手动初始化变量

介绍

本文说明如何对对变量进行手动初始化/归零操作，对需要手动初始化/归零变量的操作作为参考。

全部变量和静态变量的初始值默认是在系统初始化阶段，由编译器提供的初始化代码自动完成。但在一些情况下，用户想自己实现变量的初始化或者将变量归零，可以借助 IAR 编译器函数来完成这项操作。以下的示例使用 IAR Embedded Workbench for ARM 8.42 版本，其链接器是 ILink，其他的 IAR 版本如果是链接器是 Xlink，则本文的方法不适用。

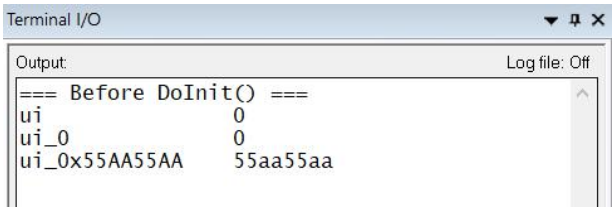
示例

在应用程序中定义了 3 个全局变量进行测试，其中 ui 是隐式零初始化，其值会被编译器自动初始化为 0；ui_0 是显示零初始化，其值也会被编译器初始化为零；ui_0x55AA55AA 是显示初始化，其初始值为 0x55AA55AA，并且编译器会创建一个内容为 0x55AA55AA 的初始式，链接时保存在 ROM 当中，初始化的实际过程就是将 ROM 中的这个初始式的内容复制到变量所在的地址。

默认初始化：

```
unsigned int ui;                          //.bss - implicit zero-init
unsigned int ui_0=0;                      //.bss - explicit zero-init
unsigned int ui_0x55AA55AA=0x55AA55AA;    //     - explicit init


printf("=== Before DoInit() ===\n");
printf("ui \t\t %x \n", ui);
printf("ui_0 \t\t %x \n", ui_0);
printf("ui_0x55AA55AA \t %x \n", ui_0x55AA55AA);
```



以上是由编译器自动完成初始化之后，各变量的值。

手动初始化：

实现手动初始化的第一步能够选出变量所在的 section，有两种方法：1、如果只是需要将模块中的部分变量进行手动初始化，则需要使用 IAR 的#pragma location 预处理命令在 C/C++源文件中将变量放置在一个特定的数据 section 当中；2、如果要操作的是一个模块当中的全部变量，可以直接在链接器脚本文件中使用命令直接对编译后的.o 文件操作。本文示例采用方法 2。

上面定义的全局变量都在 separately_inited_vars.c 文件当中：

```
ui                      0x2000'0014    0x4  Data  Gb  separately_inited_vars.o [1]
ui_0                    0x2000'0018    0x4  Data  Gb  separately_inited_vars.o [1]
ui_0x55AA55AA           0x2000'0010    0x4  Data  Gb  separately_inited_vars.o [1]
```

修改 Ilink 链接器配置文件(*.icf)：

1、将 separately_inited_vars.o 中的零初始化的 section(.bss, ui 和 ui_0 的 section)添加到“do not initialize”命令中，指示编译器不对其进行零初始化；

2、定义名为 MYZEROBLOCK 的 block，将 ui 和 ui_0 变量所在的 section 放置其中；

3、将 separately_inited_vars.o 中的带初始值的变量的 section(.data，ui_0x55AA55AA 的 section)添加到“initialize manually”，指示编译器变量需要手动初始化，生成变量的初始式；

4、定义名为“MYBLOCK”的 block，将 separately_inited_vars.o 中的带初始值的变量的 section(.data，ui_0x55AA55AA 的 section)放置在该 block 中；

5、定义名为 MYBLOCK_init 的 block，将 ui_0x55AA55AA 的初始式所在的 section 放置在这个 block 中；

6、通过 place in 指令，将定义的各个 block 放置在对应的存储区，其中初始式所在的 block 需要放置在 ROM 区。

```
do not initialize  { section .noinit,
                     section .bss object separately_inited_vars.o };

define block MYZEROBLOCK  { section .bss object separately_inited_vars.o };

initialize manually { section .data object separately_inited_vars.o };
define block MYBLOCK { section .data object separately_inited_vars.o  };
define block MYBLOCK_init { section .data_init object separately_inited_vars.o  };

place at address mem:__ICFEDIT_intvec_start__ { readonly section .intvec };

place in ROM_region   { readonly,
                        block MYBLOCK_init };
place in RAM_region   { readwrite,
                        block MYBLOCK,
                        block MYZEROBLOCK,
                        block CSTACK, block HEAP };
```

编写变量初始化函数：

```
#pragma section = "MYBLOCK"
#pragma section = "MYBLOCK_init"
#pragma section = "MYZEROBLOCK"

void DoInit(void)
{
  char * from = __section_begin("MYBLOCK_init");
  char * to = __section_begin("MYBLOCK");
  memcpy(to, from, __section_size("MYBLOCK"));
```

```
  to = __section_begin("MYZEROBLOCK");
  memset(to, 0x00, __section_size("MYZEROBLOCK") );
}
```

初始化函数的实现机制很简单，只是获取变量所在的 block 的地址、初始式 block 的地址，以及它的大小，然后调用 memcpy 函数进行复制操作。其中__section_begin、__section_size 是编译器函数，分别用于获取 section/block 的起始地址和结束地址。函数的参数是 section/block 的名称，需要在前面使用预处理命令#pragma section 列出 section/block 的名称。

在工程中通过调试宏.mac 脚本将 3 个变量的值进行了修改，见下图的 Terminal I/O 窗口的 Before DoInit 输出，在调用了 DoInit 手动初始化函数之后，3 个变量的值变成了它们原本的初始值。