AVR-GCC 中如何使用 volatile 关键字

volatile 的字面含义是易变的，那么将一个变量指示为 volatile 是什么意思呢？是告诉编译器这个变量是易变的？事实上也是如此。在多任务、中断等环境下，变量可能被其他的任务改变，而编译器无法发现，volatile 就是告诉编译器这个变量在其它任务（或中断）中可能要修改。

使用 volatile 是与编译器优化有关系的。先看看下面的例子：

```
#include <avr/io.h>

unsigned char g_Flag=0;

int main(void)
{
   g_Flag=1;
   g_Flag=0;
   while(1);
}
```

main 程序中对 g_Flag 先赋值 1,后赋值 0, 显然 g_Flag=1 是没有意义的,聪明的编译器发现这个后就产生了如下的代码:

```
int main(void)
{
  5c:  cf e5         ldi r28, 0x5F ; 95
  5e:  d4 e0         ldi r29, 0x04 ; 4
  60:  de bf         out 0x3e, r29 ; 62
  62:  cd bf         out 0x3d, r28 ; 61
   g_Flag=1;
   g_Flag=0;
  64:  10 92 60 00   sts 0x0060, r1
   while(1);
  68:  ff cf         rjmp   .-2       ; 0x68 <main+0xc>
```

编译器没有生成 g_Flag=1 的机器码, 在优化过程中 g_Flag=1 被忽略了。在一般的 C 程序中这是没有问题的，但是如果 g_Flag 是个单片机 I/O 口变量那就麻烦了，本来想从 I/O 口输出一逻辑正脉冲的程序就这样被编译器优化掉了，显然不是我们所希望的。这时候该怎么办呢？请拿出利剑 volatile，他能处理我们所遇到的麻烦。

将 unsigned char g_Flag=0; 该为 volatile unsigned char g_Flag=0;后编译所产生的汇编代码如下：

```
   int main(void)
   {
     5c:  cf e5         ldi r28, 0x5F ; 95
```

```
     5e:  d4 e0         ldi r29, 0x04 ; 4
     60:  de bf         out 0x3e, r29 ; 62
     62:  cd bf         out 0x3d, r28 ; 61
      g_Flag=1;
     64:  81 e0         ldi r24, 0x01 ; 1
     66:  80 93 60 00   sts 0x0060, r24
      g_Flag=0;
     6a:  10 92 60 00   sts 0x0060, r1
      while(1);
     6e:  ff cf         rjmp   .-2       ; 0x6e <main+0x12>
```

在《AVR 单片机 GCC 程序设计》第二章中我们展开了 PORTB 这个宏，对于 AT90S2313 它等同于 *(volatile unsigned char *)(0x38) ，现在你一定相信这里的 volatile 是必不可少的。尽管如此，有人会认为 “在一般的程序中不需要定义端口宏，因为 avr-libc 都为我们定义好了，我只要包含 io.h 就可以了，在用户程序中 volatile 是无关紧要的” ，那么让我们再来看一下 volatile 必不可少的另一种情况。

```
#include <avr/io.h>
#include <avr/interrupt.h>

unsigned char g_Flag=0;//串口接收标记

ISR(SIG_UART_RECV)
{
    ... ...
    g_Flag=1;
}

int main(void)
{
    ...  ...
    while(!g_Flag);//等待串口接收到数据

    ... ...
}
```

上面的程序看似没有任何问题，实际上 main 函数是永远也发现不了串口接收数据的。由于编译器的优化，while(!g_Flag)将生成一个非常有意思的代码，它首先从 g_Flag 对应的内存读一次数据到一个寄存器中，之后不停的测试此寄存器是否为非零,即使中断程序中已经改变了 g_Flag 对应内存的值，它还是始终检查一个不再更新的寄存器。那么如何让 while 循环每次都要从内存读取后再测试它是否为零呢？你应该猜到了，就是将 g_Flag 变量指示为 volatile，告诉编译器 g_Flag 是易变的，要对它进行保守处理。

volatile 是一种优化指示，编译器优化操作使用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，从而可以消除死代码。但有时这优化不是程序所希望的，这时可以用 volatile 关键字关闭这个优化开关。

最后提醒读者，没有必要将所有的变量都指示为 volatile ，那样做的后果将是代码膨胀和执行效率降低。如果到现在还是不能确定自己哪些变量应指示为 volatile，请记住一个原则：将那些中断（或操作系统中其它任务）中改变，主程序中循环检查的状态变量指示为 volatile。