# AVR单片机应用专题精讲

邵子扬 编著

北京航空航天大学出版社
BEIHANG UNIVERSITY PRESS

CD-ROM INCLUDED

# AVR 单片机应用专题精讲

邵子扬　编著

北京航空航天大学出版社

## 内 容 简 介

本书介绍了 AVR 单片机实际应用方面的常用技巧，包括 5 个专题，分别是宏的使用技巧、编程技巧、通信接口的使用技巧、AVRUSB 的使用技巧以及 Bootloader。每个专题都在实践的基础上深入讲解，并且提供了完整而详细的参考程序和 proteus 仿真例程（参见配套光盘），方便读者快速练习，或者在此基础上进行修改或移植。

虽然本书是以 AVR 单片机为基础进行介绍的，但是很多方法和内容同样适用于其他系列微控制器，如 ARM Cortex 系列，详细请参考相关章节。

本书适合有一定基础的单片机工程师和爱好者阅读参考。

**图书在版编目(CIP)数据**

AVR 单片机应用专题精讲 / 邵子扬编著. --北京 :
北京航空航天大学出版社，2013.3

ISBN 978-7-5124-1070-1

Ⅰ. ①A… Ⅱ. ①邵… Ⅲ. ①单片微型计算机 Ⅳ.
①TP368.1

中国版本图书馆 CIP 数据核字(2013)第 035474 号



**AVR 单片机应用专题精讲**

邵子扬 编著

责任编辑 董立娟

*

北京航空航天大学出版社出版发行

北京市海淀区学院路 37 号(邮编 100191) http://www.buaapress.com.cn

发行部电话：(010)82317024 传真：(010)82328026

读者信箱：emsbook@gmail.com 邮购电话：(010)82316936

涿州市新华印刷有限公司印装 各地书店经销

*

开本：710×1 000 1/16 印张：13.75 字数：293 千字

2013 年 3 月第 1 版 2013 年 3 月第 1 次印刷 印数：3 000 册

ISBN 978-7-5124-1070-1 定价：36.00 元(含光盘 1 张)

# 前言

## 本书的由来

作者是一名电子工程师和单片机爱好者，使用 AVR 单片机有较长的时间了。在项目开发过程中遇到过很多问题，其中很多问题都是书中和数据手册中没有提到的，或者是解答比较简略而不太容易解决的。因此，作者在长期解决问题的过程中，逐渐积累了一些经验和技巧，能够对 AVR 单片机的开发起到一些作用。作者在网络论坛、技术交流活动和研讨会上，也经常看到很多工程师提出一些作者以前碰到过的类似问题，却不知道怎样去解决；或者解决问题中使用的方法不太适当，造成开发过程中走了弯路；有时还会因为某个具体的应用缺少相关的资料和说明文档，结果在使用过程中出现一些困难。因此，想把自己在 AVR 单片机开发过程中积累的一些经验和技巧拿出来和大家分享、交流和探讨，希望本书介绍的内容能够对大家有所启发，对实际工作带来促进作用，少走一些弯路。同时，也想把这本书作为这些年开发工作的一个小结。

## 本书特点

本书深入介绍了一些有实用价值的 AVR 单片机使用技巧。和其他书不同，这里不是简单地介绍一下单片机的原理，然后给出一些原理图和参考代码就完了，而是有针对性地通过 5 个专题（宏的使用、编程、通信接口、AVRUSB、Bootloader）详细介绍一些应用的具体使用方法，讲解这种方法的工作原理，分析使用中的常见问题和注意事项，再给出解决方法或者改进方法，以及关键部分的参考代码。“授人以鱼不如授人以渔”，掌握方法非常重要，这样才能举一反三。所以本书的重点不在于程序代码和设计图纸等，而在于问题的分析、思路和解决方法。

此外，作者不是想写一本大而全的 AVR 单片机的参考书，也不是数据手册的翻译或者是网络论坛中各种应用文摘的收集。这本书讲述了一些 AVR 单片机实际应用方面的技巧，均来自于作者实际工作中解决过的问题，或是从实用角度出发，加上了作者的理解和对问题的分析，使读者更容易接受和理解，从而更快掌握解决问题的方法，并能应用到实际项目中。

本书也不是一本专门搜集各种技巧、使用大全、使用指南方面的书，而只介绍了作者认为比较重要、能够在实际项目中起到一定作用的方面。书中的内容是非常有针对性的，如 AVR 单片机中非常重要应用 Bootloader 和 AVRUSB。

## 本书读者对象

本书不适合于AVR单片机的初学者，而是针对有一定基础的单片机爱好者和工程师写的，是一本对单片机实际应用进行深入探讨的书。

本书是基于AVR单片机介绍的，但是其中的方法并不局限于AVR单片机，在很多其他的单片机或微控制器上（如ARM Cortex系列）也可以使用。因为现在的单片机，虽然型号、制造厂家、内核的架构不同，内部的指令和编译器也不同，但是它们的很多功能和特性是相似的，特别是基本外设功能模块是类似的，这就给用户系统的移植带来了可能和方便，所以也可以将很多方法应用到其他不同的单片机中。

本书所有的例子都使用AVRGCC编译器进行开发，在Windows下使用AVRStudio 4.18集成开发环境和WinAVR20100110编译器（或者使用AVR Tool-Chain作为编译器，因为目前WinAVR已经停止更新）。如果使用不同版本的AVRGCC编译器和其他IDE软件，个别地方会有一点差异，但是对整体应用没有影响。对于其他的C编译器（如ICC、IAR、CodeVisionAVR等），程序不能直接编译，需要适当修改和移植。同时大部分例子还提供了Proteus软件仿真的例程，方便读者在没有目标板的情况下直接验证设计、仿真运行、查看结果。

## 配套资源

本书不会在正文中列出过多完整的例子程序，因为往往程序中真正关键的部分就是很少的几个函数或者几行代码，完整而复杂的程序并不能帮助读者了解作者的思路和掌握使用方法，反而容易将思路误导到其他地方，增加了理解的难度。所以本书先讲解问题和思路，再列举出相关程序的关键函数和代码加以分析。有些地方甚至会使用伪代码来表示，这样更清晰。完整的程序和代码可以在本书配套光盘、作者的网站和博客中找到。

作者的联系方式：Email：shaoziyang@126.com。

博客：http://bbs.ednchina.com/BLOG_shaoziyang_8293.HTM。

## 致 谢

本书最初计划是由作者、青岛网友郭建和杭州网友马可一起编写，各自完成一部分内容。因为工作繁忙和其他一些原因，很遗憾郭建和马可没有继续下去，但是这里还是要对他们曾经的帮助和努力表示感谢。最后还要对北京航空航天大学出版社的支持表示感谢，使作者有机会能够和大家在AVR单片机上进行交流和探讨。

限于作者的理解和水平，书中给出的方法不一定是最好的，程序代码也不一定是最优的，有些地方甚至可能还会存在着错误。如果读者发现了书中的错误，或者有更好的解决方法，欢迎大家指出，或者一起进行深入探讨。

邵子扬

2012年10月

# 目 录


**专题一 宏的使用技巧** …… 1

1.1 常用的宏 …… 2

1.2 几个宏的特殊用法 …… 5

1.2.1 井号 # …… 5

1.2.2 双井号 # # …… 6

1.2.3 取特定参数 …… 7

1.2.4 将编译时间保存到目标代码中 …… 7

1.2.5 编译版本号的问题 …… 9

1.3 宏在 AVR 单片机中的应用 …… 11

1.3.1 使用宏简化程序的移植 …… 11

1.3.2 关于波特率计算时的四舍五入 …… 12

1.3.3 使用宏检查串口波特率误差的方法 …… 14

1.3.4 AVR 单片机中定义的常数 …… 14

1.4 使用宏管理 IO …… 15

1.4.1 基本方法 …… 15

1.4.2 改进的方法 …… 17

1.4.3 跨平台的 IO 管理 …… 20

1.5 使用宏时需要注意的问题 …… 27

1.5.1 宏定义中的表达式 …… 28

1.5.2 宏定义参数时需要注意的问题 …… 28

**专题二 编程技巧** …… 30

2.1 函数和变量在 Flash 中的定位 …… 30

2.2 软件定时器的使用 …… 32

2.2.1 使用方法 …… 33

2.2.2 简单示例 …… 34

2.2.3 使用软件定时器的优缺点 …… 35

2.3 多个中断共用一个中断服务程序 …… 36

2.4 超长低功耗延时 …… 36

2.5 CRC 校验计算方法的比较 …… 39

2.6 变量不自动初始化 …… 44
2.7 不使用中断向量表 …… 47
2.8 使用比较器做低成本高精度的 ADC …… 49
2.8.1 原 理 …… 49
2.8.2 优缺点 …… 51
2.8.3 参考例程 1 …… 51
2.8.4 参考例程 2 …… 56
2.9 使用查表法计算 NTC 热敏电阻的温度 …… 57
2.9.1 原 理 …… 57
2.9.2 参考例程 …… 59
2.10 使用内部基准计算电池电压 …… 62
2.11 FreeRTOS …… 65
2.11.1 为什么使用 FreeRTOS …… 65
2.11.2 FreeRTOS 的 3 种版本 …… 66
2.11.3 FreeRTOS 的使用方法 …… 67
2.11.4 参考例程 …… 74
**专题三 通信接口的使用技巧** …… 84
3.1 USI 接口的使用 …… 84
3.1.1 USI 的硬件结构 …… 84
3.1.2 USI 的控制寄存器 …… 86
3.1.3 USI 的中断 …… 86
3.1.4 使用 USI 作为主 $I^2C$ 接口 …… 87
3.1.5 使用 USI 作为主 SPI 接口 …… 93
3.2 使用 SPI 驱动数码管 …… 97
3.2.1 原 理 …… 97
3.2.2 参考例程 …… 101
3.3 1-Wire 的使用 …… 103
3.3.1 基本总线信号 …… 104
3.3.2 基本函数 …… 106
3.3.3 参考例程 …… 110
3.4 软件串口的使用 …… 114
3.4.1 串口的时序 …… 114
3.4.2 延时函数法 …… 115
3.4.3 使用普通定时器产生半双工软件串口 …… 120
3.4.4 利用定时器 1 产生全双工软件串口 …… 127

3.4.5 软件串口小结 …… 131
**专题四 AVRUSB 的使用技巧** …… 132
4.1 AVRUSB 简介 …… 132
4.2 AVRUSB 的发展 …… 133
4.3 硬件结构 …… 134
4.3.1 使用稳压二极管的连接方法 …… 134
4.3.2 使用二极管串联降压的方式 …… 135
4.3.3 使用 3.3 V LDO 供电 …… 135
4.3.4 使用单片机内部带 PLL 的 RC 振荡器 …… 136
4.3.5 使用外部电源的连接方法 …… 136
4.3.6 使用 3 个 IO 时的连接方法 …… 137
4.3.7 硬件结构分析 …… 138
4.4 软件架构 …… 139
4.4.1 基本说明 …… 139
4.4.2 AVRUSB 的程序文件结构 …… 140
4.4.3 参数配置 …… 141
4.4.4 使用 AVR Studio 创建 AVRUSB 项目 …… 142
4.5 主要的 AVRUSB 开源参考项目 …… 144
4.5.1 PowerSwitch …… 144
4.5.2 RemoteSensor …… 144
4.5.3 HIDKeys …… 145
4.5.4 BootloadHID …… 145
4.5.5 EasyLogger …… 145
4.5.6 AVR - CDC …… 145
4.5.7 AVR - Doper …… 146
4.6 AVRUSB 的应用实例 …… 146
4.6.1 使用 HID 方式显示数据 …… 147
4.6.2 使用 CDC 方式通信 …… 156
4.6.3 基于 AVRUSB 的 STK502 编程器 …… 160
4.7 AVRUSB 的优点 …… 163
4.8 AVRUSB 的使用限制 …… 163
4.9 AVRUSB 使用中的常见问题 …… 163
4.9.1 安装 CDC 驱动失败的问题 …… 164
4.9.2 计算机无法识别 USB 设备的问题 …… 164
4.9.3 设备可以识别但是运行不正常的问题 …… 164

4.10 AVRUSB 的授权方式 …… 165
4.11 AVRUSB 的相关资源 …… 165
**专题五 Bootloader** …… 166
5.1 概述 …… 166
5.2 Bootloader 的原理 …… 167
5.3 AVR 单片机 Bootloader …… 169
5.3.1 AVR 单片机的 Flash 结构 …… 169
5.3.2 与 Bootloader 相关的熔丝位和加密位 …… 172
5.4 使用 Bootloader …… 174
5.5 AVR 通用 Bootloader …… 174
5.5.1 简介 …… 174
5.5.2 AVR 通用 Bootloader 的主要特点 …… 176
5.5.3 软件流程 …… 176
5.5.4 单片机部分 …… 177
5.5.5 上位机软件使用说明 …… 187
5.5.6 加 密 …… 196
5.5.7 V4.5 版的错误修正 …… 203
5.5.8 Bootloader 使用中的常见问题 …… 205
5.5.9 改进 AVR 通用 Bootloader …… 207
**参考文献** …… 209

# 专题一

# 宏的使用技巧

本章将重点介绍宏指令的使用技巧。这一章是本书重要的环节，介绍了在 AVR 单片机中使用宏的一些方法和技巧，使编写代码变得有趣而简单，同时它也是一种非常有效和灵活的代码移植的方法。

宏指令（简称宏）是编译指令的一部分，是为了辅助编程而设计的。宏本身并不是程序运行流程的一部分，在用户程序运行时是没有任何影响的。宏指令只在编译过程时起作用，在编译过程中编译器根据宏指令的内容执行相应的动作，而在编译完成后生成的二进制代码中是没有宏的。宏不直接控制用户程序，而是间接影响用户程序。从这个角度来说，宏属于编译器那一部分，而并不是属于用户程序这一部分的。随着 C 语言的发展，宏的功能也越来越强，可以实现很多功能，对编程的帮助也越来越大，已经成为编程中不可缺少的部分。宏的很多指令和 C 语言是类似的，语法也是类似的。

正是因为宏指令可以在编译时影响到编译的流程，所以如果我们能够正确使用宏指令，在某些时候就可以起到简化编程、方便调试的效果。特别是现在大部分高级语言编译器的宏指令本身功能已经非常强大，只要有针对性地使用宏，再加上灵活使用一些小技巧、小方法，就可以实现简化代码编写、代码维护、方便程序移植等目的，甚至还可以实现一些使用正常代码难以实现的功能。

根据功能的不同，宏可以划分为以下几类：

- 宏定义；
- 特殊用法；
- 编译控制；
- 编译器内部预定义的宏；
- 编译器扩充的宏。

这几种宏的用法中，编译器扩充的宏不具有普遍性，因此它不在本文的讨论范围之内。

宏指令是与编译器相关的，所以它的功能和实现是依赖于编译器的。同样的宏

在不同的编译器中就可能存在一定的差异，编译后的结果也不尽相同。幸运的是，大部分 C 编译器（包括 Windows、Linux 和单片机的 C 编译器）对宏指令的解析是类似的，所以宏使用的基本方法也是通用的，这样我们运用宏指令时也就具有了一定的通用性，特别是在不同单片机或嵌入式系统的 C 编译器中就具有了通用性。这也给我们利用宏实现一些特殊功能带来了可能，如单片机 I/O 的使用、外设的控制等，下面将详细介绍这些方法。

## 1.1 常用的宏

在介绍一些相对复杂的使用技巧前，让我们先回顾一下常用的一些宏指令。这些应该是宏最基本的使用方法，大家可能早就都知道了。不过这里还是简单归纳一下，也是为后面的内容做一点铺垫，从而更加容易理解。

### 1. #define

#define 可以说是使用频率最高的宏指令，几乎在任何程序中，特别是头文件中都可以看到它。一般情况下使用 #define 可以将使用频率较高而又不需要在程序运行中变化的常数或参数使用一个比较容易记忆或理解的符号来代替，方便编写代码。在程序编译时，编译器会自动用 #define 定义的内容替换。如果以后需要改变这个常数或参数，只需要修改 #define 后对应的内容即可，无须修改程序中的代码，从而简化编程，提高代码的使用效率。也可以使用 #define 定义一个表达式，或者定义成任意内容。

从某种意义上说，#define 和 const 关键字是有些类似的。它们都可以定义一个常量，方便在后面的代码中使用。不过 #define 的功能更多一些，它可以用任何一个符号或字符串代替另外的内容；而 const 只能作为定义变量时的修饰符。#define 是使用频率最高的宏之一，也是后面介绍的宏使用技巧的基础。

### 2. #include

使用 #include 可以在一个程序中引用另外一个文件的内容，如：

```
#include <avr/io.h>
#include <stdio.h>
#include "user.h"
```

使用 include 可以将程序中常用或者共用部分的内容分离出来，保存到一个文件中，供其他文件引用。它可以极大地提高代码复用率，是一种非常有效的编程技巧。#include 也是最常用的宏之一。

在使用 #include 时，后面跟随的是头文件的文件名。如果文件名是使用尖括号 <> 括起来的，那么说明这个文件通常是编译器自带的系统头文件，首先在编译器的

include 文件夹里查找这个文件；如果是使用双引号“”括起来的，通常是用户自定义头文件，优先在文件当前目录下查找这个文件。

### 3. #if

C 语言中，if 是条件判断指令，是常用的功能之一。而在宏指令里面，同样也有条件判断指令：#if。它和 C 语言中 if 关键字的功能很类似：可以在 #if 中使用表达式，根据表达式的真假（和 C 语言中一样，0 代表假，非 0 代表真）决定是否编译 #if 中包含的代码，实现更为复杂的功能。宏里面的表达式和 C 语言的表达式非常相似，不过又有它的特点，这将在后面逐步说明。

#if 的基本用法是：

```
#if XXX
//用户代码
#endif
```

C 语言的 if 中允许多个条件进行组合逻辑判断，在宏定义中也是允许这样的，用法类似，如：

```
#if (A > 1) || (B < 2)
#if (A > 1) && (B < 2)
#if ! (F_CPU < 1000000)
```

C 语言中除了 if 外，还有关键字 else，用来改变程序流程。宏里面同样也有 #else，功能和用法也是类似的。和 C 语言中的 else 需要与 if 配对使用一样，#else 也必须配合 #if 使用，不能单独使用。同样，#if 是可以嵌套使用的。

此外，#if 必须和 #endif 成对使用，代表 #if 的定义范围。在 C 语言中可以用大括号对{}来设定 if 的定义范围，而宏里面没有这样的用法，所以需要使用 #endif 来设定 #if 的定义范围。

虽然 #if 和 C 语言的 if 很类似，但是它们还是有很大不同的。在 C 语言中，无论 if 中的代码是怎么样的，if 包含部分的所有代码都会生成有效二进制代码，会占用程序空间；而在宏里面，只有满足条件的那部分才会加入到编译过程中，允许编译器产生代码，否则这部分代码是不会进行编译的。所以在程序调试时，往往会使用 #define 预先定义一些宏，然后用 #if 包含一些特定的调试语句，方便调试；调试完成后只要去掉预定义的宏就不会将调试代码包含到最终的目标文件中。而在程序移植时，经常使用 #if 来使程序适应不同型号的单片机，提高代码的通用性。

### 4. #ifdef / #ifndef

#ifdef 和 #ifndef 也属于条件分支类的宏，用来判断一个宏是否定义。它的用法有些类似于 #if，但是又有一些不同。

#ifndef 的基本用法是：

```
#ifndef XXXX
//其他宏定义或申明
#endif
```

XXXX 没有定义过时，就会将包含的代码进行编译。使用#include 包含一个头文件时，通常就会使用到#ifndef，可以防止在多个文件引用相同的头文件时造成文件的内容被重复引用或者递归引用。所以几乎在所有头文件的一开始，都会看到#ifndef 这个宏，如系统头文件"stdlib.h"中是这样定义的：

```
#ifndef _STDLIB_H_
#define     _STDLIB_H_ 1
...
...
#endif /* _STDLIB_H_ */
```

#ifdef 的用法和#ifndef 类似，只是一个判断宏是否被定义，一个判断宏是否没有定义。就像是特殊的条件语句，一个是判断真，一个是判断假。和#if 类似，#ifdef/#ifndef 必须与#endif 成对使用，就像在程序中大括号{}总是成对使用一样。

还有一个与#ifdef 类似的宏指令是 defined，也可以用来判断是否定义了某个宏变量。defined 的用法是：

```
#if defined(XXX)
//其他宏定义或申明
#endif
```

如在头文件<avr/io.h>中，我们可以看到这样的定义：

```
#if defined (__AVR_AT94K__)
#   include <avr/ioat94k.h>
#elif defined (__AVR_AT43USB320__)
#   include <avr/io43u32x.h>
#elif defined (__AVR_AT43USB355__)
#   include <avr/io43u35x.h>
#elif defined (__AVR_AT76C711__)
#   include <avr/io76c711.h>
...
...
#else
#   if!defined(__COMPILING_AVR_LIBC__)
#     warning "device type not defined"
#   endif
#endif
```

### 5. #error 和 #warning

这是两个不算太常用的宏，可能很多人都不太熟悉。这两个宏是用来在编译时输出特定消息的，从某种意义上看，有点像 C 语言中的 printf 函数，只不过它们是在编译过程中输出消息，输出的内容显示在编译提示信息中；而 printf 是在程序运行时打印消息，输出对象是控制台。#error 用于输出错误提示，在编译时强制产生编译错误，并中止编译进程；而 #warning 则是产生一个告警消息，但不会中止编译。它们往往是和前面介绍的 #if、#else 等宏配合使用的，在特定条件下输出告警消息或者中止编译，如：

```
#if XXXX
#warning "some warning message"
#endif
```

或

```
#if XXXX
#error "any error message"
#endif
```

灵活使用 #error 和 #warning 可以在编译程序时实现类似 printf 那样的效果，方便程序调试和检查错误。

#error 和 #warning 后的内容可以不用引号“”括起来，它会将后面的内容自动作为字符串；如果使用了引号字符“”，它也会作为字符串的一部分。此外，#error 和 #warning 只能输出标准的字符串，而不支持表达式，也就是说，它会将后面的内容原封不动地全部显示出来。需要注意的是，目前在 GCC 中，#error 和 #warning 输出的内容不支持中文或特殊字符，这些特殊字符会显示成乱码，就是说在输出的消息中只能使用英文和数字等标准 ASCII 码（有的编译器是支持输出中文的，如 Keil C51）。

## 1.2　几个宏的特殊用法

前面简单总结了一下常用的宏指令以及宏的基本用法，下面将进一步介绍一些宏的特殊用法。这里虽然是基于 AVR GCC 来介绍这些技巧的，但是因为 C 语言的通用性和标准性，所以很多技巧在其他的嵌入式 C 编译器上也可以使用，甚至在计算机上的 C 编译器中也能使用（或者说嵌入式 C 编译器保留了 PC 上 C 编译器的许多特性）。

### 1.2.1　井号 #

我们知道，C 语言的很多编译指令都是以 # 字符开头的，如 #defnie、#if、#

else、#elif、#endif 等。但是在宏指令中，# 还有一些特殊的用法：如果在宏 #define 的参数中使用 #，则可以将参数转换为字符串，如：

```
#define STR(s)     #s
```

那么语句：

```
printf(STR(ABC));
```

就等效为：

```
printf("ABC");
```

## 1.2.2 双井号 # #

还有一个更有用的特殊用法：# #（注意这两个井号是相连的，中间没有空格，也没有其他字符）。# # 代表什么呢？这个宏的含义就是将 # # 前后的两个字符串连接起来，形成一个新的字符串。例如：

```
#define CONCAT(a, b)   a ## b
#define DDRLED  CONCAT(DDR, C)
```

那么 DDRLED 的结果就是 DDRC。编译的时候，在程序中使用 DDRLED 的地方编译器会自动将结果替换为 DDRC。

在 AVRUSB 中就大量使用了这样的宏定义，如在文件 usbdrv.h 中就有如下定义：

```
#define USB_CONCAT(a, b)            a ## b
#define USB_CONCAT_EXPANDED(a, b)   USB_CONCAT(a, b)
#define USB_OUTPORT(name)           USB_CONCAT(PORT, name)
#define USB_INPORT(name)            USB_CONCAT(PIN, name)
#define USB_DDRPORT(name)           USB_CONCAT(DDR, name)

#define USBOUT          USB_OUTPORT(USB_CFG_IOPORTNAME)
#define USB_PULLUP_OUT  USB_OUTPORT(USB_CFG_PULLUP_IOPORTNAME)
#define USBIN           USB_INPORT(USB_CFG_IOPORTNAME)
#define USBDDR          USB_DDRPORT(USB_CFG_IOPORTNAME)
#define USB_PULLUP_DDR  USB_DDRPORT(USB_CFG_PULLUP_IOPORTNAME)
```

正是因为灵活使用了 # # 宏，所以在 VUSB 中只需要在配置文件 usbconfig.h 中修改 USB 对应的 IO 引脚号，就可以快速将 VUSB 从一个型号的单片机移植到另外的型号上，而不用修改源程序中任何与 IO 部分相关的代码。程序会自动根据分配的 IO 利用宏生成对应的寄存器，这种方法非常有效。

上面是使用了 # # 将两个字符串连接起来，同样也可以用类似的方法将更多的字符串连接起来，如：

```
#define CONCAT3(A, B, C)    A ## B ## C
```

使用 ## 还可以连接更多的字符串，连接后的宏定义是所有字符串的累加。虽然不知道字符串连接次数是否有数量限制（与具体的编译器有关），但是需要注意连接后字符串的总长度不要超过编译器的限制。使用 ## 这个特殊方法可以实现一些特别的技巧，后面会更加详细地说明。

## 1.2.3　取特定参数

下面是宏的一种特殊用法，可以用来获取宏引用时特定的参数：

```
#define UTIL_ARG1(a, b) a
#define UTIL_ARG2(a, b) b
```

第一个宏是取两个参数中的第一个，第二个宏则是取第二个参数。

如果定义下面的宏：

```
#define LED C, 1
```

那么 UTIL_ARG1(LED)的结果就是 C，UTIL_ARG2(LED)的结果就是 1。

利用这个方法还可以获取 3 个参数中的任意一个，或者更多参数中的一个，如：

```
#define UTIL_ARG31(a, b, c) a
#define UTIL_ARG42(a, b, c, d) b
```

这个用法非常特殊，但是也非常简洁，它是我们后面介绍的一些宏使用技巧的基础。关于这个宏的更多使用方法和作用将在后面详细说明。

## 1.2.4　将编译时间保存到目标代码中

有时我们希望将程序编译时的时间和日期嵌入到最终生成的目标代码中，就像有些数码相机拍照时可以将拍摄的时间嵌入到照片的画面中一样，这样对于今后软件的维护是很有帮助的。这时就可以使用到编译器预定义的宏：__DATE__和__TIME__。（注意，这两个宏在 DATE 和 TIME 单词的前后各有两个单下划线，缺少下划线就会造成错误，下划线有时会因为印刷或者字体缘故造成显示不太清晰；此外字母 DATE 和 TIME 都必须是大写的，因为默认情况下 C 语言是区分大小写的。宏 __DATE__代表编译时的日期，而宏 __TIME__代表编译时的时间，在编译过程中，编译器会自动使用计算机的当前时间替换宏的内容，替换的结果是一个字符串。

为了将程序编译时的时间和日期嵌入到编译后的目标的代码中，我们可以在程序中做如下定义：

```
const char BUILD_TIME[] = __TIME__;
const char BUILD_DATE[] = __DATE__;
```

或者使用 PROGMEM 修饰符将变量定义到 Flash 空间中，这种方法更加符合

AVRGCC 的风格。而且这样产生的代码更小，效率更高，因为它将数值直接保存到 Flash 中，不占用 RAM 空间，而前一种方法会占用一定的 RAM 空间。在使用 PROGMEM 前，不要忘记需要包含相应的头文件"pgmspace.h"。

```
#include <avr/pgmspace.h>
PROGMEM char BUILD_TIME[] = __TIME__;
PROGMEM char BUILD_DATE[] = __DATE__;
```

使用任意支持 HEX/BIN 转换的软件时（比如本书后面介绍的 AVR 通用 Bootloader 的下载软件 AVRUBD），将编译后的 HEX 文件转换成 BIN 格式就可以看到本例编译时的时间和日期。AVRUBD 的 BIN 缓冲区的显示效果如图 1-1 所示。

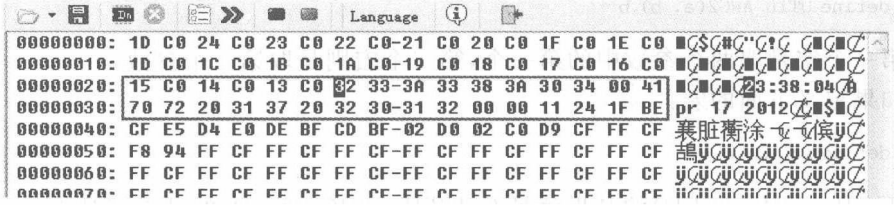


**图 1-1　AVRUBD 的 BIN 缓冲区显示效果**

关于变量在编译后 BIN 文件中的位置，可以用文本编辑软件打开编译临时文件 *.map 来查找对应的变量名。如这个例子产生的 map 文件中，我们可以看到这样的声明：

```
*(.vectors)
*(.progmem.gcc*)
*(.progmem*)
.progmem.data   0x00000026        0x15 main.o
                0x00000026                BUILD_TIME
                0x0000002f                BUILD_DATE
                0x0000003c                . = ALIGN(0x2)
*fill*          0x0000003b        0x1 00
                0x0000003c                __trampolines_start = .
```

这就很容易看出变量所在的地址了。当然也可以指定变量在 Flash 中的位置，从而方便地从程序中指定位置引用变量了。关于__DATE__和__TIME__的例子可以参考附带光盘的 macro/01 目录下的例程。

除了__DATE__和__TIME__外，还有__FILE__和__LINE__等宏，__FILE__代表当前的文件名，__LINE__代表当前的行号。__FILE__和__LINE__宏在调试时很有用，可以在编译时，使用前面介绍的 #warning 和 #error 输出告警或故障点的位置。__DATE__、__TIME__、__FILE__、__LINE__等几个编译器预定义的宏是标准 C 规定的，几乎所有的 C/C++编译器都支持这些宏，如果灵活使用这些宏，可以给程序调试和维护带来很大的方便。