文轩科技

单片机那些事儿

中级篇——模块化编程

残弈悟恩

2014

官方淘宝店铺：HTTP://SHOP109195762.TAOBAO.COM

郑重声明

本资料以残弈悟恩开发的 MGMC-V1.0 实验板为硬件平台，以残弈悟恩编写的《深入浅出玩转 51 单片机》为辅助教程，以残弈悟恩录制的《31 天环游单片机》为基础视频。

本资料以个人学习和工作的经验为素材，以单片机初学者、单片机项目开发者为对象。教大家如何走进单片机的世界，如何开发工程项目。限于时间和水平关系，资料中难免有过失之处，望各位高手批评指教，多多拍砖，拍累了，你们休息，我继续上路。

版本：20140506（V1.0）　　制造者：残弈悟恩

让爱充满大地——花 1 秒时间，拯救 1 个人，传递 1 份爱

声明：只是残弈悟恩爱心的喷发，我得不到一分钱，各位不要多想，谢谢！

你知道吗？在非洲北边的某个地区，每一秒都有许许多多的人正在挨饿，每一天至少有一位儿童死于营养不足。你的一次点击就能让某位穷人得到 1.1 杯食物。当然你可以不相信有这样的链接或者是骗点击什么的。事实上，网站确实是帮穷人得 1.1 杯食物的，只要你点进去单机一下中间的黄色按钮，就会出来一系列介绍各种商品的网页（绝对免费的并且不会下载任何软件，也不会有电脑病毒），同时也会有人因为您的一次点击而得到 1.1 杯食物，食物是由商家提拱的，但爱心却是您献出的。如果你觉得残弈悟恩在忽悠大家，你不妨可以在网上查一下是真与假。

看到这本资料的朋友多数都是电子爱好者、单片机初学者，或者干电子这一行的，管你穷学生还是穷工人，只要能上网，只要愿花一秒种就可以了。人生在世，有两件事不能等：一、孝顺；二、行善。无论你是 LED 小灯、普通灯泡也好，还是荧光灯也吧，最重要就是要懂得用自身的光去照耀别人，光的强度并不重要。

点击链接：

http://www.thehungersite.com/clickToGive/home.faces?siteId=1&link=ctg_ths_home_from_ths_thankyou_sitenav

第六章　模块化编程

通过前面 6 章的学习，相信大家对单片机、C 语言、C51 编程都有了一个基本的掌握，既然有了这些基础，那我们就不应该一直停留在点灯上，而是要学习一些新知识，只有这样，我们才能够玩好单片机，用好单片机。

在真正讲述模块化编程之前，我们先来来补充一点 Keil4 的“软仿真”。所谓“软仿真”，就是用 Keil4 来做软件仿真，这样做，我们可以初步判断我们的程序是否正确，等正确之后，我们才可以将其编译生成 HEX 文件，最后下载到单片机中。当然不是所有的程序都先需要进行软仿真，再下载到单片机，而是对一些有问题的程序，我们首先可以做一下软件仿真，以便排除一些显而易见的“傻”问题。其实 Keil4 软件的仿真功能还是比较强，这里我们简单介绍几点，剩余的就留读者自行研究了。

说道 Keil4 软件的仿真，不得不提它还可以借助一些编程器实现“硬仿真”，这个这里不做介绍，等大家以后学习 C8051F 系列或 STM32 的单片机时再具体了解吧。

6.1 Keil4 的软仿真

特别提醒：对 Keil4 不熟的，赶紧去复习一下第一章。我默认看此章节的人对 Keil4 已经很熟了。

仿真的步骤或者方法其实很简单，由编程界面进入仿真界面之前需检查一个选项设置，具体操作是：单击“Target Options ...”按钮，也即如图 6-1 所示的“4 按钮”，打开“Options for Target ‘Targer 1’”对话框，对话框如图 6-2 所示，第二个选项卡（Target）下的“Xtal”处一定要设置为：11.0592，否则后续时间仿真的参数值会有出入。接着选择倒数第二个选项卡（Debug），其界面如图 6-3 所示。这里读者需要注意图中箭头所指的两个复选框，其意义大不相同。

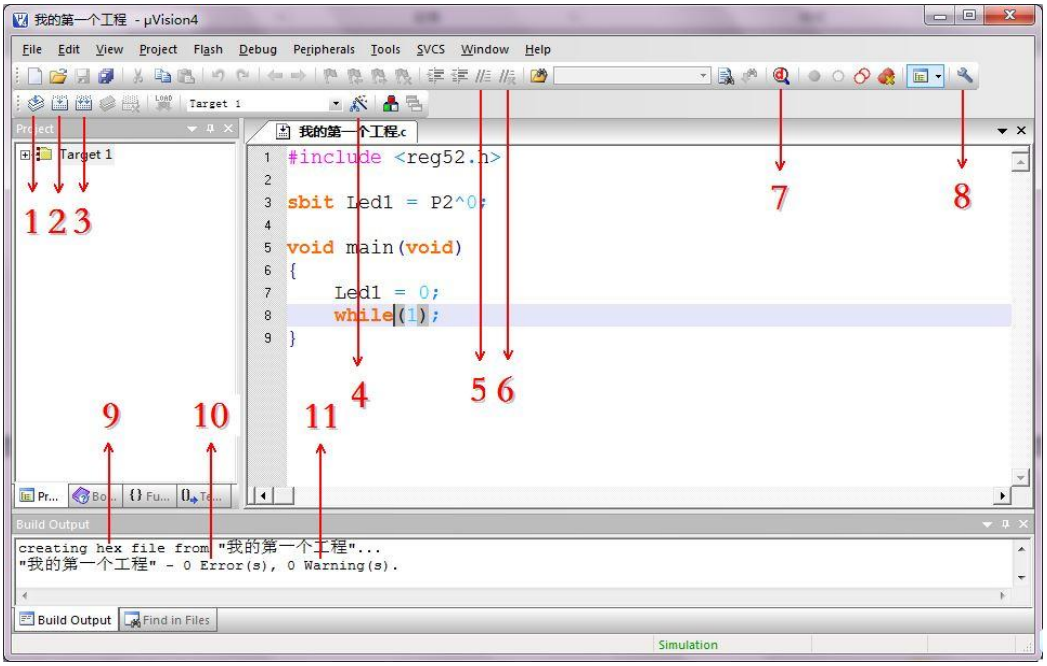

图 6-1 Keil4 的编辑界面图

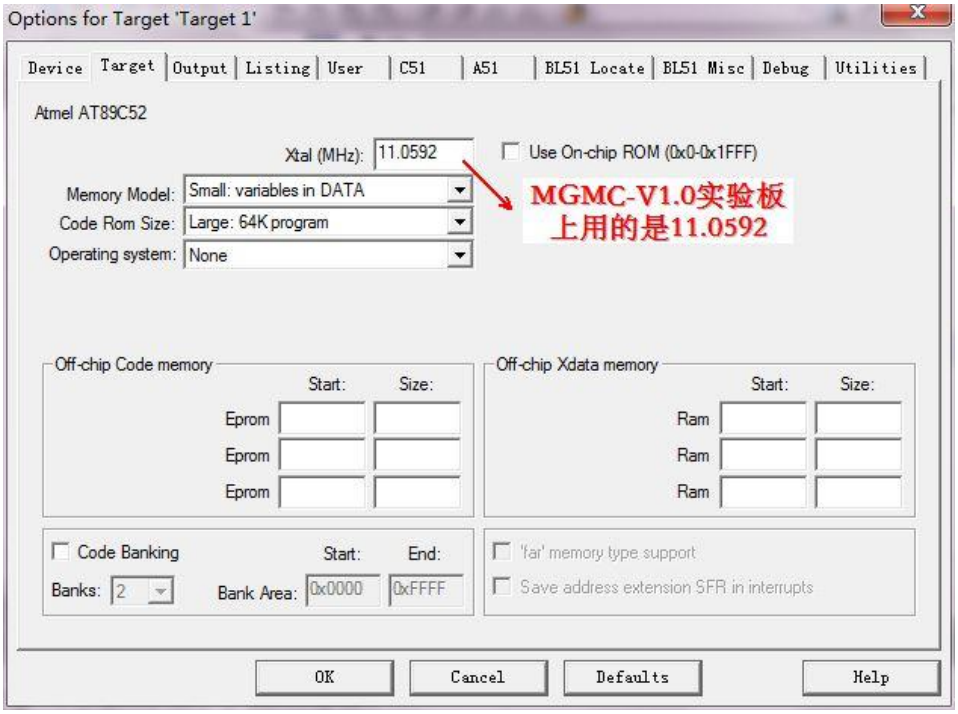

图 6-2 Options for Target ‘Target1’界面图

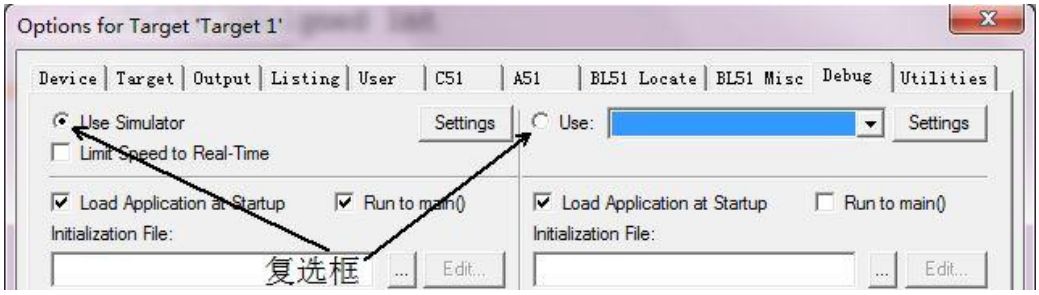

图 6-3 Debug 选项卡设置对话框

第一个（Use Simulator）：意思是软件模拟仿真，就是只在软件上做一些仿真动作，与硬件无关。

第二个（Use：...）：这里是用硬件仿真，意思是软件上面的仿真动作也会对应到硬件上。51 单片机需要借助一块特殊单片机芯片或仿真器；C8051F 系列需要借助 EC6 等仿真器；STM32 需要借助 J-link 等。这些读者只需要知道有这么回事就是了，不需深入的了解，以后学到了自然就会明白。

这里我们需要进行软件模拟仿真，因此选择软件默认的第一个选项就是了。别的选择默认项，最后单击“OK”按钮。

接着选择菜单项：Debug→Start/Stop Debug Session（或者选择如图 6-1 所示的 7 按钮），由编辑界面进入仿真界面，这时若 Keil4 软件没破解，则会有一个“2K”的代码限制，软件的破解，我在第一章中已经讲述过了，这里就不费口舌了。进入仿真界面的 Keil4 如图 6-4 所示。

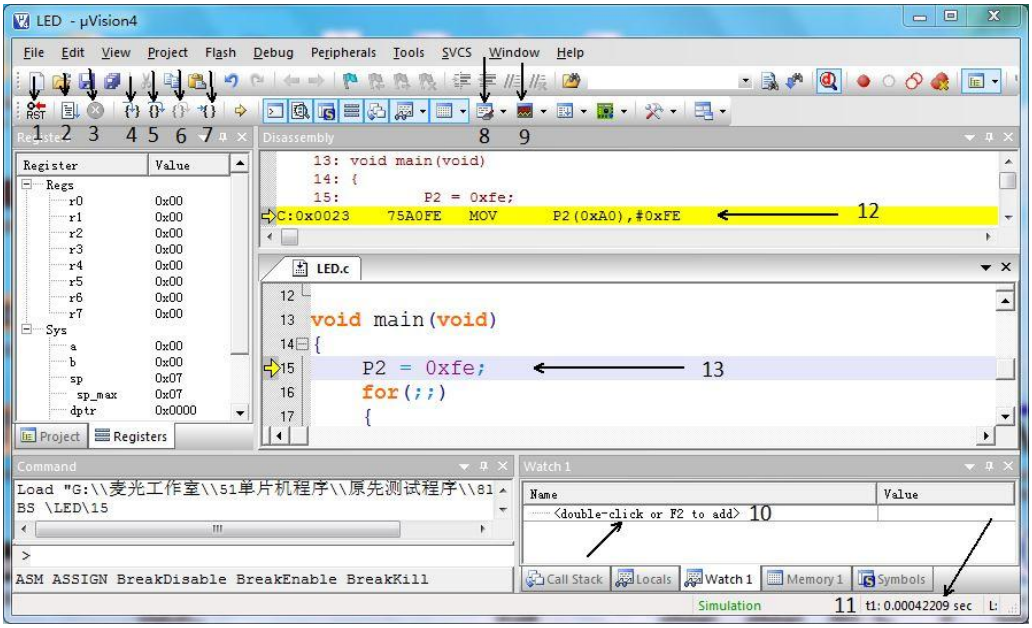

图 6-4 Keil4 的仿真界面图

接下来笔者主要说明图中所标识的 13 个选项，其中 12、13 只是为了好说明，没有操作的地方不多。

X 入广告。断点：这里的断点不是张敬轩唱的《断点》歌哈，而是在程序中添加一个断点，让程序运行到断点处时停止，以便读者作出别的操作，例如观察某一变量值，或者通过单击单步来运行程序等。插入的方法最直接，就是在想插入行的最后面双击鼠标左键，取消断点的方法也是双击。

1）Reset CPU：复位选项，意思是当程序执行一段以后，读者想让其重新开始，单击此处程序执行点就会回到开始处，即 main 函数的开头处。

2）Run：程序从头开始全速执行。当有断点时运行到断点处停止，没有时程序按程序规定一直运行。

3）Stop：顾名思义，停止运行的程序。

4）Step：单步运行。当碰见子函数时，会进入子函数。

5）Step Over：单步运行。碰见子函数时，不进入，将子函数当做一个整体来运行。

6）Step Out：单步运行。程序若在子函数内部运行，则会跳出子函数。

7）Run to Cursor Line：运行到光标处。

8）Serial Windows：串口输出窗口。

9）Analysis Windows：逻辑分析窗口。该窗口下有三个子选项。笔者这里以“Logic Analyzer”为例来讲解。别的两个读者自行研究。

10）变量等数值的观察窗口。

11）程序运行的时间。

12）反汇编窗口，这个对于新手来说，估计有些难，读者们只知道有这么回事就是了。

13）C 语言程序窗口，可以观察程序此时运行到什么地方了。

当然该界面可操作的地方不止这些，例如观察左边寄存器的值、PC 指针、内存数值等，这些就留读者慢慢去琢磨，笔者很懒就不说了。

6.1.1 Keil4 的 I/O 口仿真

这里以跑马灯为例，来仿真 P2 口的状态值。进入仿真界面后选择菜单项：Peripherals

→I/O-Ports→Port 2，此时界面中会出现一个如图 6-5 所示的复选框。由于此时程序未运行，因此 P2 口的状态值都为高电平，因此界面显示为：0xFF。当单击“Step”或者“Step Over”按钮时程序运行：P2 = 0xfe，这时就变为：0xFE;界面如图 6-6 所示。之后就会依次变为：0xFD、0xFB...0x7F。别的端口仿真类似。

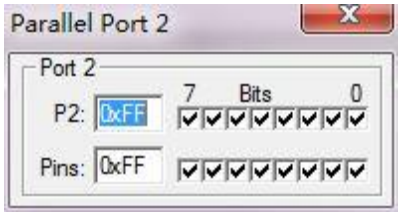

图 6-5 程序未运行是 P2 口的状态值

图 6-6 程序未运行是 P2 口的状态值

当程序运行到 DelayMS(50)时有两种选择，一种是单击“Step”按钮进入 DelayMS()函数，一种是单击“Step Over”不进入 DelayMS()函数，直接将函数当做整体运行，这个望读者自行调试，加以区别，笔者就不再说了。至于这里的时间，稍后再说。

6.1.2 Keil4 的逻辑分析仪

同样以跑马灯为例。进入仿真界面以后单击“Analysis Windows”（如图 6-7 的 1 处），则会默认选中第一个“Logic Analyzer”，这时仿真界面如图 6-7 所示。

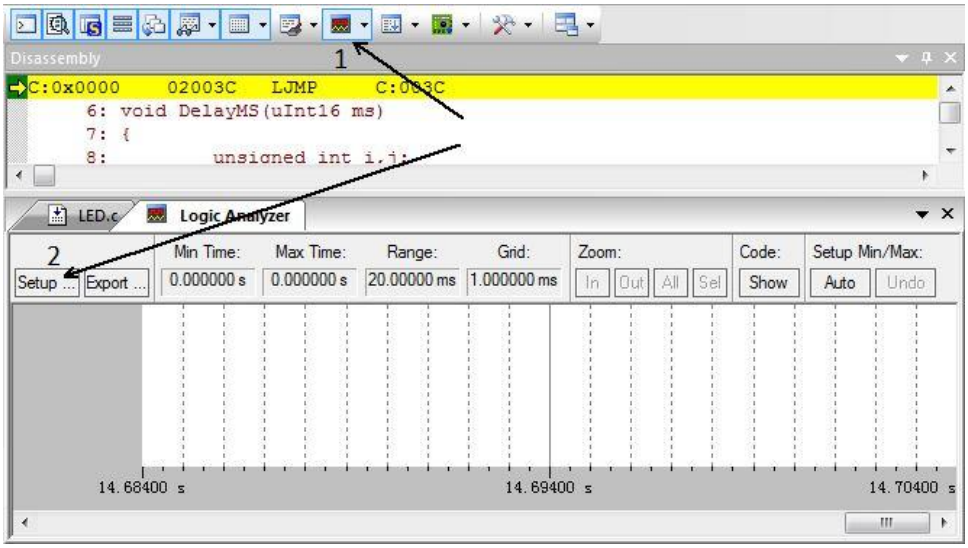

图 6-7 Logic Analyzer 界面图

接着单击“Setup...”按钮（图 6-7 所示的序号 2 处），打开“Setup Logic Analyzer”对话框，如图 6-8 所示。

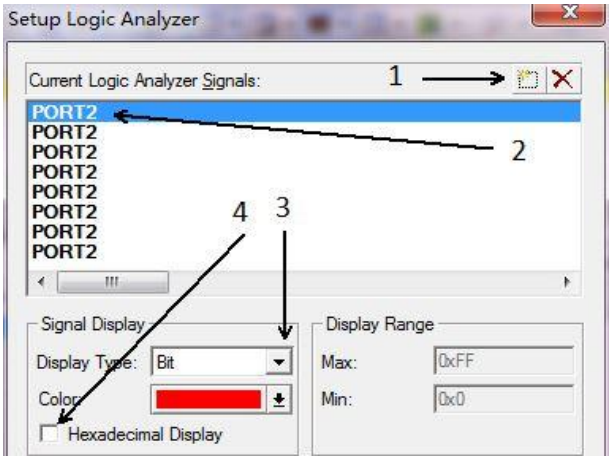

图 6-8 Setup Logic Analyzer 对话框

具体操作为：先单击图 6-8“1”处的“New”选项，之后在“2”所示的地方填“PORT2.0”，接着再单击“New”按钮，再在“2”处填“PORT2.1”，这样依次再新建 6 个，在“2”处分别填：PORT2.2、PORT2.3...PORT2.7，最后如图 6-8 所示。

其中序号 3 用于以什么方式显示，这里选择位：Bit，当然还可以选择为：Analog 和 State。序号 4 是当单击选中该复选框时数值以十六进制方式显示。这些读者可以自行实验。

以上设置好之后单击“Close”按钮，接着选择如图 6-4 所示的“Run”按钮（全速运行），过几秒钟之后在单击“Stop”按钮，停止运行，这时就可得到如图 6-9 所示的波形图，界面是不是蛮漂亮的，读者们还不赶紧亲自试一试。

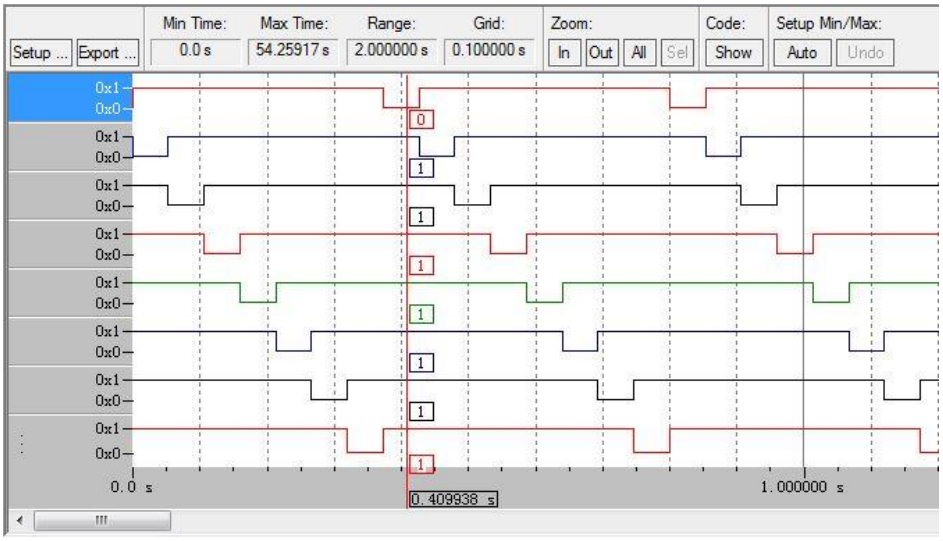

图 6-9 Logic Analyzer 的仿真波形图

6.1.3 Keil4 的变量值仿真

意思是通过 Keil4 来观察程序运行中各个变量的数值变化是否正确，这里还是以实例 4 来做讲解。

回到仿真主界面，先在图 8-20 所示的序号 10 处添加两个变量：i、j，添加方法是双击或者按键盘上的 F2，之后填写 i、j，添加完变量之后“Watch1”窗口如图 6-10 所示。接着分别右键单击选择 i、j 行，之后选择：Number Base→Decimal，意思是数值以十进制的形式显示。

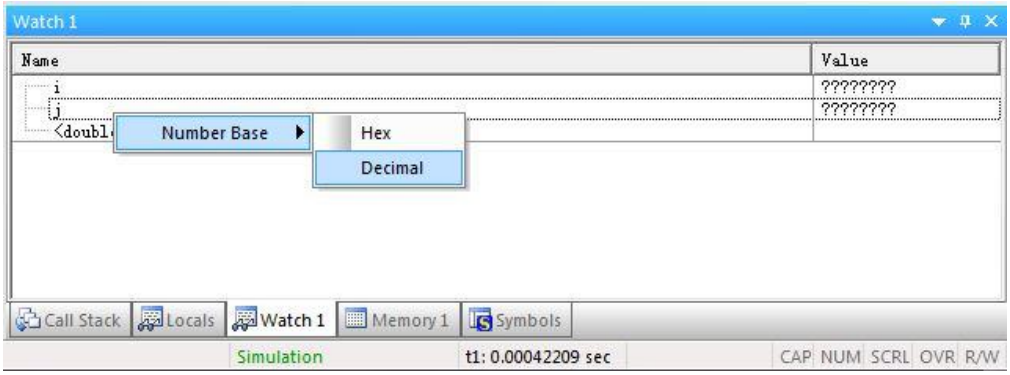

图 6-10 “Watch1”窗口中添加完变量之后的界面图

用“Step Over”来运行程序到 DelayMS(50)时改为“Step”运行程序，这样就可以进入到 DelayMS()函数，这时细心的读者已经注意到变量后面的“Value”已经由“????????”变为了 0，接着再单击“Step”运行程序，这时 i、j 是不是在有规律的变化，其实 j 变为 113 自后就再不会变化了，而 i 则一直再自增（肯定是有范围的）。这时读者还可以观察下面的 t1 时间如何变化，当读者单击 50 次“Step”之后，这里的时间是：0.05031250，读者可以想一想，这个时间与我们想要延时的 50ms 有没有联系，若有的话，又是怎么联系到一起的，或者说起到了延时的作用没？读者们带着这些问题，就可以进入下一章的学习了。

6.2 各种版本的延时

在玩单片机的过程中，有一种“功能”一直陪伴在左右，那就是——延时。原因很简单，玩单片机无非是玩一些人能看得见、能听得见、能用得着在一些东西。前面笔者说过，人很聪明，但是眼睛、耳朵等都很“慢”，速度远远跟不上单片机的速度，这样就需单片机在做某件事时等一等，这一等，给人一点反映的时间，这就引出了延时的概念。

笔者这里将延时分为两类：不精确的延时和精确的延时。这两者在概念和实现的方法上大不相同。接下来分别阐述一下这两种延时，希望对读者有所帮助。

6.2.1 续 Keil4 的时间仿真

在讲述精确延时和非精确延时之前，先来补补上章的内部，Keil4 下的延时仿真。这里还是以实例 4 为例来讲解 Keil4 软件的时间仿真，在讲述过程中，读者需考虑一个问题，为什么执行 DelayMS()函数就能起到延时的作用，难道执行别的函数就起不到延时作用吗？

接下来就打破沙锅问到底，为何 DelayMS()函数能起到延时的作用，这里从仿真入手来，来看个究竟。读者按上章所讲述的操作方法，让其 Keil4 软件进入到仿真界面，这时记下右下角（如图 8-20 的序号 11 处）的时间值为：t1=0.00000000s，接着单击一次“Step Over”按钮运行程序，这时程序运行之进入了 main()函数，则时间变为：t1=0.00042209s；再单击“Step Over”，程序运行了：P2 = 0xfe 和 for(;;)，时间变为：t1=0.00042426s，这样运行时间为：t=0.00042426s-0.00042209s≈2us；再运行程序：P2 = _crol_(P2,1)，时间变为：t1=0.00043837s，则程序运行的时间：t=0.00043837s-0.00042426s=0.00001411s≈14us。由此可得出以下三条结论：

（1）任何程序执行是需要时间的。例如这里的 2us、14us 等。

（2）C 语言编写的程序，每条语句运行的时间是不确定的（2us≠14us）。至于汇编，笔者在这里就不说了，读者可以自行了解一下。

（3）程序编写中，一般将这些时间忽略不计。

接着再说“DelayMS(50)”为何能起到延时 50ms 的作用。操作步骤是先在 DelayMS(50)和倒数第二个大括号后打两个断点（两行后面双击就可以打上断点），之后单击“Run”按钮，此时如图 8-20 所示的标号 11 处的时间值为：0.00043837s，也即 4us 多一点，再单击“Run”按钮，这时时间变为：0.05031467s，这样就可以算出执行程序：DelayMS(50)所用的时间，时间值为：0.05031467s-0.00043837s=0.0498763s≈50ms。为何上面的程序执行时间可以忽略，而这里又要借助程序执行的时间来达到延时的目的呢？那是因为，上面的程序执行次数少（先不考虑 while(1)），而这里的 DelayMS(50)执行了 5650（50×113）次，这个时间就要算了，因为其花了 50ms 了，再不算就 OUT 了。

6.2.2 真实的时间判定

以上时间都是用软件来仿真，接下来残弈悟恩带领读者们看看这个 50ms 的真、假程度究竟有多高，是比天还高，还是比海底更低呢。这里借助逻辑分析仪（简易型的哈，高级的笔者买不起）或者示波器来抓取其时间值，两者得到时间值分别如图 6-11 和 6-12 所示，现在来对比一下这三个值，软件仿真得到的为：49.87ms，逻辑分析仪抓取的时间为：49.88ms，示波器获取的时间为：50ms，三个值几乎接近我们想要的理论值：50ms，这说明上面的延时函数（DelayMS()）还是写的很“牛 X”啊，但再牛也只是一个不精确的延时，要想有精确的延时，那就看下面精确延时的讲述了。

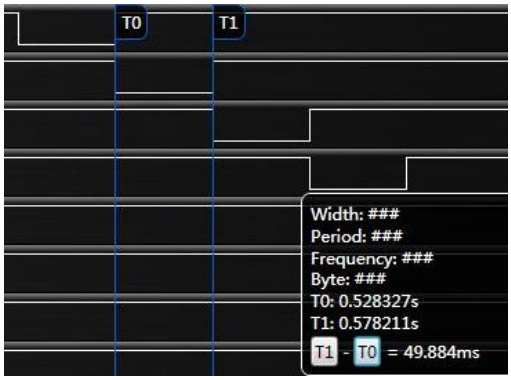

图 6-11 逻辑分析仪抓取的时间值

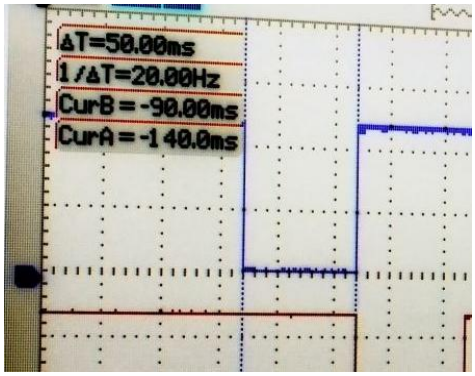

图 6-12 示波器抓取的时间值

6.2.3 精确的延时时间

俗话说，只有相对的，没有绝对的。这里的精确也是相对于上面的不精确而言，因为这里的精确也不是绝对的精确。笔者举个例子，这里所谓的精确的是用定时器和库函数_nop_()来产生，但这两者最终的参考源都是外部晶振，可晶振也是人生产的，怎么可能做到绝对准确的了，如 MGMC-V1.0 实验板上搭载的 11.0592MHz 晶振的实际频率为 11.045MHz。即使有些高级的单片机内部集成了晶振，也是靠 RC 来产生的，这些也会受到温度等影响而不稳定。还有定时器在装初值、产生中断等时也需要时间，这些时间都是无法估算到里面的。因此读者要有个清醒地认识，不要一说精确，就要精确到皮秒、飞秒去。

关于定时器和库函数_nop_()的延时，笔者这里就不举例说明了，以后具体的例程中再详细讲解。当然读者朋友们也可以提前研究研究，何乐而不为呢？

/* == */

你若不离不弃，我便生死相依。

你若不会模块化编程，我便认为你程序写不咋滴。

/* == */

当一个项目小组做一个相对比较复杂的工程时，就需要小组成员分工合作，一起完成项目，意味着不再是某人独自单干，而是要求小组成员各自负责一部分工程。比如你可能只是负责通讯或者显示某一块，这个时候，就应该将自己的这一块程序写成一个模块，单独调试，留出接口供其它模块调用。最后，小组成员都将自己负责的模块写完并调试无误后，由项目组长进行综合调试，像这些场合就要求程序必须模块化。模块化的好处非常多，不仅仅是便于分工，它还有助于程序的调试，有利于程序结构的划分，还能增加程序的可读性和可移植性。

其实说到程序的模块化编程，笔者早都按捺不住，因为前面程序的程序在一遍又一遍、一页又一页的重复，那为何笔者不放在前面讲解，那是怕读者说：别的书上都没讲，你怎么一上来就“胡扯”这东西，想“忽悠”我们是吧，没门！直接一把将书压在十八层地狱下面，书只能呜呜大哭，^_^。因而残弈悟恩将该笔记放到后面偷偷地讲。接下来就跟随笔者揭开模块化编程的神秘面纱，一窥其真面目吧。

6.3 Keil4 的进阶应用——建模

由于这里的模块化编程是基于 Keil4 的，因此这里先来讲述 Keil4 中如何“建模”，或者说自己的工程如何管理比较妥当。接下来通过以下几个步骤来讲述 Keil4 的模块化编程。

第一步 新建工程文件夹

读者可能看到这个标题有点“发疯”，新建文件夹谁不会啊，还需讲解吗？可对于新手来说，还真不一定。

新建文件夹，并命名为：模块化编程（当然也可以是别的）。接着选择好路径，如笔者的为：G:\模块化编程。 打开此文件夹，接着在下面再新建四个文件夹，分别命名为：inc、listing、output、src。这么多文件夹有何用，稍等、稍等，会用到的。

第二步 新建工程

打开 Keil4 软件，选择 Project→New uVision Project 菜单项，此时弹出工程保存对话框，这里定位到自己新建的工程文件下（如：G:\模块化编程）。输入工程名：模块化编程。就不附图了哈，简单。接着就是什么器件选型、是否添加启动代码等，残弈悟恩相信读者们很熟。此时的工程只是一个“骨架”，没有“血肉”哦。

第三步 Target Options 对话框的设置

打开“Target Options”对话框，除了在“Target”选项卡下的“Xtal”处填：11.0592 和在“Output”选项卡下“Create HEX File”前打勾之外，还需增加以下两个步骤。

1、选择“Output”选项卡，接着单击“Select Folder for Objects…”，打开文件夹选择对话框，定位到刚建立的 output 文件夹下（路径为：G:\模块化编程\output\），最后单击“OK”，具体操作如图 6-13 所示。这样编译产生的一些文件就会存放在该文件夹里，需要注意的是 HEX 文件就在此文件夹下，不要下载时找不到了哈。

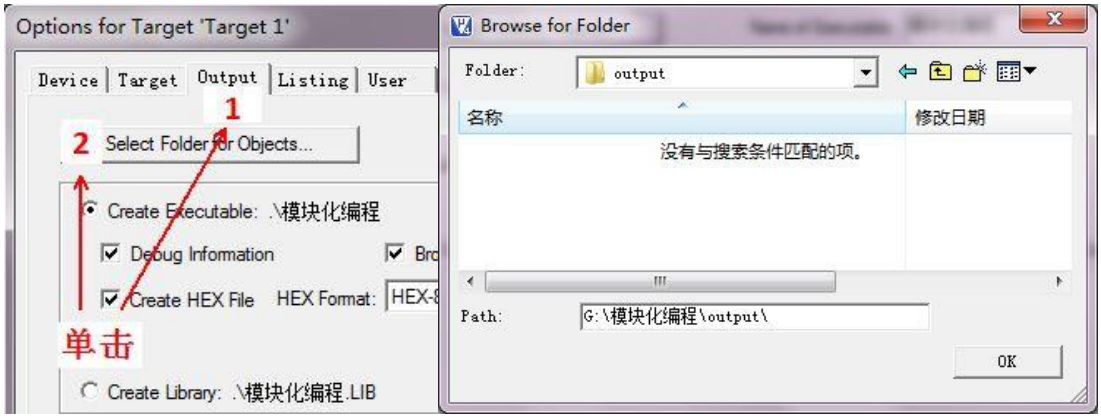

图 6-13 output 文件夹设置示意图

2、选择“Listing”选型卡，接着单击“Select Folder for Listing...”，打开文件夹选择对话框，定位到刚建立的 listing 文件夹下，最后单击“OK”，如图 6-14 所示。该文件夹里最后存放编译过程中产生的一些连接文件，不予理睬。

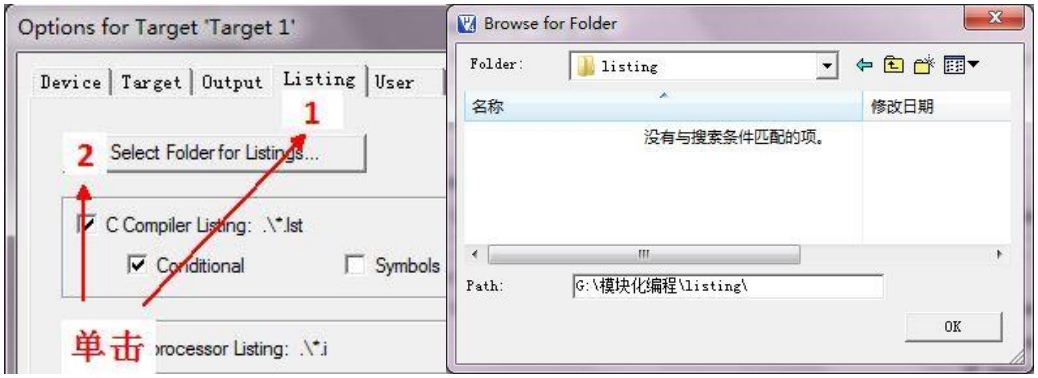

图 6-14 Listing 文件夹设置示意图

第四步 Components 对话框的设置

右键单击“Target1”，接着选择“Manage components...”打开工程组件对话框，操作过程如图 6-15 所示，所打开的对话框如图 6-16 所示。

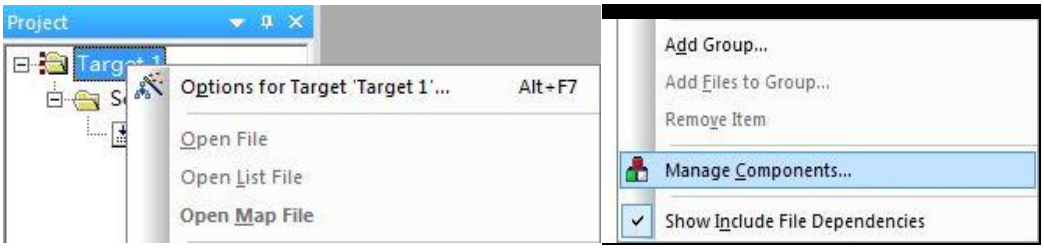

图 6-15 打开 Components 对话框

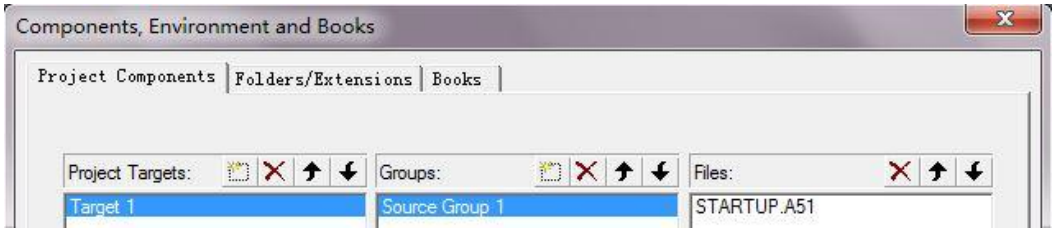

图 6-16 Components 对话框（未修改的）

接着双击图 6-16 左边的“Target1”，将其修改为“模块化编程”（当然可以修改成自己喜欢的名称）。若想再添加工程目标文件，只需轻轻单击上面的新建图标，之后单击下面“Set

as Current Target”，将其设置为当然的目标文件。要删除直接单击红 X，这些作为扩展内容，读者自行研究。

接着双击图 6-17 中间的“Source Group1”，将其修改为“System”；之后单击此列上放的新建图标，新建一个组，命名为“USER”；由于此时没有源文件，所以还没法添加源文件，若已经有源文件，这时就可以选择右下角的“Add Files”来添加文件了。关于这些后面步骤将会详细讲解。最后修改之后的界面如图 6-17 所示。

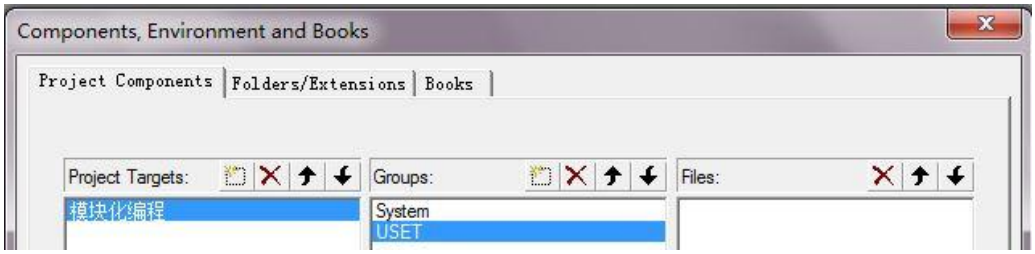

图 6-17 Components 对话框（已修改的）

第五步 新建源文件

当然，若只新建一个 XX.c，相信读者很熟，估计都快熟透了，^_^。可问题是这里要新建的远远不止一个啊，而是 N^n（n 肯定不是无穷大，呵呵）个。

回到 Keil4 的主界面，直接 8 次快捷键“CTRL+N”，意思是新建 8 个文件，此时 Keil4 的编辑界面应该是 8 个文本文件，名称依次为：Text1...Text8（也有可能是 Text3...Text10，这都没关系）。

接下来就是保存这 8 个文件，关于保存是有讲究的，有何讲究呢？请看下面两个步骤：

1、保存“.c”文件。按快捷键“CTRL+S”，此时弹出文件保存路径选择对话框，默认是工程文件夹“模块化编程”，这时在下面的文件名处写：main.c，意思你懂得，再之后单击“保存”则 OK。

同理，再保存 3 个“.c”文件，有区别的是文件再不能保存在工程文件夹下了，而是要定位到前面新建的“src”文件夹下，其中文件名称依次为：led.c、delay.c、uart.c。界面如图 6-18 所示。

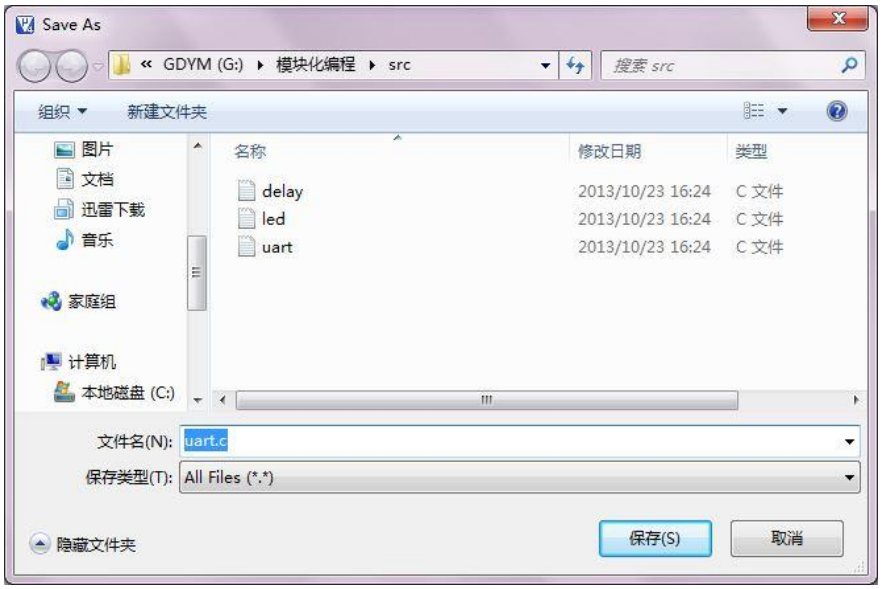

图 6-18 三个.c 文件保存之后的界面图

2、保存“.h”文件。同理 4 次“CTRL+S”，此时弹出的保存路径默认在“src”文件下，可我们的目的是该文件夹（src）用来保存“.c”，而将“.h”保存到“inc”文件夹下，为何

这样做做，让笔者慢慢道来。当选定到“inc”文件夹下以后，依次输入文件名：led.h、delay.h、uart.h、common.h，此时界面如图 6-19 所示。

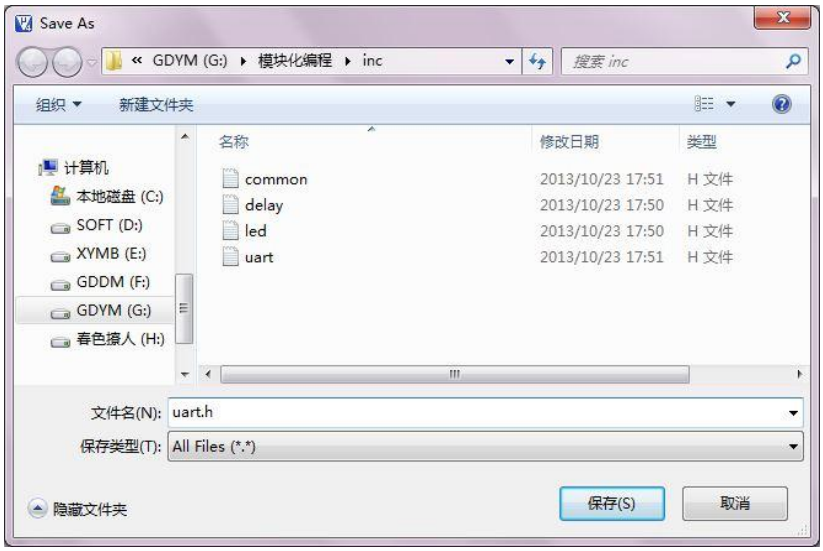

图 6-19 四个.h 保存之后的界面图

第六步 添加源文件到工程

在笔记 2 中已经讲述过如何将源文件添加到工程中，过程是右键单击“USER”文件组，在弹出的下拉菜单中选择“Add Files to Group ‘USER’”，接着依次选中：main.c、delay.c、led.c、uart.c，并添加到“USER”组中。

这里再讲述一种文件的添加方式。按图 6-15 的方式打开“Components”对话框，所打开的对话框如图 6-17 所示，接着单击选择“USER”组（选中之后会由灰色变成蓝色），之后再单击右下角的“Add Files”按钮，如图 6-20 所示，之后也会弹出“Add Files to Group ‘USER’”对话框，依次选中：main.c、delay.c、led.c、uart.c，并添加到“USER”组中，添加完之后单击“OK”。添加完源文件之后的 Keil4 主界面如图 6-21 所示。

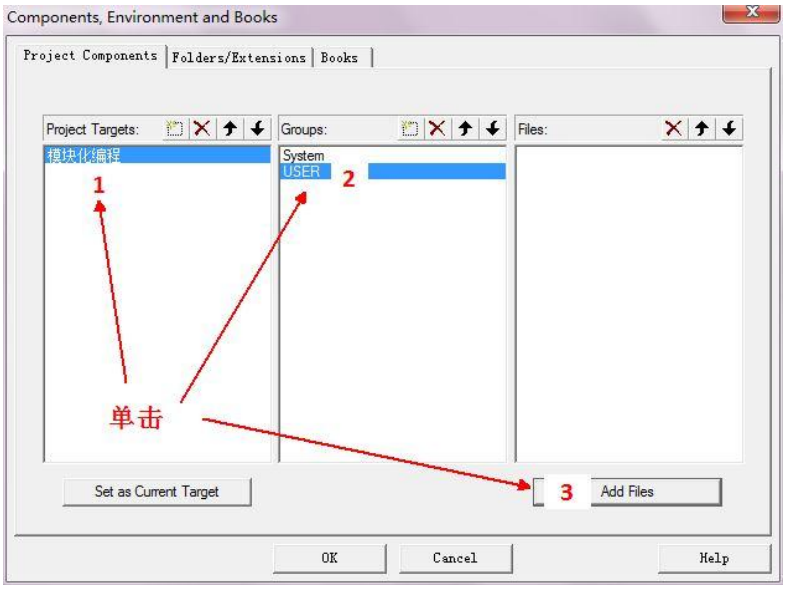

图 6-20 Components 对话框中添加源文件

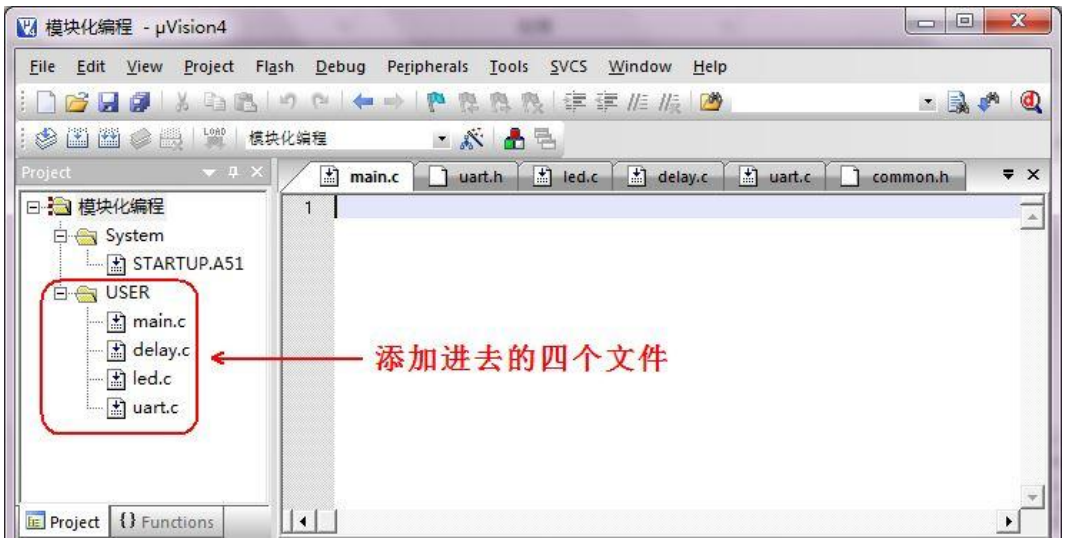

图 6-21 添加完源文件的 Keil4 主界面图

这样模块化工程的建立就讲述完毕，接下来的任务就是编写代码，也就是让各个“.c”、“.h”能表里如一，别表面风光、内心彷徨。

6.4 单片机之模块化编程

这部分应该就是该笔记的核心了，也是有别于其他书籍的地方之一，因此望读者多留一份心思。既然是难点、重点，就应该用清晰的思路、严谨的结构来讲述，但问题是所涉及的知识点多、结构混乱，又不好讲解，无赖之下，笔者计划边往上述 8 个源文件中加添加内容，边讲解其中的来龙去脉，望读者们能够接受。

6.4.1 借说明开头

说明一：模块即是一个.c 和一个.h 的结合，头文件（.h）是对该模块的声明。

说明二：某模块提供给其他模块调用的外部函数以及数据需在所对应的.h 文件中冠以 extern 关键字来声明。

说明三：模块内的函数和变量需在.c 文件开头处冠以 static 关键字声明。

说明四：永远不要在.h 文件中定义变量。

先解释一下说明中的两个关键词：定义和声明。相信读者都是学过 C 语言的，本应该对这两个词理解的很透彻，可笔者在培训时发现，好多人都搞不清楚，都是凭着感觉写的，高兴了就用定义，不高兴了就用声明，这样做当然是不对的，换句话说是错的。

强势的 X 入广告：什么是定义和声明？

所谓的定义就是（编译器）创建一个对象，为这个对象分配一块内存并给它取上一个名字，这个名字就是我们经常所说的变量名或者对象名。但注意，这个名字一旦和这块内存匹配起来（可以想象是这个名字嫁给了这块空间，没有要彩礼啊），它们就同生共死，终生不离不弃，并且这块内存的位置也不能被改变。一个变量或对象在一定的区域内（比如函数内）只能被定义一次，如果定义多次，编译器会提示你重复定义同一个变量或对象。

什么是声明？声明确切的说应该有两重含义：

第一：告诉编译器，这个名字已经匹配到一块内存上了(伊人已嫁，吾将何去何从？何以解忧，唯有稀粥)，下面的代码用到变量或对象是在别的地方定义的。声明可以出现多次。

第二：告诉编译器，我这个名字我先预定了，别的地方再也不能用它来作为变量名或对象名。比如你在图书馆自习室的某个座位上放了一本书，表明这个座位已经有人预订，别人再也不允许使用这个座位。其实这个时候你本人并没有坐在这个座位上。这种声明最典型的例子就是函数参数的声明，例如：void fun(int i, char c)。

那他们的区别也很清晰了。记住，定义声明最重要的区别：定义创建了对象并为这个对象分配了内存，声明没有分配内存(一个抱伊人，一个喝稀粥，^_^)。

6.4.2 用实践解释

有了以上的概述，接下来就请读者跟随残弈悟恩一起享受模块化编程这顿大餐吧。说明一概括了模块化的实现方法和实质：将一个功能模块的代码单独编写成一个.c 文件，然后把该模块的接口函数放在.h 文件中。这里就分别来介绍一下它们里面究竟该包括什么，又不该包括什么。

一、源文件.c

提到 C 语言源文件，大家都不会陌生。因为我们平常写的程序代码几乎都在这个.C 文件里面。编译器也是以此文件来进行编译并生成相应的目标文件。作为模块化编程的组成基础，所有要实现功能源代码均在这个文件里。理想的模块化应该可以看成是一个黑盒子，即只关心模块提供的功能，而不予理睬模块内部的实现细节。好比读者买了一部手机，只需会用手机提供的功能即可，而不需要知晓它是如何把短信发出去的，又是如何响应按键输入的，这些过程对用户而言，就是一个黑盒子。

在大规模程序开发中，一个程序由很多个模块组成，很可能，这些模块的编写任务被分配到不同的人。例如当读者在编写模块时很可能需要用到别人所编写模块的接口，这个时候读者关心的是它的模块实现了什么样的接口，该如何去调用，至于模块内部是如何组织、实现的，读者无需过多关注。特此说明，为了追求接口的单一性，把不需要的细节尽可能对外屏蔽起来，只留需要的让别人知道。

Eg:不知读者发现了，以上笔记共用最多的一个函数是：DelayMS(uInt16 ms)，那这里就先拿这个函数来开刀，写个 delay.c 源文件，具体代码如下：

```
1.  #include "delay.h"
2.  static void Delay1MS(void)
3.  {
4.      uChar8 i = 2,j = 199;
5.      do
6.      {
7.          while (--j);
8.      }
9.      while (--i);
10. }
11. void DelayMS(uInt16 ValMS)
12. {
13.     uInt16 uiVal;
14.     for(uiVal = 0; uiVal < ValMS; uiVal++)
15.     {
16.         Delay1MS();
```

```
17.     }
18. }
```

看到此程序，读者们是不是疑惑三点。第一点：第 1 行代码是从哪个星球来的？第二点：static 是那方妖魔，竟敢欺负函数；第三点：void DelayMS(uInt16 ValMS)函数怎么换了别国的发动机了，因为国产的是两个 for 循环也。笔者这样写肯定是有用意滴，具体缘由待残弈悟恩慢慢道来。

二、头文件.h

谈及到模块化编程，必然会涉及到多文件编译，也就是工程编译。在这样的一个系统中，往往会有多个 C 文件，而且每个 C 文件的作用不尽相同。在我们的 C 文件中，由于需要对外提供接口，因此必须有一些函数或变量需提供给外部其它文件进行调用。例如上面新建的 delay.c 文件，提供最基本的延时功能函数。

```
void DelayMS(uInt16 ValMS);      // 延时 ValMS(ValMS=0~65535)毫秒
```

而在另外一个文件（eg:led.c）中需要调用此函数，那该如何做呢？兵来将挡、水来土淹，头文件的作用正是在此。具体过程是先创建一个 delay.h 头文件，在该头文件中对 DelayMS()函数进行封装（声明），这种封装的内容不应包含任何实质性的函数代码。有了这样一个封装好的接口文件，每当 led.c 文件需要调用 DelayMS()函数时，直接在 led.c 中包含 delay.h 头文件即可。读者可将头文件形象的理解为连接 delay.c 和 led.c 的桥梁。同时该文件也可以包含一些宏定义以及结构体的信息，离开了这些信息，很可能就无法正常使用接口函数或者是接口变量。但是总的原则是：不该让外界知道的信息就不应该出现在头文件里，而外界调用模块内接口函数或者是接口变量所必须的信息就一定要出现在头文件里，否则外界就无法正确调用。因而为了让外部函数或者文件调用我们提供的接口功能，就必须包含我们提供的这个接口描述文件----**头文件**。同时，我们自身模块也需要包含这份模块头文件(因为其包含了模块源文件中所需要的宏定义或者是结构体)，好比三方协议，除了给学校、公司有之外，自己总的留一份吧。下面我们来定义这个头文件，一般来说，头文件的名字应该与源文件的名字保持一致，这样便可清晰的知道哪个头文件是哪个源文件的描述。

于是便得到了 delay.c 如下的 delay.h 头文件，具体代码如下：

```
1.  #ifndef   __DELAY_H__
2.  #define   __DELAY_H__
3.  #include "common.h"
4.  extern void DelayMS(uInt16 ValMS);
5.  #endif
```

第 3 行先放一放，稍安勿躁，后面讲解，这里需要详细解释三点。

1）.c 源文件中不想被别的模块调用的函数、变量就不要出现在.h 文件中。例如本地函数 static void Delay1MS(void)，即使出现在.h 文件中也是在做无用功，因为其它模块根本不去调用它,实际上也调用不了它(static 关键字起了限制作用啊)。例如，国家主席的专车不是每个人都能坐。

2）.c 源文件中需要被别的模块调用的函数、变量就声明现在.h 文件中。例如 void DelayMS(uInt16 ValMS)函数，这与以前所写源文件中的函数声明有些类似，为何没说一样了，因为前面加了修饰词 extern，表明是一个外部函数。通俗点，就想公交车一样，谁都可以坐（没钱人当然坐不了，例如残弈悟恩）。

特别提醒，在 Keil4 编译器中，extern 这个关键字即使不声明，编译器也不会报错，且程序运行良好，但不保证使用其它编译器也如此。因此，强烈建议加上，养成良好的编程习惯也不是一件坏事。

3）1、2、5 行是条件编译和宏定义，目的是为了防止重复定义。假如有两个不同的源文件都需要调用 void DelayMS(uInt16 ValMS)这个函数，他们分别都通过#include “delay.h”把这个头文件包含进去。在第一个源文件进行编译时候，由于没有定义过__DELAY_H__，因此#ifndef __DELAY_H__条件成立，于是定义__DELAY_H__并将下面的声明包含进去。在第二个文件编译时，由于第一个文件包含的时候，已经将__DELAY_H__定义过了。因而此时#ifndef __DELAY_H__不成立，整个头文件内容就不再被包含。假设没有这样的条件编译语句，那么两个文件都包含了 extern void DelayMS(uInt16 ValMS)，就会引起重复包含的错误。

特别说明，可能新手们看到 DELAY 前后的这些“__”、“_”时，又会模糊一阵，残弈悟恩告诉你，这又是一只“纸老虎”，看着吓人，一捅就破。举几个例子：DELAY_H__、DELAY_H、DELAYH、____DELAY_H、__Delay_H，经调试，这些版本都是对的，所以，请读者自便，怎么高兴怎么来，残弈悟恩这么（__DELAY_H__）写是出于编程习惯。

看看上面预留的问题——#include ″common.h″，这里面又包含着什么东东，打开一看原来是下面几行不起眼的家伙，代码如下。

```
1.  #ifndef __COMMON_H__
2.  #define __COMMON_H__
3.  typedef unsigned char uChar8;
4.  typedef unsigned int   uInt16;
5.  #endif
```

这里简单说一下条件编译（1、2、5 行）。在一些头文件的定义中，为了防止重复定义，一般用条件编译来解决此问题。如第 1 行的意思是如果没有定义：__COMMON_H__，那么就定义：#define __COMMON_H__（第 2 行），定义的内容包括：3、4 行，代码含义就不说了，只要读者还能记得起前面讲述的“typedef”，那就能理解这两句的意思和这么写的好处。

三、位置决定思路—变量

上面说明四中提到，变量不能定义在.h 中，是不是有点危言耸听的感觉，都不敢用全局变量，其实也没这么严重。对于新手来说，或许是一个难点，再难也有解决的办法啊，世上无难事，只怕有心人嘛。解决这个问题的良方当然可以借鉴嵌入式操作系统——uCOS-Ⅱ，该操作系统处理全局变量的方法比较特殊，也比较难理解，但学会之后妙用无穷啊。感兴趣的读者可以自行研究一下。

残弈悟恩依个人的编程习惯，介绍一种处理方式。概括的讲，就是在.c 中定义变量，之后在该.c 源文件所对应的.h 中声明即可。注意，一定要给它带上一顶“奴隶帽”，即变量声明前加一修饰词——extern，这样无论“他”走到哪里，别人都可以指示“他”干活，想怎么修改就怎么修改，但读者用“他”时，可别太过分，“他”也是人，累了会生病。同理，滥用全局变量会使程序的可移植性、可读性变差。接下来用两段代码来比较说明全局变量的定义和声明。

电脑爆炸式的代码：

```
1.  module 1.h                  // 编写一个.h
2.  uChar8   uaVal = 0;         // 在模块 1 的.h 文件中定义一个变量 uaVal
3.  /* ============================================================*/
4.  module1 .c                  // 编写一个.c
5.  #include   "module1.h"      // .c 模块 1 中包含模块 1 的.h
6.  /* ============================================================*/
```

```
7.  module2 .c
8.  #include   "module1.h"        // .c 模块 2 中包含模块 1 的.h
```

以上程序的结果是在模块 1、2 中都定义了无符号 char 型变量 uaVal，uaVal 在不同的模块中对应不同的内存地址。如果这个世间都这么写程序，那电脑就会爆炸，当然是夸张的修辞手法。

推荐式的代码：

```
1.  module 1.h                    // 编写一个.h
2.  extern uChar8   uaVal;        // 在.h 中声明 uaVal
3.  /* ============================================================*/
4.  module1 .c
9.  #include   "module1.h"        // .c 模块 1 中包含模块 1 的.h
10. uChar8   uaVal = 0;           // 在模块 1 的.h 文件中定义一个变量 uaVal
5.  /* ============================================================*/
6.  module2 .c
11. #include   "module1.h"        // 在模块 2 的.h 文件中定义一个变量 uaVal
```

这样如果模块 1、2 操作 uaVal 的话，对应的是同一块内存单元。

四、符号决定出路—头文件之包含

以上模块化编程中，要大量的包含头文件。学过 C 语言的都知道，包含头文件的方式有两种，一种是“<xx.h>”，第二种是“”xx.h””，那何时用第一种，又何时用第二种，可能读者会从什么相对路径啊、绝对路径啊、系统的用什么、工程中的用什么，当然如果你知道，肯定是一件好事，若那个读者和残弈悟恩一样笨，记不下，那就听笔者一句话：自己写的用**双引号**，不是自己写的用**尖括号**。

五、模块的分类

一个嵌入式系统通常包括两类模块（注意：是两类，不是两个）。

（1） 硬件驱动模块。一种特定硬件对应一个模块。

（2） 软件功能模块。其模块的划分应满足低耦合、高内聚的要求。

低耦合、高内聚这是软件工程中的概念。简单说是六个字，但是所涉及的内容比较多，笔者就不过多讲解了，若读者感兴趣，可以自行查阅资料，慢慢理解、总结、归纳其中的奥秘。笔者这里简单说明两点。

第一点：内聚和耦合。

内聚是从功能角度来度量模块内的联系，一个好的内聚模块应当恰好做一件事。它描述的是模块内的功能联系。

耦合是软件结构中各模块之间相互连接的一种度量，耦合强弱取决于模块间接口的复杂程度、进入或访问一个模块的点以及通过接口的数据。

理解了以上两个词的含义之后，那“低耦合、高内聚”就好理解了，通俗点讲，模块与模块之间少来往，模块内部多来往。当然对应到程序中，就不是这么简单，这需要大量的编程和练习才能掌握其真正的内涵，这就留给读者去慢慢研究吧。

第二点：硬件驱动模块和软件功能模块的区别。

所谓硬件驱动模块是指所写的驱动（也就是.c 文件）对应一个硬件模块。例如 led.c 是用来驱动 LED 灯的，smg.c 是用来驱动数码管的，lcd.c 是用来驱动 LCD 液晶的，key.c 是用来检测按键的，等等，将这样的模块统称为硬件驱动模块。

所谓的软件功能模块是指所编写的模块只是某个功能的实现，而没有所对应的硬件模

块。例如 delay.c 是用来延时的，main.c 是用来调用各个子函数的。这些模块都没有对应的硬件模块，只是起某个功能而已。

6.5 源文件路径的添加

假如此时读者都理解了上述所讲述的知识点，并且按实例 45 编写好了代码，应该是大功告成了，可编译之后发现错误多多是也，此时读者若按以前常规方法去分析，应该认为是代码哪儿写错了，其实原因不在这儿，那原因何在，敢问路在何方啊？路就在脚下。接下来分两步来解决此问题。

第一步：打开“Target Options”对话框，并选择“C51”选项卡，此时界面如图 6-22 所示，注意笔者添加的圆角矩形框 。

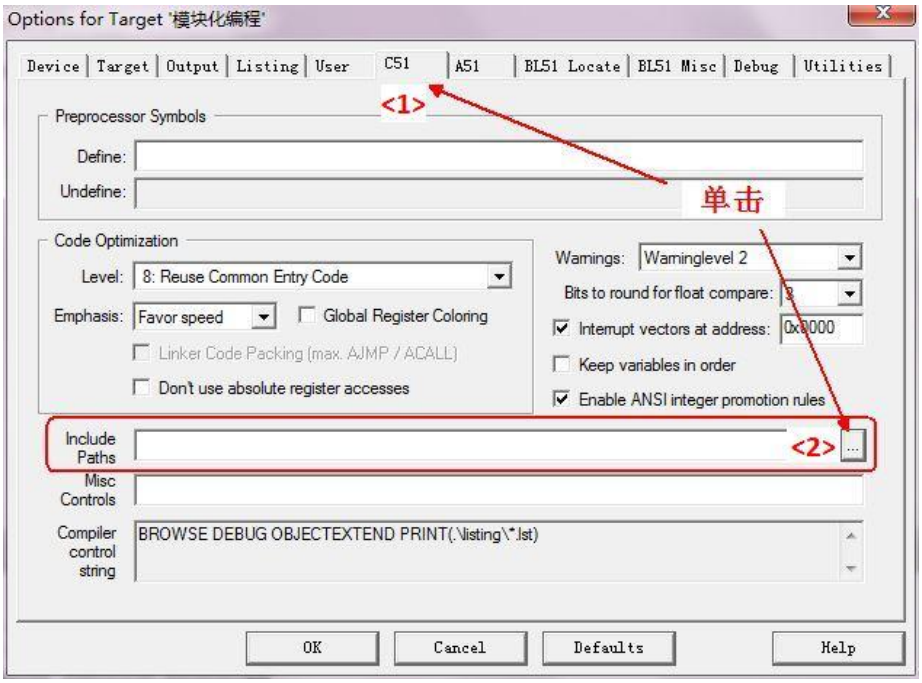

图 6-22 Options 路径添加对话框

第二步：单击圆角矩形宽后的“<2>”处进入“Floder Setup”对话框。接着单击如图 6-23（左图）所示的“New”处，此时该对话框会变成右图所示的摸样，接着单击路径浏览框，这时会弹出如图 6-24 所示的浏览文件夹，这里定位到自己的“inc”文件夹路径下（例如笔者的为：G:\模块化编程\inc），接着单击“确定”按钮，最后一路“OK”，这样路径就添加完成了。

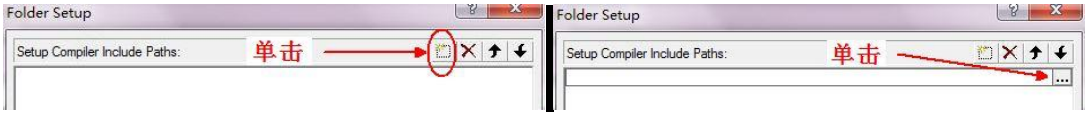

图 6-23 新建路径对话框

图 6-24 路径选择框

6.6 模块化编程的应用实例

上面接近 10 页的原理估计将读者带进了原始森林，在雾里来，雾里去的，连东南西北都辨别不清了，那就赶紧上两个实例，帮助读者消化、消化。

傻瓜实例——8 灯闪烁

先来个简单的实例，主要理一理模块化编程的过程。该实例就是利用 MGMC-V1.0 实验板，编写程序（不是一个.c 的，而是要用模块化编程），让其 8 个 LED 灯实现流水灯的效果，同时通过串口打印此时演示的是什么实验。程序当然不难，但是望读者还是虚心点、认真点，掌握其模块化编程的思想，以便为下一个实例做铺垫。整个过程的建立、文件的新建、添加请看 16.1 节，各部分源码如下。

(一) main.c 的源码。

```
1.  #include <reg52.h>
2.  #include "led.h"
3.  void main(void)
4.  {
5.      while(1)
6.      {
7.          LED_FLASH();
8.      }
9.  }
```

(二) common.h 的源码。

```
1.  #ifndef __COMMON_H__
2.  #define __COMMON_H__
```

```
3.  typedef unsigned char uChar8;
4.  typedef unsigned int   uInt16;
5.  #endif
```

(三) delay.c 的源码。

```
1.  #include "delay.h"
2.  static void Delay1MS(void)
3.  {    /* 同.c 源码的讲解部分 */      }
4.  void DelayMS(uInt16 ValMS)
    {    /* 同.c 源码的讲解部分 */      }
```

(四) delay.h 的源码。

```
1.  #ifndef __DELAY_H
2.  #define __DELAY_H__
3.  #include "common.h"
4.  extern void DelayMS(uInt16 ValMS);
5.  #endif
```

(五) led.c 的源码。

```
1.  #include "led.h"
2.  void LED_FLASH(void)
3.  {
4.      P2 = 0x00;          // 点亮 8 个灯
5.      DelayMS(1000);
6.      P2 = 0xFF;          // 熄灭 8 个灯
7.      DelayMS(1000);
8.  }
```

(六) led.h 的源码。

```
1.  #ifndef __LED_H__
2.  #define __LED_H__
3.  #include <reg52.h>      // 程序用到了 P2 口，所以包含此头文件
4.  #include "delay.h"      // 程序用到延时函数，所以包含此头文件
5.  extern void LED_FLASH(void);
6.  #endif
```

该程序注释也不详细，本应该分析一下代码，可笔者发现，无论从结构上还是原理上，前面都不止一次讲过，估计读者比残弈悟恩还熟悉，因此就不费口舌了。

这样模块化编程就讲述完毕了，可能这里的实例不够复杂，读者看不出模块化编程的重要性，既然这样，那就在这里随便再附一个程序，只供读者欣赏，里面用到了按键、定时器、中断、液晶、状态机检测等，这些知识我们还没讲述，所以大家不必能看懂此程序，只需感觉一下模块化编程就好了，若自己能看懂，那就更好了。具体实例的讲解，等我学了按键、定时器、中断、液晶、状态机检测之后，我再来详细讲解。

漂亮的花瓶——基于定时器的时钟

实例简介：以 MGMC-V1.0 实验板为硬件平台，以模块化的方式编写程序，让其能在 1602 液晶的第一行显示：2013-12-08 SUN，其中“SUN”代表星期天；第二行显示：11:27:11，如图 6-25 所示，当然这些时间是“活”的哦，:-) :-)。接着增加按键调时功能。如按一次 S4 键时，时间停止“走”动，并且秒个位处的光标开始闪烁，此时，短按一次 S8，则秒数加一，

长按 S8 时，数值连续加一；同理，若此时短按一次 S12，则秒数减一，长按 S12，则数值连续减一，调节的界面如图 6-26 所示。之后，若再按一次 S4，秒数位置处的数值正常显示，调整的时间数切换到分钟处，这时按下 S8、S12 分钟数做相应的递增和递减，如图 6-27 所示。若再按一次 S4，同理可以调节小时数了，如图 6-28 所示。限于时间关系，笔者没有写年、月、日、星期的调节功能，这个就留读者自己摸索呢。同时还需增加蜂鸣器功能，当短按一次 S4、S8、S12 时，蜂鸣器响一次，若长按 S4、S8、S12 时，蜂鸣器不响。

接下来看几张“美”图，先给读者视觉上的冲击，这样读者看代码时会有一个整体上的把握，之后就是万众瞩目的程序源码，最后来点笔者当初调试时的心得和总结。

图 6-25 时钟显示界面图

图 6-26 调节“秒”界面图

图 6-27 调节“分”界面图

图 6-28 调节“时”界面图

先来看看源码，其中主程序的源码“main.c”如下：

```
#include<reg52.h>
#include "common.h"
#include "delay.h"
#include "lcd1602.h"
#include "KeyScan.h"
uChar8 Count50MS;          //定时器 50MS 计数
char Second,Hour,Minute;   //秒、时、分
void Timer0Init(void)
{
    TMOD = 0x01;           //设置定时器 0 工作模式 1
    TH0 = 0x4c;            //定时器装初值
    TL0 = 0x00;
    EA = 1;                //开总中断
    ET0 = 1;               //开定时器 0 中断
    TR0 = 1;               //启动定时器 0
}
void Timer1Init(void)
{
    TMOD |= 0x10;          // 设置定时器 0 工作在模式 1 下
    TH1 = 0xDC;
    TL1 = 0x00;            // 赋初始值
    TR1 = 1;               // 开定时器 0
}
void Init(void)
{
```

```
26.     LCD_Init();                 //液晶初始化
27.     Timer0Init();               //定时器 0 初始化
28.     Timer1Init();               //定时器 1 初始化
29.     CONTROL = 0;                //软件将矩阵按键第 4 列一端置低用以分解出独立按键
30.     Minute = 0;                 //初始化种变量值
31.     Second = 0;
32.     Hour = 0;
33.     Count50MS = 0;
34.     WrTimeLCD(10,Second); //分别送去液晶显示
35.     WrTimeLCD(7,Minute);
36.     WrTimeLCD(4,Hour);
37. }
38. void main(void)
39. {
40.     Init();                         //首先初始化各数据
41.     while(1)                    //进入主程序大循环
42.     {
43.         ExecuteKeyNum();  //不停的检测按键是否被按下
44.     }
45. }
46. void timer0(void) interrupt 1
47. {
48.     TH0 = 0x4c;                                 //再次装定时器初值
49.     TL0 = 0x00;
50.     Count50MS++;                                //中断次数累加
51.     if(Count50MS == 20)                         //20 次 50 毫秒为 1 秒
52.     {
53.         Count50MS = 0;
54.         Second++;
55.         if(Second == 60)                        //秒加到 60 则进位分钟
56.         {
57.             Second = 0;                         //同时秒数清零
58.             Minute++;
59.             if(Minute == 60)                    //分钟加到 60 则进位小时
60.             {
61.                 Minute = 0;                     //同时分钟数清零
62.                 Hour++;
63.                 if(Hour == 24)                  //小时加到 24 则小时清零
64.                 {
65.                     Hour = 0;
66.                 }
67.                 WrTimeLCD(4,Hour);              //小时若变化则重新写入
68.             }
69.             WrTimeLCD(7,Minute);        //分钟若变化则重新写入
```

```
70.             }
71.             WrTimeLCD(10,Second);           //秒若变化则重新写入
72.         }
73. }
```

这里有个小小的头文件，虽小功能大，特别是对残弈悟恩这样的懒人说，更有用。源码如下：

```
1.  #ifndef __COMMON_H__
2.  #define __COMMON_H__
3.  typedef unsigned char uChar8;
4.  typedef unsigned int   uInt16;
5.  #endif
```

别忘了，延时函数虽然有问题，但是某些时候还是值得一用，因此还是增加到这里。其头文件“delay.h”还是那么的简单。

```
1.  #ifndef __DELAY_H__
2.  #define __DELAY_H__
3.  #include "common.h"
4.  void DelayMS(uInt16 ValMS);
5.  #endif
```

其中功能源码“delay.c”就更简单了。

```
1.  #include "delay.h"
2.  void DelayMS(uInt16 ValMS)
3.  {
4.      uInt16 uiVal,ujVal;
5.      for(uiVal = 0; uiVal < ValMS; uiVal++)
6.          for(ujVal = 0; ujVal < 113; ujVal++);
7.  }
```

LCD1602 液晶的头文件源码“LCD1602.H”如下：

```
1.  #ifndef _LCD1602_H_
2.  #define _LCD1602_H_
3.  #include<reg52.h>
4.  #include "common.h"
5.  #include "delay.h"
6.  sbit SEG_SELECT = P1^7;      //段选
7.  sbit BIT_SELECT = P1^6;      //位选
8.  sbit RS = P3^5;              //数据/命令选择端(H/L)
9.  sbit RW = P3^4;              //数/写选择端(H/L)
10. sbit EN = P3^3;              //使能信号
11. extern void WrComLCD(uChar8 ComVal);
12. extern void LCD_Init(void);
13. extern void WrTimeLCD(uChar8 Add,uChar8 Data);
14. #endif
```

LCD1602 液晶的驱动源码“LCD1602.c”如下，其中读者需要注意的是 WrTimeLCD()函数，该函数的主要功能是将待显示的数据实时的再液晶上刷新，其中位的分离前面已经讲述过了，剩下的就是先确定位置，后向确定的位置写数据。这种方式在讲述实例 23 的时候也用

过，原理都是相同的，这里就在不重复了。

```
#include "LCD1602.h"
uChar8 code table[]=" 2013-12-08 SUN";//定义初始上电时液晶默认显示状态
void DectectBusyBit(void)
{
    P0 = 0xff;                          // 读状态值时，先赋高电平
    RS = 0;
    RW = 1;
    EN = 1;
    DelayMS(1);
    while(P0 & 0x80);                   // 若 LCD 忙，停止到这里，否则走起
    EN = 0;                             // 之后将 EN 初始化为低电平
}
void WrComLCD(uChar8 ComVal)
{
    DectectBusyBit();
    RS = 0;
    RW = 0;
    EN = 1;
    P0 = ComVal;
    DelayMS(1);
    EN = 0;

}
void WrDatLCD(uChar8 DatVal)
{
    DectectBusyBit();
    RS = 1;
    RW = 0;
    EN = 1;
    P0 = DatVal;
    DelayMS(1);
    EN = 0;
}
/* ********************************************************* */
// 函数名称：WrTimeLCD()
// 函数功能：向液晶的某个位置写数据
// 入口参数：液晶地址（Addr）,数据（Data）
// 出口参数：键值（num）
/* ********************************************************* */
void WrTimeLCD(uChar8 Addr,uChar8 Data)       //写时分秒函数
{
    uChar8 shi,ge;
    shi = Data / 10;                          //分解一个 2 位数的十位和个位
```

```
44.         ge = Data % 10;
45.         WrComLCD(0x80 + 0x40 + Addr);          //设置显示位置
46.         WrDatLCD(0x30 + shi);                  //送去液晶显示十位
47.         WrDatLCD(0x30 + ge);                   //送去液晶显示个位
48.  }
49.  void LCD_Init(void)
50.  {
51.         uChar8 Num;
52.         SEG_SELECT = 0;                 // 关段选
53.         BIT_SELECT = 0;                 // 关位选
54.         WrComLCD(0x38);                 // 16*2 行显示、5*7 点阵、8 位数据接口
55.         DelayMS(1);                     // 稍作延时
56.         WrComLCD(0x38);                 // 重新设置一遍
57.         WrComLCD(0x01);                 // 显示清屏
58.         WrComLCD(0x06);                 // 光标自增、画面不动
59.         DelayMS(1);                     // 稍作延时
60.         WrComLCD(0x0C);                 // 开显示、关光标、并不闪烁
61.         for(Num = 0;Num < 15;Num++)     //显示年月日星期
62.         {
63.                 WrDatLCD(table[Num]);
64.                 DelayMS(5);
65.         }
66.         WrComLCD(0x80 + 0x40 + 6);      //写出时间显示部分的两个冒号
67.         WrDatLCD(':');
68.         DelayMS(5);
69.         WrComLCD(0x80 + 0x40 + 9);
70.         WrDatLCD(':');
71.         DelayMS(5);
72.  }
```

再来看看笔者认为此程序最难的——按键扫描部分，其实也不难，就是用了一个状态机来检测按键，这个难点，笔者在实例 17 已经讲述过了，也画了状态图，读者在看这部分源码时，不妨先去复习一下实例 17，在回过头来看此源码，这样就会好过度一点，废话少说，来看看这些源码的真面目吧，头文件“KeyScan.h”源码如下：

```
1.  #ifndef _KEYSCAN_H_
2.  #define _KEYSCAN_H_
3.  #include <reg52.h>
4.  #include "common.h"
5.  #include "delay.h"
6.  #include "LCD1602.h"
7.  sbit Beep = P1^4;                  //定义蜂鸣器端
8.  sbit KEY1 = P3^0;
9.  sbit KEY2 = P3^1;
10. sbit KEY3 = P3^2;
11. sbit CONTROL = P3^7;                //分离按键用
```

```
12. extern void KeyScan(void);
13. extern void ExecuteKeyNum(void);
14. #endif
```

最后、最长的按键驱动源码“KeyScan.h”终于出世了，赶紧围观围观。

```
1.  #include "KeyScan.h"
2.  extern char Second,Hour,Minute;        //秒、时、分     外部变量
3.  extern uChar8 code table[];            //LCD 显示数组   外部变量
4.  char FunctionKeyNum;                   //功能键键值
5.  char FuncTempNum;                          //功能键临时键值
6.  typedef enum KeyState{StateInit,StateAffirm,StateSingle,StateRepeat};    //键值状态值
7.  /* ******************************************************* */
8.  // 函数名称：DropsRing()
9.  // 函数功能：蜂鸣器发声
10. // 入口参数：无
11. // 出口参数：无
12. /* ******************************************************* */
13. void DropsRing(void)
14. {
15.     Beep = 0;
16.     DelayMS(100);
17.     Beep = 1;
18. }
19. /* ******************************************************* */
20. // 函数名称：KeyScan(void)
21. // 函数功能：扫描按键
22. // 入口参数：无
23. // 出口参数：键值（num）
24. /* ******************************************************* */
25. void KeyScan(void)
26. {
27.     static uChar8 KeyStateTemp1 = 0;              //按键状态临时存储值 1
28.     static uChar8 KeyStateTemp2 = 0;              //按键状态临时存储值 2
29.     static uChar8 KeyStateTemp3 = 0;              //按键状态临时存储值 3
30.     static uChar8 KeyTime = 0;                    //按键延时时间
31.     bit KeyPressTemp1;                            //按键是否按下存储值 1
32.     bit KeyPressTemp2;                            //按键是否按下存储值 2
33.     bit KeyPressTemp3;                            //按键是否按下存储值 3
34.
35.     KeyPressTemp1 = KEY1;                         //读取 I/O 口的键值
36.     switch(KeyStateTemp1)
37.     {
38.         case StateInit:                           //按键初始状态
39.             if(!KeyPressTemp1)                    //当按键按下，状态切换到确认态
40.                 KeyStateTemp1 = StateAffirm;
```

```
41.                 break;
42.             case StateAffirm:                       //按键确认态
43.                 if(!KeyPressTemp1)
44.                 {
45.                     KeyTime = 0;
46.                     KeyStateTemp1 = StateSingle;    //切换到单次触发态
47.                 }
48.                 else KeyStateTemp1 = StateInit;     //按键已抬起，切换到初始态
49.                 break;
50.             case StateSingle:                       //按键单发态
51.                 if(KeyPressTemp1)                   //按下时间小于 1s
52.                 {
53.                     DropsRing();                    //每当有按键释放蜂鸣器发出滴声
54.                     KeyStateTemp1 = StateInit;      //按键释放，则回到初始态
55.                     FuncTempNum++;                  //键值加一
56.                     if(FuncTempNum  > 4) FuncTempNum  = 0;
57.                 }
58.                 else if(++KeyTime > 100)            //按下时间大于 1s(100*10ms)
59.                 {
60.                     KeyStateTemp1 = StateRepeat;    //状态切换到连发态
61.                     KeyTime = 0;
62.                 }
63.                 break;
64.             case StateRepeat:                       //按键连发态
65.                 if(KeyPressTemp1)
66.                     KeyStateTemp1 = StateInit;      //按键释放，则进初始态
67.                 else                                //按键未释放
68.                 {
69.                     if(++KeyTime > 10)              //按键计时值大于 100ms（10*10ms）
70.                     {
71.                         KeyTime = 0;
72.                         FuncTempNum++;              //键值每过 100ms 加一次
73.                         if(FuncTempNum  > 4) FuncTempNum  = 0;
74.                     }
75.                     break;
76.                 }
77.                 break;
78.             default: KeyStateTemp1 = KeyStateTemp1 = StateInit; break;
79.         }
80.         if(FuncTempNum)                 //只有功能键被按下后，增加和减小键才有效
81.         {
82.             KeyPressTemp2 = KEY2;                   //读取 I/O 口的键值
83.             switch(KeyStateTemp2)
84.             {
```

```
85.             case StateInit:                         //按键初始状态
86.                 if(!KeyPressTemp2)                  //当按键按下，状态切换到确认态
87.                     KeyStateTemp2 = StateAffirm;
88.                 break;
89.             case StateAffirm:                       //按键确认态
90.                 if(!KeyPressTemp2)
91.                 {
92.                     KeyTime = 0;
93.                     KeyStateTemp2 = StateSingle;    //切换到单次触发态
94.                 }
95.                 else KeyStateTemp2 = StateInit;     //按键已抬起，切换到初始态
96.                 break;
97.             case StateSingle:                       //按键单发态
98.                 if(KeyPressTemp2)                   //按下时间小于 1s
99.                 {
100.                    KeyStateTemp2 = StateInit;  //按键释放，则回到初始态
101.                    DropsRing();                //每当有按键释放蜂鸣器发出滴声
102.                    if(FunctionKeyNum == 1)     //若功能键第一次按下
103.                    {
104.                        Second++;               //则调整秒加 1
105.                        if(Second == 60)        //若满 60 后将清零
106.                            Second = 0;
107.                        WrTimeLCD(10,Second); //每调节一次送液晶显示一下
108.                        WrComLCD(0x80 + 0x40 + 11);//显示位置重新回到调节处
109.                    }
110.                    if(FunctionKeyNum == 2)     //若功能键第二次按下
111.                    {
112.                        Minute++;               //则调整分钟加 1
113.                        if(Minute == 60)        //若满 60 后将清零
114.                            Minute = 0;
115.                        WrTimeLCD(7,Minute);    //每调节一次送液晶显示一下
116.                        WrComLCD(0x80 + 0x40 + 8); //显示位置重新回到调节处
117.                    }
118.                    if(FunctionKeyNum == 3)     //若功能键第三次按下
119.                    {
120.                        Hour++;                 //则调整小时加 1
121.                        if(Hour == 24)          //若满 24 后将清零
122.                            Hour = 0;
123.                        WrTimeLCD(4,Hour);      //每调节一次送液晶显示一下
124.                        WrComLCD(0x80 + 0x40 + 5); //显示位置重新回到调节处
125.                    }
126.                }
127.                else if(++KeyTime > 100)//按下时间大于 1s(100*10ms)
128.                {
```

```
129.                    KeyStateTemp2 = StateRepeat;//状态切换到连发态
130.                    KeyTime = 0;
131.                }
132.                break;
133.            case StateRepeat:                    //按键连发态
134.                if(KeyPressTemp2)
135.                    KeyStateTemp2 = StateInit;   //按键释放，则进初始态
136.                else                             //按键未释放
137.                {
138.                    if(++KeyTime > 10)       //按键计时值大于 100ms（10*10ms）
139.                    {
140.                        KeyTime = 0;
141.                        if(FunctionKeyNum == 1)     //若功能键第一次按下
142.                        {
143.                            Second++;               //则调整秒加 1
144.                            if(Second == 60)    //若满 60 后将清零
145.                                Second = 0;
146.                            WrTimeLCD(10,Second); //每调节一次送液晶显示一下
147.                            WrComLCD(0x80 + 0x40 + 11);
148.                            //显示位置重新回到调节处
149.                        }
150.                        if(FunctionKeyNum == 2)     //若功能键第二次按下
151.                        {
152.                            Minute++;               //则调整分钟加 1
153.                            if(Minute == 60)        //若满 60 后将清零
154.                                Minute = 0;
155.                            WrTimeLCD(7,Minute);  //每调节一次送液晶显示一下
156.                            WrComLCD(0x80 + 0x40 + 8);//重新回到调节处
157.                        }
158.                        if(FunctionKeyNum == 3)     //若功能键第三次按下
159.                        {
160.                            Hour++;                 //则调整小时加 1
161.                            if(Hour == 24)          //若满 24 后将清零
162.                                Hour = 0;
163.                            WrTimeLCD(4,Hour);      //每调节一次送液晶显示一下
164.                            WrComLCD(0x80 + 0x40 + 5); //重新回到调节处
165.                        }
166.                    }
167.                    break;
168.                }
169.                break;
170.            default: KeyStateTemp2 = KeyStateTemp2 = StateInit; break;
171.        }
172.
```

```
173.        KeyPressTemp3 = KEY3;                    //读取 I/O 口的键值
174.        switch(KeyStateTemp3)
175.        {
176.            case StateInit:                      //按键初始状态
177.                if(!KeyPressTemp3)               //当按键按下，状态切换到确认态
178.                    KeyStateTemp3 = StateAffirm;
179.                break;
180.            case StateAffirm:                    //按键确认态
181.                if(!KeyPressTemp3)
182.                {
183.                    KeyTime = 0;
184.                    KeyStateTemp3 = StateSingle;//切换到单次触发态
185.                }
186.                else KeyStateTemp3 = StateInit;  //按键已抬起，切换到初始态
187.                break;
188.            case StateSingle:                    //按键单发态
189.                if(KeyPressTemp3)                //按下时间小于 1s
190.                {
191.                    KeyStateTemp3 = StateInit;   //按键释放，则回到初始态
192.                    DropsRing();                 //每当有按键释放蜂鸣器发出滴声
193.                    if(FunctionKeyNum == 1)      //若功能键第一次按下
194.                    {
195.                        Second--;                //则调整秒减 1
196.                        if(Second == -1)         //若减到负数则将其重新设置为 59
197.                            Second = 59;
198.                        WrTimeLCD(10,Second);        //每调节一次送液晶显示一下
199.                        WrComLCD(0x80 + 0x40 + 11);//显示位置重新回到调节处
200.                    }
201.                    if(FunctionKeyNum == 2)      //若功能键第二次按下
202.                    {
203.                        Minute--;                //则调整分钟减 1
204.                        if(Minute == -1)         //若减到负数则将其重新设置为 59
205.                            Minute = 59;
206.                        WrTimeLCD(7,Minute);         //每调节一次送液晶显示一下
207.                        WrComLCD(0x80 + 0x40 + 8); //显示位置重新回到调节处
208.                    }
209.                    if(FunctionKeyNum == 3)      //若功能键第二次按下
210.                    {
211.                        Hour--;                  //则调整小时减 1
212.                        if(Hour == -1)           //若减到负数则将其重新设置为 23
213.                            Hour = 23;
214.                        WrTimeLCD(4,Hour);           //每调节一次送液晶显示一下
215.                        WrComLCD(0x80 + 0x40 + 5); //显示位置重新回到调节处
216.                    }
```

```
217.                }
218.                else if(++KeyTime > 100)                //按下时间大于 1s(100*10ms)
219.                {
220.                    KeyStateTemp3 = StateRepeat;    //状态切换到连发态
221.                    KeyTime = 0;
222.                }
223.                break;
224.            case StateRepeat:                           //按键连发态
225.                if(KeyPressTemp3)
226.                    KeyStateTemp3 = StateInit;          //按键释放，则进初始态
227.                else                                    //按键未释放
228.                {
229.                    if(++KeyTime > 10)      //按键计时值大于 100ms（10*10ms）
230.                    {
231.                        KeyTime = 0;
232.                        if(FunctionKeyNum == 1)     //若功能键第一次按下
233.                        {
234.                            Second--;               //则调整秒减 1
235.                            if(Second == -1)    //若减到负数则将其重新设置为 59
236.                                Second = 59;
237.                            WrTimeLCD(10,Second); //每调节一次送液晶显示一下
238.                            WrComLCD(0x80 + 0x40 + 11);//重新回到调节处
239.                        }
240.                        if(FunctionKeyNum == 2)     //若功能键第二次按下
241.                        {
242.                            Minute--;               //则调整分钟减 1
243.                            if(Minute == -1)    //若减到负数则将其重新设置为 59
244.                                Minute = 59;
245.                            WrTimeLCD(7,Minute);  //每调节一次送液晶显示一下
246.                            WrComLCD(0x80 + 0x40 + 8); //重新回到调节处
247.                        }
248.                        if(FunctionKeyNum == 3)     //若功能键第二次按下
249.                        {
250.                            Hour--;                 //则调整小时减 1
251.                            if(Hour == -1)          //若减到负数则将其重新设置为 23
252.                                Hour = 23;
253.                            WrTimeLCD(4,Hour);      //每调节一次送液晶显示一下
254.                            WrComLCD(0x80 + 0x40 + 5); //重新回到调节处
255.                        }
256.                    }
257.                    break;
258.                }
259.                break;
260.            default: KeyStateTemp3 = KeyStateTemp3 = StateInit; break;
```

```
261.            }
262.        }
263. }
264. /* ******************************************************* */
265. // 函数名称：ExecuteKeyNum(void)
266. // 函数功能：按键值来执行相应的动作
267. // 入口参数：无
268. // 出口参数：无
269. /* ******************************************************* */
270. void ExecuteKeyNum(void)
271. {
272.     if(TF1)
273.     {
274.         TF1 = 0;
275.         TH1 = 0xDC;
276.         TL1 = 0x00;
277.         KeyScan();
278.     }
279.     switch(FuncTempNum)
280.     {
281.         case 1:
282.             FunctionKeyNum  = 1;
283.             TR0 = 0;                        //关闭定时器
284.             WrComLCD(0x80 + 0x40 + 11);     //光标定位到秒位置
285.             WrComLCD(0x0f);                 //光标开始闪烁
286.         break;
287.         case 2:
288.             FunctionKeyNum  = 2;            //第二次按下光标闪烁定位到分钟位置
289.             WrComLCD(0x80 + 0x40 + 8);
290.         break;
291.         case 3:
292.             FunctionKeyNum  = 3;            //第三次按下光标闪烁定位到小时位置
293.             WrComLCD(0x80 + 0x40 + 5);
294.         break;
295.         case 4:
296.             FunctionKeyNum  = 0;            //记录按键数清零
297.             WrComLCD(0x0c);                 //取消光标闪烁
298.             TR0 = 1;
299.             FuncTempNum = 0;
300.         break;
301.     }
302. }
```

该程序的内容有详细的注释，这里不解释程序了，主要主要说明几点笔者的编程、调试心得。笔者在为学生培训时，一直在说，程序觉得不是从第一行按次序写到最后一行，而是

首先的划分模块，再来将各个模块分别击破，同样，在击破各个模块时，也不是从第一行写到最后一行，而是先将模块按功能划分成几个函数，之后就需要画流程图，最后才是按流程图编写程序，接下来，就以此实例为例，来简述一下整个程序的编写过程。

第一步：模块的划分。该实例中，模块的划分已经很清晰了，因为上面的源码就是按模块贴上去的。如该实例主要包括主模块、LCD 模块、调试模块。

第二步：模块到函数的划分。以主程序为例，笔者将其划分为五个函数，其实中断函数是定的；之后再加三个初始化函数，分别用来初始化定时器 0（1）、LCD 的初始界面。最后一个当然是主函数 main()了，这个的功能还用重复吗。

第三步：画流程图。这部分的子流程图比较多，笔者就不画了，但是读者一定要画，这对编程很管用，有句话是这么说的，望读者能铭记于心：做一件事，规划的时间占 70%，执行的时间占 30%。笔者为何说这个，就是想告诉读者画流程图也算做一件事的规划，只要流程图画的好，编程思路清晰，是很容易编写程序的。相反，没有好的规划，一上来就是写，变写边删，最后形成一种恶性循环，导致整个思路中断，这就是好多“浮躁”读者所采用的方法，又恰恰是一种错误的方法。

再来说说整个实例的调试心得。笔者当初就是按这个步骤来调试的，读者可以去其糟怕，取其精华。

第一步：编写程序，让液晶能正常显示字符。

第二步：增加液晶刷新函数，并随便写一个数，看是否能显示到液晶上。

第三步：增加定时器功能，让时、分、秒三个全局变量动起来，看是否能正常显示到液晶上。读者刚开始，可以将秒时间设置的快一点，同时将秒逢 60 进一暂时改为逢 5 进一，这样做主要是便于调试。否则要让小时数变一次，读者真希望等 1 小时？？

第四步：增加按键扫描功能。这里编程风格很重要，各个大括号一定要按程序语句对齐，否则最后会很乱。同时全局变量的处理一定要到位，否则会造成时间数不统一。这里读者特别要注意，什么时候开定时器，什么时候关定时器；什么时候光标闪烁，什么时候光标不闪烁，更要注意光标闪烁的位置。笔者认为这第四大步是最难、最重要的，读者一定要耐心、仔细。

第五步：增加蜂鸣器功能。这个就很 easy 了，一个小小的函数就可搞定。

最后，留读者一个作业：

（1）增加年、月、日、星期的调节、动态走时功能。

（2）增加阴历功能。

这个作业的答案，读者可以关注残弈悟恩的博客或电子工程师论坛。