**Keil C51中函数指针使用注意事项**

在我们的代码中大量使用了函数指针。当函数指针用在Keil C51中时，一定要注意编译器自动生成的函数调用树通常是不正确的，需要手动调整。否则可能造成无法预知的后果。

这是因为，Keil C51编译器并不把函数参数和局部变量压入堆栈中，而是放在寄存器或固定的内存位置。

C51的编译器监视函数调用的嵌套顺序，把几个函数的变量放在同样固定的位置。在C51编译器中连接器会搜索所有函数中变量占用存储区间最多的函数，然后以这个函数的变量的占用空间开辟一片空间，其他函数的变量也放在该空间中，同时实现了变量的覆盖（无相互调用）与地址的共享。例如函数A占10个字节，函数B占20个字节，函数C占15个字节，如果它们之间没有相互调用则仅需20个字节就可以满足45个字节的变量需要。

正是由于所有函数的参数和局部变量的共享一个覆盖区，函数没有相互的调用时，在执行一个函数时，会将另一个函数的变量的存储区覆盖。如果函数有调用，那么不会覆盖原来函数的局部变量的区间。

调用树(call tree)是由Keil链接器自动生成的，用于描述函数的调用关系（调用树可通过编译生成的\*.M51文件的OVERLAY MAP OF MODULE部分查看，该部分详细的说明了函数的调用关系以及对覆盖存储区的使用情况）。链接器通过分析调用树来确定哪些寄存器或内存位置是可安全覆盖的。这样两个不同时调用的函数就可以共享同一块内存用于传递参数和存储局部变量。但对于函数指针来说，编译器并不知道函数指针将指向哪个函数。这导致了调用树构造出错的可能，函数的参数和局部变量也可能被错误覆盖（例如，函数A通过函数指针调用了函数B，但编译器并不知道它们之间存在调用关系，所以认为它们是可以共享同一块内存的。这样当函数A调用了函数B，回到函数A后，函数A的参数和局部变量可能已经被改变了，再往下运行就出错了）。

对此，Keil提供了链接器OVERLAY伪指令，可让用户自行修改调用树，调整函数的调用关系。

删除调用关系，命令格式：

OVERLAY (sfname-caller ~ sfname-callee)

OVERLAY (sfname-caller ~ (sfname-callee, sfname-callee))

举例：OVERLAY(?PR?\_FUNC?DMAIN ~ (?PR?\_FUNC\_A?DMAIN,?PR?\_FUNC\_B?DMAIN))

意思是从FUNC函数中删除对FUNC\_A和FUNC\_B的调用。

添加调用关系，命令格式：

OVERLAY (sfname-caller ! sfname-callee)

OVERLAY (sfname-caller ! (sfname-callee, sfname-callee))

举例：OVERLAY(?PR?\_MAIN?DMAIN ! (?PR?\_FUNC\_A?DMAIN,?PR?\_FUNC\_B?DMAIN))

意思是添加FUNC函数对FUNC\_A和FUNC\_B的调用。

**方法一**： 可在链接命令行输入命令。或在Keil集成开发环境中，在“BL51 Misc”－“Overlay”中填入()中的内容。

**方法二： 函数指针的建议和技巧**

  有些函数指针的应用技巧。  
使用指定空间的指针

  把函数指针从一个普通的指针变成一个指定空间的指针。用一个字节保存指针。因为函数属于CODE存储区（在8051里），一个字节可以用来保存声明的函数指针作为CODE指针。例如：

  void (code \*function\_ptr) (void) = another\_function;

如果你选择在你的函数指针声明中包含code关键字，就可以在任何地方使用它。如果你声明一个函数，它接收一个3字节的普通指针，通过指定空间传递，2字节函数指针，坏事将要产生。