单片机 C 语言模块化编程

下面让我们揭开模块化神秘面纱，一窥其真面目。

C 语言源文件 *.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 XX.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，我们所要实现的所有功能的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。即我们只关心模块提供的功能，而不管模块内部的实现细节。好比我们买了一部手机，我们只需要会用手机提供的功能即可，不需要知晓它是如何把短信发出去的，如何响应我们按键的输入，这些过程对我们用户而言，就是是一个黑盒子。

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。而你在编写这个模块的时候很可能就需要利用到别人写好的模块的借口，这个时候我们关心的是，它的模块实现了什么样的接口，我该如何去调用，至于模块内部是如何组织的，对于我而言，无需过多关注。而追求接口的单一性，把不需要的细节尽可能对外部屏蔽起来，正是我们所需要注意的地方。

C 语言头文件 *.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者是变量提供给外部其它文件进行调用。

假设我们有一个 LCD.C 文件，其提供最基本的 LCD 的驱动函数

```
LcdPutChar(char cNewValue) ;  //在当前位置输出一个字符
```

而在我们的另外一个文件中需要调用此函数，那么我们该如何做呢？

头文件的作用正是在此。可以称其为一份接口描述文件。其文件内部不应该包含任何实质性的函数代码。我们可以把这个头文件理解成为一份说明书，说明的内容就是我们的模块对外提供的接口函数或者是接口变量。同时该文件也包含了一些很重要的宏定义以及一些结构体的信息，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，外界就无法正确的调用我们提供的接口功能。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----即头文件。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)，好比我们平常所用的文件都是一式三份一样，模块本身也需要包含这个头文件。

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，这样我们便可以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 LCD.C 的头文件 LCD.h 其内容如下。

```
#ifndef      _LCD_H_
        #define       _LCD_H_
        extern     LcdPutChar(char cNewValue) ;
#endif
```

这与我们在源文件中定义函数时有点类似。不同的是，在其前面添加了 extern 修饰符表明其是一个外部函数，可以被外部其它模块进行调用。

```
#ifndef      _LCD_H_
        #define       _LCD_H_
#endif
```

这个几条条件编译和宏定义是为了防止重复包含。假如有两个不同源文件需要调用 LcdPutChar(char cNewValue)这个函数，他们分别都通过#include “Lcd.h”把这个头文件包含了进去。在第一个源文件进行编译时候，由于没有定义过 _LCD_H_ 因此 #ifndef _LCD_H_ 条件成立，于是定义_LCD_H_ 并将下面的声明包含进去。在第二个文件编译时候，由于第一个文件包含时候，已经将_LCD_H_定义过了。因此#ifndef _LCD_H_ 不成立，整个头文件内容就没有被包含。假设没有这样的条件编译语句，那么两个文件都包含了 extern LcdPutChar(char cNewValue) ; 就会引起重复包含的错误。

不得不说的 typedef

很多朋友似乎了习惯程序中利用如下语句来对数据类型进行定义

```
#define uint    unsigned int
#define uchar    unsigned char
```

然后在定义变量的时候 直接这样使用

```
uint    g_nTimeCounter = 0 ;
```

不可否认，这样确实很方便，而且对于移植起来也有一定的方便性。但是考虑下面这种情况你还会 这么认为吗？

```
#define PINT unsigned int *    //定义 unsigned int 指针类型
PINT    g_npTimeCounter, g_npTimeState ;
```

那么你到底是定义了两个 unsigned int 型的指针变量，还是一个指针变量，一个整形变量呢？而你的初衷又是什么呢，想定义两个 unsigned int 型的指针变量吗？如果是这样，那么估计过不久就会到处抓狂找错误了。

庆幸的是 C 语言已经为我们考虑到了这一点。typedef 正是为此而生。为了给变量起一个别名我们可以用如下的语句

```
typedef   unsigned   int      uint16 ;      //给指向无符号整形变量起一个别名 uint16
  typedef   unsigned   int   * puint16 ;   //给指向无符号整形变量指针起一个别名 puint16
```

在我们定义变量时候便可以这样定义了：

```
uint16      g_nTimeCounter   =   0 ;   //定义一个无符号的整形变量
puint16   g_npTimeCounter   ;      //定义一个无符号的整形变量的指针
```

在我们使用51单片机的 C 语言编程的时候，整形变量的范围是16位，而在基于32的微处理下的整形变量是32位。倘若我们在8位单片机下编写的一些代码想要移植到32位的处理器上，那么很可能我们就需要在源文件中到处修改变量的类型定义。这是一件庞大的工作，为了考虑程序的可移植性，在一开始，我们就应该养成良好的习惯，用变量的别名进行定义。

如在8位单片机的平台下，有如下一个变量定义

```
uint16      g_nTimeCounter   =   0 ;
```

如果移植32单片机的平台下，想要其的范围依旧为16位。

可以直接修改 uint16 的定义，即

```
typedef   unsigned   short   int      uint16 ;
```

这样就可以了，而不需要到源文件处处寻找并修改。

将常用的数据类型全部采用此种方法定义，形成一个头文件，便于我们以后编程直接调用。

文件名 MacroAndConst.h

其内容如下：

```
#ifndef    _MACRO_AND_CONST_H_
#define    _MACRO_AND_CONST_H_
typedef      unsigned int      uint16;
typedef      unsigned int     UINT;
typedef      unsigned int     uint;
typedef      unsigned int     UINT16;
```

```
typedef       unsigned int     WORD;
typedef       unsigned int     word;
typedef          int             int16;
typedef          int             INT16;
typedef       unsigned long    uint32;
typedef       unsigned long        UINT32;
typedef       unsigned long       DWORD;
typedef       unsigned long       dword;
typedef       long                   int32;
typedef       long                   INT32;
typedef       signed    char        int8;
typedef       signed    char        INT8;
typedef       unsigned char          byte;
typedef       unsigned char        BYTE;
typedef       unsigned char        uchar;
typedef       unsigned char        UINT8;
typedef       unsigned char       uint8;
typedef       unsigned char       BOOL;
#endif
```

至此，似乎我们对于源文件和头文件的分工以及模块化编程有那么一点概念了。那么让我们趁热打铁，将上一章的我们编写的 LED 闪烁函数进行模块划分并重新组织进行编译。

在上一章中我们主要完成的功能是 P0口所驱动的 LED 以1Hz 的频率闪烁。其中用到了定时器，以及 LED 驱动模块。因而我们可以简单的将整个工程分成三个模块，定时器模块，LED 模块，以及主函数

对应的文件关系如下

main.c

Timer.c --?Timer.h

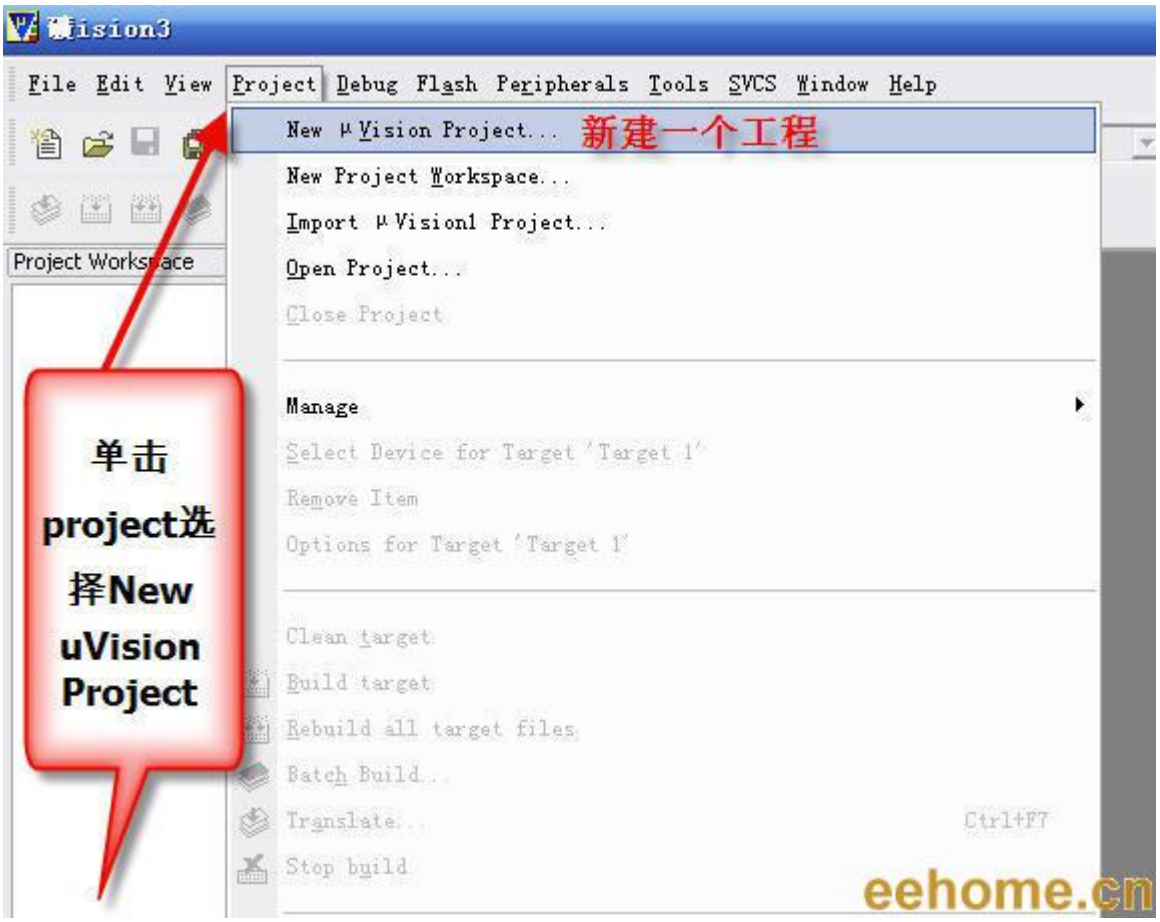
Led.c --?Led.h

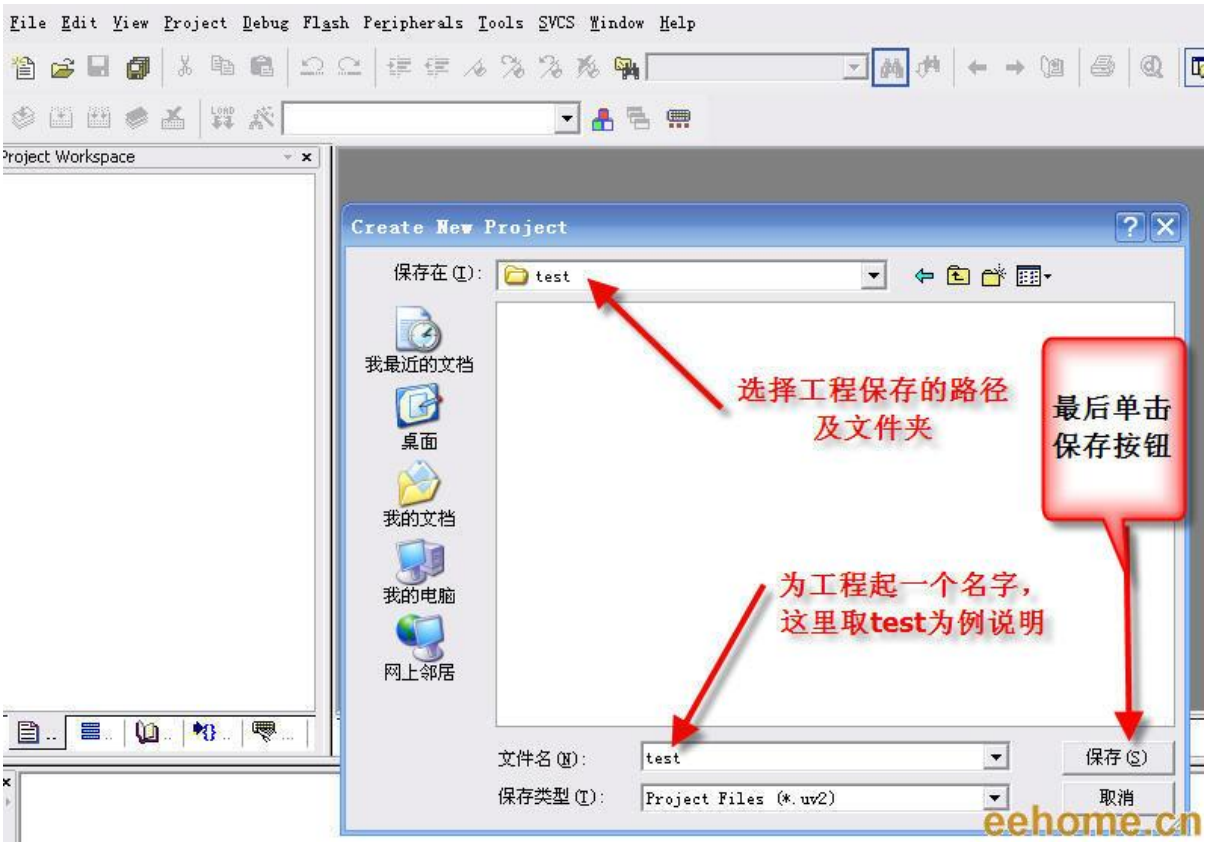
在开始重新编写我们的程序之前，先给大家讲一下如何在 KEIL 中建立工程模板吧，这个模板是我一直沿用至今。

希望能够给大家一点启发。

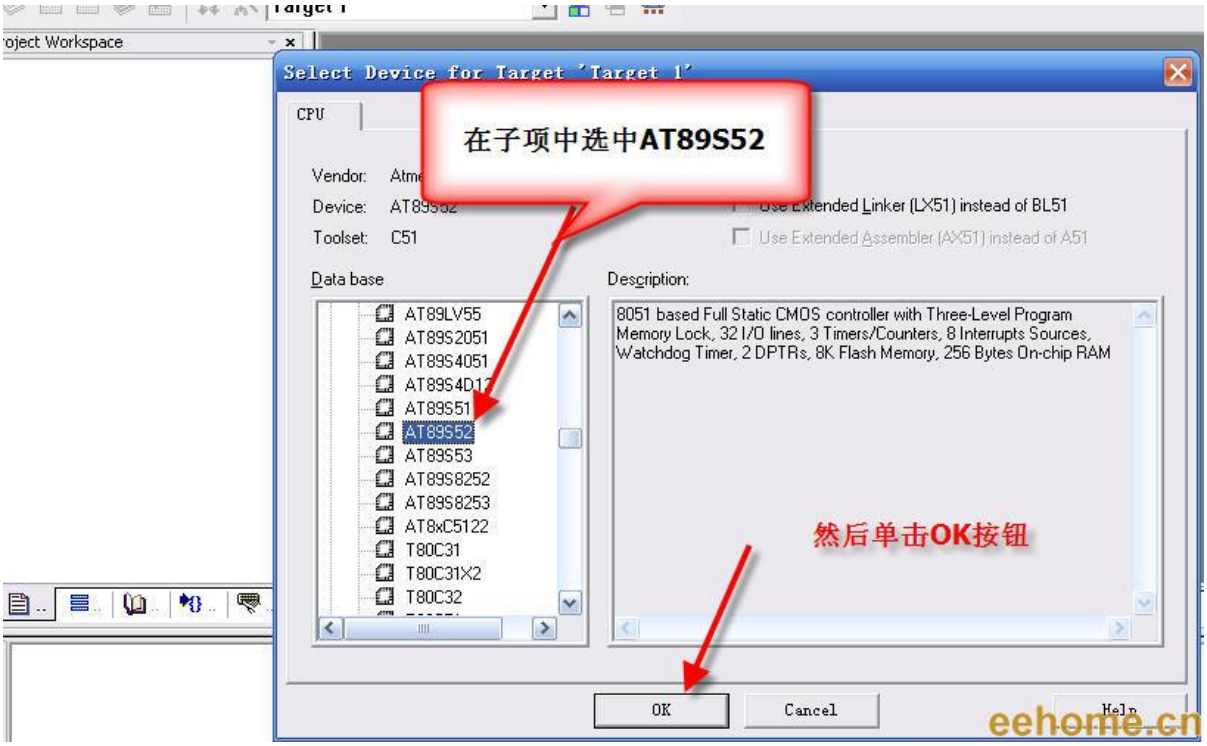
下面的内容就主要以图片为主了。同时辅以少量文字说明。

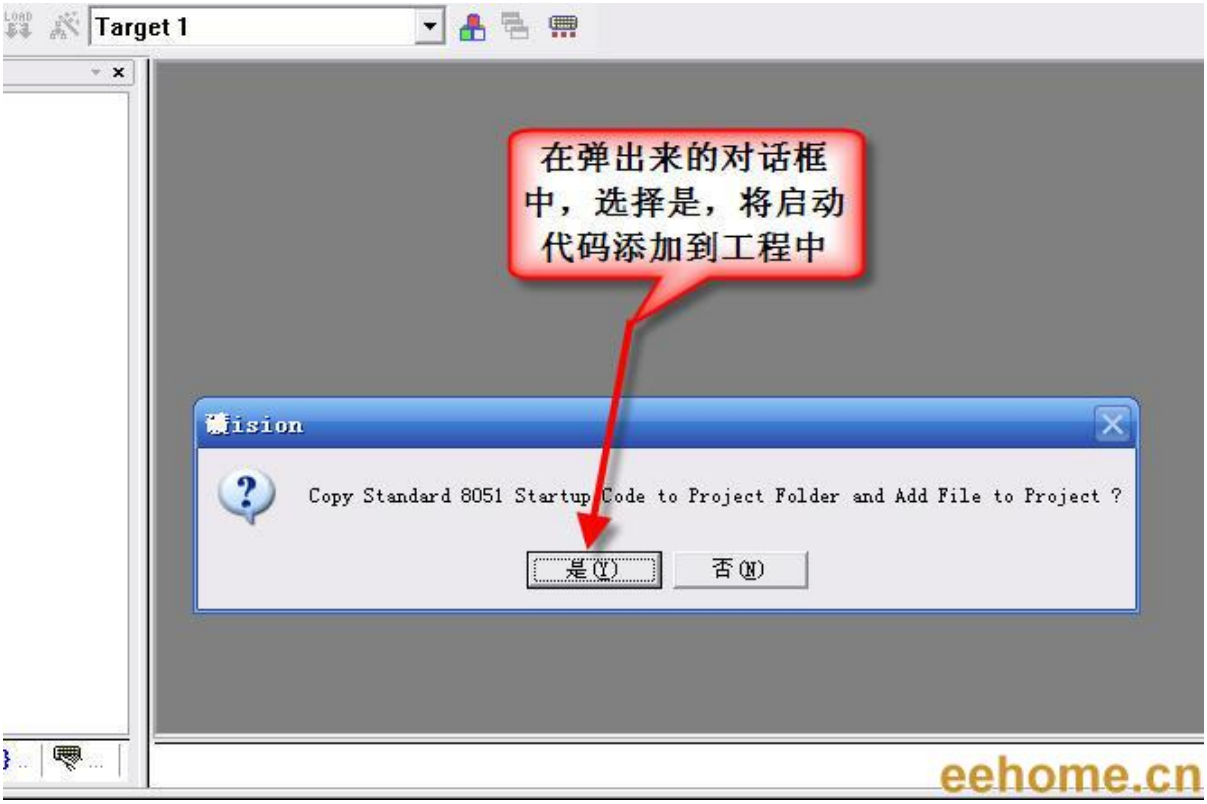
我们以芯片 AT89S52为例。

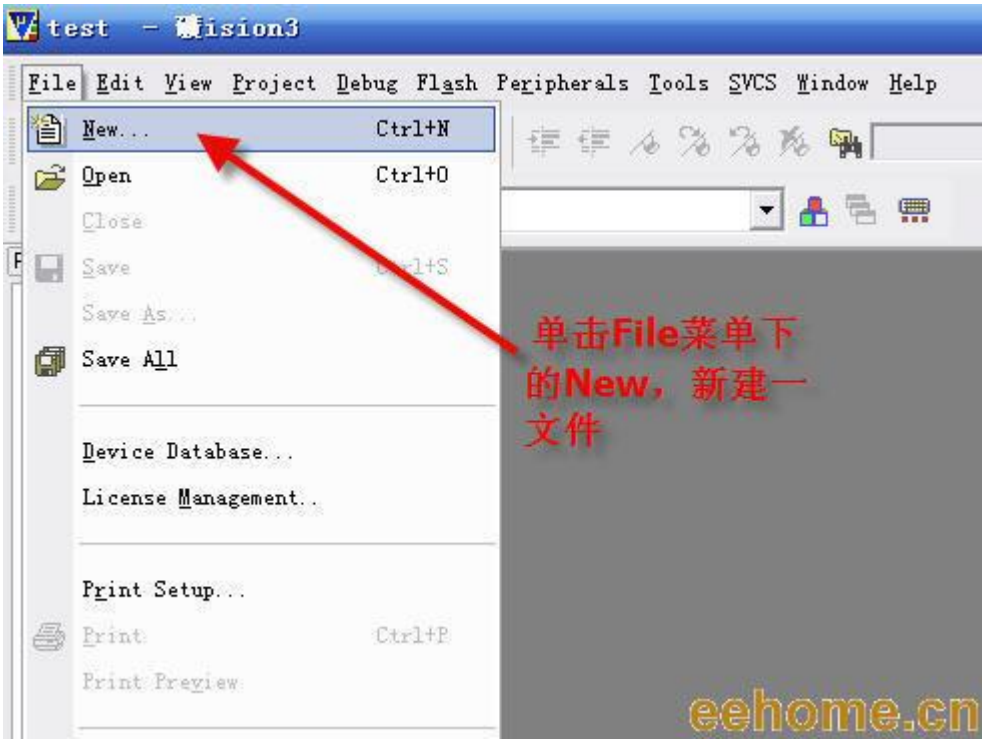


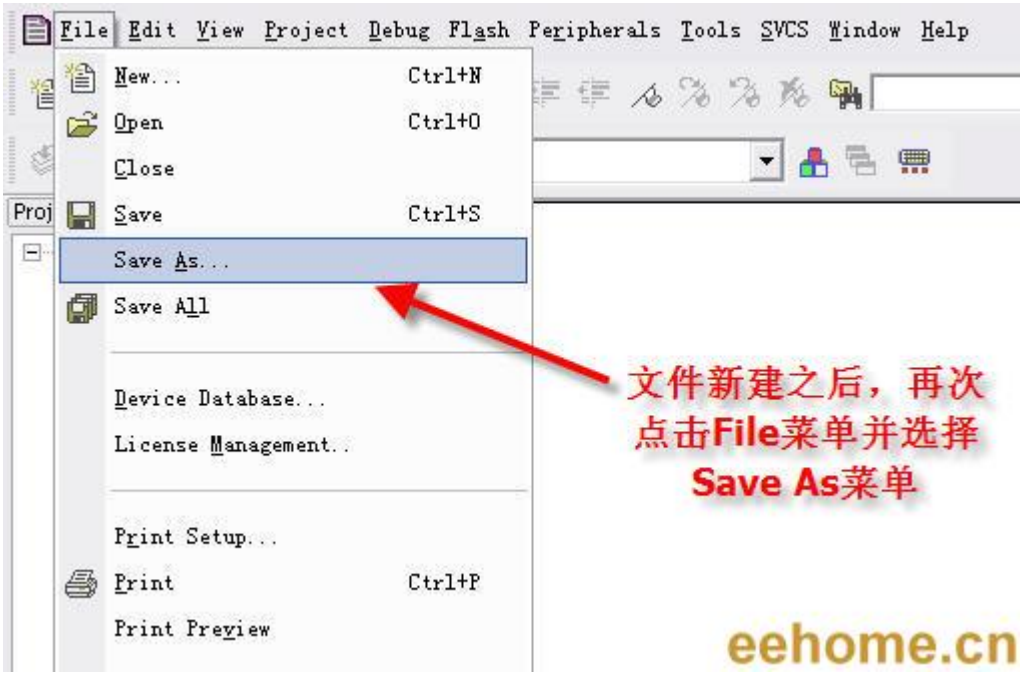

File Edit View Project Debug Flash Peripherals Tools SVCS Window Help
Project Workspace
Create New Project
保存在(I):
test
我最近的文档
桌面
我的文档
我的电脑
网上邻居
选择工程保存的路径
及文件夹
最后单击
保存按钮
为工程起一个名字，
这里取test为例说明
文件名(N):
test
保存(S)
保存类型(T):
Project Files (*.uv2)
取消
eehome.cn


Target 1
Workspace
Select Device for Target 'Target 1'
CPU
Vendor: Acer Labs
Device:
Toolset:
在弹出来的对话框中，通过滚动
条找到Atmel并展开其子项
Data base
Description:
AnchorChips
ASIX Electronics Corporat
Atmel
80C32E
AT48801
AT8032X2
AT80C31X2
AT83/87C5103
AT83/87C5111
AT83/87C5112
AT83C5134
AT83C5135
AT83C5136
OK
Cancel
Help
eehome.cn

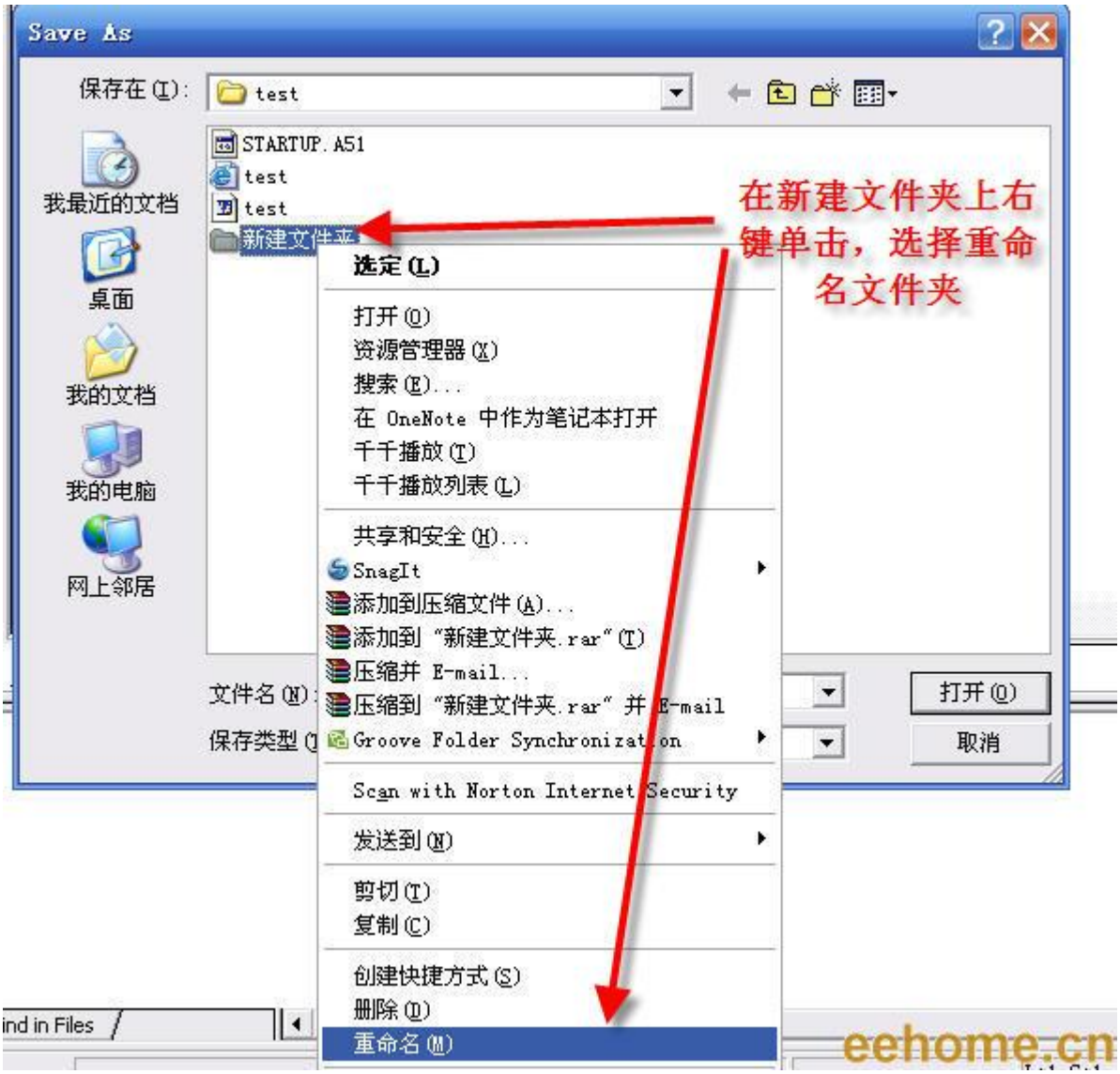

Select Device for Target 'Target 1'
CPU
在子项中选中AT89S52
Vendor:
Device:
Toolset: C51
Use Extended Linker (LX51) instead of BL51
Use Extended Assembler (AX51) instead of A51
Data base
Description:
AT89LV55
AT89S2051
AT89S4051
AT89S51
AT89S52
AT89S53
AT89S8252
AT89S8253
AT8xC5122
T80C31
T80C31X2
T80C32
8051 based Full Static CMOS controller with Three-Level Program Memory Lock, 32 I/O lines, 3 Timers/Counters, 8 Interrupts Sources, Watchdog Timer, 2 DPTRs, 8K Flash Memory, 256 Bytes On-chip RAM
然后单击OK按钮
OK
Cancel
eehome.cn


Target 1
在弹出来的对话框中，选择是，将启动代码添加到工程中
Vision
Copy Standard 8051 Startup Code to Project Folder and Add File to Project ?
是(Y)
否(N)
eehome.cn


test - Vision3
File Edit View Project Debug Flash Peripherals Tools SVCS Window Help
New... Ctrl+N
Open Ctrl+O
Close
Save Ctrl+S
Save As...
Save All
Device Database...
License Management...
Print Setup...
Print Ctrl+P
Print Preview
单击File菜单下的New，新建一文件
eehome.cn

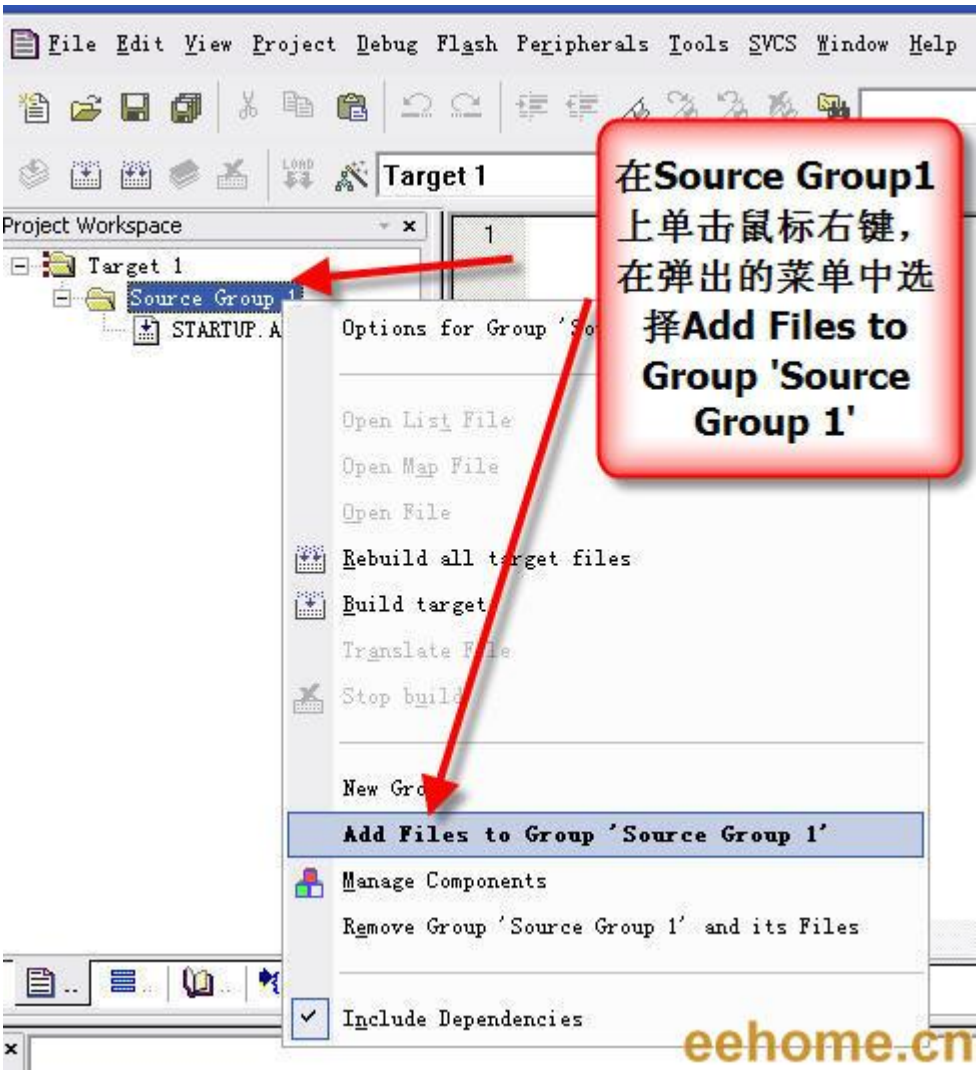

File Edit View Project Debug Flash Peripherals Tools SVCS Window Help
New... Ctrl+N
Open Ctrl+O
Close
Save Ctrl+S
Save As...
Save All
Device Database...
License Management...
Print Setup...
Print Ctrl+P
Print Preview
文件新建之后，再次点击File菜单并选择Save As菜单
eehome.cn


Target 1
Save As
保存在(I):
test
STARTUP.A51
test
test
新建文件夹
我最近的文档
桌面
我的文档
我的电脑
网上邻居
文件名(N):
保存类型(T):
All Files (*.*)
打开(O)
取消
在弹出的保存文件对话框中，单击新建文件夹按钮，新建一个空的文件夹
eehome.cn

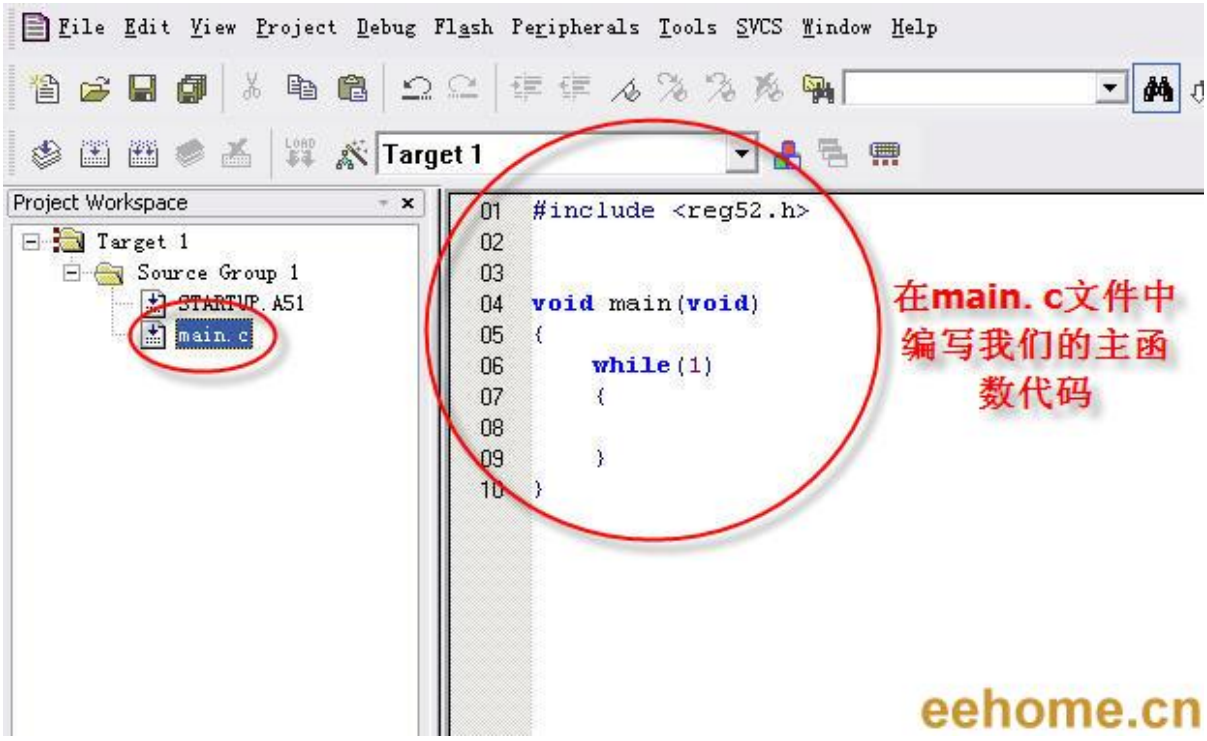

Save As
保存在(I):
test
STARTUP.A51
test
test
新建文件夹
我最近的文档
桌面
我的文档
我的电脑
网上邻居
选定(L)
打开(O)
资源管理器(X)
搜索(E)...
在 OneNote 中作为笔记本打开
千千播放(T)
千千播放列表(L)
共享和安全(H)...
SnagIt
添加到压缩文件(A)...
添加到 "新建文件夹.rar"(T)
压缩并 E-mail...
压缩到 "新建文件夹.rar" 并 E-mail
Groove Folder Synchronization
Scan with Norton Internet Security
发送到(N)
剪切(T)
复制(C)
创建快捷方式(S)
删除(D)
重命名(M)
文件名(N):
保存类型(T):
打开(O)
取消
在新建文件夹上右键单击，选择重命名文件夹
ind in Files
eehome.cn

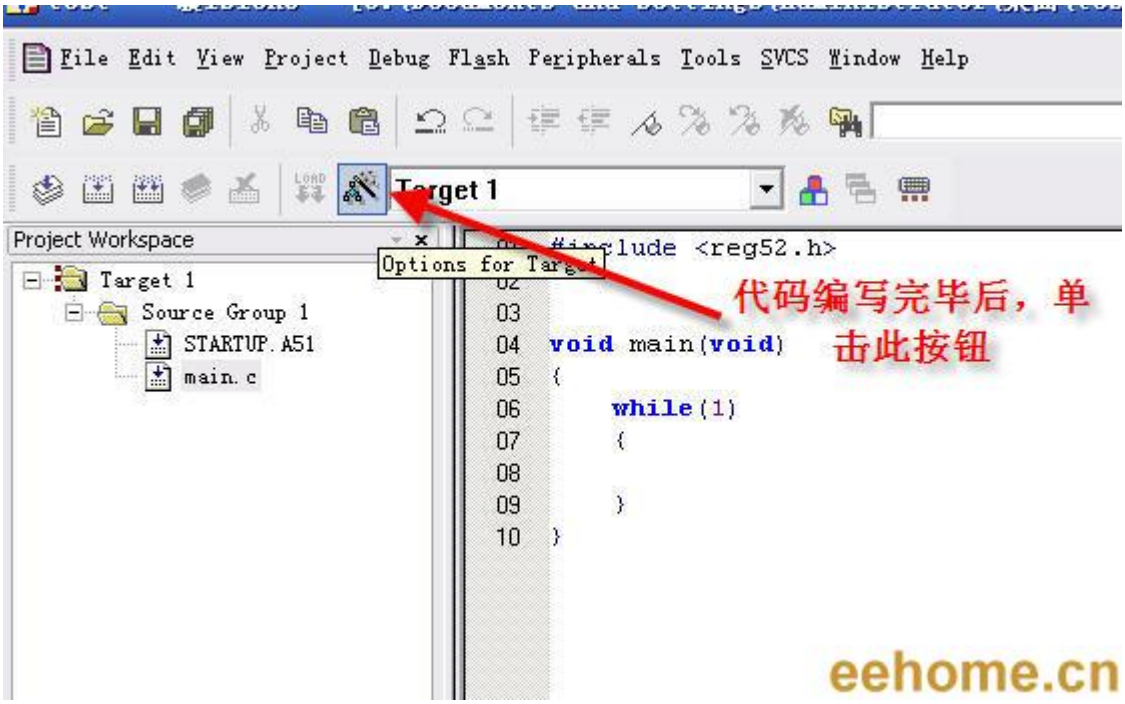

Save As
保存在(I):
test
STARTUP.A51
test
test
src
我最近的文档
桌面
我的文档
我的电脑
网上邻居
将文件夹重命名为src
双击进入文件夹
文件名(N):
保存类型(T):
All Files (*.*)
打开(O)
取消
eehome.cn

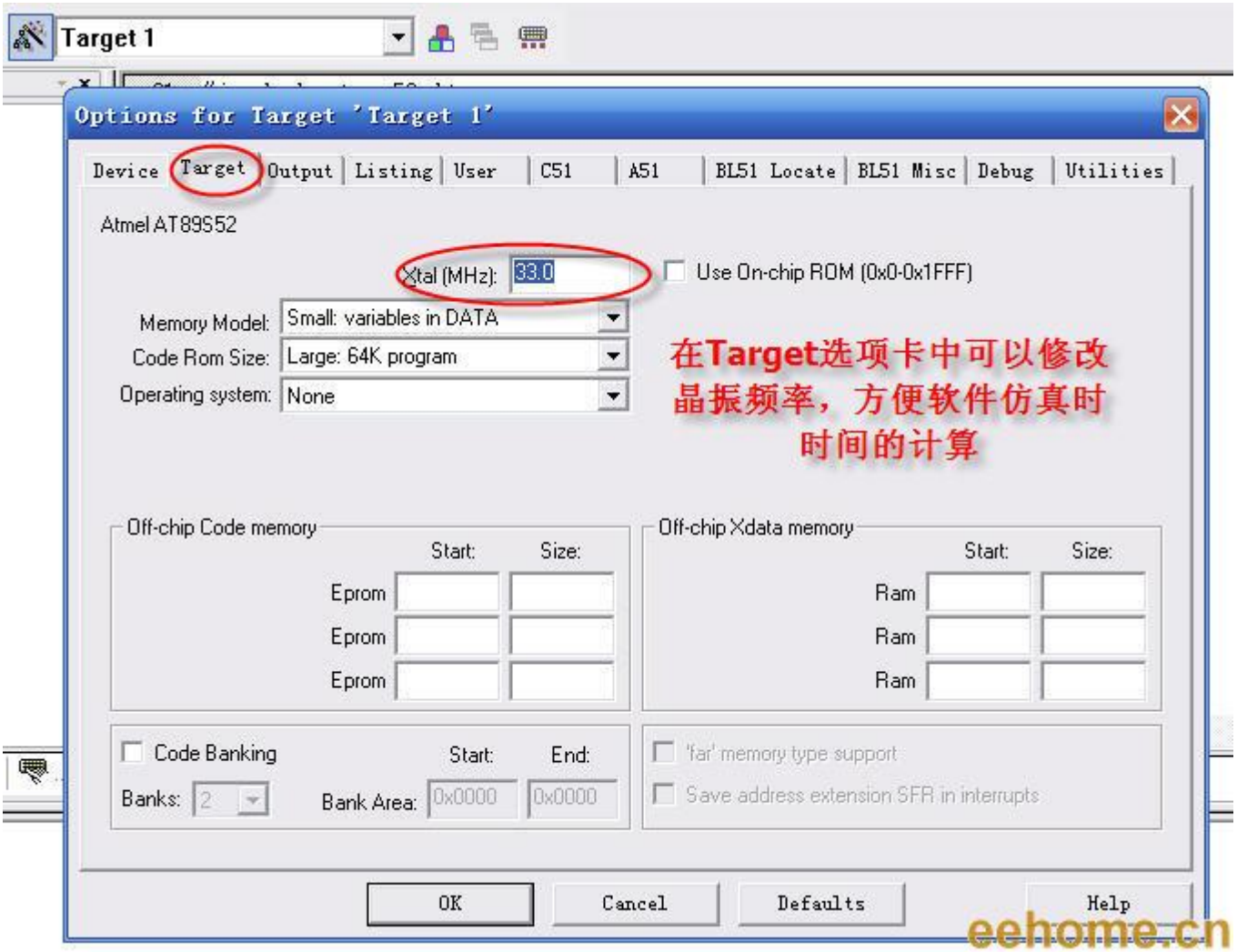

Save As
保存在(I):
src
我最近的文档
桌面
我的文档
我的电脑
网上邻居
给文件取一个名字为
main. c, 然后单击保存按
钮
文件名(N):
main.c
保存类型(T):
All Files (*.*)
保存(S)
取消
eehome.cn

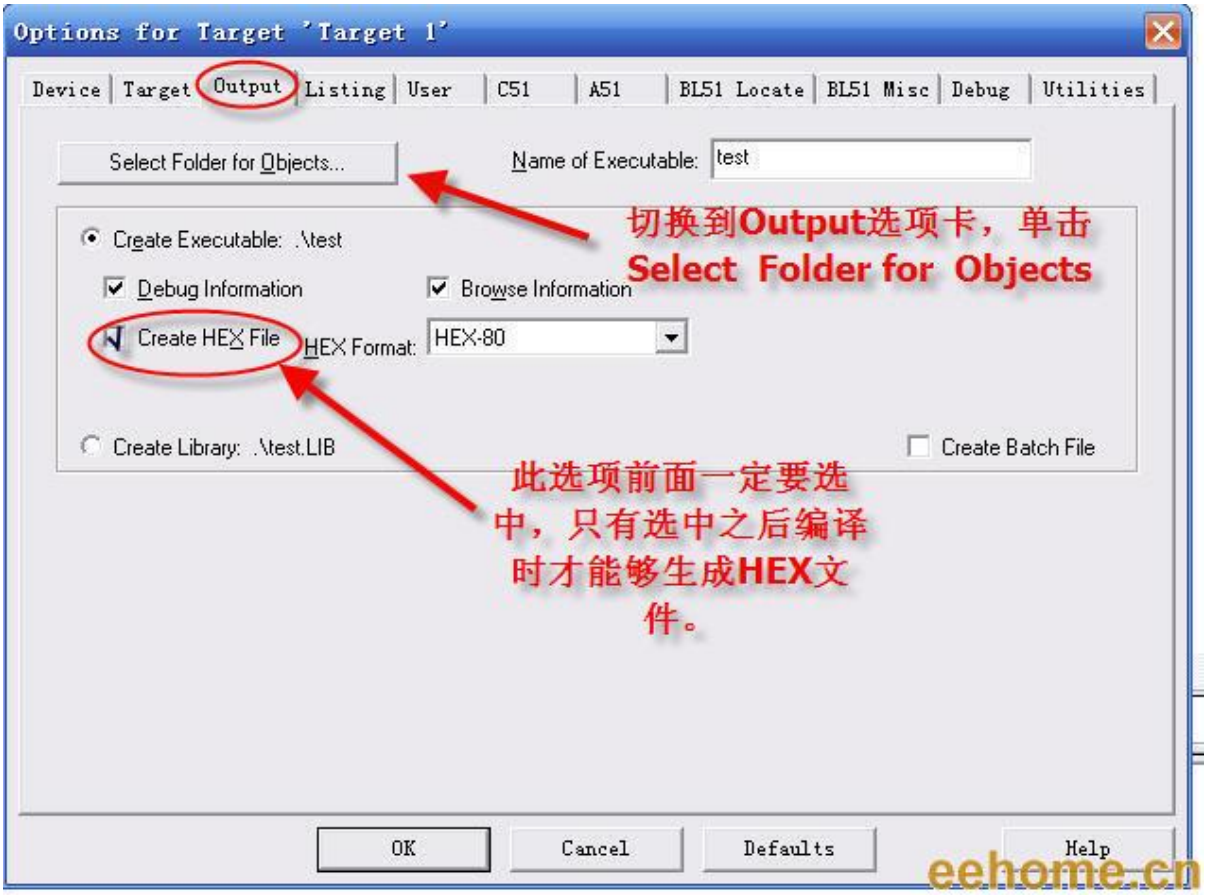

File Edit View Project Debug Flash Peripherals Tools SVCS Window Help
Target 1
Project Workspace
Target 1
Source Group 1
STARTUP.A
在Source Group1上单击鼠标右键，在弹出的菜单中选择Add Files to Group 'Source Group 1'
Options for Group 'So
Open List File
Open Map File
Open File
Rebuild all target files
Build target
Translate File
Stop build
New Gro
Add Files to Group 'Source Group 1'
Manage Components
Remove Group 'Source Group 1' and its Files
Include Dependencies
eehome.cn

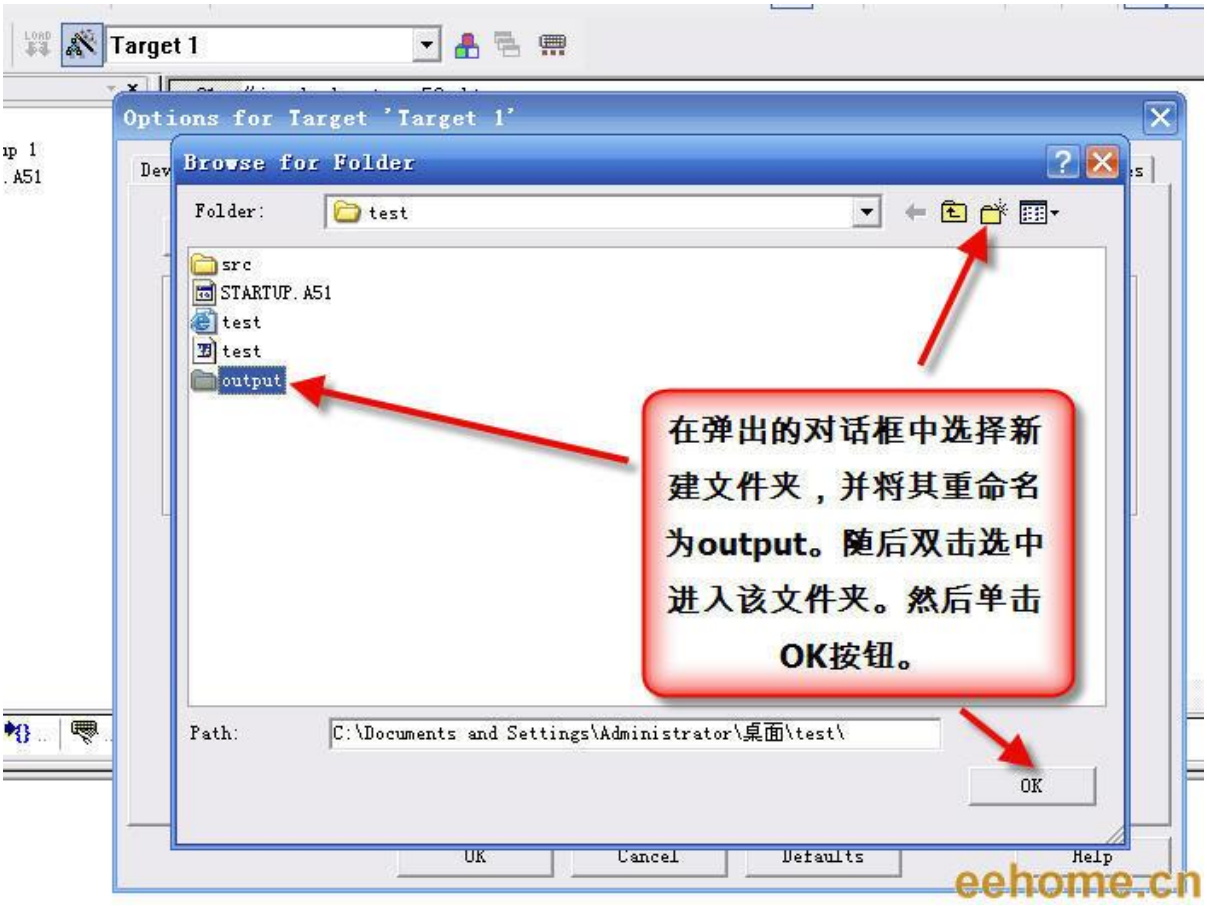

Add Files to Group 'Source Group 1'
查找范围(I):
src
main.c
选中我们之前新建的src文件夹，然后选中main.c文件，单击Add按钮后，再单击Close按钮
文件名(N):
main
文件类型(T):
C Source file (*.c)
Add
Close
eehome.cn

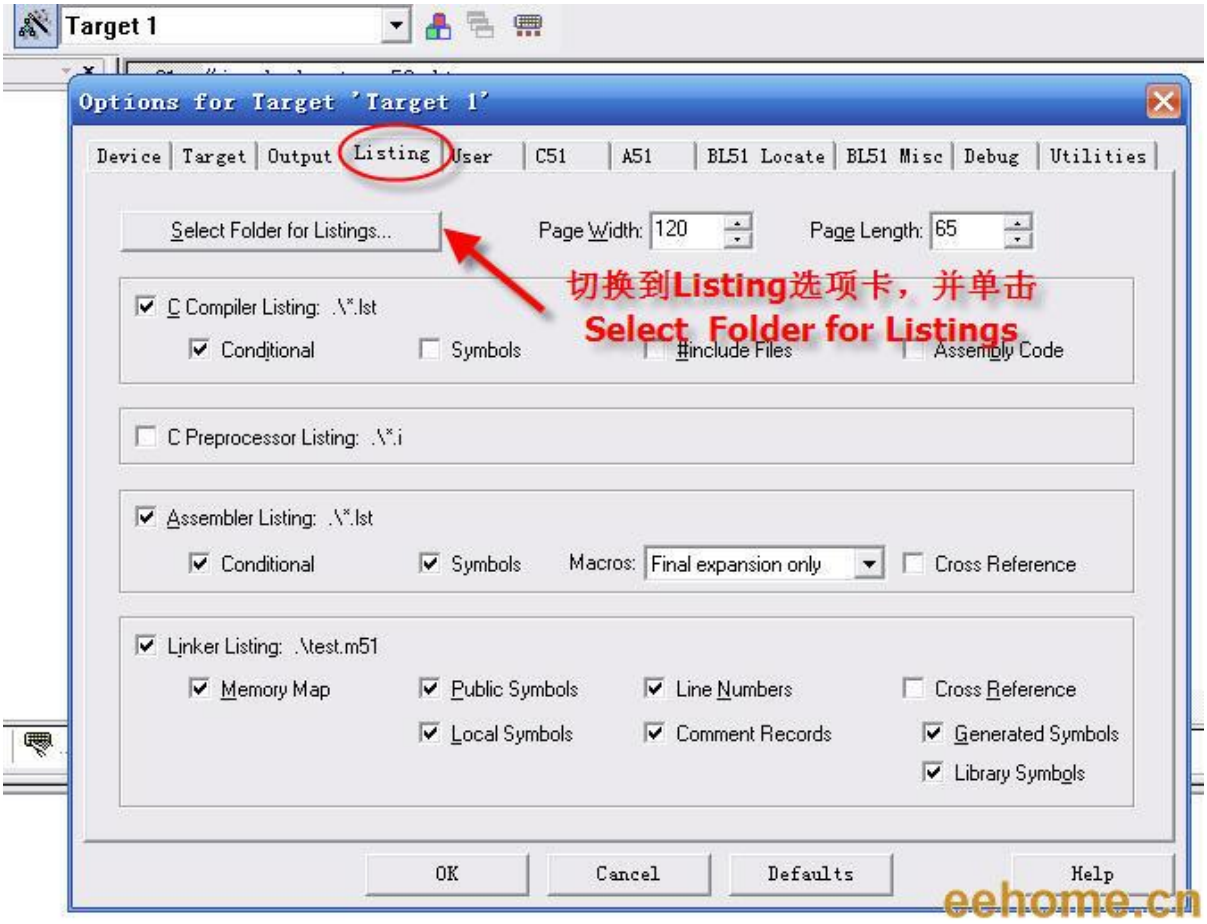

File Edit View Project Debug Flash Peripherals Tools SVCS Window Help
Target 1
Project Workspace
Target 1
Source Group 1
STARTUP.A51
main.c
#include <reg52.h>
void main(void)
{
while(1)
{
}
}
在main.c文件中编写我们的主函数代码
eehome.cn

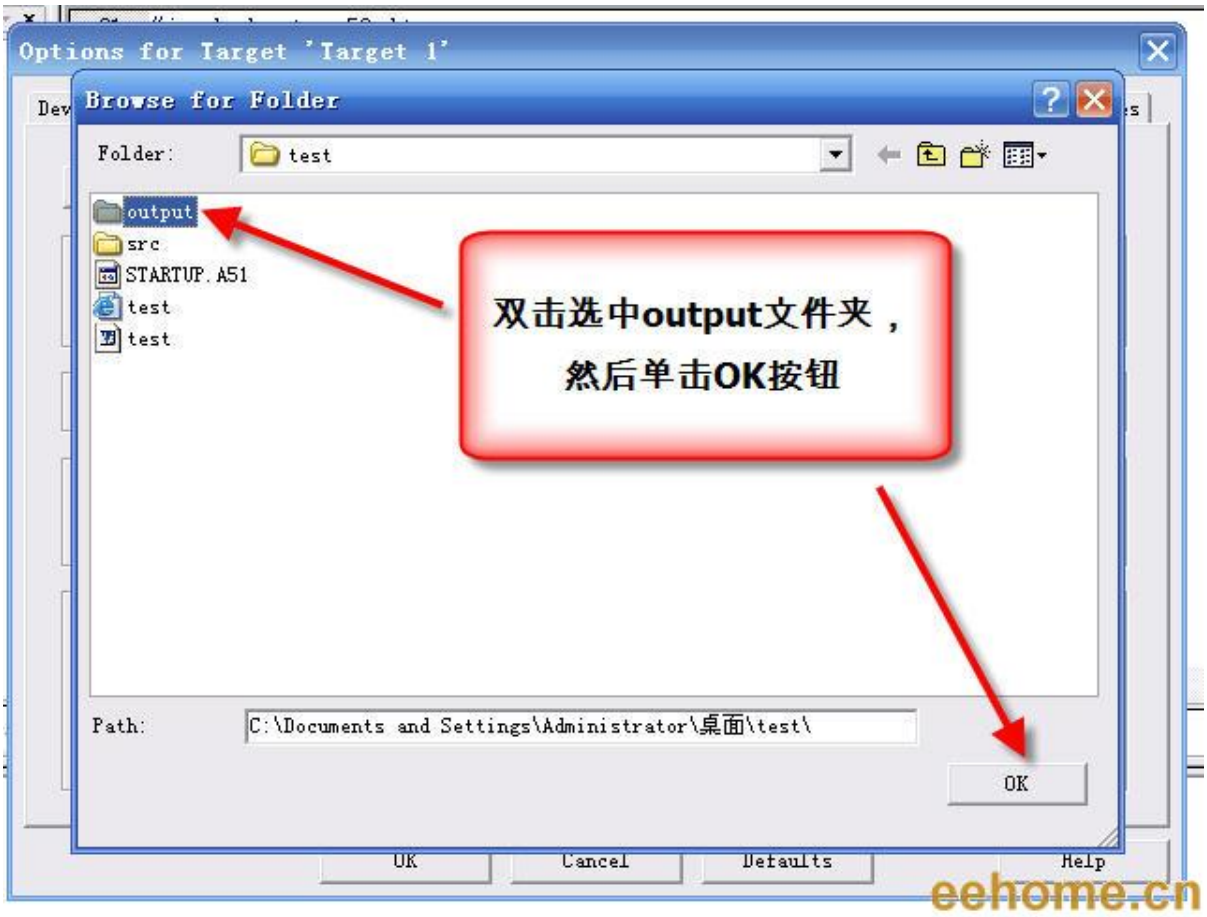

File Edit View Project Debug Flash Peripherals Tools SVCS Window Help
Target 1
Project Workspace
Options for Target
Target 1
Source Group 1
STARTUP.A51
main.c
#include <reg52.h>
void main(void)
{
while(1)
{
}
}
代码编写完毕后，单击此按钮
eehome.cn

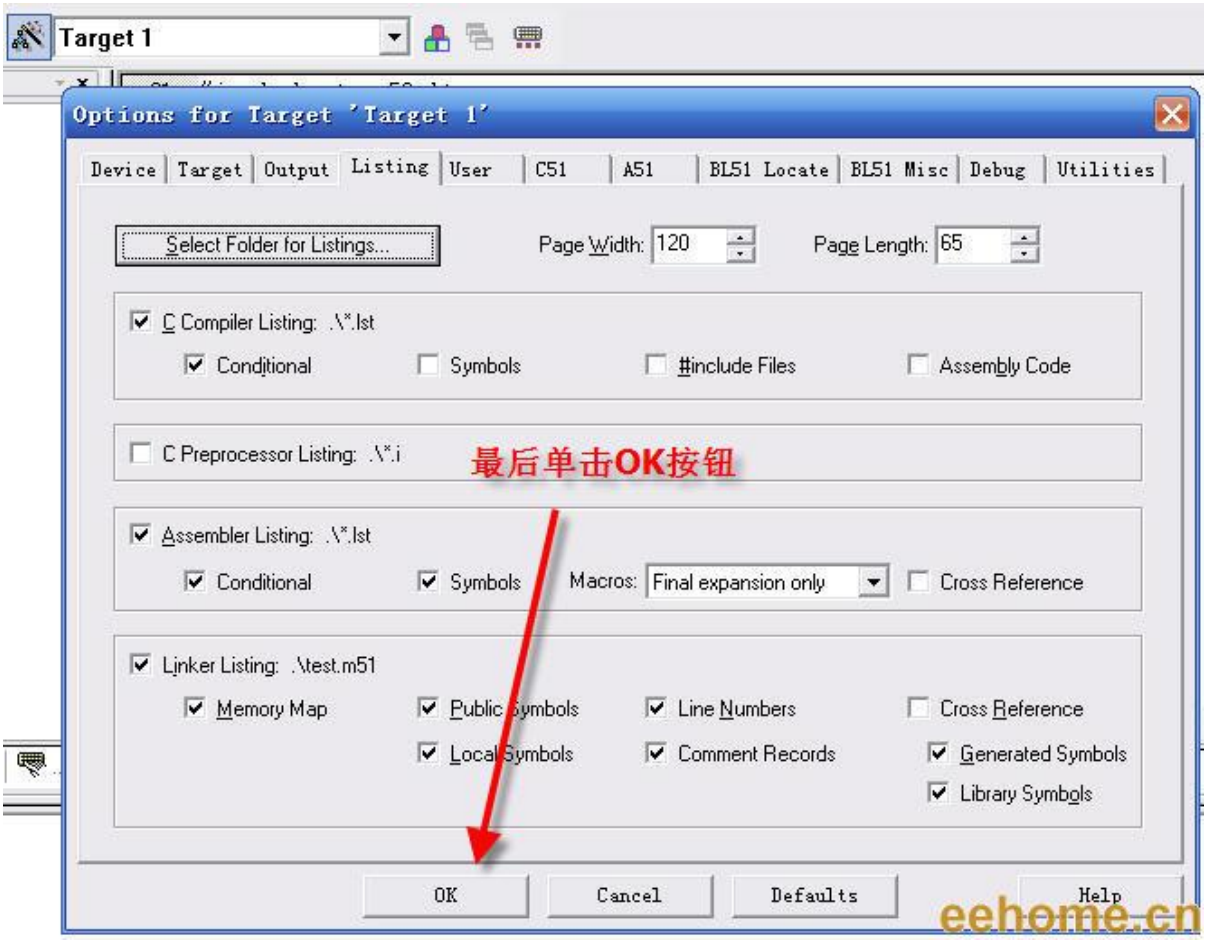

Target 1
Options for Target 'Target 1'
Device
Target
Output
Listing
User
C51
A51
BL51 Locate
BL51 Misc
Debug
Utilities
Atmel AT89S52
Xtal (MHz): 33.0
Use On-chip ROM (0x0-0x1FFF)
Memory Model: Small: variables in DATA
Code Rom Size: Large: 64K program
Operating system: None
在Target选项卡中可以修改晶振频率，方便软件仿真时时间的计算
Off-chip Code memory
Start:
Size:
Eprom
Eprom
Eprom
Off-chip Xdata memory
Start:
Size:
Ram
Ram
Ram
Code Banking
Start:
End:
Banks: 2
Bank Area: 0x0000 0x0000
'far' memory type support
Save address extension SFR in interrupts
OK
Cancel
Defaults
Help
eehome.cn

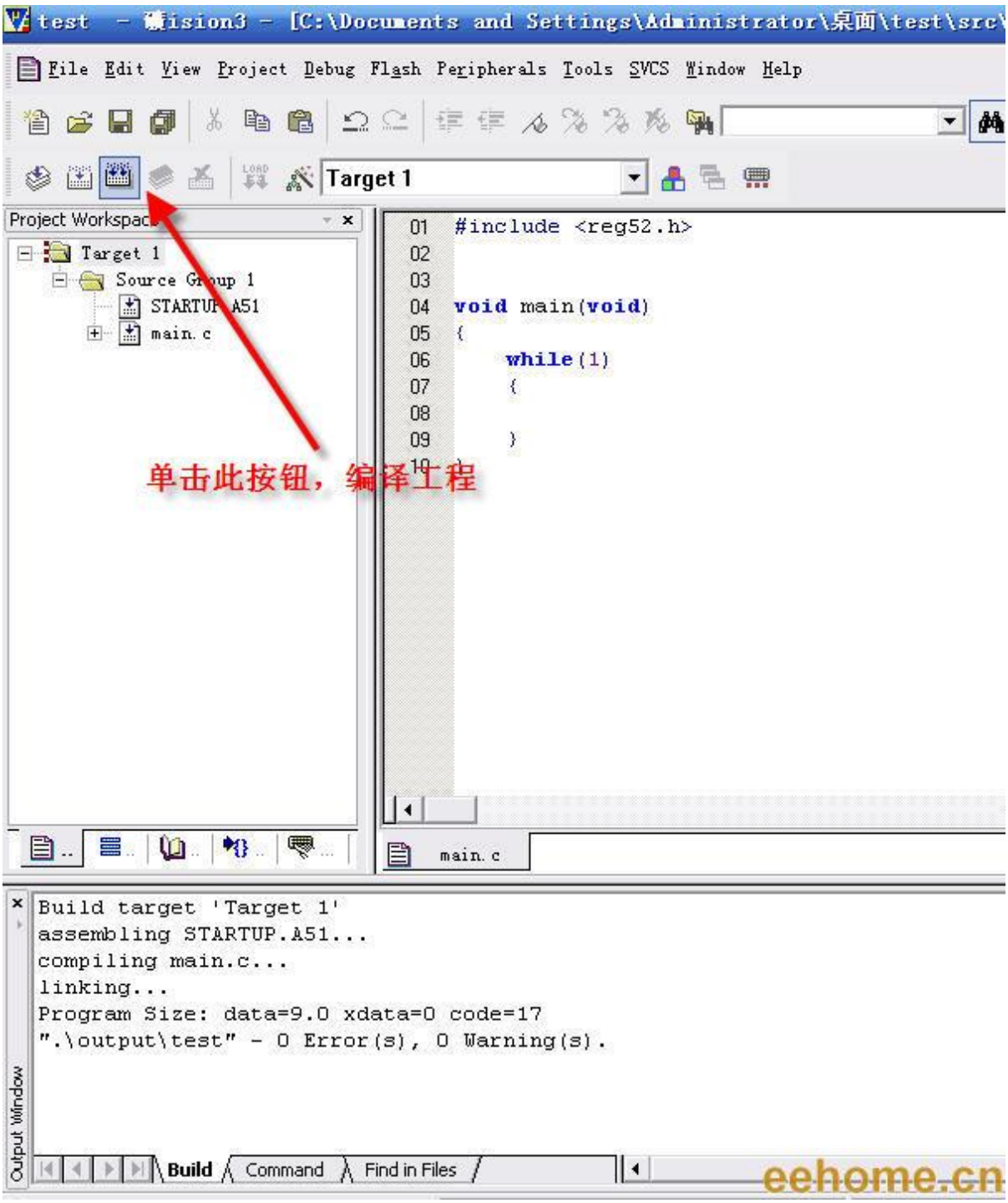

Options for Target 'Target 1'
Device
Target
Output
Listing
User
C51
A51
BL51 Locate
BL51 Misc
Debug
Utilities
Select Folder for Objects...
Name of Executable: test
Create Executable: .\test
Debug Information
Browse Information
Create HEX File
HEX Format: HEX-80
Create Library: .\test.LIB
Create Batch File
OK
Cancel
Defaults
Help
切换到Output选项卡，单击
Select Folder for Objects
此选项前面一定要选
中，只有选中之后编译
时才能够生成HEX文
件。
eehome.cn


Target 1
Options for Target 'Target 1'
Browse for Folder
Folder:
test
src
STARTUP.A51
test
test
output
在弹出的对话框中选择新建文件夹，并将其重命名为output。随后双击选中进入该文件夹。然后单击OK按钮。
Path:
C:\Documents and Settings\Administrator\桌面\test\
OK
OK
Cancel
Defaults
Help
eehome.cn


Target 1
Options for Target 'Target 1'
Device
Target
Output
Listing
User
C51
A51
BL51 Locate
BL51 Misc
Debug
Utilities
Select Folder for Listings...
Page Width: 120
Page Length: 65
切换到Listing选项卡，并单击
Select Folder for Listings
C Compiler Listing: .*.lst
Conditional
Symbols
#include Files
Assembly Code
C Preprocessor Listing: .*.i
Assembler Listing: .*.lst
Conditional
Symbols
Macros:
Final expansion only
Cross Reference
Linker Listing: .\test.m51
Memory Map
Public Symbols
Line Numbers
Cross Reference
Local Symbols
Comment Records
Generated Symbols
Library Symbols
OK
Cancel
Defaults
Help
eehome.cn


Options for Target 'Target 1'
Browse for Folder
Folder:
test
output
src
STARTUP.A51
test
test
双击选中output文件夹，
然后单击OK按钮
Path:
C:\Documents and Settings\Administrator\桌面\test\
OK
OK
Cancel
Defaults
Help
eehome.cn


Target 1
Options for Target 'Target 1'
Device
Target
Output
Listing
User
C51
A51
BL51 Locate
BL51 Misc
Debug
Utilities
Select Folder for Listings...
Page Width: 120
Page Length: 65
C Compiler Listing: .*.lst
Conditional
Symbols
#include Files
Assembly Code
C Preprocessor Listing: .*.i
最后单击OK按钮
Assembler Listing: .*.lst
Conditional
Symbols
Macros: Final expansion only
Cross Reference
Linker Listing: .\test.m51
Memory Map
Public Symbols
Line Numbers
Cross Reference
Local Symbols
Comment Records
Generated Symbols
Library Symbols
OK
Cancel
Defaults
Help
eehome.cn


test - Vision3 - [C:\Documents and Settings\Administrator\桌面\test\src\
File Edit View Project Debug Flash Peripherals Tools SVCS Window Help
Target 1
Project Workspace
Target 1
Source Group 1
STARTUP.A51
main.c
01 #include <reg52.h>
02
03
04 void main(void)
05 {
06 while(1)
07 {
08
09 }
10 }
单击此按钮，编译工程
main.c
Build target 'Target 1'
assembling STARTUP.A51...
compiling main.c...
linking...
Program Size: data=9.0 xdata=0 code=17
".\output\test" - 0 Error(s), 0 Warning(s).
Output Window
Build
Command
Find in Files
eehome.cn

OK ，到此一个简单的工程模板就建立起来了，以后我们再新建源文件和头文件的时候，就可以直接保存到 src 文件目录下面了。

下面我们开始编写各个模块文件。

首先编写 Timer.c 这个文件主要内容就是定时器初始化，以及定时器中断服务函数。其内容如下。

```
#include <reg52.h>

bit g_bSystemTime1Ms = 0 ;                    // 1MS 系统时标


void Timer0Init(void)

{

    TMOD &= 0xf0 ;

    TMOD |= 0x01 ;        //定时器0工作方式1

    TH0   =     0xfc ;        //定时器初始值

    TL0   =   0x66 ;

    TR0   = 1 ;

    ET0   = 1 ;

}

void Time0Isr(void) interrupt 1

{

    TH0   =     0xfc ;              //定时器重新赋初值

    TL0   =   0x66 ;
```

```
    g_bSystemTime1Ms = 1 ;      //1MS 时标标志位置位
}
```

由于在 Led.c 文件中需要调用我们的 g_bSystemTime1Ms 变量。同时主函数需要调用 Timer0Init()初始化函数，所以应该对这个变量和函数在头文件里作外部声明。以方便其它函数调用。

Timer.h 内容如下。

```
#ifndef _TIMER_H_
#define _TIMER_H_
extern void Timer0Init(void) ;
extern bit g_bSystemTime1Ms ;
#endif
```

完成了定时器模块后，我们开始编写 LED 驱动模块。

Led.c 内容如下：

```
#include <reg52.h>
#include "MacroAndConst.h"
#include "Led.h"
#include "Timer.h"
static uint16   g_u16LedTimeCount = 0 ; //LED 计数器
static uint8    g_u8LedState = 0 ;        //LED 状态标志, 0表示亮，1表示熄灭
#define LED P0                  //定义 LED 接口
#define LED_ON()        LED = 0x00 ;   //所有 LED 亮
#define LED_OFF()       LED = 0xff ;   //所有 LED 熄灭
void LedProcess(void)
{
    if(0 == g_u8LedState)   //如果 LED 的状态为亮，则点亮 LED
    {
        LED_ON() ;
    }
    else                    //否则熄灭 LED
```

```
    {
        LED_OFF() ;
    }
}

void LedStateChange(void)
{
    if(g_bSystemTime1Ms)                    //系统1MS 时标到
    {
        g_bSystemTime1Ms = 0 ;
        g_u16LedTimeCount++ ;          //LED 计数器加一
        if(g_u16LedTimeCount >= 500) //计数达到500,即500MS 到了,改变 LED 的状态。
        {
            g_u16LedTimeCount = 0 ;
            g_u8LedState    = ! g_u8LedState       ;
        }
    }
}
```

这个模块对外的借口只有两个函数，因此在相应的 Led.h 中需要作相应的声明。

Led.h 内容：

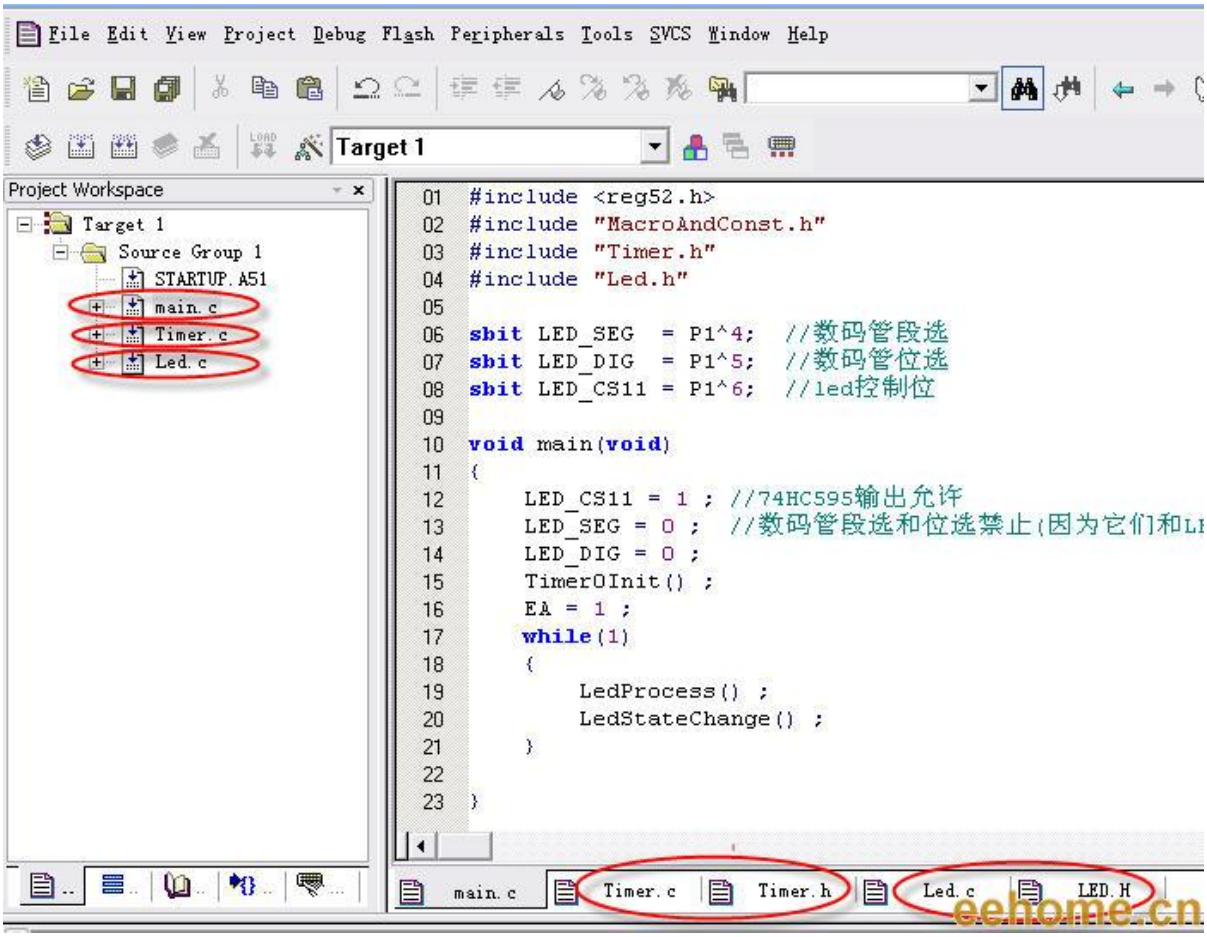
```
#ifndef _LED_H_
#define _LED_H_
extern void LedProcess(void) ;
extern void LedStateChange(void) ;
#endif
```

这两个模块完成后，我们将其 C 文件添加到工程中。然后开始编写主函数里的代码。

如下所示：

```
#include <reg52.h>
#include "MacroAndConst.h"
#include "Timer.h"
#include "Led.h"
sbit LED_SEG    = P1^4;    //数码管段选
sbit LED_DIG    = P1^5;    //数码管位选
sbit LED_CS11 = P1^6;    //led 控制位
void main(void)
{
            LED_CS11 = 1 ; //74HC595输出允许
      LED_SEG = 0 ;    //数码管段选和位选禁止(因为它们和 LED 共用 P0口)
            LED_DIG = 0 ;
            Timer0Init() ;
            EA = 1 ;
            while(1)
          {
      LedProcess() ;
      LedStateChange() ;
          }
}
```

整个工程截图如下：



至此，第三章到此结束。

一起来总结一下我们需要注意的地方吧

1. C 语言源文件(*.c)的作用是什么
2. C 语言头文件(*.h)的作用是什么
3. typedef 的作用
4. 工程模板如何组织
5. 如何创建一个多模块(多文件)的工程