应用说明

C环境中的VOLATILE关键字

2017.02

Author: Lu Xiang

Revision History

NO	日期	描述	作者
1	2017-2	新建	鲁翔

1. 概要

volatile 提醒编译器它后面所定义的变量随时都有可能改变，因此编译后的程序每次需要存储或读取这个变量的时候，都会直接从变量地址中读取数据。如果没有 volatile 关键字，则编译器可能优化读取和存储，可能暂时使用寄存器中的值，如果这个变量由别的程序更新了的话，将出现不一致的现象。或者直接将该变量作为中间变量优化掉了。下面举例说明。在 C 开发中，经常需要等待某个事件的触发，所以经常会写出这样的程序：

```
unsigned int flag;
void test()
{
  do1();
  while(flag==0);
  do2();
}
```

这段程序等待内存变量 flag 的值变为 1 之后才运行 do2()。变量 flag 的值由别的程序更改，这个程序可能是某个硬件中断服务程序。例如：如果某个按钮按下的话，就会产生外部中断，在按键中断程序中修改 flag 为 1，这样上面的程序就能够得以继续运行。但是，编译器并不知道 flag 的值会被别的程序修改，因此在它进行优化的时候，可能会把 flag 的值先读入某个寄存器，然后等待那个寄存器变为 1。如果不幸进行了这样的优化，那么 while 循环就变成了死循环，因为寄存器的内容不可能被中断服务程序修改。为了让程序每次都读取真正 flag 变量的值，就需要定义为如下形式：

```
volatile unsigned int flag;
void test()
{
    do1();
    while(flag==0);
    do2();
}
```

需要注意的是，即使没有 volatile 程序也可能正常运行，但是可能修改了编译器的优化级别之后就又不能正常运行了。因此经常会出现 debug 版本正常，但是 release 版本却不能正常的问题。所以为了安全起见，只要是等待别的程序修改某个变量的话，就加上 volatile 关键字。

2. Volatile的本意是“易变的”

由于访问寄存器的速度要快过 RAM，所以编译器一般都会作减少存取外部 RAM 的优化。比如：

```
static int flag=0;
int main(void)
{
    ...
    while (1)
    {
      if (flag) do_something();
    }
}

/* Interrupt service routine. */
void ISR(void)
{
    flag=1;
}
```

程序的本意是希望 ISR 中断产生时，在 main 当中调用 do_something 函数，但是，由于编译器判断在 main 函数里面没有修改过 flag，因此可能只执行一次对从 flag 到某寄存器的读操作，然后每次 if 判断都只使用这个寄存器里面的“flag 副本”，导致 do_something 永远也不会被调用。如果变量加上 volatile 修饰，则编译器保证对此变量的读写操作都不会被优化（肯定执行）。所以此例中 flag 也应该如此说明。

一般来说，volatile 用在如下几个地方：

- 中断服务程序中修改的，供其它程序检测的变量。
- 多任务环境下各任务间共享的标志。
- 存储器映射的硬件寄存器通常也要加 volatile 说明，因为每次对它的读写都可能由不同意义。

3. Volatile与编译优化

volatile 总是与优化有关，编译器有一种技术叫做数据流分析，分析程序中的变量在哪里赋值、在哪里使用、在哪里失效，分析结果可以用于常量合并，常量传播等优化，进一步可以死代码消除。但有时这些优化不是程序所需要的，这时可以用 volatile 关键字禁止做这些优化，volatile 的字面含义是易变的，它有下面的作用：

第一，不会在两个操作之间把 volatile 变量缓存在寄存器中。在多任务、中断环境下，变量可能被其他的程序改变，编译器自己无法知道，volatile 就是告诉编译器这种情况。

第二，不做常量合并、常量传播等优化，所以像下面的代码：

```
volatile int i = 1;
if (i>0) ...
```

由于 i 变量申明时 volatile 的存在，if 的条件不会当作无条件真。

第三，对 volatile 变量的读写不会被优化掉。如果你对一个变量赋值但后面没用到，编译器常常可以省略那个赋值操作，然而对 Memory Mapped IO 的处理是不能这样优化的。