ANSIC 程序到 KeilC51 的移植心得

李章林　张立民

（　南开大学信息技术科学学院，　天津　300071　）

摘要：文章讲述了将 ANSIC 程序移植到 KeilC51 上应该注意的事项。文章在总结作者使用 KeilC51 编写程序和移植程序的基础上，讲述了存储类型、指针类型、重入函数、根据目标系统 RAM 的分布的段定位和仿真栈设置、函数指针、NULL 指针问题、字节顺序、交叉汇编等移植时需要注意的事项。对存储类型、指针类型、重入函数对程序的效率的影响进行了分析，从而对如何进行高效的移植给出了指导。最后文章以将 ucosii 移植到 KeilC51 的小模式下为实例，讲述了移植的一般步骤，希望给读者正确而高效的移植 ANSIC 程序到 KeilC51 提供参考。

关键字：ANSI C 程序 移植 KeilC

1 引言

C 语言是应用很广泛的计算机语言。因为它具有很强的移植性等优点，在编写单片机程序时，有时系统的可读性、易维护性往往比程序的效率更重要，这时候我们可以选择 C 语言作为程序语言。使用 C 语言的另一个优点是可以利用大量的已有程序资源，为 X8086 等 CPU 编写的 C 程序只要稍加修改就可以拿过来用，避免了重复开发。KeilC51 是 51 系列单片机上的优秀的 C 编译器，了解 KeilC 的特点将有利于编写和移植高效的 C51 程序。

2 指定存储类型，尽量使用小模式编译

KeilC 中的变量除了可以设置数据类型以外还可以设置存储类型(Memory type)。对于变量常需要在 data，idata，pdata 和 xdata 这几个存储类型之间做一个选择，它们分别将变量放在内部 RAM，间接寻址内部 RAM，用 R0、R1 寻址的外部 RAM，用 DPTR 寻址的外部 RAM。KeilC 编译器使用的存储模式(memory model)有小模式、紧凑模式和大模式。在各个模式下，如果变量没有指定存储类型，默认分别对应 data、pdata、xdata 存储类型。四种存储类型访问速度依次降低，但是可用空间依次增多。

假如程序有较多的外部变量，如果从 ANSI 移植到 KeilC 不给变量指定存储类型，那么一般只能使用大模式编译，这样程序速度较慢。为了能在小模式下编译，我们可以将数据量大、访问量小的变量定义为 xdata 类型，我的做法是将所有的外部变量都定义为 xdata 或者 pdata，局部变量不指定存储类型，这样一般能在小模式下编译。

3 尽量使用指定存储类型的指针(memory-specific pointer)不使用一般指针(generic pointer)

如果程序移植的时候不做修改，所有的指针将都是“一般指针”，我们的建议是尽量修改为“指定存储类型”的指针，因为它的效率要高很多[(1)]。

首先一般指针使用三个字节，第一个字节指示是什么存储类型，后两个字节是指针指向的地址。“指定存储类型”的指针则只用一个或者两个字节[(1)]。可见“一般指针”占用内存多。

另外，为了取得“一般指针”指向的数据，程序必须调用?C?CLDPTR 函数，在？C？CLDPTR 中根据指针第一字节指示的存储类型采取不同的读取 RAM 的方式。而使用“指定存储类型”的指针时，采取哪种读取 RAM 的方式在编译时已经确定，不用在运行时动态判断。可见“一般指针”运行效率低。

“指定存储类型”的指针指向的变量必须要有明确的存储类型。一般情况下程序中使用指针是为了指向大块内存，而 KeilC 中大块内存一般定义为外部变量。依照第一点移植建议，所有的外部变量都定义为 xdata 或者 pdata 类型了，有明确的存储类型，这说明程序中的指针基本都可以改为“指定存储类型”的指针。

4 需重入函数增加 reentrant 关键字

X8086CPU 上运行的 Dos 和 Windows 程序中的函数都是可重入函数。但是为提高效率，KeilC 默认情况下使用寄存器传递参数，局部变量放在固定的内存空间，这样函数就不可重入了。如果不加修改的将 ANSI 程序移植到 KeilC，发生不可重入函数被重入时，程序运行将出错。这时我们需要将可能被重入的函数后增加 reentrant 关键字[1]。

但是我们往往对需要移植的程序的流程不太了解，这样也就不清楚哪个函数可能被重入。这里提供一个方法：首先不添加 reentrant，在 KeilC 下编译连接，将会有警告：如果提示“recursive call to non-reentrant function”，说明此函数被递归调用而重入；如果提示“multiple call to segment”，说明此函数很可能是被中断函数和非中断函数都调用而重入。然后，在有以上警告的函数后增加 reentrant 关键字。但是以上的设置方法并不是万无一失，比如有函数指针存在的程序，函数调用树(call tree)不能反映真实调用情况；又如程序中改变压入堆栈的程序指针，使得函数返回时不回到原来的调用点，例如 ucosii 就是采用这种方式进行任务切换，这时 KeilC 编译器无法建立正确的函数调用树，无法判断函数是否被重入。

既然判断函数是否会被重入较麻烦，为何不将所有的函数都设置为 reentrant 类型？为了明白这点，我们首先要了解一下 reentrant 函数的执行速度和代码量。

为了使函数可重入，KeilC 使用了仿真栈(simulated stack)，它区别于 SP 寄存器指向的硬件栈(hardware stack)。在大模式、紧凑模式和小模式下仿真栈分别被定义在 XDATA、PDATA、IDATA 空间中。仿真栈从上向下生长。有一个全局变量（编译器自动定义的）指向栈顶，对于不同的存储模式该变量分别是：?C_XBP、 ?C_PBP、 ?C_IBP [1]。仿真栈的作用和 Dos 操作系统下的堆栈作用是类似的。重入函数和非重入函数运行时的区别主要有：

情况	非重入函数	重入函数
函数参数无法全部通过寄存器传递时	通过局部数据段传递	通过仿真栈传递
需要局部变量时	局部变量放在局部数据段中	局部变量放在仿真栈中
函数返回时		调整仿真栈顶

X0886CPU 支持类似于 mov eax，dword ptr [esp+20]的汇编语言来读取堆栈的内容，而 51 单片机没有读取仿真栈的配套指令，所以仿真栈的额外操作使得速度变慢、代码量增大。如果你的移植系统对速度和代码量有要求，要避免设置不必要的函数为 reentrant 类型。

5 目标系统的外部 RAM 起始地址影响段定位和仿真栈设置

例如你的系统的外部 RAM 为 32K，而 KeilC 默认情况下认为外部 RAM 为 64K，如果移植程序使用了超过 32K 的 RAM，编译器不会报错，但是程序运行将会出错；又如，你的系统为了某种需要将 RAM 范围设置为 0x8000-0xFFFF，这时也需要告诉 KeilC 地址范围。

设置 xdata 段定位的方法。例如外部 RAM 地址分布为 0x0000-0x4000 和 0xC000-0xFFFF。命令行方式下使用 BL51 的选项 XDATA[2]：BL51 MyProgram.obj XDATA(0x0000-0x4000,0xC000-0xFFFF)；在 KeilC 集成开发环境下，找到菜单 project/option for target1/BL51 location，在 Xdata 输入框中输入 0x0000-0x4000,0xC000-0xFFFF。

设置 pdata 段定位的方法。如果让 pdata 使用 0x8000-0x80FF 之间的外部 RAM，在命令行方式下使用 BL51 的选项 PDATA[2]：BL51 MyProgram.obj PDATA(0x8000)；在集成开发环境下，找到菜单 project/option for target1/BL51 location，在 Pdata 输入框中输入 0x8000。其中 0x8000 就是 pdata 的起始地址。还要修改 Startup.a51，修改如下： ① 增加 Startup.a51 到工程：将 KeilC\C51\LIB\Startup.a51 拷贝一份到你的工作目录下，然后添

加到你的工程中。② 找到 startup.a51 中的

```
PPAGEENABLE EQU 0   ; set to 1 if pdata object are used.
PPAGE       EQU 0   ; define PPAGE number.
```

修改为：

```
PPAGEENABLE EQU 1   ; set to 1 if pdata object are used.
PPAGE       EQU 80H ; define PPAGE number.
```

初始化时，PPAGE 将被赋予单片机 P2 口寄存器，当程序使用类似 MOVX A,@R0 时，高 8 位地址就是 PPAGE 的值。使用 pdata 类型数据时，要特别注意不能随意在程序中修改 P2 寄存器的值。

大模式下设置仿真栈顶。在大模式下仿真栈在 xdata 空间。如果外部 RAM 地址范围是 0x0000 到 0x8000。此时需要设置栈顶为 0x8000，默认情况下的(0xFFFF+1)将会使程序出错。设置方法是：① 增加 startup.a51。② 修改 startup.a51 中的部分代码为如下代码：

```
XBPSTACK        EQU 1   ; set to 1 if large reentrant is used.
XBPSTACKTOP EQU 7FFFH+1; set top of stack to highest location+1..
```

紧凑模式下设置仿真栈顶。默认的情况下为 0xFF+1。但是某些时候采用默认值会出错。比如 pdata 所有变量占用 0x80 字节的空间,并且你的程序中有 0x80 字节的 xdata 类型的数据。那么默认情况下 pdata 数据放到 0-0x007F，xdata 放到 0x0080-0x00FF。这时默认的仿真栈顶在 0x00FF，它和 xdata 数据区冲突。一个解决的办法是将 pdata 段定位到 xdata 段的后面，例如这里将 pdata 段起始地址定位在 0x100。

6 KeilC 中的函数指针

如果被移植的程序中使用了函数指针，那么就要注意覆盖分析的出错问题[3]。问题的产生在于“覆盖分析”(overlay)技术。在小模式下编译的 C51 程序局部变量都放在 data 空间中，为了重复利用 data 空间，KeilC 采用了 overlay 技术：一个程序中函数的层层调用会形成一个函数“调用树”(call tree)，处于函数调用树的不同树枝上的函数可以共享一块内存空间（即覆盖），这样就节省了内存空间的使用。KeilC 能够根据函数调用树进行正确的覆盖分析。使用函数指针一般有两种操作：① 将一个函数名赋给一个函数指针，这时 KeilC 编译器误认为调用了这个函数名对应的函数。② 使用函数指针调用函数，这时 KeilC 编译器不能发现调用了函数。这都使得函数调用树构造出错，根据此调用树进行的覆盖分析也将出错，致使局部变量冲突，程序出错。对此有两种措施：① 手动修正调用树：使用 BL51 的 OVERLAY 选项增删调用树的树枝[3]。② 将通过函数指针调用的函数都设置为 reentrant 类型。由于 reentrant 类型局部变量在仿真栈中，不会引起局部变量冲突。

ANSIC 中，通过函数指针调用的函数的参数的个数没有限制，但是 KeilC 对此有限制，至多 3 个参数[3]。因为，KeilC 编译时，无法通过函数指针找到该函数的局部数据段，也就无法通过局部数据段传递参数，只能通过寄存器传递参数，所以参数个数是有限制的。碰到这个问题时解决办法是：① 将该函数改为 reentarnt 类型。② 修改源程序，将多个参数放在一个结构体中传递。

7 NULL 指针问题

C 程序一般规定任何变量都不能使用地址为 0 的内存。但是单片机的 xdata 空间的 0 地址内存在默认的情况下是可以被使用的。现假如有内存分配函数 malloc(int size)，malloc 函数成功分配了一块 0 地址开始的内存，返回首地址 0，当程序发现返回值等于 NULL（即 0）时误认为内存分配失败。为了防止以上错误，我们在移植时要增加以下一个全局变量：

```
Char xdata NULLAddr _at_ 0
```

这里使用了 KeilC 的_at_关键字将一个变量 NULLAddr 指定在 0 地址，从而避免了其它变量占用 0 地址。

8 字节顺序(byte order)

X8086 等 CPU 在内存中双字节变量：高字节在高地址，低字节在低地址。KeilC51 默认双字节变量则顺序相反。字节顺序引起修改的一个典型例子：TCP/IP 程序中的 htons()函数将主机字节顺序转化为网络字节顺序，对于 X8086 和 KeilC51 这个 htons()函数是不同的。

9 交叉汇编

移植的时候可能还需要编写少量的 51 汇编程序。汇编和 C 互相调用应该遵守 KeilC 的参数传递和返回值传递规则[1]。为了使汇编程序也能够进行 overlay 分析，汇编的书写要有一定的格式[1]。另外需要强调的一点是：被 C 程序调用的汇编函数可以使用所有的寄存器，而不用担心会修改 C 程序中使用的寄存器[1]。

10 关键字

pdata、data 等 KeilC 关键字可能被 ANSIC 程序中用作变量名，必须修改之。

11 实例：Ucosii 到 KeilC 小模式下的移植

Ucosii 已经由杨屹移植到 KeilC 的大模式下[4]，本文讲述将其修改为小模式的方法。移植步骤如下：

(1)将所有的外部变量定义为 xdata 储存类型。

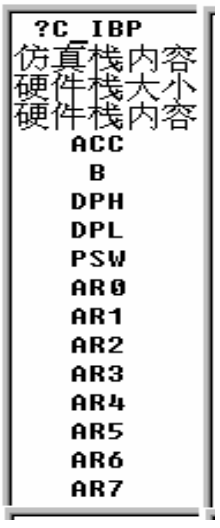
(2)修改指针：查找’*’符号，发现是指针定义的地方在’*’号前加 xdata。

(3)在所有的函数申明后增加 reentrant 关键字。对 Ucosii，无法用上文提到的方法判断哪些函数可能被重入，只好全部设置为可重入函数。

(4)根据你的目标系统的外部 RAM 起始地址定义 xdata 段的起始地址。下面具体讲一下移植到小模式下仿真栈的使用。

在小模式下仿真栈顶默认设置在内部 RAM 空间的顶端 0xFF。硬件栈顶初始值由 KeilC 自动分配，实际上在决定栈顶以前 KeilC 先安排所有的 data 类型变量，然后设置 SP 指向空余 data 空间的开始。这时两个堆栈上下相对增长。对于堆栈是否会溢出，KeilC 本身不提供编译警告，只能在程序运行时调试。

Ucosii 任务栈中是否需要保存堆栈，因移植系统的不同而不同。① 移植到堆栈在外部 RAM 中的系统上（例如 Dos）时，只要保存当前堆栈的指针就可以了。② 移植到 KeilC 大模式下时，需要保存硬件栈的内容和仿真栈的指针[5]。③ 移植到 KeilC 小模式下，需要保存硬件栈的内容和仿真栈的内容，它的任务栈的结构如右图所示。



通过变量“？C_IBP”可以知道仿真栈所在的内部 RAM 区间。用以下的方法可以获得初始硬件栈顶[4]，在汇编程序中增加以下代码：

```
?STACK SEGMENT IDATA
        RSEG ?STACK
StkBottom:
```

标号 StkBottom 即为硬件栈的初始栈顶。通过硬件栈大小和初始栈顶可以知道硬件栈所在内部 RAM 的区间。图中的寄存器的排列顺序和 KeilC 在进入中断以后保存寄存器的顺序是一致的，和中断时寄存器压栈顺序一致是 ucosii 所要求的。

(5)函数指针问题。Ucosii 有任务切换，KeilC 得到函数调用树是错误的。另外在 main 函数中一般将任务函数(例如 Task1)作为参数传递给 OSTaskCreate 函数，KeilC 误认为 main 函数调用了 Task1。由于已经将所有的函数都申明为 reentrant 类型，所以没有必要手动修正调用树，实际上也很难修正。

(6)NULL 指针问题。使用以上提到的方法，避免 NULL 指针问题。

(7)交叉汇编。Ucosii 移植的需要编写一部分 51 汇编程序。

(8)关键字。Ucosii 中使用 pdata、data 作为变量名，修改这些变量名[4]。

参考文献：

[1]德国 KeilC 公司 《Cx51 Compiler》http://www.keil.com 2001 年 5 月 P103-P108，P126,P155-P158

[2]德国 KeilC 公司 《Macro Assembler and Utilities for 8051 and Variants》http://www.keil.com 2000 年 7 月 p325,p317

[3]德国 KeilC 公司 《Function Pointers in C51》http://www.keil.com/appnotes/files/apnt_129.pdf 1999 年 4 月 27

[4]杨屹 《uCOS51 移植心得》http://www.zlgmcu.com/philips/philips-embedsys.asp 2002 年 10 月 3 P3,P3

[5]杨屹 《uCOS51 重入问题的解决》http://www.zlgmcu.com/philips/philips-embedsys.asp 2002 年 10 月 9

What I learned from Porting of ANSI C program to KeilC51

Li Zhanglin Zhang Limin

(.College of Information Technology Science,NanKai University,Tianjin 300071)

ABSTRACT：The thesis introduces what should be noted when porting an ANSI C program to KeilC51. It explains memory type, pointer type, reentrant function, segment locating and simulated stack setting based on your target system, function pointer, NULL pointer issue, byte order, cross assembly and so on about notation when porting, based on summary of the author's programming and porting with KeilC51. The thesis gives a analysis in how memory type, pointer type, reentrant function affect efficiency of program and give a direction on efficient porting. Finally it illustrates porting of ucosii to KeilC small model as a example.

Key word：ANSI C program porting KeilC