基于 C51 单片机的 C 语言编程实践

作者：合肥工业大学　杨铸 方彧
网址：www.200yi.com　电邮：fairyfar@msn.com
2007.10.01 国庆日
Scripted by FairyFar.

1. 关于本文

单片机，亦称单片微电脑或单片微型计算机。它是把中央处理器（CPU）、随机存取存储器（RAM）、只读存储器（ROM）、输入/输出端口（I/O）等主要计算机功能部件都集成在一块集成电路芯片上的微型计算机。

毋庸置疑，单片机应用领域十分广泛。随便找个 WWW 搜索引擎，检索“单片机”，结果可算是汗牛充栋，所以，这里不准备赘述那些基本知识。本文是笔者实际工作的点滴经验总结，内容比较零散，随机罗列如下。

2. 实践总结

八股地，还是先来简要介绍一下本文讨论的适用范围。

关于硬件，AT89S52 是一种低功耗、高性能 CMOS8 位微控制器，具有 8K 在系统可编程 Flash 存储器。使用 Atmel 公司高密度非易失性存储器技术制造，与工业 80C51 产品指令和引脚完全兼容。片上 Flash 允许程序存储器在系统可编程，亦适于常规编程器。在单芯片上，拥有灵巧的 8 位 CPU 和在系统可编程 Flash，使得 AT89S52 为众多嵌入式控制应用系统提供高灵活、超有效的解决方案。

关于软件（程序开发环境），Keil 软件是目前最流行开发 MCS-51 系列单片机的软件，这从近年来各仿真机厂商纷纷宣布全面支持 Keil 即可看出。Keil 提供了包括 C 编译器、宏汇编、连接器、库管理和一个功能强大的仿真调试器等在内的完整开发方案，通过一个集成开发环境（μVision）将这些部份组合在一起。

本文所有案例适用 Keil 版本为：Keil μVision3，编程语言：C。尽管本文是关于 Keil 编程环境下基于 AT89S52 单片机的 C 语言编程实践，但不失一般性，文中大部分内容适用于 C51 系列单片机。

2.1 看门狗定时器

AT89S52 内置看门狗定时器（WDT，Watch Dog），它由 13 位的计数器组成，复位后，调用喂狗指令清零计数器，然后，每个机器周期该计数器自动加 1，当记数值达到 8191（1FFF）时，产生一个复位信号，系统复位。因此，在启用看门狗后，至少要在 8191 个机器周期内，调用一次喂狗指令，防止看门狗溢出复位。

以下给出 Watch Dog 使用方法及注意事项：

1、Watch Dog 寄存器地址为 0A6H，为便于应用，可以在 Keil 中定义 Watch Dog：

```
sfr WDTRST = 0xA6;        //定义 Watch Dog
```

2、启动 Watch Dog 指令：向 Watch Dog 寄存器连续送入 1EH 和 E1H，即

```
WDTRST = 0x1E;            //启动 Watch Dog
WDTRST = 0xE1;
```

3、Watch Dog 复位与喂狗指令同 2。

4、Watch Dog 一旦启动，是不能通过软件停止的，所以须要及时喂狗，Watch Dog 启动

后，如果程序员忘了喂狗，将造成系统误复位。

5、系统在掉电模式下，晶振停止了震荡，看门狗停止。当外部中断唤醒掉电模式时，最好调用一次喂狗指令，防止看门狗溢出复位。

2.2 避免存储器扩展

AT89S52 和其它单片机芯片一样，允许片外扩展存储器，但是除非必要，还是尽可能不要扩展。对于纯粹的程序员来说，可能是针对已经设计好的硬件电路板编程，不需要关心是否扩展存储器，但是，我们仍然建议，如果你拿到的板子已经扩展了存储器，那么还是尽可能不用它。当然，这条建议是从实际工作角度来说的。如果是练习，还是不要“逃避”这个问题为宜。

2.3 常量的存储位置

AT89S52 单片机片内数据存储器 RAM 只有 256 字节（其中高 128 字节为 SFR 区），而程序存储器对于一般应用来说足够大，因此应当尽可能将常数和系数（运行期无需改变的量）定义在程序存储器中。譬如，经常用于数码管显示的七（八）段码，温度转换系数表等。尤其是数据存储器不够用时，你还必须这样做。例如，

```
unsigned int data tab[] = {38629, 38157,…};     //tab[]有 171 个整型元素
```

Keil 编译无法通过，原因是片内数据存储器空间不够用，所以，必须将 tab[]定义在程序存储器中：

```
unsigned int code tab[] = {38629, 38157,…};     //定义在程序存储器中
```

2.4 代码优化

1、对于复杂的数学表达式，应尽可能简化，原表达式写成注释形式。这样做的好处是可以节省程序存储器空间以及减少程序执行时间。例如，

```
e = (a + b)/2 + (c + d)/2;        //a,b,c,d,e 均为 unsigned int
```

简化成：

```
e = (a + b + c + d)/2;            //e = (a + b)/2 + (c + d)/2
```

可以减少 7 字节代码空间。

2、避免浮点运算。我们知道 AT89S52 以及其它常用单片机的指令集没有浮点运算指令，如果你的 C 程序中有浮点运算语句，Keil 将用一系列汇编指令来完成一条浮点运算语句。我们发现每引入一条浮点运算，代码空间将以几十乃至上百字节增加，而且偶尔会出现一些我们现在还无法解释的意外现象。因此，应尽可能将浮点运算转化为整型运算。举一个我们在实际工作中遇到的例子，现有下面代码片断：

```
float code tab[] = {38.629, 38.157,…};      //tab[]有 171 个浮点型元素
…
if (adc * 0.195 >= tab[i])                  //查表。adc 和 i 是 unsigned int
…
```

以上代码功能：读取温度传感器经 ADC 采集转化后的离散值，经查表计算获得温度值。tab[]实际上是电压温度转换表，该过程需要进行浮点计算和存储，但是经过简化，发现完全可以不使用浮点计算。因为我们可以将 tab[]元素值和转换系数（0.195）同时扩大 1000 倍，这样浮点运算就转换成了整型运算。修改后代码如下：

```
unsigned int code tab[] = {38629, 38157,…};//tab[]的 171 个元素转换为整型
…
if (adc * 195 >= tab[i])                    //转换系数 0.195 也扩大 1000 倍
```

```
...
```

测试表明，以上对代码的修改，节省了 342 字节代码空间。

3、变量定义时，赋与不赋初值，汇编后代码多少是不一样的。例如，

```
unsigned int sample;
```

与

```
unsigned int sample = 0;
```

后者节省了 4 字节代码空间。因此，变量定义时是否初始化可以根据实际情况而定。通常，你被建议：变量初始化是一个好的编程习惯。但是，在单片机编程中，我们认为，应当灵活应用这条建议。

4、变量作用域会影响代码空间大小。例如，unsigned int 若定义为文件型变量（作用域为整个文件）和定义为局部变量相比，汇编后代码增加了 4 字节。

2.5 关于中断

建议不要开启那些未用到的中断，尽可能让系统轻装上阵。单片机经常被应用在工业现场，而工业现场电磁环境复杂，干扰很可能产生“弹飞”程序或者打开某个未用的中断或者关闭某个中断等等不确定的行为。程序员应根据实际情况，考虑是否应当加入程序“自救”功能。譬如，为未启用的中断编写中断服务程序（意外处理）。

2.6 语法检查

Keil 编译器语法检查相当宽松，主要表现在：

1、数组越界不报错。如果程序员不注意，很可能埋下潜在的错误（运行期错误）。例如，定义一个数组：

```
unsigned char array[5];
```

我们知道 C 语言中，这表示定义了一个大小为 5 字节的存储空间，引用时应当为 array[0]~array[4]，如果程序员在之后的程序中，使用了 array[5]，编译器不报错，但是在运行期难保不出错，因为我们并不能保证 array[4]后面的单元不被其它变量使用。

2、变量溢出不报错。假定我们有这样一条变量定义语句：

```
unsigned int sample;
```

意味着 sample 的数值范围为 0~65535，问题是当给 sample 赋一个该范围之外的数，如，

```
sample = 65536;
```

对于这样显式的超范围赋值，VC ++ 6.0 编译器会给出警告错误，而 Keil 编译器不报错，但实际上，sample 已经溢出了。因此，程序员必须对变量的范围十分小心，尤其要注意变量之间算术运算后是否溢出，例如，

```
unsigned int sample1 = 1;
unsigned int sample2 = 65535;
sample1 += sample2;                // sample1 溢出
```

这个例子理所当然是简单的，但是在程序越来越复杂时，你还能保持大脑“不缺氧”吗？

3. 结束语

上文多个例子对比了代码修改前后的代码空间大小，有的效果不明显，只节省了几个字节，或许你要说这点节省对于可用的 8KB 空间来说实在是微不足道。是的，无可否认，多那么几个字节算不了什么。但是，谁让咱们是程序员呢。正如音乐家对待自己的曲子一丝不苟到音节，我们是程序员，代码就是我们的艺术品，编程就是我们的艺术创作——精益求精，壹字（节）千金。