TKS-B系列专业仿真器“复杂断点”的使用技巧

（2004/8/18 V1.0）

本章节主要阐述了在Keil IDE（μVision2/μVision3）集成开发环境下，TKS-B系列专业仿真器中“复杂断点”的特点和使用方法。

1. 断点的概念

断点是仿真器中经常使用的一个概念。它的基本含义是，当程序运行到其当前状态满足断点要求的位置时，仿真器将执行指定操作（一般是程序停止运行）。最简单的断点是程序断点，即程序运行到断点指定的位置时停止运行。

断点是仿真器的重要功能。通过断点我们可以控制仿真器在指定的位置停止运行，然后就可以分析当前的运行状态，判断程序中可能存在的问题或调试整个系统的硬件。

断点的种类有很多，大体分为简单断点和复杂断点两种。不同的仿真器提供的断点种类也不同，一般都能提供简单的程序断点。高档仿真器一般能提供更多的断点种类，例如数据读/写断点，代码读取断点和组合断点等。

2. TKS-B中的特点

B系列仿真器具备4×64K用户实时断点，用户可以根据自己的需要设置不同的断点来调试程序。

- 64K程序运行断点：程序运行到该位置时停止运行。
- 64K代码读取断点：MOVC指令读取该地址数据时，程序在完成操作后停止运行。
- 64K数据读取断点：MOVX指令读取该地址数据时，程序在完成操作后停止运行。
- 64K数据写入断点：MOVX指令写入该地址数据时，程序在完成操作后停止运行。

用户可以单独或混合使用上述4种断点，每种断点的使用个数最大为64K。

3. 断点的使用方法

- 简单程序断点的设置

程序运行断点有非常简单的定义方法：在程序窗口（包括C语言、汇编和反汇编窗口）中，双击想要设置断点的程序行，则在窗口左边的状态条中出现红色的断点标志块；再次双击该程序行，则为取消断点，之后窗口左边的红色的断点标志块消失。

注：在源程序窗口中，使用这种方法有时不能在某些程序行中设置断点。出现这种情况一般是因为该程序没有被编译或者由于语言优化而没有产生最终的程序代码，在这种情况下不可能在该源程序行设置断点。

用户也可以使用断点菜单和断点工具条：在程序窗口（包括C语言、汇编和反汇编窗口）中，使用鼠标或键盘上的上移/下移键将光标移动到想要设置的程序行，点击主菜单“Debug→Insert/Remove Breakpoint”或快捷图标来插入/删除一个断点；对于已经定义的断点，可以点击主菜单“Debug→Enable/Disable Breakpoint”或快捷图标来启动/关闭一个断点。

注：断点的操作有插入断点/关闭断点/删除断点三种，其中关闭断点和删除断点是不同的。删除断点是把该断点取消，当需要时需再次设置；关闭断点是对已经插入的断点暂时关闭，可以在以后再次简单地启动。

- 复杂断点的设置

由于程序运行断点直接和程序相对应，用户可以使用简单的方法来设置。对于B系列中的其它三种断点，用户需要启动**断点管理器**来进行断点的处理。在硬件仿真环境中点击主菜单“Debug→Breakpoints”打开断点管理器，如图1所示。

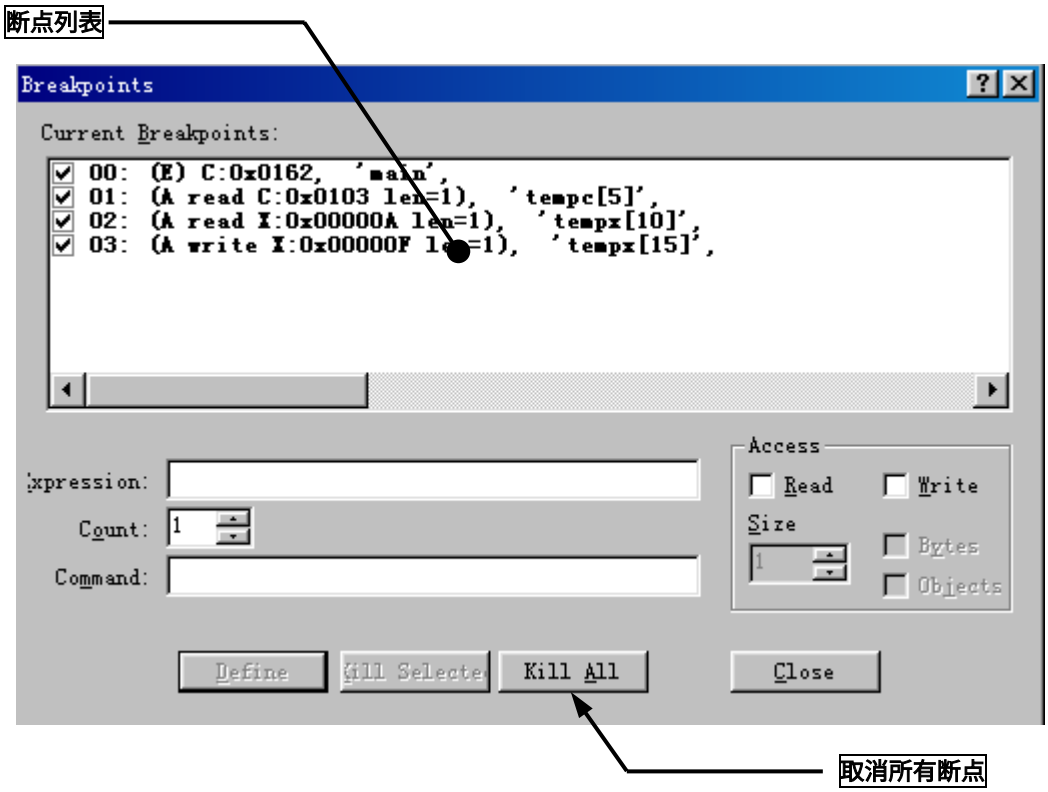

图 1 断点管理窗口

断点管理器主要有下面的功能部件：

Current Breakpoints：	当前所有断点的显示窗口。
Expression：	断点的表达式。
Count：	触发断点的次数（B系列不支持）。
Command：	断点命令输入（B系列不支持）。
Read：	断点操作属性为读。
Write：	断点操作属性为写。
Size：	断点的尺寸以Byte为单位，B系列只支持8bits尺寸（Size=1）。

Byte： B系列不支持选择。
Objects： B系列不支持选择。
Define： 定义一个断点。
Kill Selected： 删除选择的断点。
Kill All： 删除全部的断点。

下面举一个例子具体说明“复杂断点”的使用方法。

以下的例程“duandian.c”通过简单的赋值语句演示了“复杂断点”功能。使用TKS-58B对其进行硬件仿真，在硬件仿真环境中设置断点，掌握“复杂断点”的使用方法。

```
/*****************************************************************
 函数：duandian.c
 功能：通过简单的赋值语句演示“复杂断点”功能
 说明：在硬件仿真环境中设置断点，掌握“复杂断点”的使用方法
*****************************************************************/
#include"reg51.h"

code char tempc[100] = {0x00, 0x01, 0x02, 0x03, 0x04,};
xdata char tempx[100] = {0x10, 0x11, 0x12, 0x13, 0x14,};

unsigned char cnt = 0;
unsigned char dat = 0;

void main(void)
  {
      while(1)
        {
            dat = tempc[cnt];
            dat++;                          // 无意义，用于间隔不同操作
            tempx[cnt] = dat;
            dat++;                          // 无意义，用于间隔不同操作
            dat = tempx[cnt];
            dat++;                          // 无意义，用于间隔不同操作

            if( cnt>100 ) cnt = 0;
            else cnt++;
        }
    }
```

定义程序运行断点：

方法一：在“Expression”中写入函数名称。例如，在本例程中，在“Expression”中写入“main”；“Access”中的“Read”和“Write”属性均不选择，如图2所示。点击“Define”按钮，则可以在main函数入口位置添加断点。设置完毕后，可以观察到源程序（duandian.c）窗口以及反汇编窗口中对应的程序行出现程序断点标志（红色方块）。

使用函数名来定义断点的优点是不用输入断点的直接地址，方便用户记忆。即使重新编译后函数地址发生变化，调试器能自动重新调整该断点的位置。

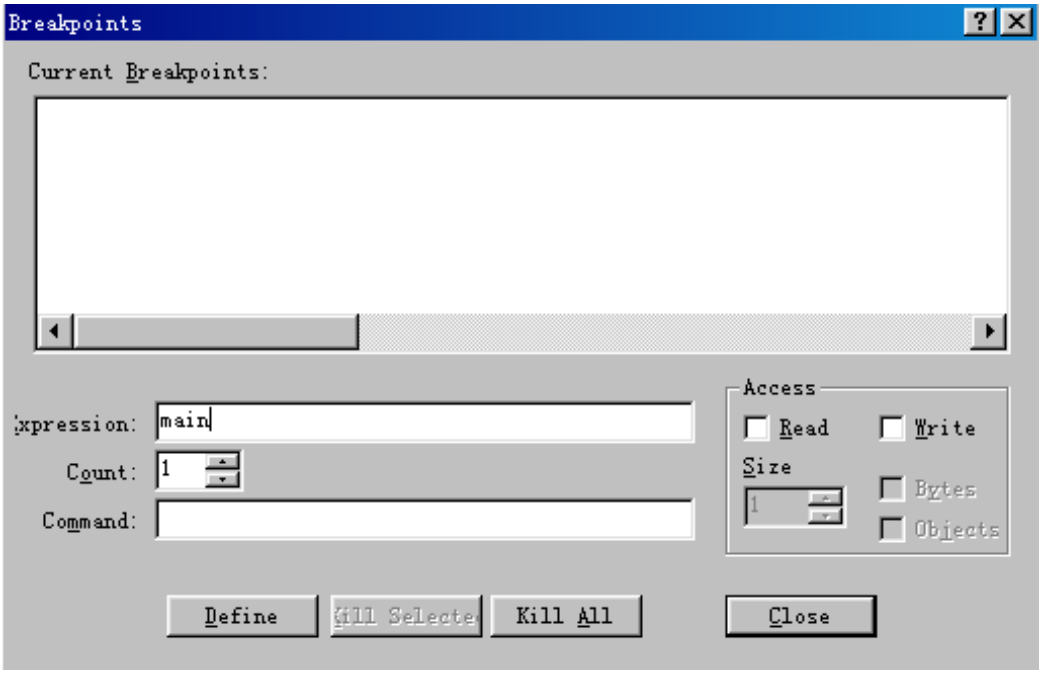

图2 程序运行断点定义窗口

方法二：在“Expression”中写入“C:0x0162”（C表示是程序属性，0x0162为main函数的程序代码地址，不同的编译环境可能数值有变化），其他设置同上。然后点击“Define”按钮，同样可以在main函数处设置一个断点。**注：**使用这种直接地址定义断点的方法时，用户必须知道程序代码的确切地址。

定义MOVC代码读取断点：

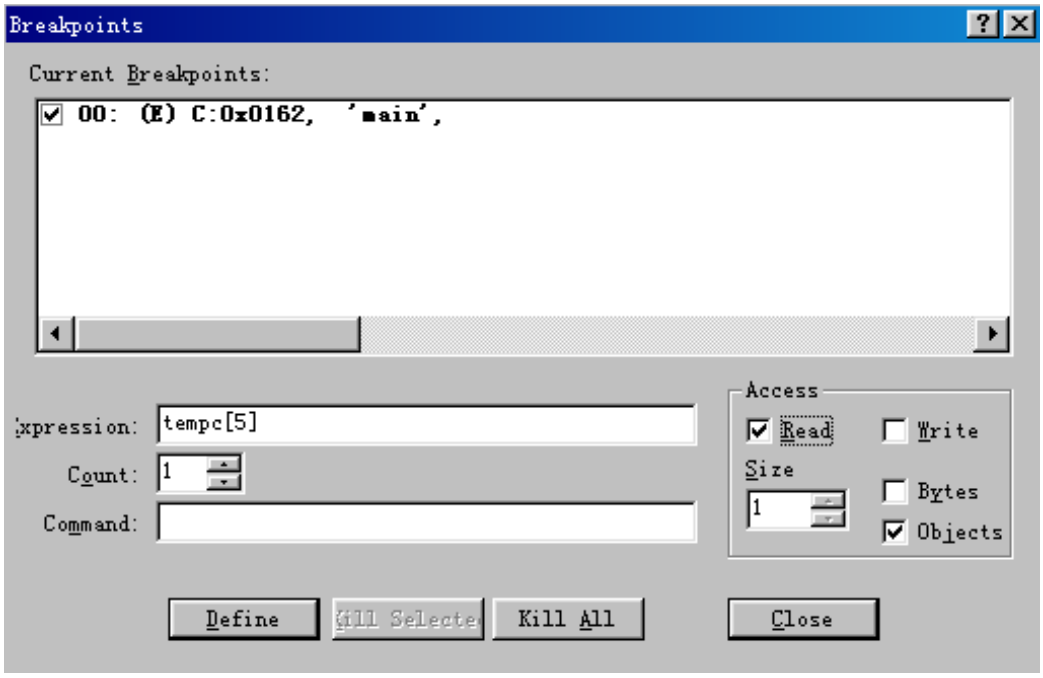

图3 MOVC读取断点定义窗口

方法一：在“Expression”中输入变量名称。例如，在本例程中定义了一个“code char tempc[100]”的数组，则可以如下设置：

1）在“Expression”写入“tempc[5]”；

2）选中“Access”中的“Read” 属性，而“Write”不选中；

3）“Size”选择为“1”，若选择其它数值仍将按1处理。

如图3所示。按动“Define”按钮，可以在tempc[5]位置添加一个MOVC读取断点，即在MOVC指令读取tempc[5]后中断程序的运行。

方法二：在“Expression”中写入“C:0x0103”（C表示是程序属性，0x0103为tempc[5]的程序代码地址），其他设置同上。然后点击“Define”按钮，同样可以在tempc[5]位置添加一个MOVC读取断点。**注：**使用这种直接地址定义断点的方法时，用户必须知道变量的确切地址。

注意：在定义MOVC读取断点时，“Read”属性一定要选中，否则将成为程序运行断点。“Write”不能选中，因为程序代码无法进行写操作。

定义MOVX数据读取断点：

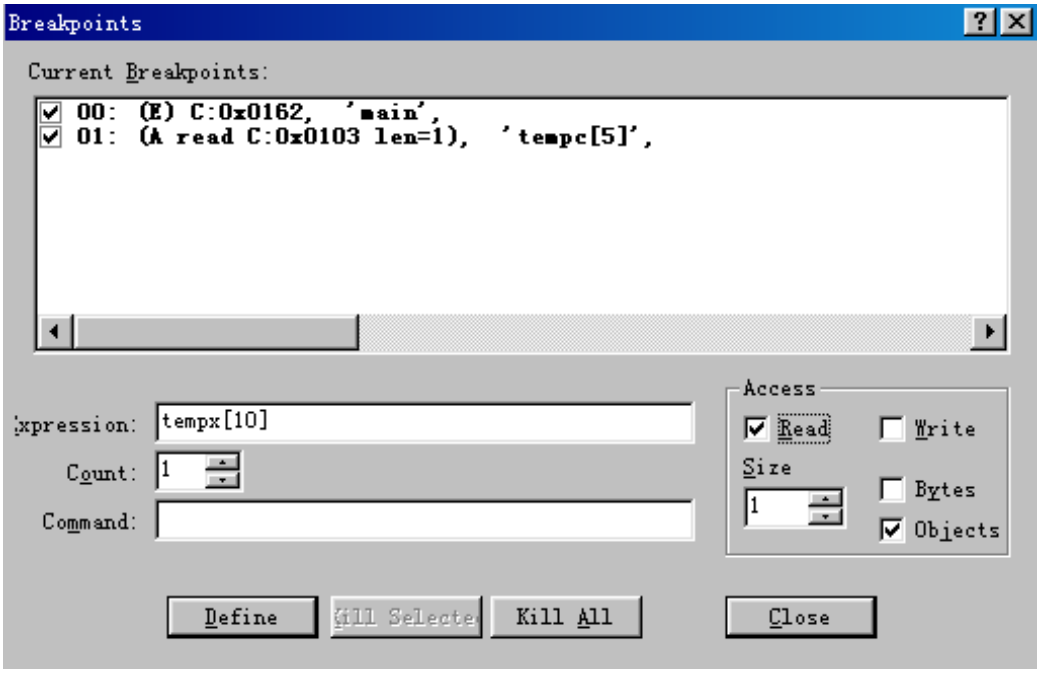

图4 MOVX读取断点定义窗口

方法一：在“Expression”中输入外部数据变量名称。例如在本例程中，定义了一个“xdata char tempx[100]”的外部数据空间数组，则可以如下设置：

1）在“Expression”写入“tempx[10]”；

2）选中“Access”中的“Read” 属性，而“Write”不选中；

3）“Size”选择为“1”，若选择其它数值仍将按1处理。

如图4所示。按动“Define”按钮，可以在tempx[10]添加一个MOVX读取断点，即在MOVX指令读取tempx[10]后设置一个断点。

方法二：在“Expression”中写入例如“X:0x00000A”的表达式（X表示是外部数据空间属性，0x00000A为tempx[10]的数据地址），其他设置同上。然后点击“Define”按钮，同样可以在tempx[10]位置添加一个MOVX读取断点。**注：**使用这种直接地址定义断点的方法时，用户必须知道数据变量被定义存放的确切地址。

注意：在定义MOVX读取/写入断点时，“Read/Write”至少必须有一项选中，否则将不能形成有效的MOVX操作断点；如果“Read/Write”全部选中，则该断点在读取/写入任何操作发生时都会形成有效的断点。

定义MOVX数据写入断点：

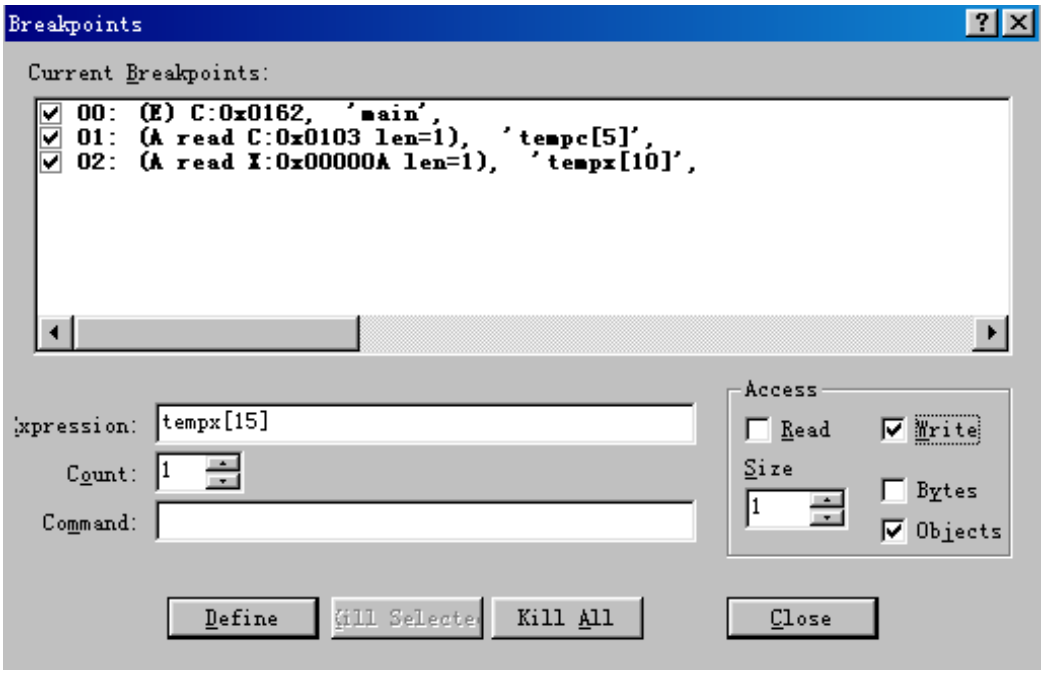

图5 MOVX写入断点定义窗口

方法一：在“Expression”中输入外部数据变量名称。例如在本例程中，定义了一个“xdata char tempx[100]”的外部数据空间数组，则可以如下设置：

1）在“Expression”写入“tempx[15]”；

2）选中“Access”中的“Write” 属性，而“Read”不选中；

3）“Size”选择为“1”，若选择其它数值仍将按1处理。

如图5所示。按动“Define”按钮，可以在tempx[15]添加一个MOVX写入断点，即在MOVX指令写入tempx[15]后设置一个断点。

方法二：在“Expression”中写入例如“X:0x00000F”的表达式（X表示是外部数据空间属性，0x00000F为tempx[15]的数据地址），其他设置同上。然后点击“Define”按钮，同样可以在tempx[15]位置添加一个MOVX写入断点。**注：**使用这种直接地址定义断点的方法时，用户必须知道数据变量被定义存放的确切地址。

观察运行停止后断点的位置：

设置完上述断点后，全速运行，在**反汇编窗口**中，可以看到程序分别在运行到以下位置停止运行：

main函数入口处（反汇编指令为“C:0x0162 E509 MOV A,cnt(0x09) ”）；
MOVC读取tempc[5]（反汇编指令为“C:0x0167 93 MOVC A,@A+DPTR”）后；
MOVX读取tempx[10]（反汇编指令为“C:0x0187 E0 MOVX A, @DPTR”）后；
MOVX写入tempx[15]（反汇编指令为“C:0x0179 F0 MOVX @DPTR,A”）后。

关闭已经存在的断点：

对于“Current Breakpoints”窗口中存在的每个断点，用户都可以暂时关闭，使其断点作用失效。方法是点击断点表达式前面的打勾框，使之取消打勾。由于该失效的断点仍然存在于“Current Breakpoints”窗口中，因此用户可以在以后很方便重新起用它，这比删除该断点然后再重新增加一个断点要方便得多。

4. 局限性

在断点管理器中，B系列仿真器不支持以下的断点设置：

Count： 触发断点的次数（B系列不支持）。
Command： 断点命令输入（B系列不支持）。
Size： 断点的尺寸，以Byte为单位。B系列只支持8bits尺寸（Size=1）。
Byte： B系列不支持选择。
Objects： B系列不支持选择。