定要小心以下几种情况。

① 一定要保证 malloc 和 free 语句成对出现，谁申请的空间就要由谁来释放。如有下面这样的一段程序：

```
char * function(void)
{
  char *p;
  p = (char *)malloc(…);
  if(p==NULL)
  …;
   …                              /* 一系列针对p的操作 */
  return p;
}
```

在某处调用 function()，用完 function 中动态申请的内存后将其 free，如下：

```
char *q = function();
…
free(q);
```

上述代码实际上存在很大的隐患，不小心很容易导致存储空间泄露。存储空间由 function 来申请，却由另外一个函数来释放。如果在外函数中没有使用 free（q），那么就会导致每使用一次 function 函数就会损失一块存储空间，不满足 malloc 和 free 成对出现的原则。另外，这样使用还会导致代码的耦合度增大，因为用户在调用 function 函数时需要知道其内部细节。

正确的操作方法是在调用子函数时申请存储空间，并把指针传入 function 函数，如下：

```
char *p=malloc(…);
if(p==NULL)
…;
function(p);
…
free(p);
p=NULL;                          /*避免野指针*/

void function(char *p)    /*函数接收指针p*/
{
…                                /* 一系列针对p的操作 */
}
```

② 不要用指针动态申请存储空间。先看如下一段代码，这种错误的使用方法比较常见。

```
void GetMem(char *p, int num)
{
     p = (char *)malloc(sizeof(char) * num);
}
```