Keil 的辅助工具和部份高级技巧

在前面的几讲中我们介绍了工程的建立方法，常用的调试方法，除此之外，Keil 还提供了一些辅助工具如外围接口、性能分析、变量来源分析、代码作用分析等，帮助我们了解程的性能、查找程序中的隐藏错误，快速查看程序变量名信息等，这一讲中将对这些功工具作一介绍，另外还将介绍 Keil 的部份高级调试技巧。

一、 辅助工具

这部份功能并不是直接用来进行程序调试的，但可以帮助我们进行程序的调试、程序性能的分析，同样是一些很有用的工具。

1、外围接口

为了能够比较直观地了解单片机中定时器、中断、并行端口、串行端口等常用外设的使用情况，Keil 提供了一些外围接口对话框，通过 Peripherals 菜单选择，该菜单的下拉菜单内容与你建立项目时所选的 CPU 有关，如果是选择的 89C51 这一类“ 标准 ”的 51 机，那么将会有 Interrupt（中断）、I/O Ports（并行 I/O 口）、Serial（串行口）、Timer（定时/计数器）这四个外围设备菜单。打开这些对话框，列出了外围设备的当前使用情况，各标志位的情况等，可以在这些对话框中直观地观察和更改各外围设备的运行情况。

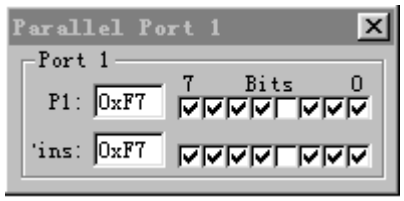

图 1 外围设备之并行端口

下面我们通过一个简单例子看一看并行端口的外围设备对话框的使用。例 4：

```
        MOV     A,#0FEH
LOOP:   MOV     P1,A
        RL      A
        CALL    DELAY   ;延时 100 毫秒
        JMP     LOOP
```

其中延时 100 毫秒的子程序请自行编写。

编译、连接进入调试后，点击 Peripherals->I/O-Ports->Port 1 打开，如图 1 所示，全速运行，可以看到代表各位的勾在不断变化（如果看不到变化，请点击 View->Periodic Window Updata），这样可以形象地看出程序执行的结果。

注：如果你看到的变化极快，甚至看不太清楚，那么说明你的计算机性能好，模拟执行的速度快，你可以试着将加长延时程序的时间以放慢速度。模拟运行速度与实际运行的速度无法相同是软件模拟的一个固有弱点。

点击 Peripherals->I/O-Ports->Timer0 即出现图 2 所示定时/计数器 0 的外围接口界面，可以直接选择 Mode 组中的下拉列表以确定定时/计数工作方式，0-3 四种工作方式，

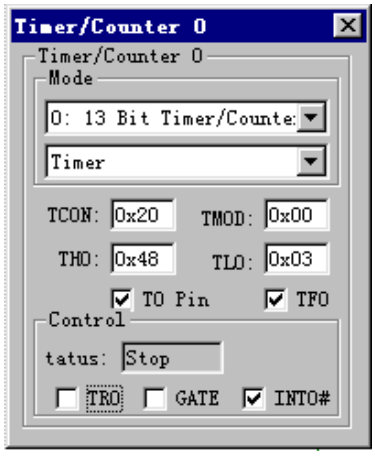

图 2 外围设备之定时器

设定定时初值等，点击选中 TR0，status 后的 stop 就变成了 run，如果全速运行程序，此时 th0,tl0 后的值也快速地开始变化（同样要求 Periodic Window Updata 处于选中状态），直观地演示了定时/计数器的工作情况（当然，由于你的程序未对此写任何代码，所以程序不会对此定时/计数器的工作进行处理）。

2、性能分析

Keil 提供了一个性能分析工具，利用该工具，我们可以了解程序中哪些部份的执行时间最长，调用次数最多，从而了解影响整个程序中执行速度的瓶颈。下面通过一个实例来看一看这个工具如何使用，例 5：

```
#include "reg51.h"
sbit P1_0=P1^0;     //定义 P1.0
void mDelay(unsigned char DelayTime)
{     unsigned int j=0;
      for(;DelayTime>0;DelayTime--)
      {    for(j=0;j<125;j++)    {;}     }
}
void mDelay1(unsigned char DelayTime)
{     unsigned int j=0;
      for(;DelayTime>0;DelayTime--)
      {    for(j=0;j<125;j++)    {;}     }
}
void main()
{     unsigned int i;
      for(;;){    mDelay(10);   // 延 时  10 毫秒
            i++;
            if(i==10)
            {       P1_0=!P1_0;
                  i=0;
                  mDelay1(10);}
 } }
```

编译连接。进入调试状态后使用菜单 View->Performance Analyzer Window，打开性能分析对话框，进入该对话框后，只有一项 unspecified，点鼠标右键，在快捷菜单中选择 Setup PA 即打开性能分析设置对话框，对于 C 语言程序，该对话框右侧的“ Function Symbol ”下的列表框给出函数符号，双击某一符号，该符号即出现在 Define Performance Analyzer 下的编缉框中，每输入一个符号名字，点击 Define 按钮，即将该函数加入其上的分析列表框。对于汇编语言源程序，Function Symbol 下的列表框中不会出现子程序名，可以直接在编缉框中输入子程序名，点击 Close 关闭窗口，回到性能分析窗口，此时窗口共有 4 个选项。全速执行程序，可以看到 mDelay 和 mDelay1 后出现一个蓝色指示条，配合上面的标尺可以直观地看出每个函数占整个执行时间的比例，点击相应的函数名，可以在该窗口的状态栏看到更详细的数据，其中各项的含义如下：

Min：该段程序执行所需的最短时间；Max：该段程序执行所需的最长时间；Avg：该段程序执行所花平均时间；Total：该段程序到目前为目总共执行的时间；%：占整个执行时间的百分比；count：被调用的次数。

本程序中，函数 mDelay 和 mDelay1 每次被调用都花费同样的时间，看不出 Min、Max、和 Avg 的意义，实际上，由于条件的变化，某些函数执行的时间不一定是一个固定的值，借助于这些信息，可以对程序有更详细的了解。下面将 mDelay1 函数略作修改作一演示。

```
void mDelay1(unsigned char DelayTime)
{     static unsigned char k;
      unsigned int j=0;
      for(;DelayTime>0;DelayTime--)
      {    for(;j<k;j++)
            {;}
      }     k++; }
```

程序中定义了一个静态变量 K，每次调用该变量加 1，而 j 的循环条件与 k 的大小有关，

这使每次执行该程序所花的时间不一样。编译、执行该程序，再次观察性能分析窗口，可以看出 Min、Max、Avg 的意义。

3、变量来源浏览

该窗口用于观察程序中变量名的有关信息，如该变量名在那一个函数中被定义、在哪里被调用，共出现多少次等。在 Source Browse 窗口中提供了完善的管理方法，如过滤器可以分门别类地列出各种类别的变量名，可以对这些变量按 Class（组）、Type（类型）、Space（所在空间）、Use（调用次数）排序，点击变量名，可以在窗口的右侧看到该变量名的更详细的信息。

4、代码作用范围分析

在你写的程序中，有些代码可能永远不会被执行到（这是无效的代码），也有一些代码必须在满足一定条件后才能被执行到，借助于代码范围分析工具，可以快速地了解代码的执行情况。

进入调试后，全速运行，然后按停止按钮，停下来后，可以看到在源程序的左列有三种颜色，灰、淡灰和绿，其中淡灰所指的行并不是可执行代码，如变量或函数定义、注释行等等，而灰色行是可执行但从未执行过的代码，而绿色则是已执行过的程序行。使用调试工具条上的 Code Coverage Window 可打开代码作用范围分析的对话框，里面有各个模块代码执行情况的更详细的分析。如果你发现全速运行后有一些未被执行到的代码，那么就要仔细分析，这些代码究竟是无效的代码还是因为条件没有满足而没有被执行到。

二、部份高级调试技巧

Keil 内置了一套调试语言，很多高级调试技巧与此有关，但是全面学习这套语言并不现实，这不是这么几期连载可以胜任的，这里仅介绍部份较为实用的功能，如要获得更详细的信息，请参考 Keil 自带的帮助文件 GS51.PDF。

1、串行窗口与实际硬件相连

Keil 的串行窗口除可以模拟串行口的输入和输出功能外还可以与 PC 机上实际的串口相连，接受串口输入的内容，并将输出送到串口。这需要在 Keil 中进行设置。方法是首先在输出窗口的 Command 页用 MODE 命令设置串口的工作方式，然后用 ASSIGN 命令将串行窗口与实际的串口相关联，下面我们通过一个实例来说明如何操作。例 6：

```
ORG     0000H
    JMP START
    ORG     3+4*8    ;串行中断入口
    JMP SER_INT
START:
    MOV     SP,#5FH   ;堆栈初始化
    CALL    SER_INIT ;串行口初始化 A
    SETB    EA  ;
    SETB    ES  ;
    JMP $    ;主程序到此结束
SER_INT:
    JBC     RI,NEXT  ;如果串口接收到字符，转
    JMP     SEND     ;否则转发送处理
NEXT:
    MOV    A,SBUF   ;从 SBUF 中取字符
```

```
    MOV   SBUF,A   ;回送到发送 SBUF 中
    JMP   OVER
SEND:
    clr   ti
OVER:
    reti

SER_INIT:              ;中断初始化
     MOV       SCON,#50H
     ORL       TMOD,#20H
     ORL       PCON,#80H
     MOV        TH1,#0FDH ;设定波特率
     SETB      TR1    ;定时器 1 开始运行
     SETB      REN    ;允许接收
     SETB      SM2
     RET
                        END
```

这个程序使用了中断方式编写串行口输入/输出程序，它的功能是将接串行口收到的字符回送，即再通过串行口发送出去。

正确输入源文件、建立工程、编译连接没有错后，可进行调试，使用 Keil 自带的串行窗口测试功能是否正确，如果正确，可以进行下一步的连机试验。

为简单实用，我们不借助于其它的硬件，而是让 PC 机上的两个串口互换数据，即 COM1 发送 COM2 接收，而 COM2 发送则由 COM1 接收，为此，需要做一根连接线将这两个串口连起来，做法很简单，找两个可以插入 PC 机串口的 DIN9 插座（母），然后用一根 3 芯线将它们连起来，连线的方法是：

2——3

3——2

5——5

接好线把两个插头分别插入 PC 机上的串口 1 与串口 2。找一个 PC 机上的串口终端调试软件，如串口精灵之类，运行该软件，设置好串口参数，其中串口选择 2，串口参数设置为：

19200，n，8，1 其含义是波特率为 19200，无奇偶校验，8 位数据，1 位停止位。

在 Keil 调试窗口的 command 页中输入：

```
>mode com1 19200,0,8,1
>assign com1 <sin>sout
```

注意两行最前面的“ > ”是提示符，不要输入，第二行中的“ < ”和“ > ”即“ 小于 ”和“ 大于 ”符号，中间的是字母“ s ”和“ input ”的前两个字母，最后是字母“ s ”和“ output ”的前三个字母。

第一行命令定义串口 1 的波特率为 19200，无奇偶校验，8 位数据，1 位停止位。第二行是将串口 1（com1）分配给串行窗口。

全速运行程序，然后切换串口精灵，开始发送，会看到发送后的数据会立即回显到窗口中，说明已接收到了发送过来的数据。切换到 uVison，查看串行窗口 1，会看到这里的确接收到了串口精灵送来的内容。

2、从端口送入信号

程序调试中如果需要有信号输入，比如数据采集类程序，需要从外界获得数据，由于 Keil 的调试完全是一个软件调试工具，没有硬件与之相连，所以不可能直接获得数据，为此必须采用一些替代的方法，例如，某电路用 P1 口作为数据采集口，那么可以使用的一种方法是利用外围接口，打开 PORT 1，用鼠标在点击相应端口位，使其变为高电平或低电平，就能输入数据。显然，这种方法对于要输获得数据而不是作位处理来说太麻烦了，另一种方法是直接在 command 页输入 port1=数值，以下是一个小小的验证程序。例 7：

```
LOOP:   MOV     A,P1
```

```
        JZ      NEXT
        MOV     R0,#55H
        JMP     LOOP
NEXT:   MOV     R0,#0AAH
        JMP     LOOP
        END
```

该程序从 P1 口获得数据，如果 P1 口的值是 0，那么就让 R0 的值为 0AAH，否则让 R0 的值为 55H。输入源程序并建立工程，进入调试后，在观察窗口加入 R0，然后全速运行程序，注意确保 View->Periodic Window Updata 处于选中状态，然后在 Command 后输入 PORT1=0 回车后可以发现观察窗口中的 R0 的值变成了 0AAH，然后再输入 PORT1=1 或其它非零值，则 R0 的值会变为 55H。

同样的道理，可以用 port0、port2、port3 分别向端口 0、2、3 输入信号。

3、直接更改内存值

在程序运行中，另一种输入数据的方法是直接更改相应的内存单元的值，例如，某数据采集程序，使用 30H 和 31H 作为存储单元，采入的数据由这两个单元保存，那么我们更改了 30H 和 31H 单元的值就相当于这个数据采集程序采集到了数据，这可以在内存窗口中直接修改（参考上一讲），也可以通过命令进行修改，命令的形式是： _WBYTE (地址,数据)，其中地址是指待写入内存单元的地址，而数据则是待写入该地址的数据。例如 _WBYTE(0x30,11)会将值 11 写入内存地址十六进制 30H 单元中。