使用 MSP430F1611 中的大 RAM 注意事项

TI 推出了拥有 10K 内置 SRAM 的单片机 MSP430F1611，许多用户在使用中发现在 IAR 环境中不能设置多个大数组。当设置多个大数组后，会无法进入硬件仿真环境（FET Debug），就连下载程序后脱机运行也不正常。为此，我们做了一个测试，并找到了原因与解决方法。

测试条件：

硬件环境：MSP430F1611 单片机；MSP430-JTAG 工具；MSP430PM-64 适配器；

软件环境：IAREW430 3.10A（IAR Embedded Workbench IDE V 3.10A）

测试程序：

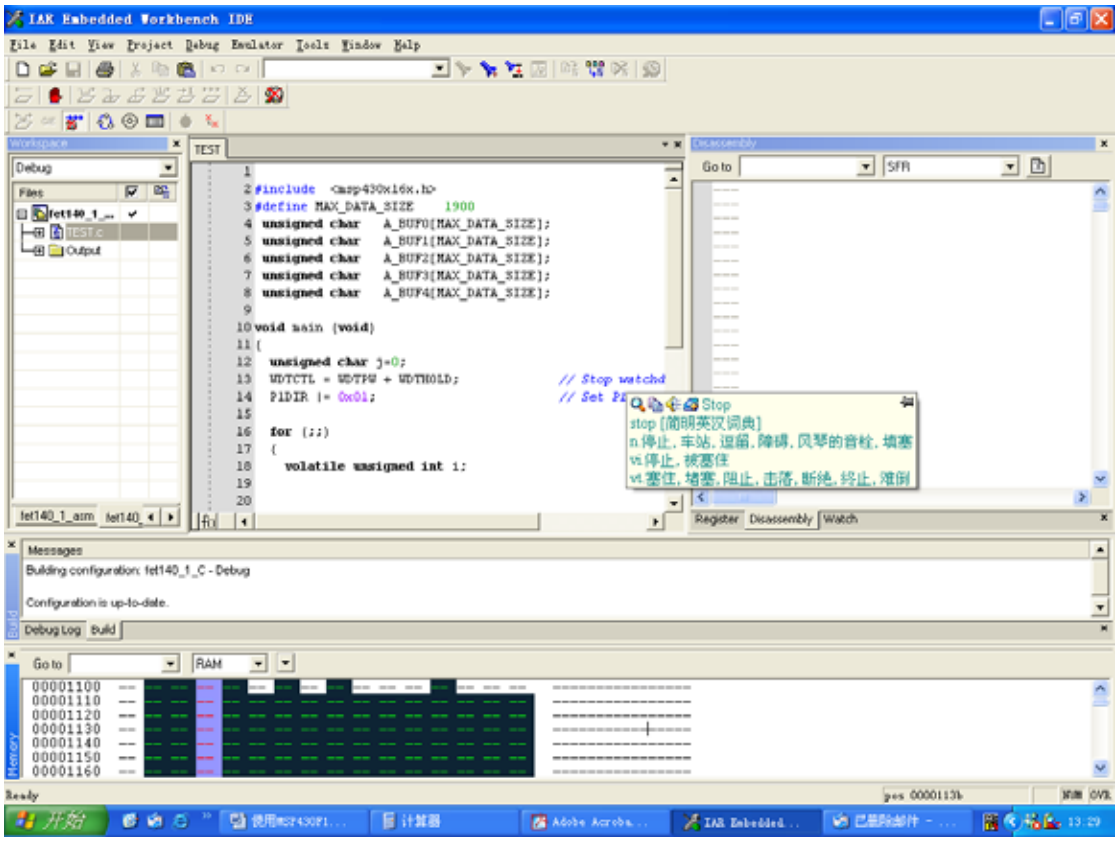
```
#include   <msp430x16x.h>
#define MAX_DATA_SIZE       1900
 unsigned char     A_BUF0[MAX_DATA_SIZE];
 unsigned char     A_BUF1[MAX_DATA_SIZE];
 unsigned char     A_BUF2[MAX_DATA_SIZE];
 unsigned char     A_BUF3[MAX_DATA_SIZE];
 unsigned char     A_BUF4[MAX_DATA_SIZE];

void main (void)
{
  unsigned char j=0;
  WDTCTL = WDTPW + WDTHOLD;                    // Stop watchdog timer
  P1DIR |= 0x01;                               // Set P1.0 to output direction

  for (;;)
  {
    volatile unsigned int i;
        P1OUT ^= 0x01;                         // Toggle P1.0 using exclusive-OR
    i = 50000;                                 // Delay
    do (i--);
    while (i != 0);
    for(i=0;i<MAX_DATA_SIZE;i++)
    {
    A_BUF0[i]=j;
    A_BUF1[i]=j;
    A_BUF2[i]=j;
    A_BUF3[i]=j;
    A_BUF4[i]=j;
    }
    j++;
  }
}
```

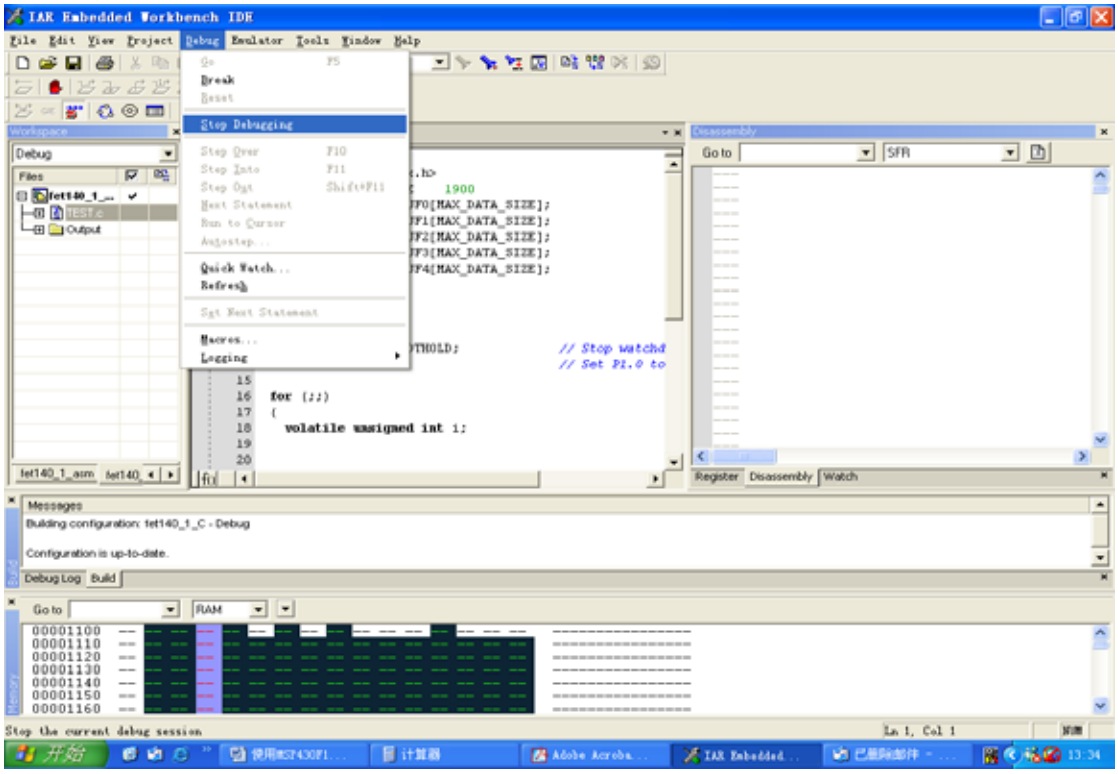
在测试程序中设置了五个 unsigned char 数组，全局变量，大小为 1900，共占用 RAM 资源 9500 个字节。

将测试程序编译下载到单片机，会出现如下图的现象

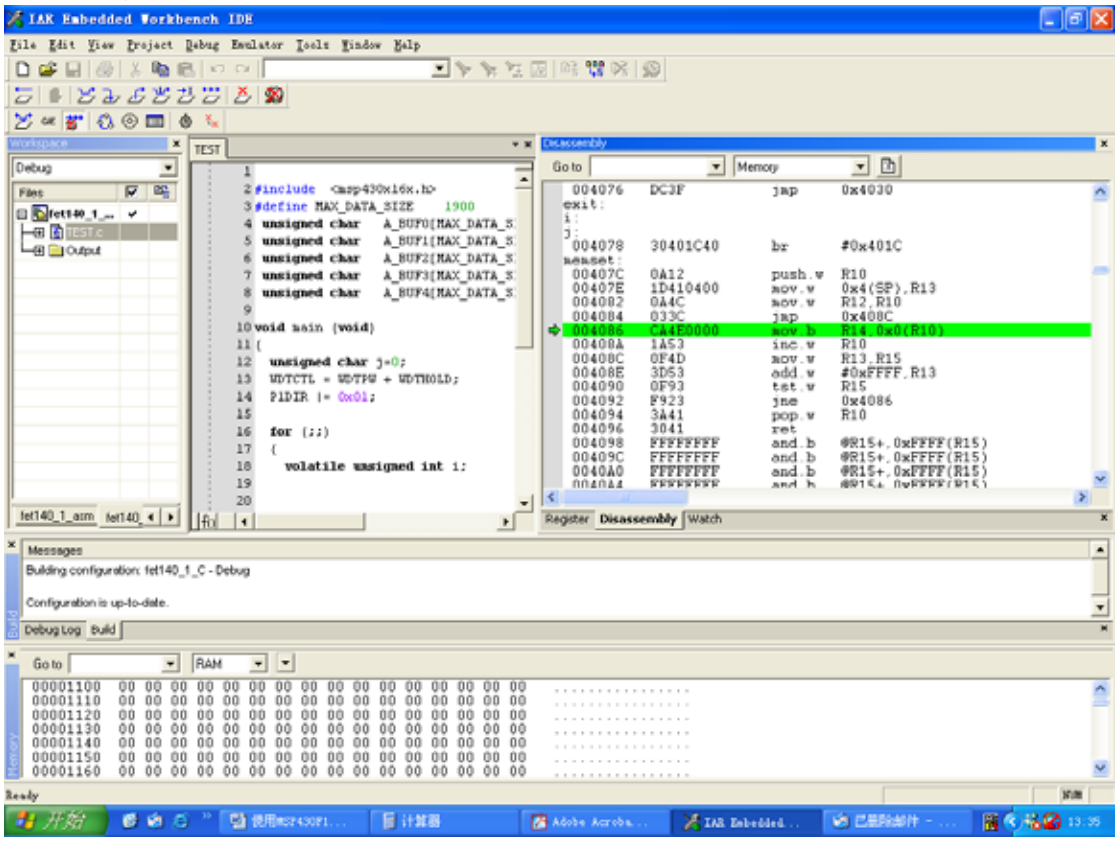


似乎单片机处于 RUN 状态，且进入不了用户编写的 MAIN 函数。

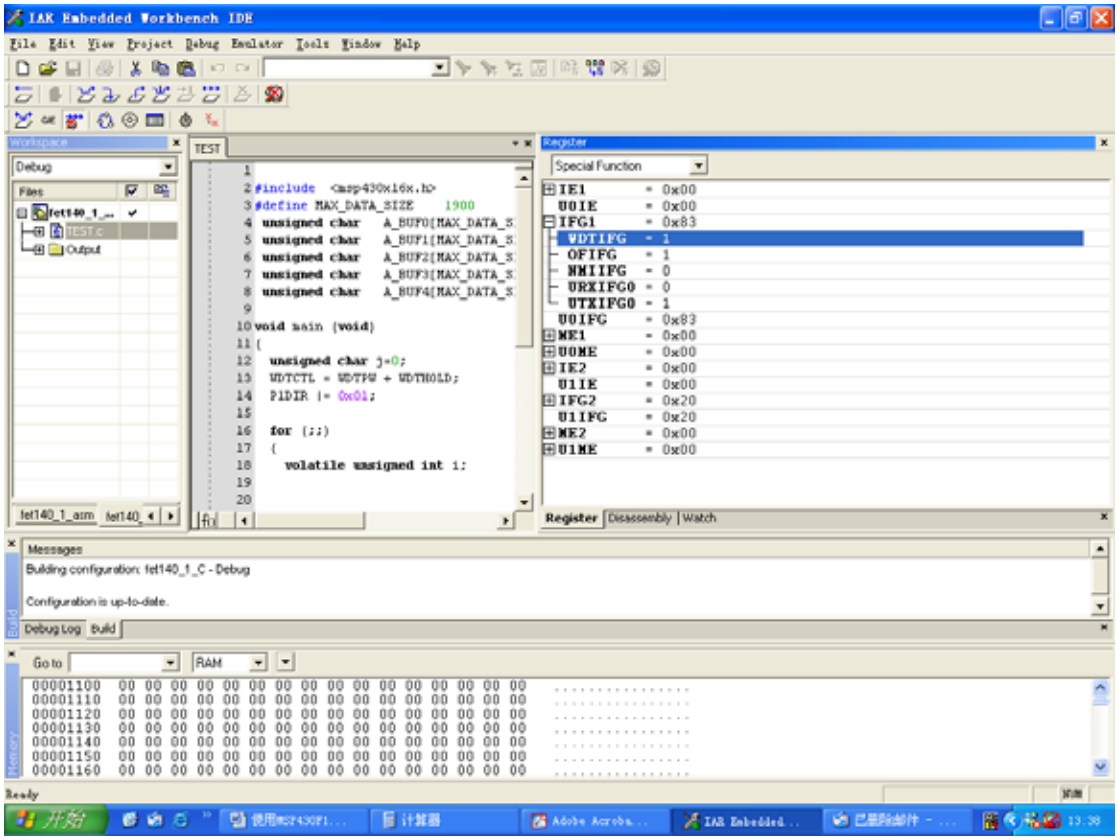
点击 Debug->Stop Debuging



在 Disassembly 窗口中发现程序停留在 MEMSET 子程序中



在查看 Register 发现 IFG0 中的 WDTIFG 被置位。



为什么 WDTIFG 会被置位呢？

众所周知，IAR C 编译器在编译链接程序时，会自动添加一些初始化的程序，在 IAR 环境中称为 Cstartup.s43，可以从这个初始化程序中找到原因。在此例程中，添加的 Cstartup.s43 程序可以从 Disassembly 窗口中获取，程序如下：

```
__program_start:
  004000     31400039          mov.w     #0x3900,SP
?cstart_init_zero:
  004004     3C400011          mov.w     #0x1100,R12
  004008     0E43              clr.w     R14
  00400A     30121C25          push.w    #0x251C
  00400E     B0127C40          call      #memset
  004012     2153              incd.w    SP
?cstart_call_main:
  004014     B0122240          call      #main
  004018     B0127840          call      #exit
_exit:
  00401C     30402040          br        #0x4020
?C_EXIT:
__exit:
  004020     FF3F              jmp       ?C_EXIT
main:
  004022     2183              decd.w    SP
  004024     4E43              clr.b     R14
  004026     B240805A2001      mov.w     #0x5A80,&WDTCTL
  00402C     D2D32200          bis.b     #0x1,&P1DIR
  004030     D2E32100          xor.b     #0x1,&P1OUT
  004034     B14050C30000      mov.w     #0xC350,0x0(SP)
  00403A     B1530000          add.w     #0xFFFF,0x0(SP)
  00403E     81930000          tst.w     0x0(SP)
  004042     FB23              jne       0x403A
  004044     81430000          clr.w     0x0(SP)
  004048     B1906C070000      cmp.w     #0x76C,0x0(SP)
  00404E     122C              jc        0x4074
  004050     2F41              mov.w     @SP,R15
  004052     CF4E0011          mov.b     R14,0x1100(R15)
  004056     2F41              mov.w     @SP,R15
  004058     CF4E6C18          mov.b     R14,0x186C(R15)
  00405C     2F41              mov.w     @SP,R15
  00405E     CF4ED81F          mov.b     R14,0x1FD8(R15)
  004062     2F41              mov.w     @SP,R15
  004064     CF4E4427          mov.b     R14,0x2744(R15)
  004068     2F41              mov.w     @SP,R15
  00406A     CF4EB02E          mov.b     R14,0x2EB0(R15)
  00406E     91530000          inc.w     0x0(SP)
```

```
 004072     EA3F            jmp       0x4048
 004074     5E53            inc.b     R14
 004076     DC3F            jmp       0x4030
exit:
i:
j:
 004078     30401C40        br        #0x401C
memset:
 00407C     0A12            push.w    R10
 00407E     1D410400        mov.w     0x4(SP),R13
 004082     0A4C            mov.w     R12,R10
 004084     033C            jmp       0x408C
 004086     CA4E0000        mov.b     R14,0x0(R10)
 00408A     1A53            inc.w     R10
 00408C     0F4D            mov.w     R13,R15
 00408E     3D53            add.w     #0xFFFF,R13
 004090     0F93            tst.w     R15
 004092     F923            jne       0x4086
 004094     3A41            pop.w     R10
 004096     3041            ret
```

从上面的反汇编程序中可知，单片机在上电复位（PUC）或复位（POR）后，程序是从__program_start 处开始的。需要进行 RAM 区域的初始化后才进入用户的 MAIN()函数。而 MEMSET（）正是 RAM 清零的函数。在 MENSET（）函数中，R10 存取 RAM 的地址，R13 存取需清零的 RAM 数量。在这个测试程序中，R10 的初始值为 0x1100，R13 的初始值为 0x251c(9500),为定义的全局变量的总数。

再分析 MEMSET（）函数，可以发现对一个字节的 RAM 单元进行清零操作需要占用十个时钟周期的时间。

最后在仔细的观察从上电复位后的__program_start 到进入用户函数 MAIN，这中间都没有对看门狗进行操作的语句，这也就意味着上电复位 PUC 后看门狗是出于默认状态的。而参考 430 的用户指南可以得知看门狗在上电复位后是处于看门狗状态，定时时间为 SMCLK/32768，也就是说 32768 个时钟周期。再结合上电后时钟系统的默认状态 MCLK=SMCLK，就可以得出结论，Cstartup.s43（）函数的运行时间不得大于 32768 个时钟周期，否则系统会被自动复位。这个是为什么 TI 推荐用户函数的第一个操作时看门狗操作的原因。

在测试程序中系统需初始化的变量为 9500 个，需要的时钟周期为 95000〉〉32768，所以会出现上面的现象。

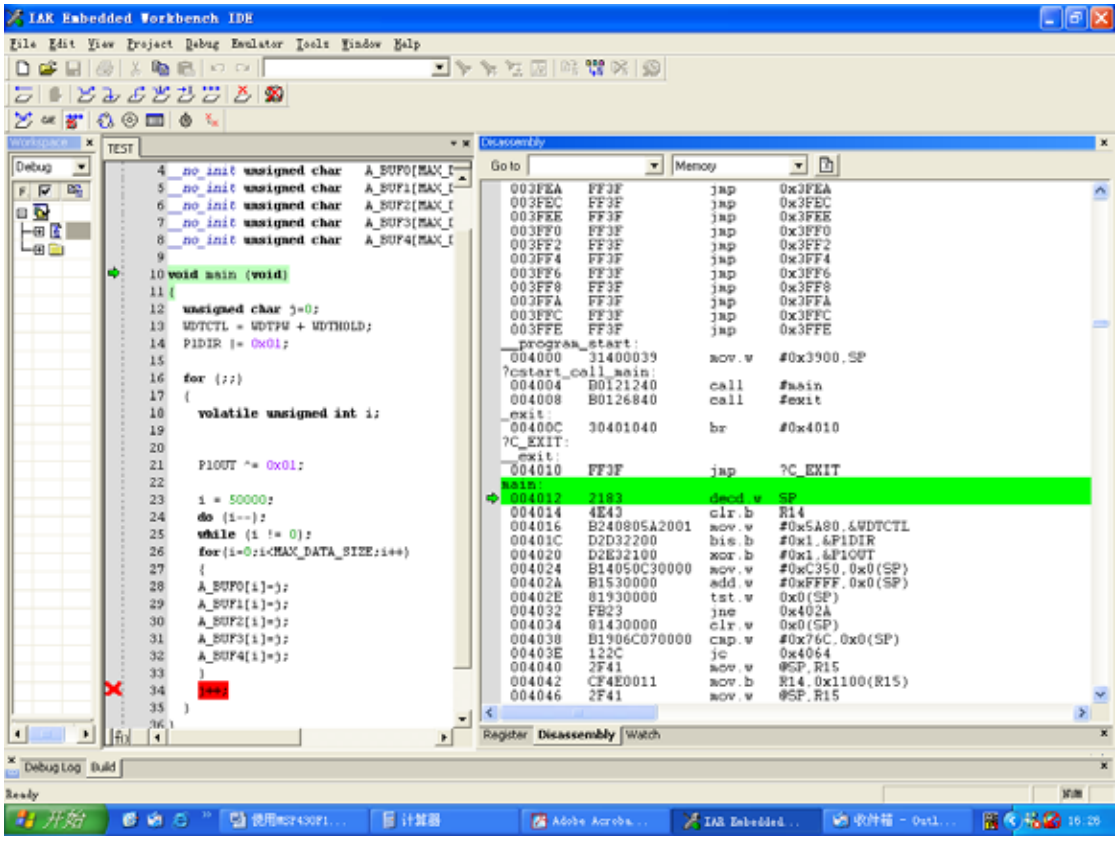
根据上面分析的原理，得出了两种解决方法。

方法一 将一些变量定义成 **no init** 类型，保证 cstartup 的初始化变量数目小于 3270 个字节

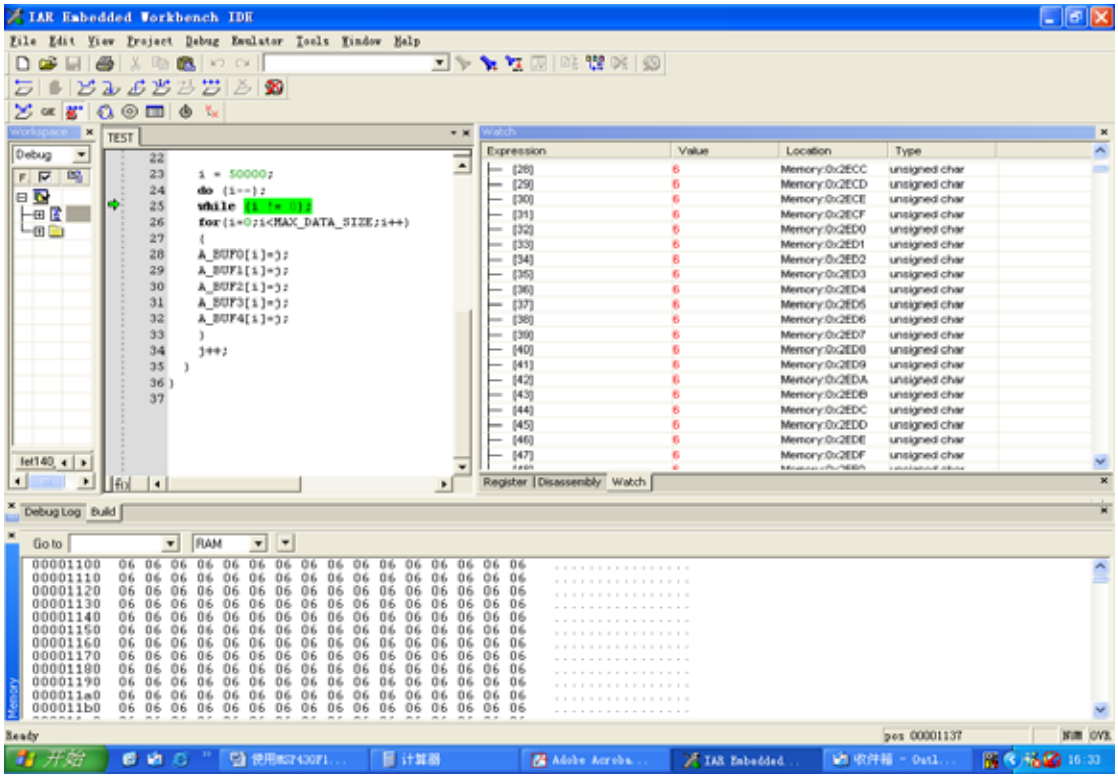
在测试程序中的修改如下：

```
__no_init unsigned char     A_BUF0[MAX_DATA_SIZE];
__no_init unsigned char     A_BUF1[MAX_DATA_SIZE];
__no_init unsigned char     A_BUF2[MAX_DATA_SIZE];
__no_init unsigned char     A_BUF3[MAX_DATA_SIZE];
__no_init unsigned char     A_BUF4[MAX_DATA_SIZE];
```

编译链接 DEBUG 后结果如下：



可以从Disassembly中清楚地看到在__program_start()函数中已经没有了RAM初始化程序MEMSET()，从__program_start 到用户的 MAIN 函数的时间<<32768 个时钟周期，所以能正常地回到用户程序 MAIN；执行程序，可以看到 RAM 中定义的变量在改变，适配器灯在闪烁。



如果需要对 RAM 进行初始化，可以在用户程序中进行。

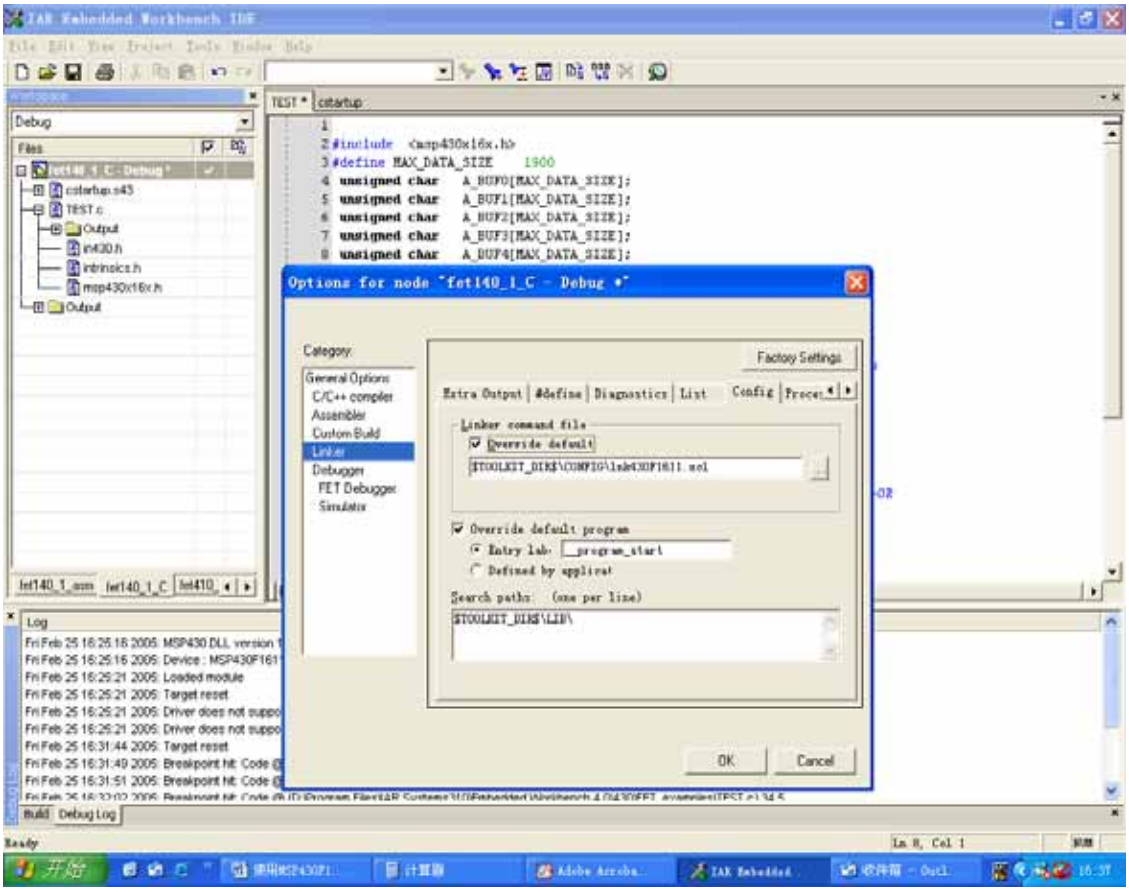
方法二 修改 IAR 的 cstartup.s43 程序，具体方法如下：(iar310a 为例)

1 将 cstartup.s43 程序加载到用户自己的项目中，cstartup.s43 在 iar310a 的路径如下 $TOOLKIT_DIR$\src\LIB\

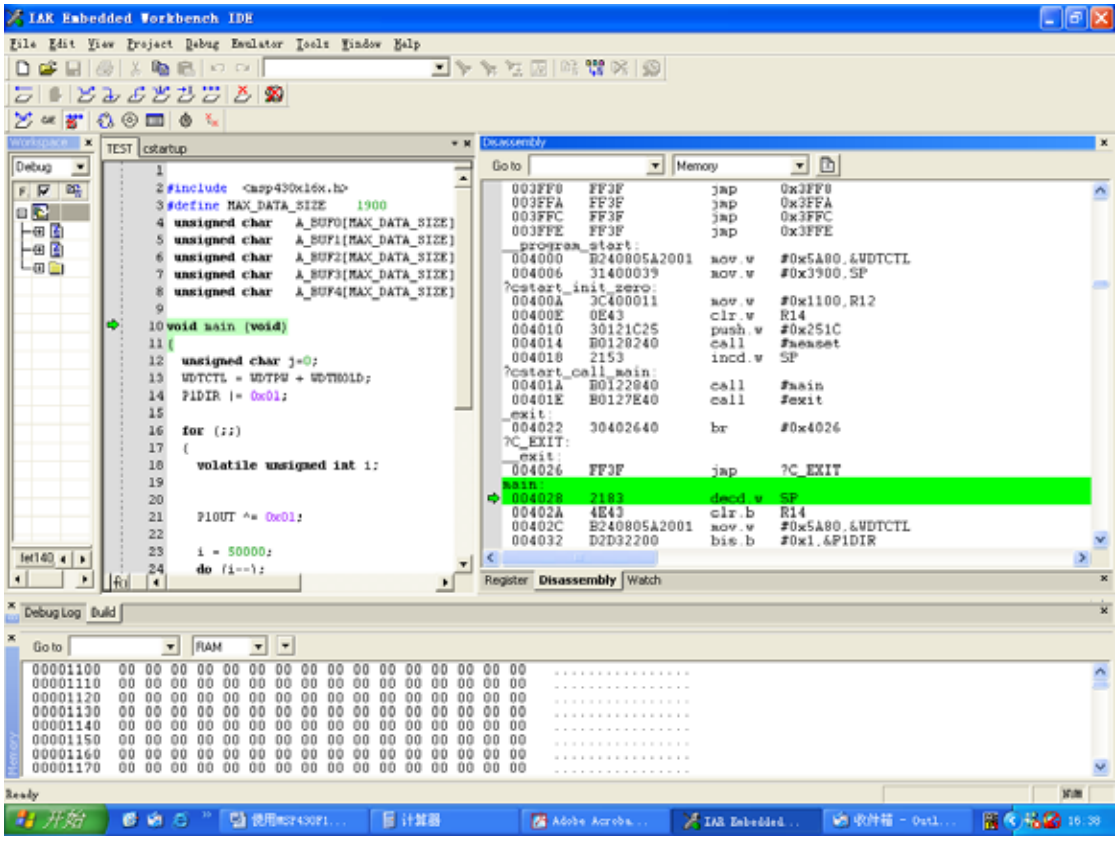
2 修改 cstartup.s43 中的 __program_start 子程序，加入关闭看门狗的命令 MOV #0x5A80,&0x0120，修改后如下：

```
__program_start:
//
// Initialize SP to point to the top of
the stack.
//
MOV #0x5A80,&0x0120
MOV #SFE(CSTACK), SP
//
// Ensure that main is
called.
//
REQUIRE ?cstart_call_main
```

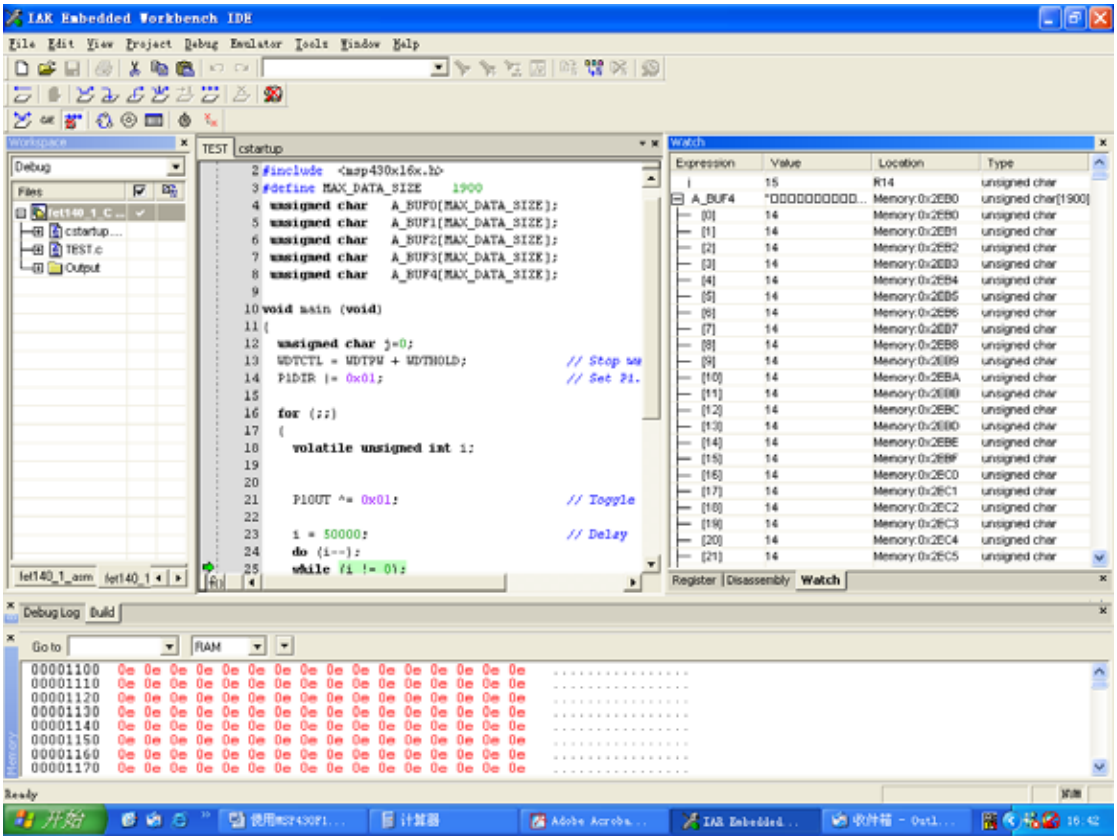
3 在 **Project->Options->Linker->Config** 页中选择 **Override default programe** ,并将 **Entry lib** 设置成 **__program_start**



然后就可以编译链接了。结果如下：



可以清楚地从 Disassembly 窗口中看到在__program_start 程序中增加了关闭看门狗的语句。程序能够正常地运行。



总结：

在 MSP430F1611 中不能使用大变量的原因是因为 IAR 的 C 编译器引起的，而不是 430 的原因。

IAR 编译器在对 C 语言编译时会在用户程序的前面加入 RAM 的初始化的程序 cstartup.s43。执行完这段程序才会执行用户的程序。这段程序中有一个 MENSET 子程序是对用户所用到的 RAM 空间进行清零操作的。每请一字节 RAM，需占用 10 个时钟周期的时间。

在 cstartup.s43 程序中没有对看门狗进行任何操作，而上电复位(PUC)后 WDT 默认的是看门狗状态，定时时间为 SMCLK/32768，也就是 32768 个时钟周期。当需初始化 RAM 的数量大于 32768/10=3270 时，430 还没有完成 cstartup 程序就被看门狗复位了，导致 430 进入不了用户自己的程序即 main 函数.