定义，所以不能再重新定义，需要加以保留。用户在定义自己的变量或者函数的时候千万不要使用关键字，否则就会出现一些错误。在 C 语言的编译系统中主要有以下几种类型的关键字。

(1) 数据类型关键字

数据类型关键字包括 auto、char、const、double、enum、extern、float、int、long、register、sizeof、short、static、struct、typedef、union、unsigned、void、volitile 等。该类型关键字主要用于定义一些变量或者函数。以后编程中最常用的如 char、int 等，例如，“int nlndex;”语句就是定义一个 int 类型的数据。这里不同的关键字有不同的含义，需要在使用的时候加以区分，总结起来就是一句话：关键字不能当做普通的变量来使用。

(2) 程序控制关键字

程序控制关键字包括 break、case、continue、default、do、else、for、goto、if、return、switch、while 等。该类型的关键字主要用于程序的控制。例如：

```
int n = 9;                    //定义一个整型变量 n,并将其赋值为 9
int m;                        //定义一个整型变量 m,如果没有赋值,一般其值默认为 0
if(n<10)                      //上面已经将 n 赋值为 9 了,所以条件(n<10)成立
{
    m = 10;
}
else                          //n 大于或等于 10 时执行
{
    m = 11;
}
```

以上代码运行后的结果是 m=10。

(3) 预处理功能关键字

预处理功能关键字包括 define、end、if、include 等。该类型的关键字主要用于进行预处理，这些实际上用得不多。例如，“#include <msp430x16x.h>”表示包括 msp430x16x.h 头文件，这个头文件在以后编程中还会遇到。说明一点，不要看这么多的关键字，实际上有些关键字以后根本就用不到，真正用到的就那么十几个而已。

2.2 数据基本类型

标准 C 语言中数据主要有整型、实型和字符型。下面就这几种类型进行具体介绍。

1. 整型数据

这是一种比较常用的数据类型，整型数据中主要包括 int、short、long(三者之间的区别见

表 2-1)等。不同整数类型的变量具有不同的整数范围。MSP430 的 C 语言除了支持标准的 C 语言整数类型外,还支持其他几种整数类型。表 2-1 给出了 MSP430 系列单片机的 C 语言支持的几种整数类型,这张表格中每一种数据的类型所表示的数据范围是值得注意的,如果在进行计算的时候超出了数制所能表示的范围而自己还没有意识到,就要出大问题了!

表 2-1　MSP430 系列单片机的 C 语言支持的整数类型

整数类型	字　节	数值范围	说　明
sfrb	1		定义特殊功能寄存器
sfrw	2		定义特殊功能寄存器
unsigned char	1	0～255	无符号字符
char(默认)	1	0～255	等效于 unsigned char
signed char	1	−128～127	有符号字符
char (−c 选项)−	1	−128～127	等效于 signed char
short	2	−32768～32767	短整数
int	2	−32768～32767	整数
unsigned short	2	0～65535	无符号短整数
unsigned int	2	0～65535	无符号整数
long	4	−2147483648～2147483647	长整数
unsigned long	4	0～4294967295	无符号长整数
float	4	±1.18E−38～±3.39E+38	浮点数
double	4	±1.18E−38～±3.39E+38	双精度浮点数
long double	4	±1.18E−38～±3.39E+38	长双精度浮点数
pointer	2		指针
enum	1～4		枚举

整型变量的定义如下:

```
main()                    //主程序开始
{
   int sum,n,m;
//定义三个整型变量 sum、n、m,这里要注意,每个整型变量占用两个存储空间,这两个存储空间都在内部
//RAM 单元中,由编译软件分配
   n = 12;                //将整型变量赋值为 12
   m = 0x15;              //将整型变量赋值为 0x15;0x 是十六进制表示形式,0x15 化成十进制是 21
   sum = n + m;           //将 n、m 进行求和
```

```
}
```

上面的例子给出了整型变量的定义方法。其他两种变量定义也可以参考这种方法。另外，在上面的例子中，整数有常用的两种形式：十进制和十六进制。在数值的前面加上 0x 就表示十六进制数了。

2. 实型数据

实型数据就是 C 语言中的浮点数据。它可以含有小数点，但是它表示的数是有精度的。实型数据有两种具体的表现形式：十进制小数点形式和指数形式。小数点形式如 10.1234。指数形式包括整数部分、尾数部分和指数部分，具体形式如 1.21E+1（十进制的值 1.21×10^1）。实型变量主要有 float 型、double 型和 long double 型。实型变量的定义方法很简单。例如：

```
float a;                    //定义一个 float 型的实型数据 a
double b;                   //定义一个 double 型的实型数据 b
```

3. 字符型数据

字符型数据主要处理与字符相关的内容，比如英文字母或者汉语句子。一般来说，会将多个字符型变量组成一个字符串来使用。在 C 语言中，字符是与 ASCII 码的值对应的，一个字符占一个字节，例如，英文字母 A 的 ASCII 码值是 41H。字符的 ASCII 码值可以在码表中查找到。在这里需要强调一下，整数和字符常量在表现形式上是有区别的，比如加单引号的'5'表示的是字符，而 5 表示的是整数；所以，字符是以单引号表示的，而单引号的输入要在英文状态下才有效，不能在汉字的状态下输入。

字符变量主要包括 char 型的字符变量，字符变量的定义方法非常简单。例如：

```
char chrTemp;
chrTemp = 'A';
```

上面的例子定义了字符变量 chrTemp，并给它赋值为'A'。在使用字符变量时，需要了解转义字符。转义字符是一种特殊的字符，通常使用转义符表示 ASCII 码字符中不可打印的控制字符和特殊功能字符。转义字符是使用反斜杠（\）后面跟一个字符来表示。例如，“\n”表示回车，“\'”表示单引号。当然这些所谓的转义字符一般用到的较少，用到时再查也来得及。在某些应用场合需要不同数据类型的转换，如把字符类型的变量转换成整数类型，把 int 类型的数据转换成 long 类型的数据等。在这里，进行数据类型转换的方法是采用强制转换类型。下面例子给出的是数据之间的转换。

```
main()
{
    char chrTemp;
    int n;
```

```
    chrTemp = 'A';
    n = (int)(chrTemp);      //强制转换类型,将字符型数据强制转换成整型数据
  while(1);                  //等待或执行其他程序
  }
```

上面的例子是从字符类型转换成 int 类型,n 的最终值为 65。需要注意的是:并不是所有类型之间都可以进行类型转换;当占字节多的类型转换成占字节少的类型时,有可能会造成数据的丢失。基于以上原因,使用类型转换的时候需要特别小心。

2.3 C 语言的运算符

C 语言的内部运算比较丰富,比如可以直接进行加、减、乘、除等运算而不再像汇编语言那么繁琐。C 语言的运算主要包括算术运算、关系运算、逻辑运算、赋值运算和位运算。下面简要介绍各种具体的运算。

1. 算术运算

算术运算主要是指加、减、乘、除等运算。表 2-2 给出了算术运算符。

表 2-2 算术运算符

运算符	含 义	说 明
++	单目加(加 1 操作)	只有一个操作数
—	单目减(减 1 操作)	只有一个操作数
+	加	需要两个操作数
—	减	需要两个操作数
*	乘	需要两个操作数
/	除	当两个操作数是整数的时候,结果为整数
%	模运算(求余)	操作数必须是整数

下面给出算术运算的例子。

```
main()
{
    int n,m;
    int y;
    n = 5;
    m = 2;
    y = m + n;              //y 的值为 7
```

```
    y=m * n;                //y 的值为 10
    y=m / n;                //y 的值为 2
    y=m % n;                //y 的值为 1
    n++;                    //执行这句代码后,n 的值为 6
}
```

上面的例子比较简单,复杂的运算也是同样如此。

2. 关系运算

关系运算符主要是对操作数进行某种逻辑上的判断,结果只有 true(结果为真)和 false(结果为假)两种结果。表 2-3 给出了关系运算符。

表 2-3 关系运算符

关系运算符	含 义	举例(设 a=3,b=4)
>	大于	a>b 结果:false
>=	大于或等于	a>b 结果:false
==	等于	a==b 结果:false
<	小于	a<b 结果:true
<=	小于或等于	a<=b 结果:true
!=	不等于	a!=b 结果:true

由表 2-3 可知,关系运算主要就是处理操作数之间的结果,这里有一点要说明,a==b 是关系运算,其结果有两种情况,即 true 或 false;a=b 是赋值运算,就是 b 的值赋给 a。这二者区别很大,大家一定要注意。

3. 逻辑运算

逻辑运算和关系运算比较相似,也是处理操作数之间的关系,结果只有 true 和 false 两种结果。表 2-4 给出了逻辑运算符。

表 2-4 逻辑运算符

逻辑运算符	含 义	举例(设 a=true,b=false)
&&	"与"运算	a&&b 结果:false
||	"或"运算	a||b 结果:true
!	"非"运算	! a 结果:false

4. 赋值运算

通常把"="称为赋值运算符。该运算符是一个二元运算符,需要两个操作数,左边的操作

数是变量或者数组，右边的是表达式。例如：

```
int n,m;
m = 5;                    //赋值运算
n = m * m                 //赋值运算
```

另外，“=”还可以和其他的运算符结合起来使用。比如+=、-=、/=、%=等，它们的意义分别是：“x+=a;”等价于“x=x+a;”、“x-=a;”等价于“x=x-a;”、“x*=a;”等价于“x=x*a;”、“x/=a;”等价于“x=x/a;”、“x%=a;”等价于“x=x%a;”。

上面几种运算形式在以后的编程中会经常用到，大家要熟记。当然“=”还可以和“>>”等运算符结合起来使用，使用的含义和上面的含义相同，所以在此不再赘述。

5. 位运算

位运算在单片机的开发中非常重要，比如设置某个引脚的输出电平为高电平的操作就是通过位运算来实现的。位运算主要包括“与”(&)、“或”(|)、“反”(~)、“左移”(<<)和“右移”(>>)等运算。例如：

```
main()
{
  int m,n,k,result;
  m = 10;
  n = 13;
  k = 0x0a;
  result = m & n;                //result 的值为 8
  result = m | n;                //result 的值为 15
  result = ~(k);                 //result 的值为 0xFA
  result = m << 2;               //result 的值为 40
  result = m >> 2;               //result 的值为 2
}
```

通过以上的例子就很容易理解位运算，说到这里，大家可能就要有疑问了，“&&”和“&”、“||”和“|”有什么区别呢？其实区别很大，首先，前者“&&”和“||”叫做关系运算，其结果只有true 或者 false 两种情况，true 就是 1，而 false 就是 0；“&” 和“|”等运算是位运算，其结果千变万化、五花八门，可以为 1、为 0，也可以为其他的数值。为了便于理解可以简单地认为关系运算是两个“事件”之间的运算，那么其结果就是 1 或 0 了，位运算是两个数据内部按照比特(位)的关系进行的运算，当然其结果就有可能出现多种情况了。

6. 运算的优先级

通过前面的介绍，读者应该对 C 语言的几种运算有了基本的了解。在实际应用中，一个

计算可能是上面的几种运算的组合，这样在进行运算的时候，执行的顺序就非常重要，这时就需要了解运算的优先级。

表 2-5 给出了运算的优先级，如果在有括号运算的时候，应该先运算括号里面的，再运算括号外面的表达式，而括号里面的表达式的运算优先级顺序应该参照表 2-5 所列。

表 2-5　运算的优先级

优先级	符　号	操作数个数
1	!，～，++，-	单操作数
2	*，/，%	双操作数
3	+，-	双操作数
4	<<，>>	双操作数
5	<，<=，>，>=，==，!=	双操作数
6	&	双操作数
7	|	双操作数
8	&&	双操作数
9	||	双操作数
10	=，+=，-=，*=，/=等	双操作数

2.4　函　数

在 C 语言中，函数是程序的基本组成单位。函数不仅可以实现程序的模块化，使程序设计得简洁和直观，提高程序的易读性和可维护性，而且还可以把程序中经常用到的一些计算或者操作做成通用的函数，以便随时调用。

函数定义的语法如下：

```
函数类型　函数名
{
语句 1;
......
语句 n;
返回;
}
```

函数类型确定了函数返回值的类型。函数的类型可以是任何一种有效的类型，比如整数类型，也可以是用户自定义的类型。

函数的参数表确定了函数的输入参数和输出参数，多个参数之间用逗号分开。

函数体主要由一系列的语句组成，语句的组成结构可以是顺序结构，也可以是分支结构或者循环结构。在函数体的最后，需要返回函数的值，如果函数定义成 void，可以不用返回值，其他类型必须返回函数值，并且返回函数值的类型必须和函数定义时函数的类型一致。下面举例来说明函数的定义。

```
memset(int a[10],n)
{
    int i;
    for(i = 0;i < 10;i ++)
    {
        a[i] = n;
    }
    return;
}
```

在上面的例子中，return 可以省略，因为这个函数的返回类型是 void。这里的 void 没有出现在函数名 memset 前，所以可以不写，默认是 void 的。

```
int sum(int n,int m)
{
    int sum;
    sum = n + m;
    return sum;
}
```

在上面的例子中，return 不可以省略，需要将计算的结果返回。

1. 局部变量与全局变量

在引入了函数定义后，需要知道变量的作用域，就是变量的使用范围。根据变量的作用域可以将变量分为全局变量和局部变量。局部变量就是在函数内部定义的变量，局部变量只被函数内部访问。全局变量与局部变量不同，它一般定义在程序的顶端，能贯穿整个程序，能被任何一个模块使用。在程序的设计中，如果全局变量和某一个局部变量的名字相同，那么在局部变量使用的函数内部当使用同一名字的两个变量时，实际使用的是局部变量，这一点需要引起注意。正因为如此，所以在定义变量的时候绝对不能重名，并且在使用变量的时候应该合理使用局部变量和全局变量。

结构化的程序需要程序代码和数据分离，C 语言是通过局部变量和全局变量来实现这一分离的。如果大量使用全局变量就破坏了结构化程序设计的要求。下面举例说明局部变量和

全局变量的使用。

```
int a = 3;                    //全局变量
int c = 3;                    //全局变量
int Mult(int x,y);
main()
{
  int sum,a;                  //全局变量
  sum = Mult(a,c);
}
int Mult(int x,int y);
{
  int res;
  res = x * y;
  return res;
}
```

上面的例子具体演示了局部变量和全局变量的使用。上面程序的运行结果应该是 48。通过例子也能明确变量的作用域，直观来讲，局部变量就是在函数或子程序里再重新定义的变量，出了这个函数或子程序变量的数值就不再保存了；而全局变量在整个程序运行过程中数值都会保留，直到程序再对其进行修改为止。全局变量一般在整个程序的起始位置来定义，后面的编程中可以直接使用。

2. 形式参数与实际参数

函数定义时的参数称为形式参数，简称形参。它们同函数内部的局部变量作用相同。形参的定义在函数名后的括号内。在进行函数调用时，传入的参数成为实际参数，简称实参。实参和形参的顺序必须一致，否则就会出现错误，这一点需要引起注意。下面举例说明。

```
int GetMax(int x,int y);
main()
{
  int m,n,k;
  m = 9;
  n = 10;
  k = GetMax(m,n);            //m,n 为实参
}
int GetMax(int x,int y);      //x,y 为形参
{
if(x > = y) return x;
```

```
else return y;
    }
```

程序中 k 的值为 10。

3. 函数调用方式

在函数体实现完成后，需要具体调用函数才能执行函数，也才能利用函数实现的功能。在 C 语言中，函数有标准的库函数，也有用户自定义的函数。对于库函数而言，需要包括具体的头文件，比如 #include<stdio.h>。对于系统预定义的函数，如果函数不在调用它的函数的那个 C 文件里面，则需要包括头文件，比如 #include<stdio.h>。对于用户自定义的函数，如果函数不在调用它的函数的那个 C 文件里面，则需要包括头文件，比如 #include"user.h"。这里需要注意的是头文件不能用"<>"括起来，只能用双引号括起来。如果函数和调用它的函数在同一个文件里，则只需要在函数前面声明一下就可以了。

关于函数的调用主要有以下形式。

① 函数作为执行语句。

```
memset (int a[10],int n);
main()
{
int a[10];
int n;
n = 10;
memset (a,n);                    //函数作为执行语句
}
memset (int a[10],int n);
{
int i;
for(i = 0;i < 10;i ++ )
{
    a[i] = n;
}
return;
}
```

最终结果是 a[0],…, a[9]的值均为 0。

② 函数作为表达式，这种形式就是将函数作为表达式里面的一部分。

```
int GetMax(int x,int y);
main()
```

```
{
int m,n,k;
m = 9;
n = 10;
k = 20;
k = k + GetMax(m,n);              //函数作为表达式
}
int GetMax(int x,int y)
{
if(x > = y) return x;
else return y;
    }
```

最终结果是 30。

③ 函数作为参数，这种形式就是将函数作为一个函数的实参进行传递。

```
int GetMax(int x,int y);
main()
{
int m,n,k;
m = 9;
n = 10;
k = 20;
k = GetMax(m,GetMax(n,k));      //函数作为参数
}
int GetMax(int x,int y)
{
if(x > = y) return x;
else return y;
    }
```

最终结果是 20。

4. 函数嵌套调用

在 C 语言中，所有的函数都可以互相调用，如果函数调用自己的函数，就是所说的递归调用。函数也可以调用其他函数，函数之间可以实现多次调用，下面举例说明。

```
int max2(int x,int y);
int max3(int x,int y,int z);
main()
```

```
{
int m,n,k,res;
n = 10;
m = 20;
k = 30;
res = max3(n,m,k);
}
int max2(int x,int y)
{
    if(x >= y) return x;
    else return y;
}
int max3(int x,int y,int z);
{
    int res;
    res = max2(x,y);
    return = max2(res,z);
}
```

上面的例子给出了函数的嵌套调用，主要强调的是当实现递归的时候，一定要注意，因为很容易引起死循环。

2.5 数 组

数组是一个由同种类型变量组成的集合，引用这些变量时可以使用同一名字。数组由连续的存储区域组成，最低地址对应数组的第一个元素，最高地址对应最后一个元素。数组可以是一维的，也可以是多维的。

1. 一维数组

一维数组的定义如下：

数组类型　数组名[数组元素个数]；

例如：

```
int a[10];                    //定义一个整数类型的数组 a,数组元素的个数为 10
```

一维数组可以在定义的时候初始化值。例如：

```
int a[10]={0,1,2,3,4,5,6,7,8,9};
```

数组的访问通过下标来实现，数组元素的下标从 0 开始，而不是从 1 开始。比如上面数组中的第 3 个元素就是 a[2]。在赋值的时候可以全部赋值，也可以部分赋值。如果是全部赋

值，那么可以写成下面的形式：

int a[]={1,2,3,4,5,6,7,8,9};

如果是部分赋值，那么没有被赋值的元素的值默认为 0。例如：

int a[10]={1,2,3,4,5};

在上面的数组中，只有前 5 个元素被赋了值，而后面几个元素没有被赋值，所以后面几个元素的值为 0。

数组元素的访问和一般变量的使用差不多。下面举例说明。

```
main()
{
int a[10];
int n;
for(n=0;n<10;n++)
{
    a[n]=n;
}
}
```

上面的例子说明数组的元素 a[n]与一般变量的使用基本相同。

2. 多维数组

C 语言中可以定义多维数组，最简单的多维数组就是二维数组。

二维数组的定义如下：

数组类型　数组名[行数][列数];

二维数组元素的个数等于行数乘以列数。例如：

int a[3][5];　　//定义一个 3 行 5 列的数组，数组元素的个数为 15

二维数组可以在初始化的时候赋值。例如：

int a[3][4]={{1,2,3,4},{5,6,7,8},{9,10,11,12}}

通过以上例子可以看出：二维数组可以看成多个一维数组的集合。

另外，二维数组也可以部分赋值。例如：

int a[3][4]={{1,2,3},{5,6,7,8},{9,10}}

对于二维数组的每个元素而言，也可以看成一般的变量。下面举例说明。

```
main()
{
int a[10][5];
int n,m;
for(n=0;n<10;n++)
```

```
{
    for(m = 0;m < 5;m ++ )
    {
a[n][m] = m + n;
    }
}}
```

2.6 自学思考题

1. MSP430 系列单片机 C 语言有哪些基本的数据类型?
2. 函数和过程有何区别? 如何定义一个函数?
3. 如何定义一个二维数组?
4. 如何使用 ASCII 码表? A 的 ASCII 值是多少?

第 3 章

MSP430 系列单片机并口 JTAG 仿真器及其使用

3.1 并口 JTAG 仿真器功能简介

对于初学者来说，仿真器是必备工具，借助于编程器可以完成程序的下载及在线调试功能，这里介绍一款价格低廉而且实用的并口 JTAG 仿真器。

图 3-1 是并口 JTAG 的实物照片，JTAG 仿真器的核心是一个 244 芯片，购买 244 芯片时一定要注意型号的问题，比如 TTL 类型的 244，如 74LS244，就不能买，因为这种 TTL 型号

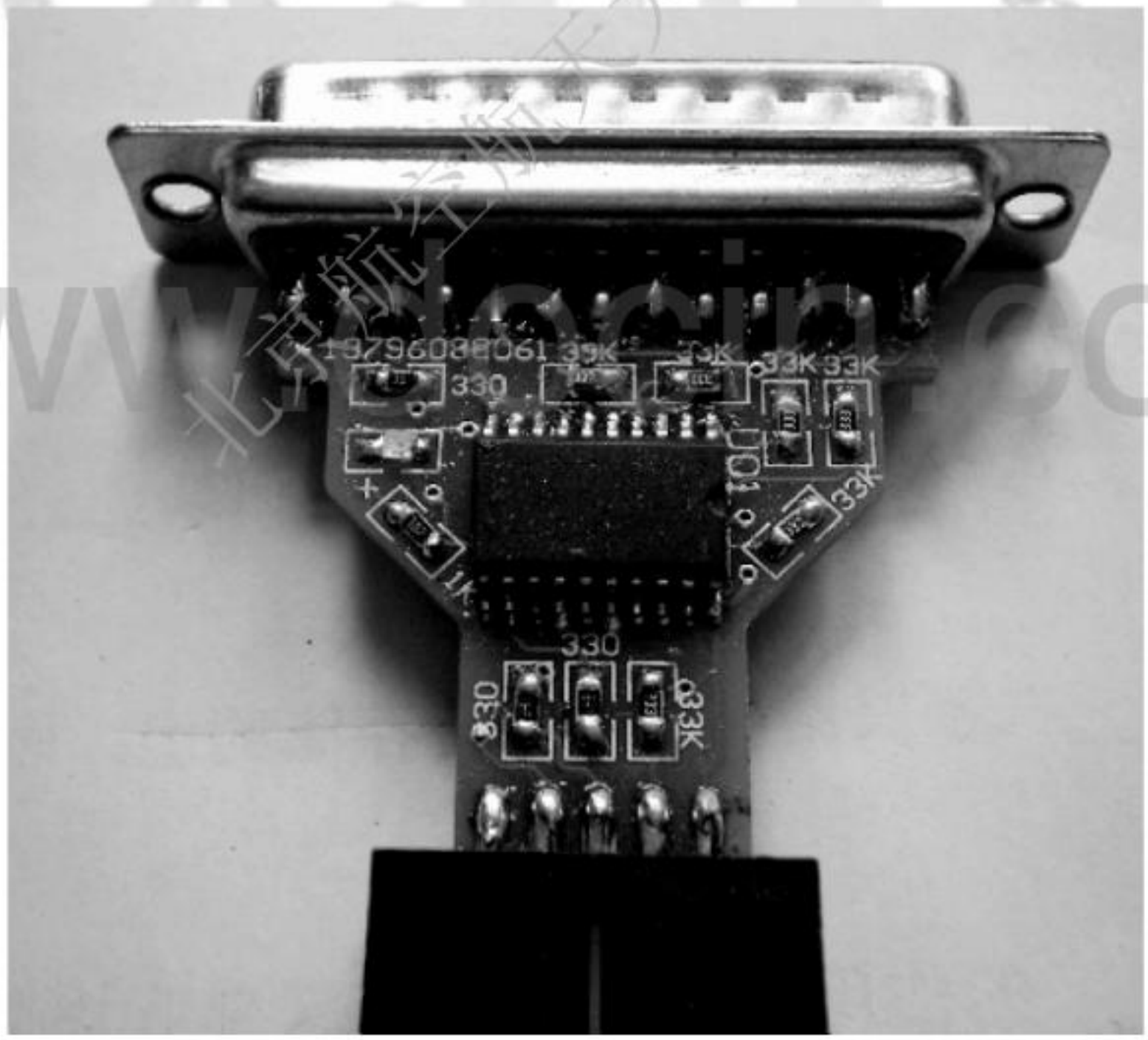


图 3-1 并口 JTAG 的实物照片

的 244 在 3.3 V 电压下可能无法正常工作。这里选用的型号是 74HC244，该芯片可以工作在 3.3 V 电压下。这种芯片目前至少有三种封装形式，一种是直插的，另外两种是贴片的。对于直插封装的 74HC244，由于其引脚间距较大(2.54 mm)，非常适合自己动手时用，可以选择该芯片在实验板上搭建自己的 JTAG 电路，省去了画板、制板的时间与繁琐；缺点是对焊工要求高一些，如果焊工一般，就千万不要去尝试了，因为那是自找麻烦。至于贴片的 74HC244，有两种封装形式，一种是宽体的，一种是窄体的，功能上没有区别，引脚也兼容，就是大小不一样。具体使用时，应先买来芯片再画 PCB 板，在本 JTAG 中采用的是宽体封装的 244 芯片。这种芯片的封装形式可以在光盘中找到。使用时通过上端的 DB25 针形(公头)接口和计算机的打印机的输出接口相连，当然二者之间可能有一定的电阻，然后经过 244 芯片输出，输出端采用一个 IDC10 针形插座输出(实际上只用了 8 根，其中 6 根信号线，2 根电源线)，与目标板的 MSP430 系列单片机相连，实现仿真、下载功能。

图 3-1 上的 DB25 插头是一个针形的插头(因为计算机主机箱后边的输出插口是一个孔型的插座，这样二者才能配合起来)，这个插头和电路板的连接位置属于非标准连接，在做 PCB 板时，要先测量它的外观尺寸，再画它的封装。当然输出接口 IDC10 的插座和电路板连接时也是非标准连接关系，这个封装也要自己来画。至于 74HC244 芯片和电阻，在 PCB 库中都可以找到，74HC244 芯片的封装是 SOL20；电阻的封装是 0805。

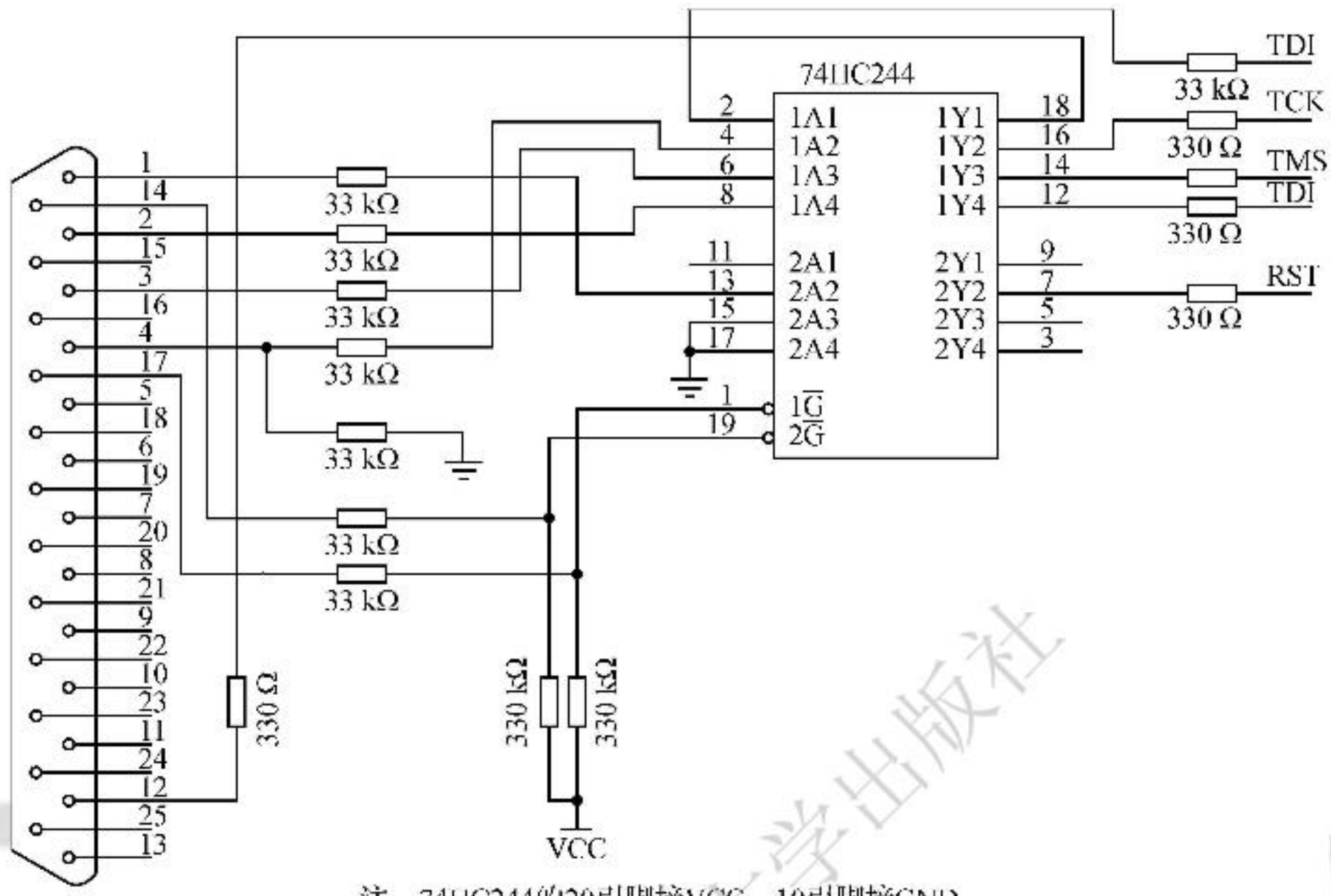
3.2 JTAG 仿真器原理图与自制过程

JTAG 仿真器是 MSP430 系列单片机学习中必不可少的工具，可以购买但价格不便宜，其实 JTAG 仿真器并不复杂，有一定经验的同学完全可以自制。下面介绍 JTAG 的原理图与自制的方法，方便大家自己动手制作。

3.2.1 JTAG 仿真器原理图

图 3-2 是并口的 JTAG 的原理图，从图中可以看出，除使用了一片 74HC244 外，就是一些电阻了。对于电阻的精度，没有什么特殊要求，常用的即可。阻值有三种，330 kΩ、33 kΩ 和 330 Ω。笔者做的编程器中，所有电阻都采用的是 0805 的封装。该编程器结构简单，价格低廉，使用方便，可以实现程序的下载、单步、全速、断点等功能，可以按照原理图自制，也可以购买。如果需要原理图及 PCB，可以发邮件给笔者，邮箱地址是 qingtengzfc@yeah.net。

该编程器中使用的 74HC244 芯片是 8 缓冲器及线驱动器，用来改善三态存储地址驱动器，其引脚如图 3-3 所示。74HC244 芯片可以工作在 3.3 V 的电源电压下，芯片是八同相三态缓冲器/线驱动器，芯片内部共有两个 4 位三态缓冲器，使用时可分别以 $1\overline{G}$和 $2\overline{G}$作为它们的选通工作信号。A 侧为输入端，Y 侧为输出端，当 $1\overline{G}$和 $2\overline{G}$都为低电平时，输入端 A 和输出端 Y 状态相同；当 $1\overline{G}$和 $2\overline{G}$都为高电平时，输出呈高阻态。74HC244 真值表如表 3-1 所



注：74HC244的20引脚接VCC，10引脚接GND。

图 3－2　并口 JTAG 的原理图

列。图 3－2 的原理图的左边与计算机的打印机接口相连，右边与目标板相连。

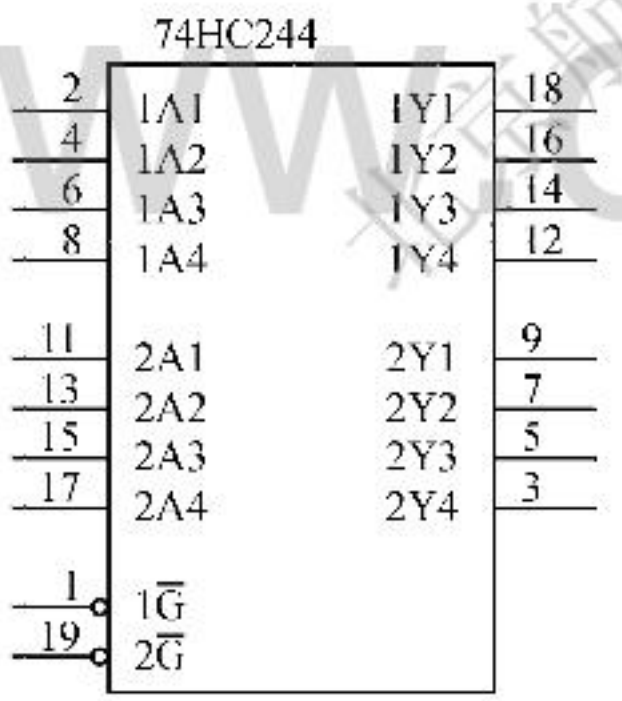


图 3－3　74HC244 芯片引脚图

表 3－1　74HC244 真值表

输　入		输　出
使　能	1Ax/2Ax	1Yx/2Yx
L	L	L
L	H	H
H	×	Z

注：L 表示低电平；H 表示高电平；Z 表示高阻抗；×表示任意状态；x 表示 1～4。

3.2.2 JTAG 仿真器的自制过程

图 3-4 是 JTAG 仿真器的 PCB 照片，该板是双面板，为节省空间，所有元件均采用贴片封装；在市场上可以买到一种并口的小盒子，大家可以把做好的编程器放到小盒子里面。这里要说明一点，大家自己动手制作的时候，可以购买一个实验板，买一个直插的 74HC244 就行了。如果没有制作 PCB 板就不要买贴片的，否则，特别不容易焊接。如果需要 PCB 图，可以联系笔者。

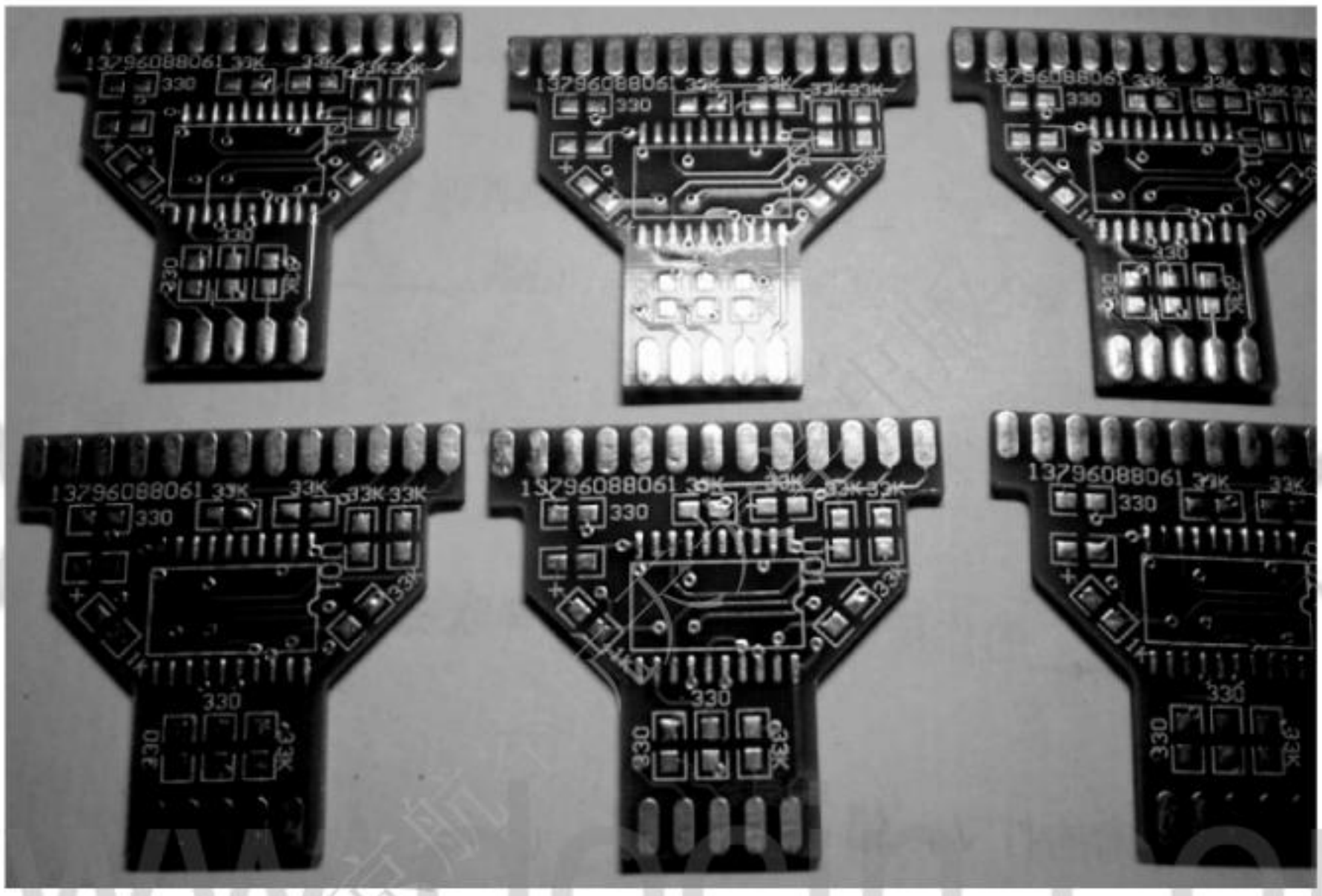


图 3-4 并口的 JTAG 的 PCB 照片

该编程器只采用了一片 74HC244 芯片及少量的贴片元件，特别适合初学者自己动手制作。贴片元件焊接时首先在 PCB 的一个焊盘上点上少许的焊锡，然后用镊子夹住元件，将其焊在刚才点过焊锡的位置，然后再将贴片元件的另一端也焊好就 OK 了。焊接 0805 的贴片大家实验几次就没有问题了。贴片的 74HC244 焊接起来有点难度，基本上有两种方法。一种是一个一个引脚慢慢焊接，这种方法比较容易理解，这里就不必细说了。下面介绍第二种方法，这种方法在焊接贴片时比较常用，通过图片的形式简单介绍一下。

如图 3-5 所示，首先，在元件的一侧挂上焊锡，起固定作用，可以随便焊几个引脚，也可以用堆锡的方式，一堆锡也没关系，不要害怕更不用担心，后面自有处理的方式。

在元件的另一侧，只采用堆锡的方式，如图 3-6 所示。看到图中的铅“疙瘩”害怕了没有？是不是有“废”了的感觉啊？没关系，下面给出解决的方法。

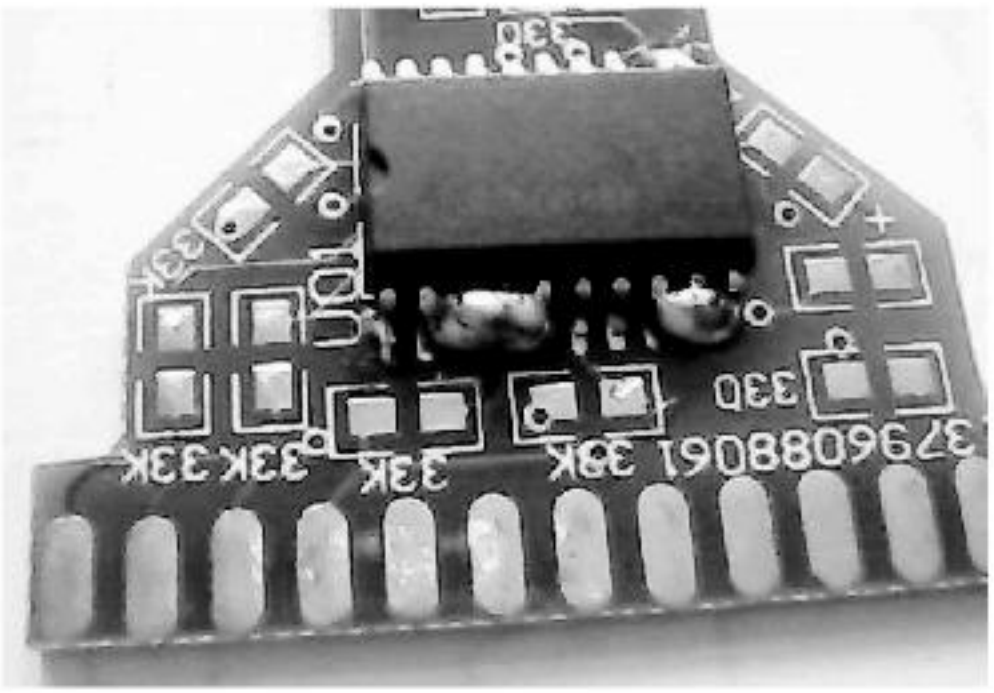


图 3-5　在元件的一侧挂锡，起固定作用

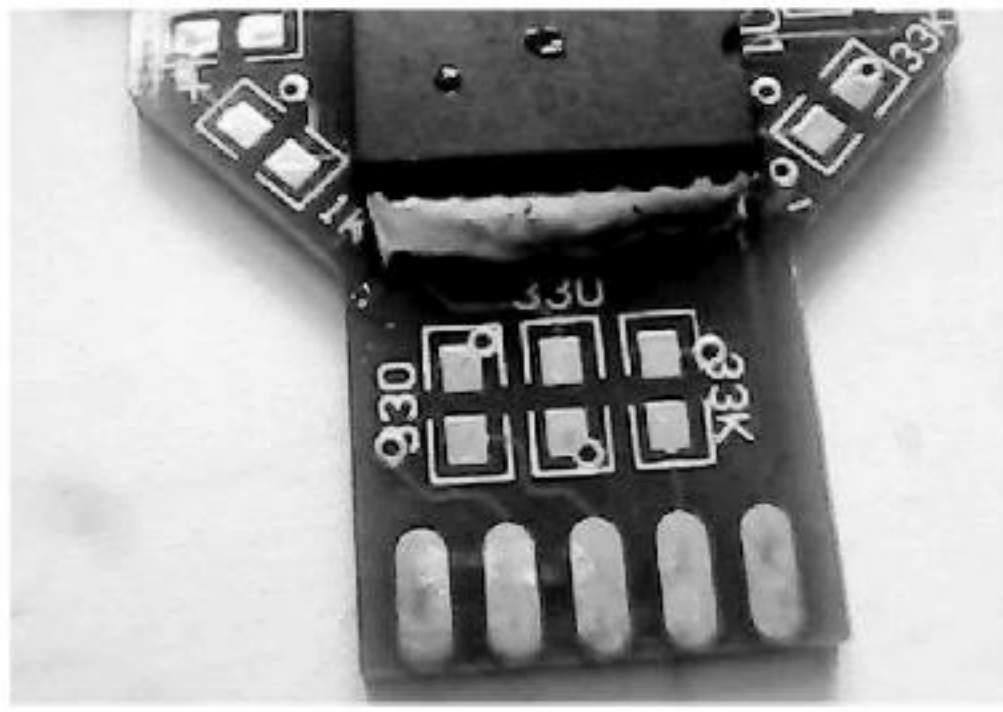


图 3-6　在另一侧大面积堆锡

在图 3-7 中，使用烙铁在堆锡的一侧不断加热，在加热过程中，不断左右移动烙铁，只要烙铁足够热，焊锡就完全变为融化状态了，就像水一样，这时就到了最关键的时候了！用手拿起电路板用力向桌面“撞”去，这时焊锡就会从元件的一侧脱落下来，引脚间只会残留很少的焊锡，但这一点焊锡就足够保证元件和电路板连接用了。如果引脚还有部分粘连在一起的话，可以重复上述方法，用同样的方法向桌面“撞”几下即可。至于元件的另一侧，也就是起固定作用的那一侧，也可以采用相同的方法，细节在这里就不再赘述了，请大家自己尝试。

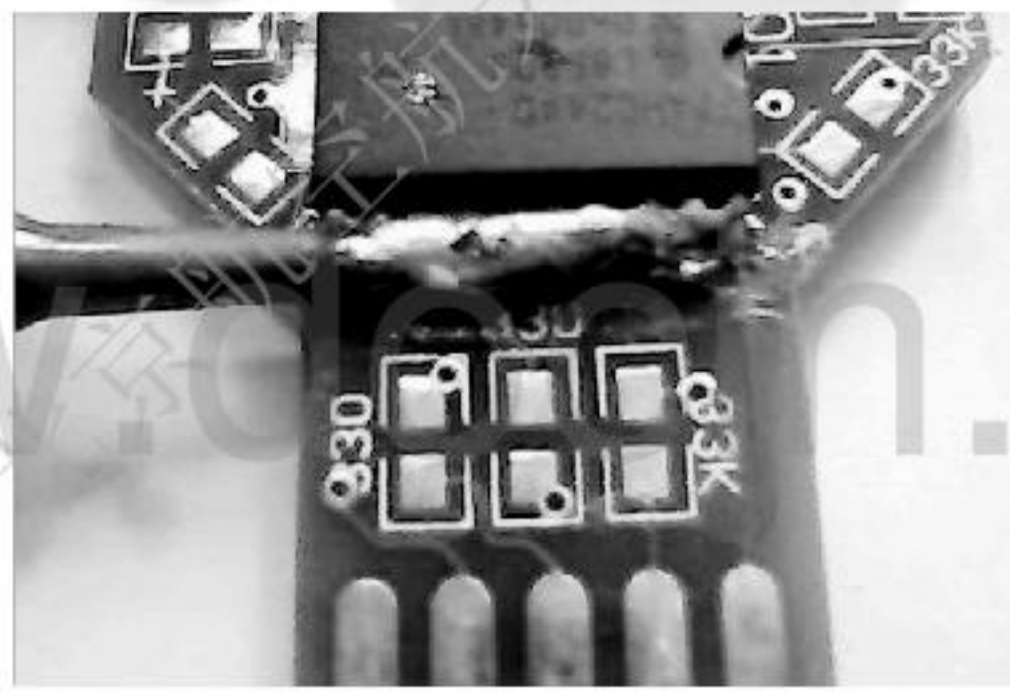


图 3-7　烙铁在一侧加热示意图

3.3　自学思考题

1. JTAG 仿真器具有哪些功能？
2. 自制 MSP430 系列单片机 JTAG 仿真器时都需要哪些元件？
3. 如何焊接贴片元件？

第 4 章

MSP430 系列单片机集成开发调试环境

目前，MSP430 系列单片机的开发调试环境版本比较多，本章介绍的是 IAR 公司提供的开发调试环境，版本是 IAR Embedded Workbench V3.42A。该编程软件可以在网上下载，选择非限制版本就行了。

4.1 IAR Embedded Workbench V3.42A 概述

IAR Embedded Workbench V3.42A 支持全系列 MSP430 单片机，它具备以下特点。

① 支持 Windows 98/Windows NT/Windows XP 操作系统；

② Windows 风格的可视化开发环境；

③ 集成所有开发、调试工具(编译、连接、仿真)，方便使用；

④ 可以采用 Make 进行重新编译、连接和调试。

IAR Embedded Workbench V3.42A 采用创建项目(Project)的方式来进行软件的开发和管理。IAR Embedded Workbench V3.42A 包含的实用工具有：编辑器、汇编器、连接器、函数管理器、Make 工具和调试工具。

用户使用集成开发环境的文本编辑器编写源程序代码，该编译器特点如下：

① 根据 C 语言的语法来区别字体的颜色。

② 具有查找和替换功能，能够非常方便地对程序进行编辑。

③ 可以从出错的列表中直接跳到文本中相应的出错位置。

④ 能够检查括号是否匹配，这一点在程序编写、纠错时非常有用。

⑤ 可以多个窗口进行编辑。

程序编写完成后，用户可以对程序代码进行编译连接。编译连接完成后，可以运行程序，可以对程序进行调试。

4.2 Electronic Workbench for MSP430 V3.42A 安装过程简介

光盘中的编译软件是非限制版本，所以安装有点复杂，需要破解才能正常使用，现将

MSP430 系列单片机的开发环境介绍如下，可以按照下列步骤完成安装。

1）双击名为 keygen 的图标，出现注册机的画面，在 Product 选项的下拉菜单里选择 Electronic Workbench for MSP430 V3.42A。

2）查看 Hardware ID 文本框中的字符，将 0x 以后的字符中的小写字母全部改为大写。具体的操作方法是：直接选中某个小写字母，然后从键盘输入大写字母即可；更改完毕以后，选中文本栏内的所有字符就能看到原来的小写字母是否已经被改为大写了。例如，笔者计算机中的操作流程如图 4-1～图 4-4 所示。

① 打开软件以后看到的 Hardware ID，见图 4-1。

图 4-1　Hardware ID

② 选中小写字母 d，见图 4-2。

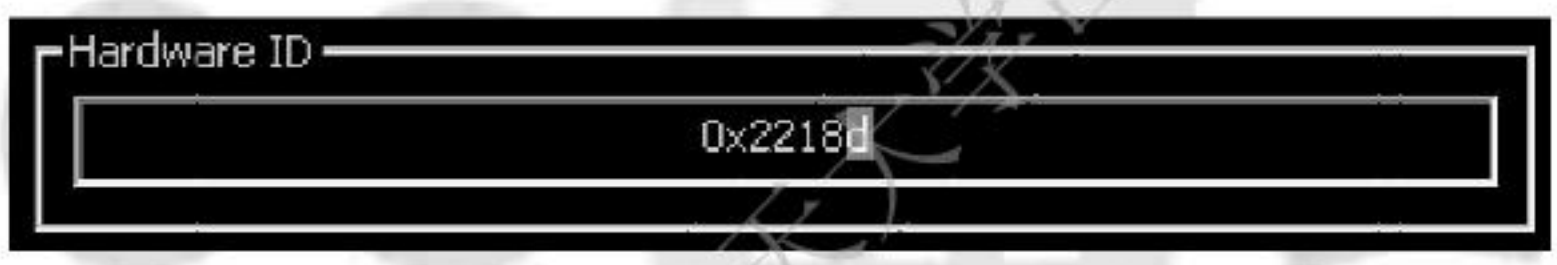


图 4-2　输入小写字母

③ 从键盘输入大写字母 D，见图 4-3，此时整个文本框内的字符会变得模糊不清，这是正常现象，不用担心；如果要看清楚，将工具条最小化后再恢复一下即可。

图 4-3　输入大写字母 D

④ 再次选择整个文本框中的内容，见图 4-4，可以看到模糊的字符又变清晰了，注意原来的 d 已经被更改为 D。

图 4-4　选择整个文本框

这时，单击注册机软件的左下角的 Generate 图标就可以得到需要的序列号了。

3）双击名为 EW430-ev-web-342A 的图标，等待解压缩完毕后出现安装界面，单击 Next 按钮后看到关于 License 的说明，再单击 Accept 按钮就可以看到如图 4－5 所示的对话框。

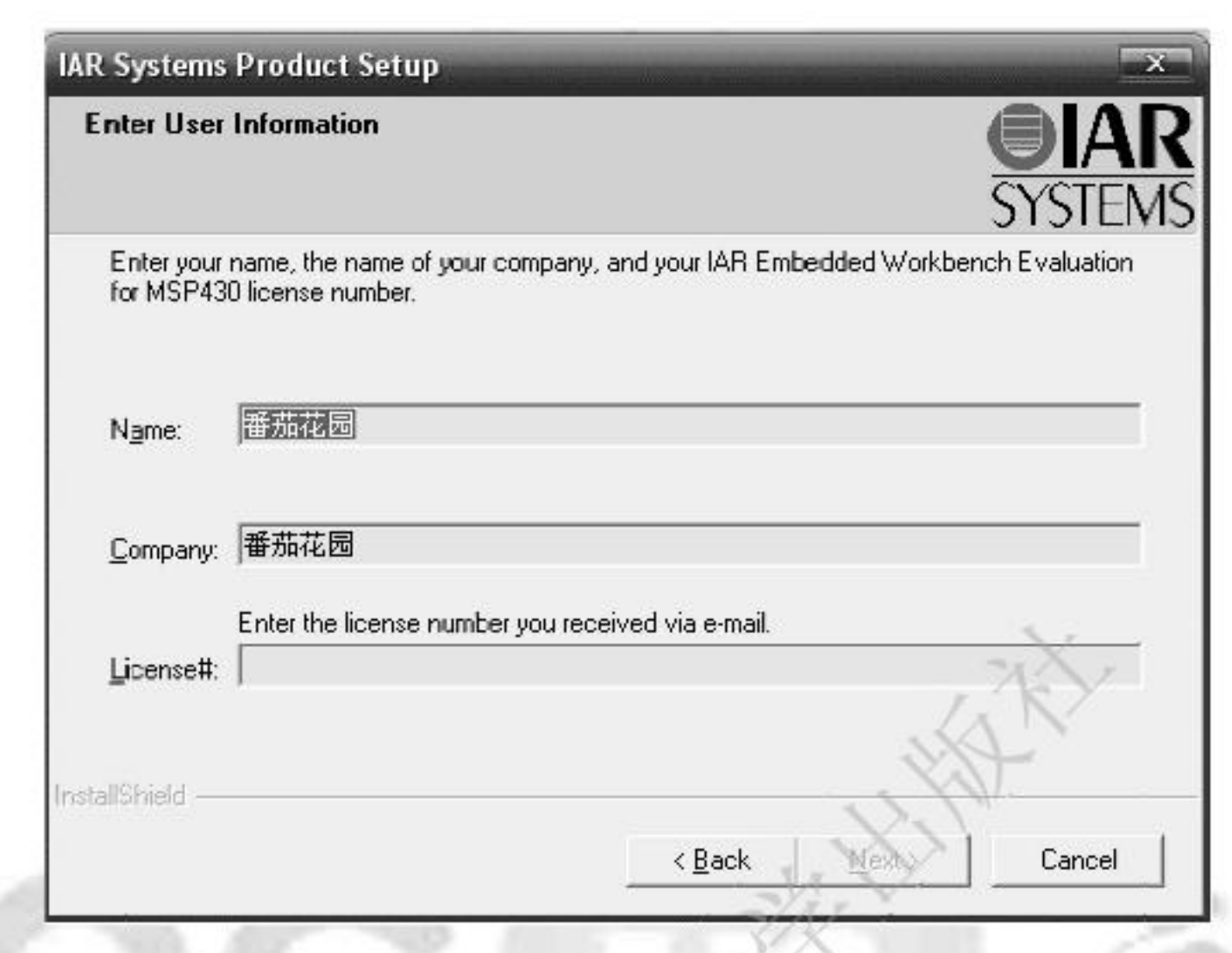


图 4－5　安装对话框 1

4）用户可以随意更改 Name 和 Company 文本框中的内容；然后选择已经打开的注册机，将 License number＋key 文本框中的数字复制到图 4－5 的 License 的文本框中，可以看到原来灰色的 Next 按钮变成了黑色，单击 Next 按钮进入下一对话框，如图 4－6 所示。

图 4－6　安装对话框 2

5）再次回到已经打开的注册机，将最后一栏文本框中的全部数字复制到 License Key 下面的文本框中，单击 Next 按钮，看到图 4－7。

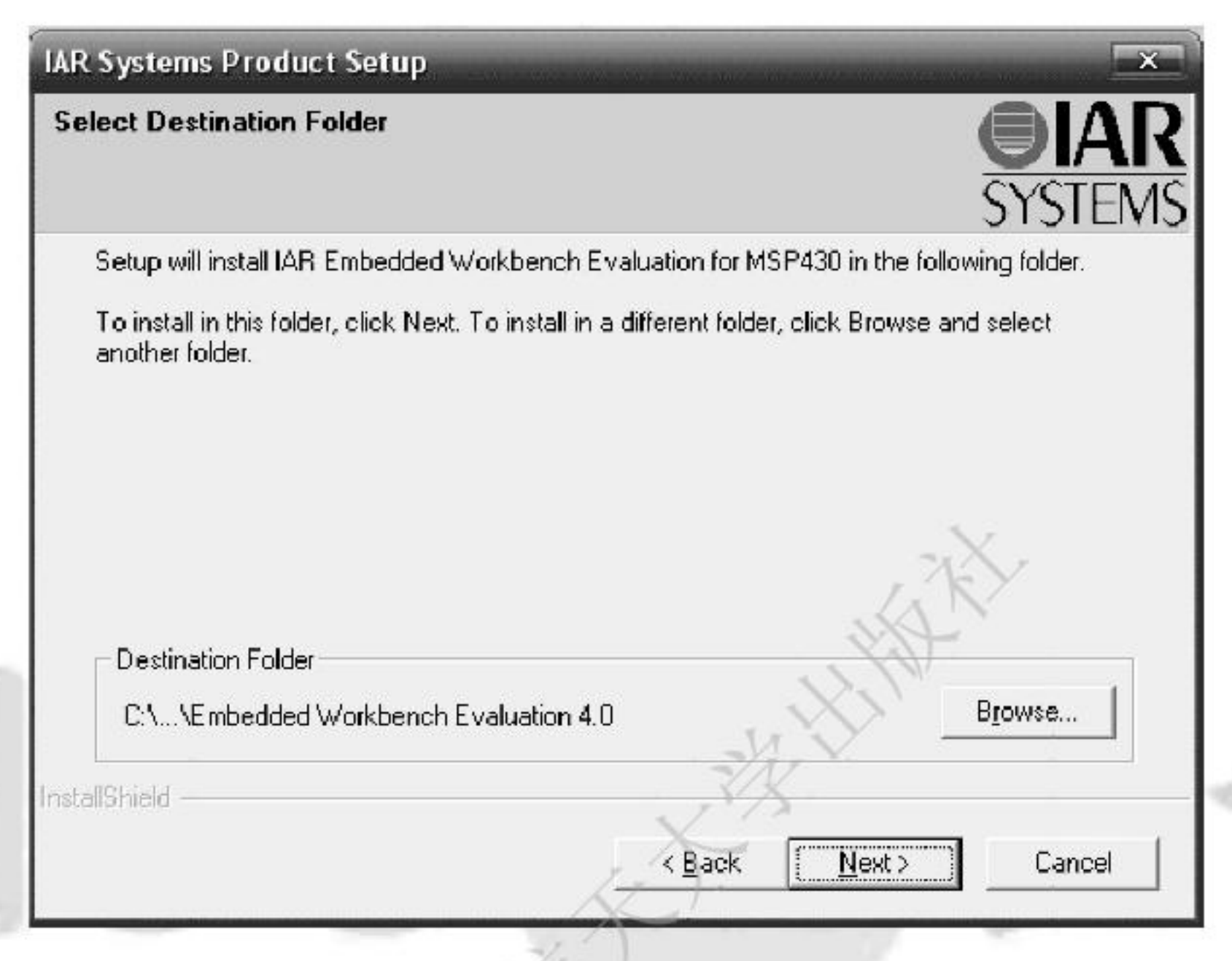


图 4－7　安装对话框 3

6）单击 Browse 按钮可以更改安装目录，然后单击 Next 按钮继续。等待安装完成，单击 Finish 按钮就大功告成了。

7）安装以后，您的桌面上会出现图标，双击即可打开这个软件了。

4.3　Electronic Workbench for MSP430 V3.42A 应用方法简介

4.3.1　Electronic Workbench 仿真设置

1）创建一个新的项目。选择 Project→Create New Project 命令，显示如图 4－8 所示对话框。选择项目模板 Empty project 后，会出现“另存为”的对话框，如图 4－9 所示，选择文件的保存位置并在文件名文本框中输入项目名字，单击“保存”按钮就可以了。这样将建立一个空的不包含任何文件的项目。

2）也可以单击图 4－8 中 C 旁边的“＋”号，选中下面新出现的 main，单击 OK 按钮以后新建一个带有 main.c 源文件的项目。单击 OK 按钮后同样会出现图 4－9 的画面，进行类似操作保存即可。

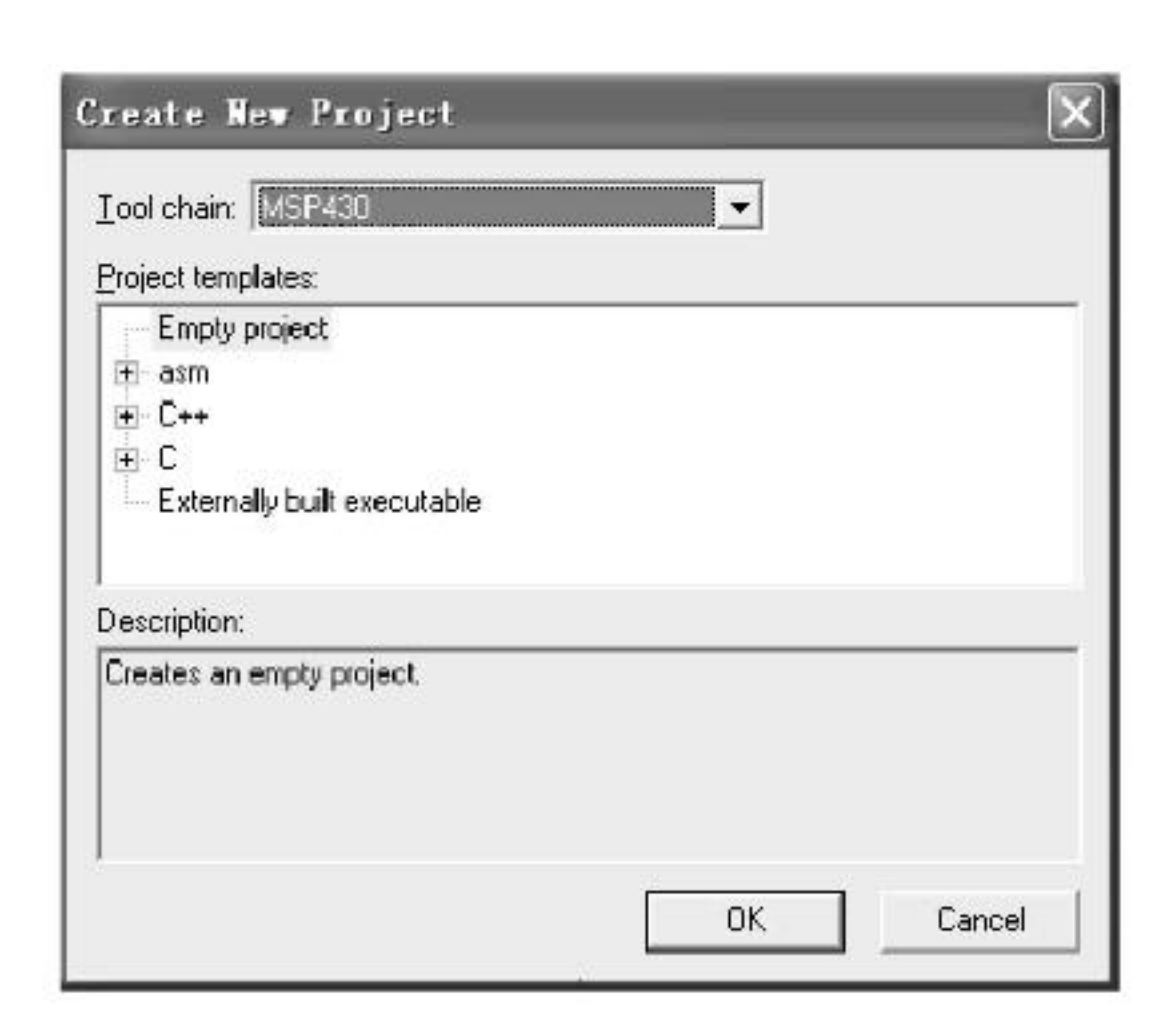


图 4-8　创建一个新的项目

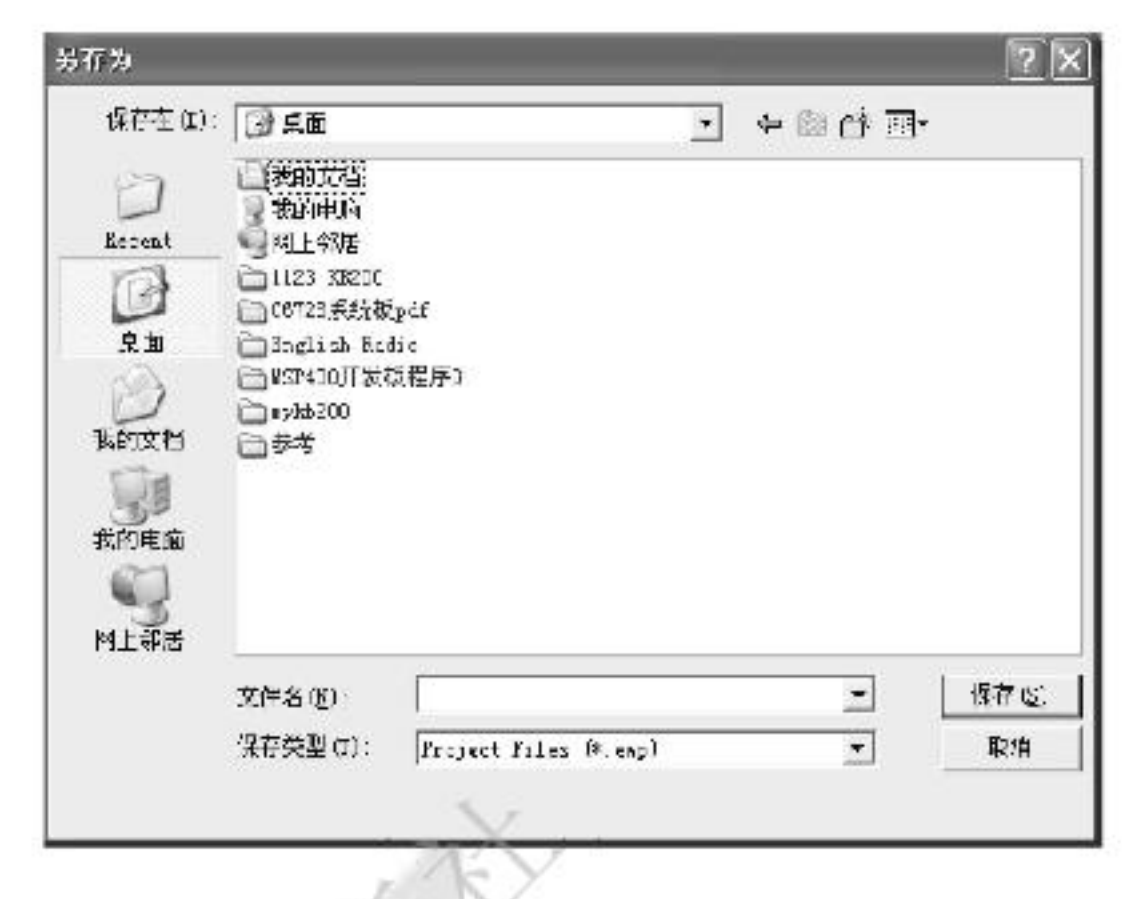


图 4-9　“另存为”对话框

3）之后可以看到当前项目出现在 Workspace 窗口内，如图 4-10 所示。在 Workspace 中有一个下拉列表框，这里有系统的创建配置（build configurations），默认时系统有两种创建（build）配置：Debug 和 Release。缺省配置是 Debug，在这种模式下，用户可以进行仿真和调试；在 Release 模式下，是不能进入调试状态的，所以建议在产品研发阶段一定不要修改这个创建配置，否则就不能进行调试了。

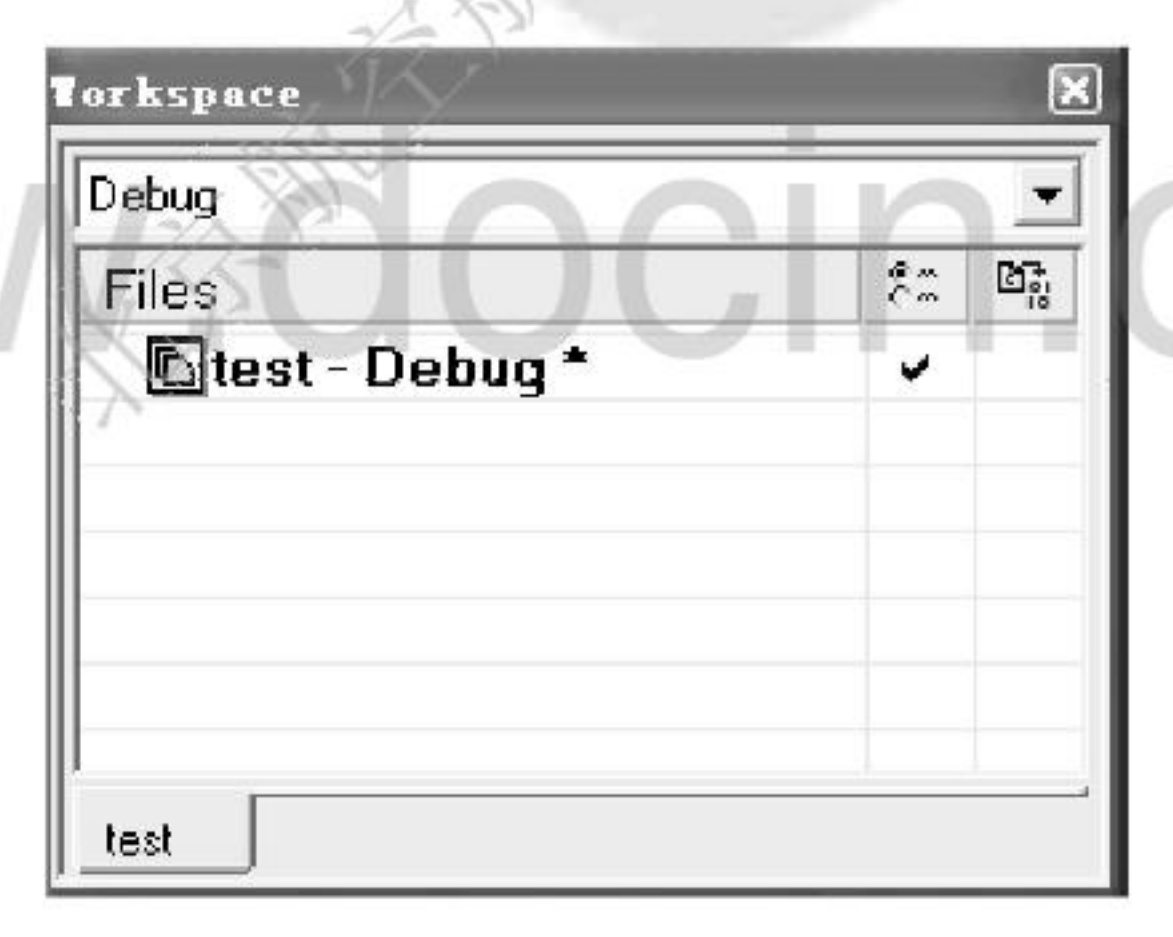


图 4-10　Workspace 窗口

4）选择 File→Save Workspace 命令，保存当前的工作空间（Workspace），以后直接打开工作空间就可以了。系统会为每个 Workspace 单独保存一套配置信息，所以不同项目的设置可以保留而不会相互冲突，因此建议用户每次建立一个项目都单独存储一个 Workspace 文

件，这样日后使用起来相当方便。

5）如果用户已经编辑好了源文件，则选择 Project→Add Files 命令就可以打开一个添加文件的对话框，见图 4－11。在这里可以向项目中添加源文件。在 Files of type 下拉列表框中可以选择要添加的文件类型。用鼠标同时选择多个文件或者按住 Ctrl 键单击多个文件名可以一次性地向项目中添加多个文件。当然，对于一般的初学者来讲，一般只有源文件是要添加的，另外，引用的那些是不需要添加的，那些一般都可以自动由编译软件自动加载进来。

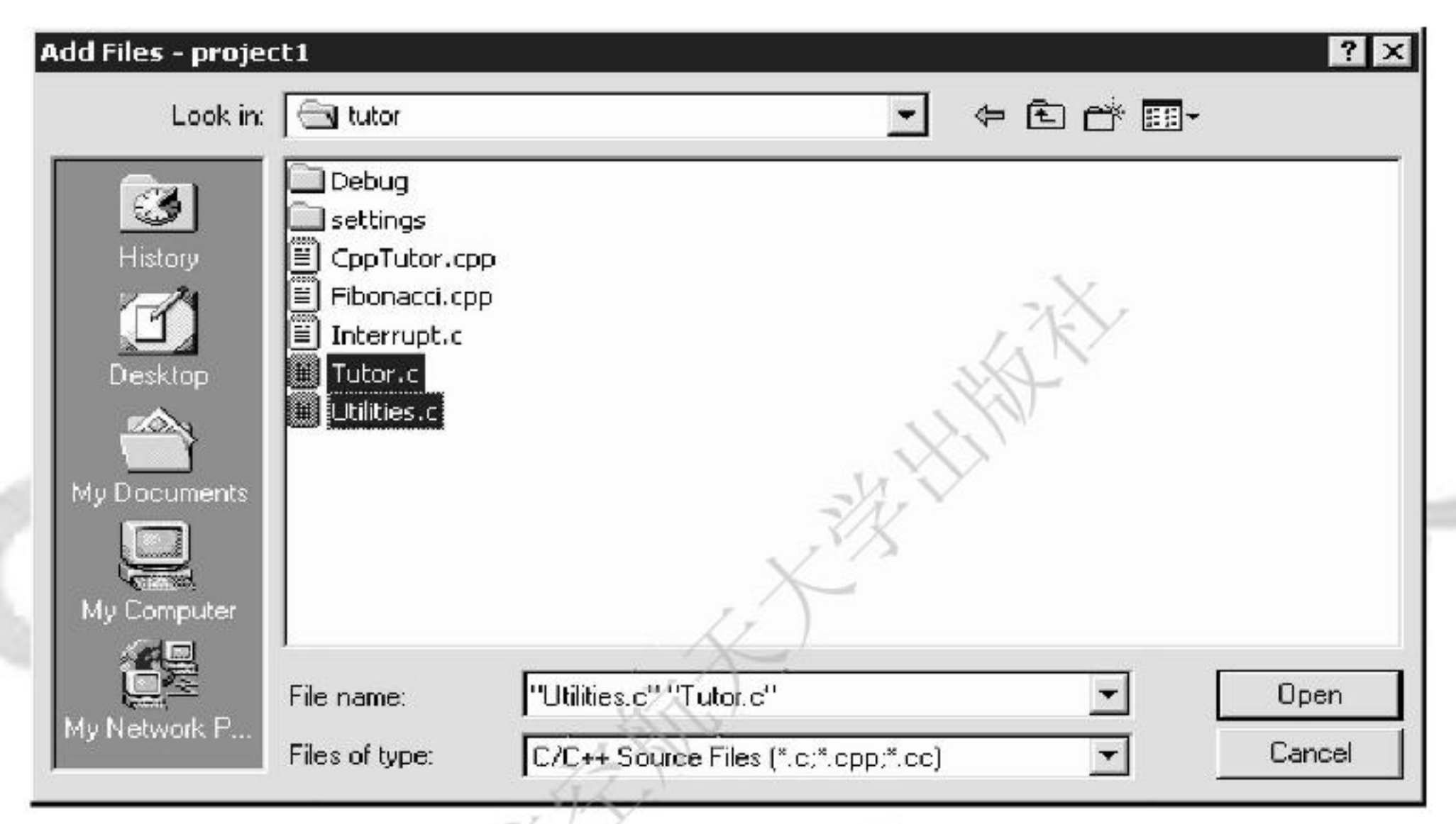


图 4－11　添加文件的对话框

6）如果用户需要手动输入源文件，则选择 File→New File 命令，也可使用工具栏左侧的图标按钮新建一个文本文件，然后用户可在其中输入自己的源程序，选择 File→Save 命令用于保存输入的文件。

7）所有的源文件都输入完毕以后，需要设置项目选项(Project Options)。选择 Project→Options 命令，或者将光标放在窗口左边的 Workspace 窗口的项目名字上单击右键(简称右击)选择 Options 命令，可以看到如图 4－12 所示对话框。

8）图 4－12 中是有关对本项目进行编译(compile)和创建(make)时的各种控制选项，系统的默认配置已经能够满足大多数应用的需求，所以通常情况下用户只需要更改两个地方。

① 单击 Category 下面的 General Options(见图 4－13)，再单击 Device 下方文本框右侧的图标按钮，显示图 4－14 所示的 MSP430 单片机型号，用户可以选择要使用的单片机。若使用本书介绍的开发板，请选择 MSP430F169，此时可以看到如图 4－15 所示的画面。

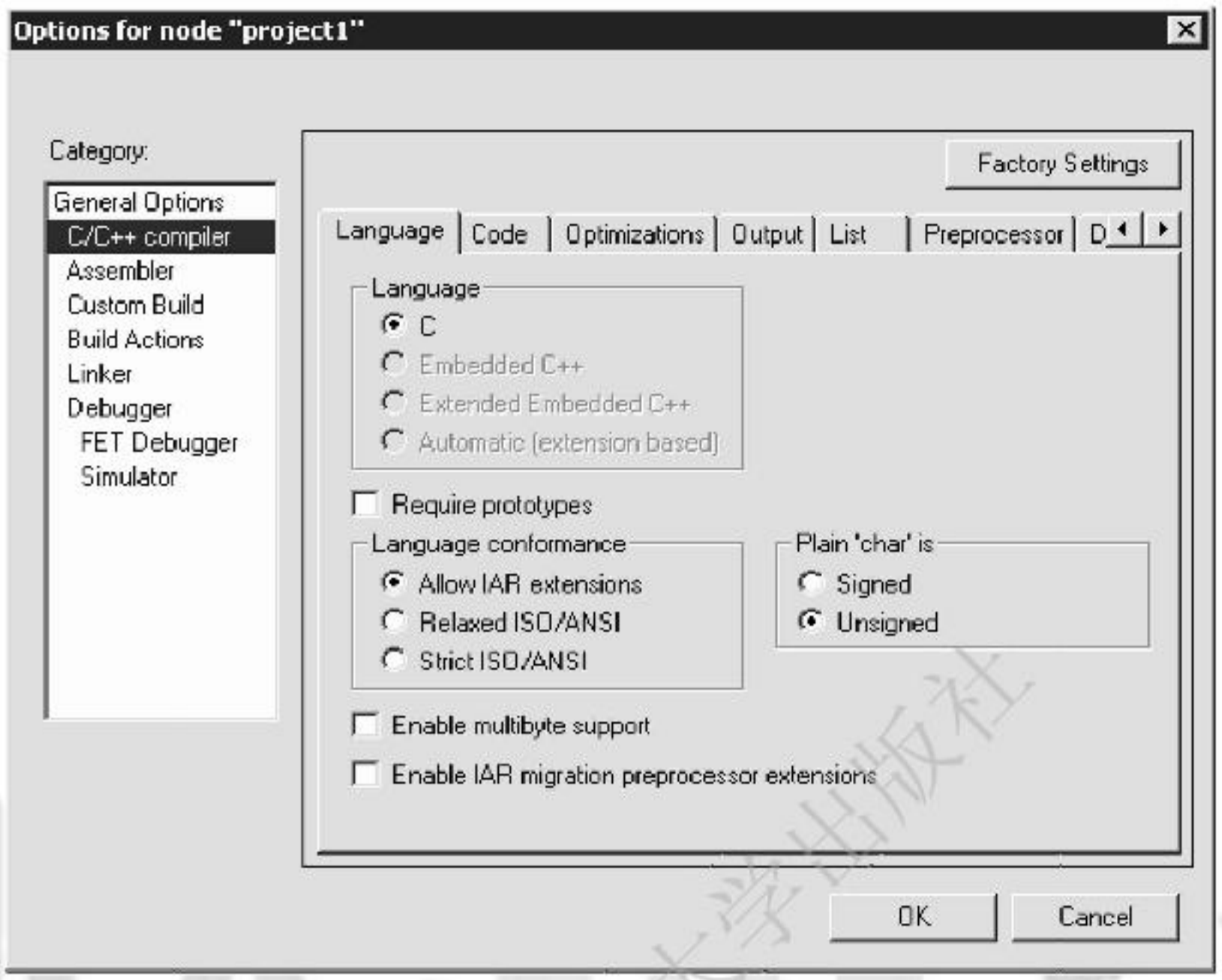


图 4－12　设置项目选项对话框

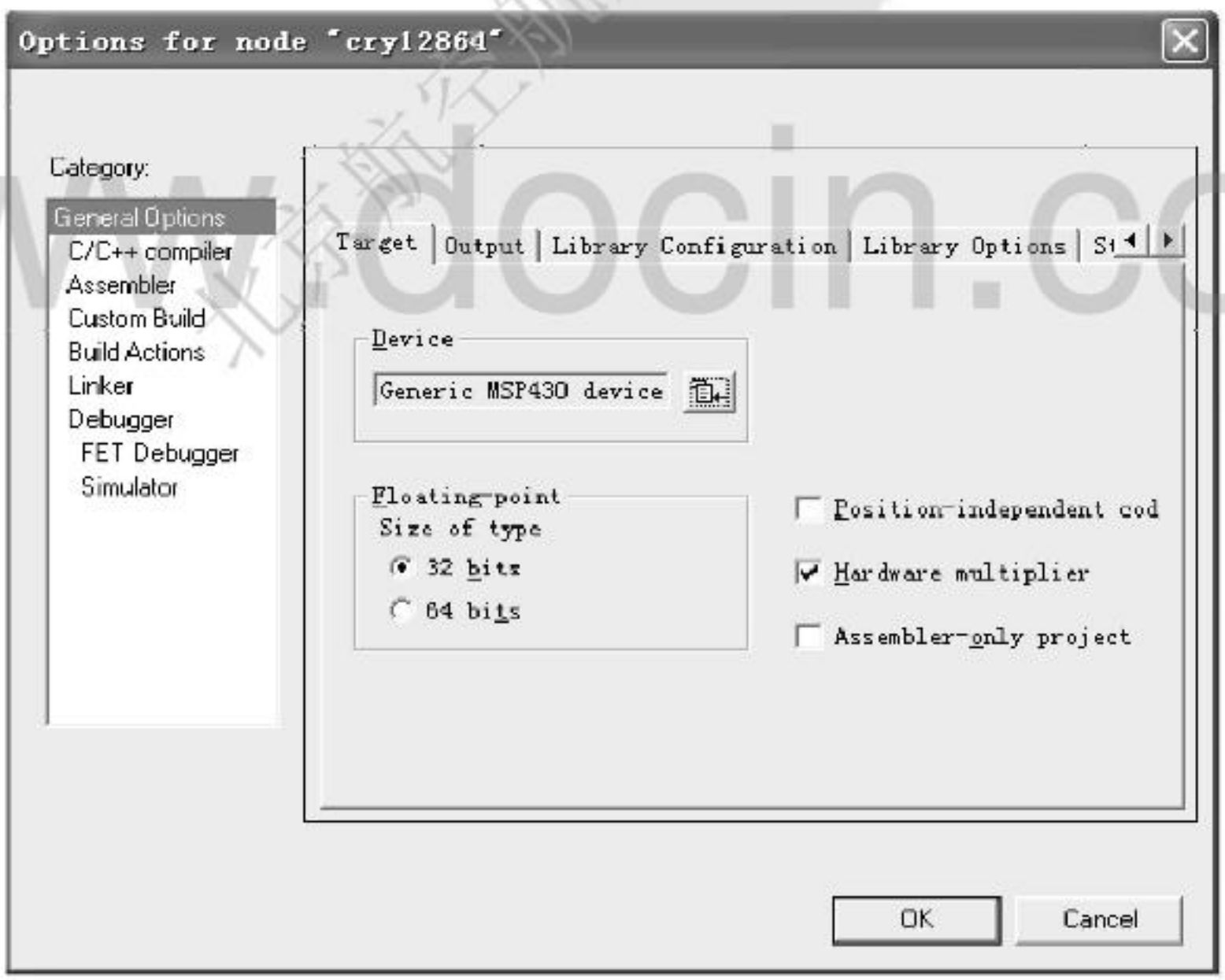


图 4－13　系统配置对话框

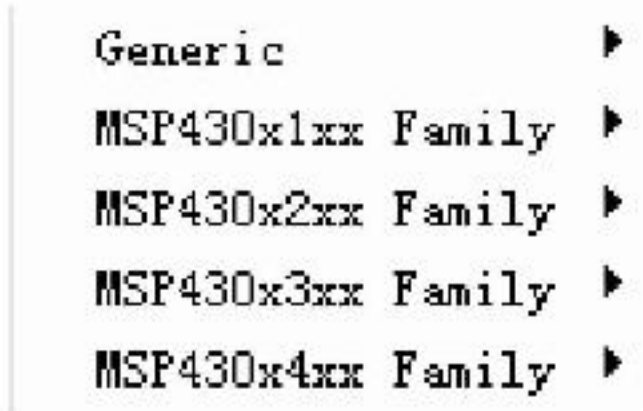


图 4-14　单片机型号选择

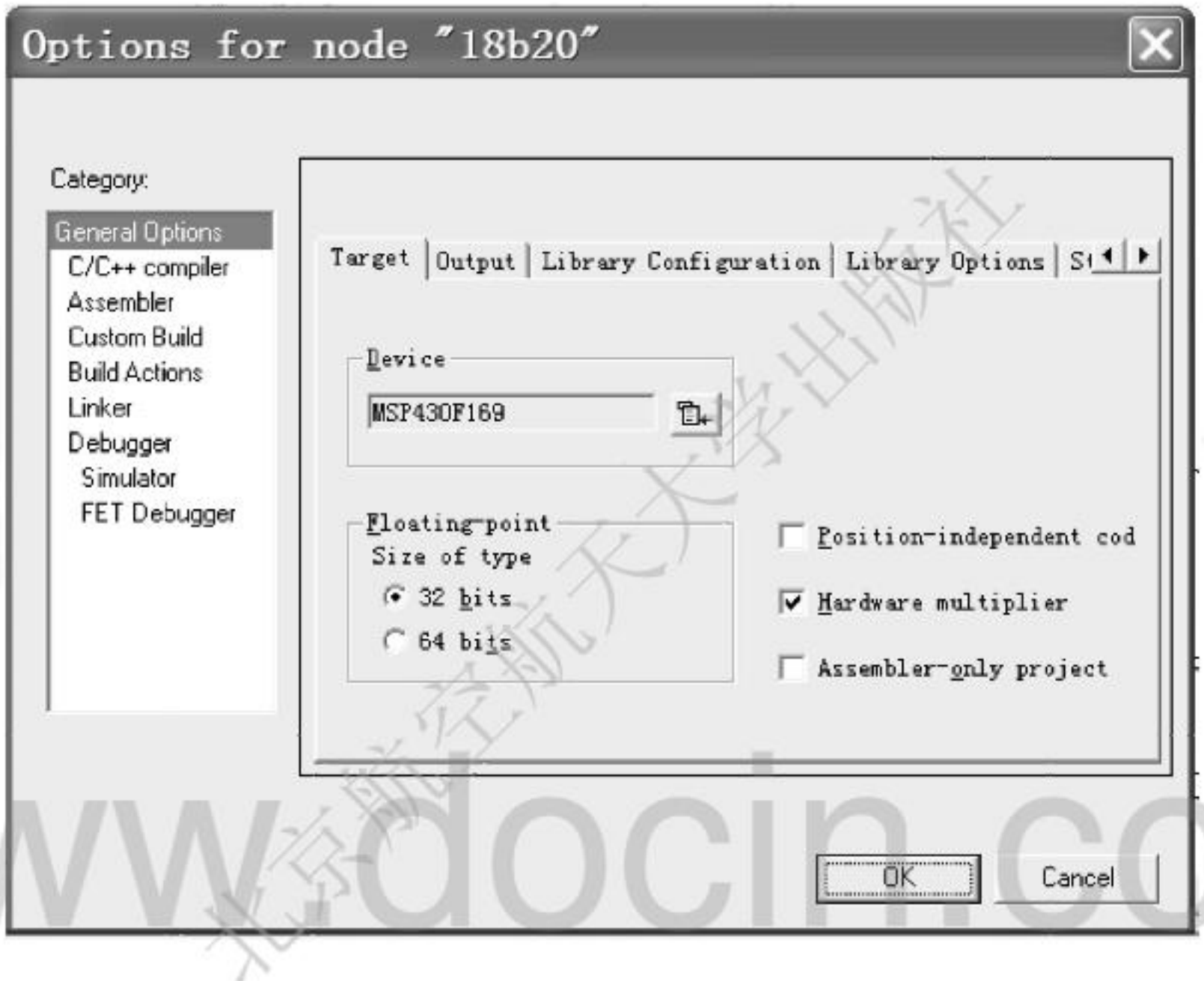


图 4-15　选择 MSP430F169

② 单击 Category 下面的 Debugger(见图 4-16),在 Driver 下拉列表框中,可以看到 Simulator 和 FET Debugger 两个选项。若选择 Simulator,则可以用软件模拟硬件时序,实现对程序运行的仿真观察;若选择 FET Debugger,则需要通过仿真器将 PC 上的软件与开发板上的 MCU 进行连接,然后就可以进行硬件仿真了。

如果用户只想进行软件仿真,那么选择 Simulator 以后单击 OK 按钮,就可以完成设置。如果用户需要进行硬件仿真,那么选择 FET Debugger 后,可以看到如图 4-17 所示的画面。在 Connection 下拉列表框中选择要使用的仿真器类型,就可以了。如果使用精简版仿真器,请选择 Texas Instrument LPT-IF。

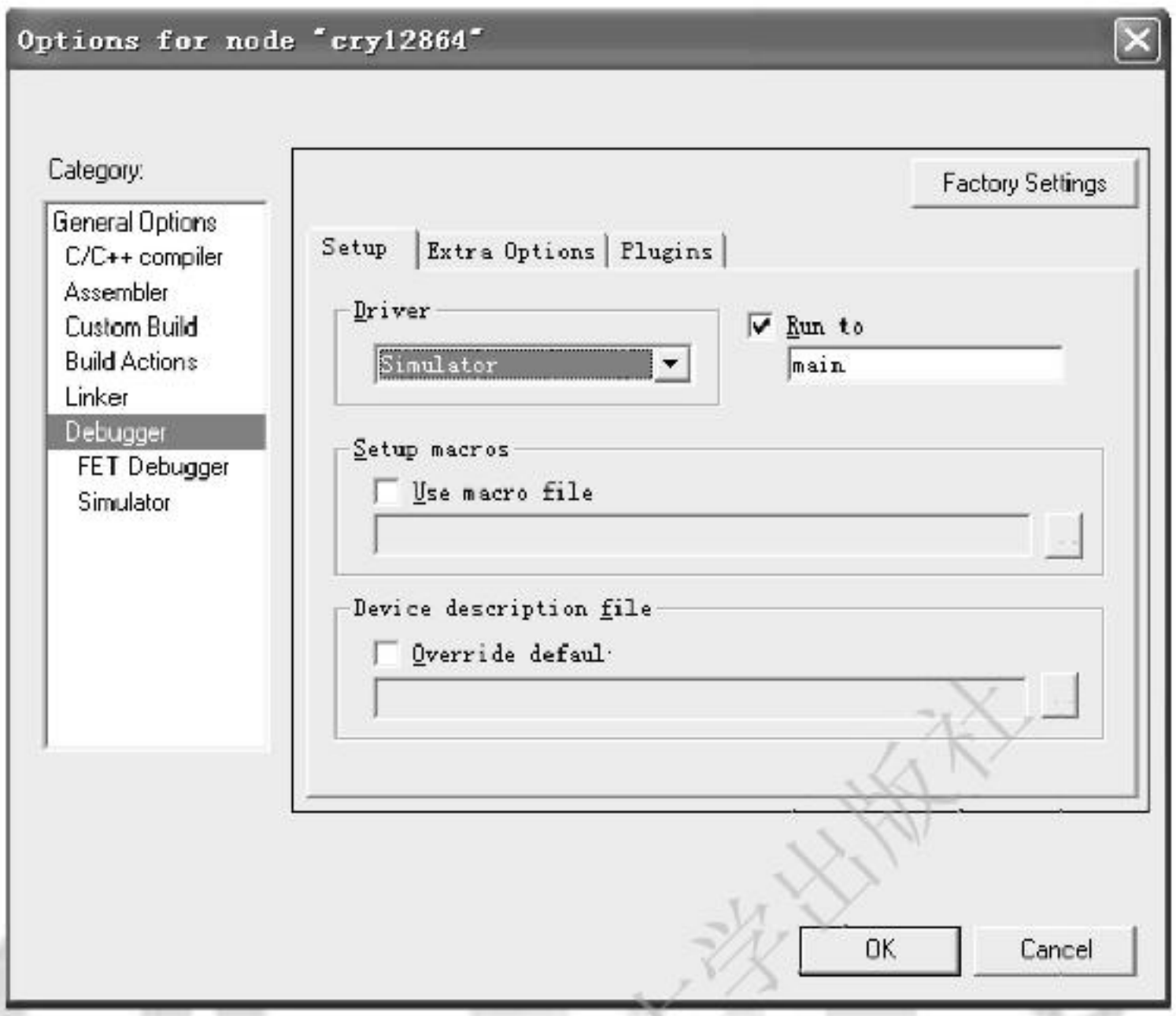


图 4－16　仿真器选择

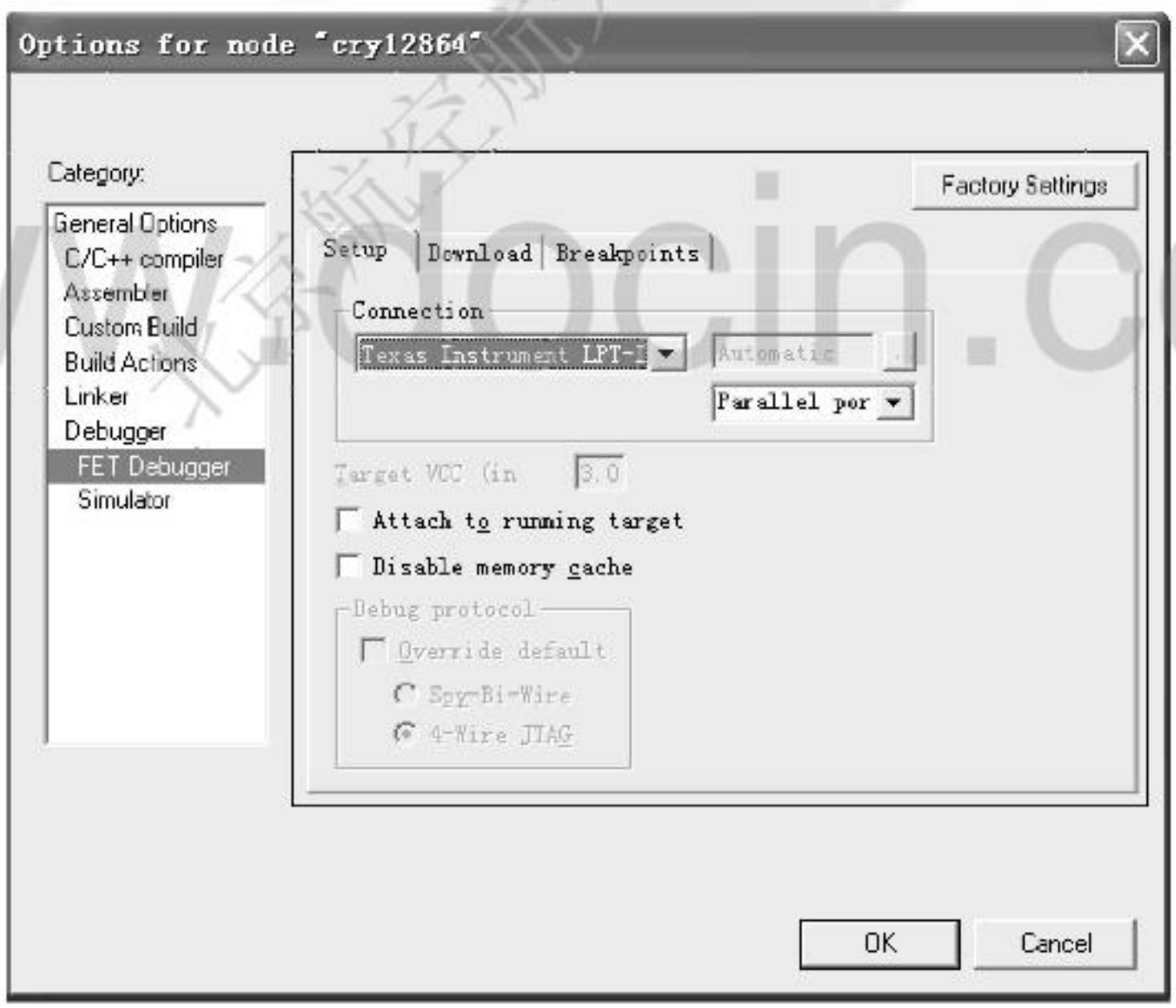


图 4－17　选择 JTAG 仿真器

4.3.2 Electronic Workbench 仿真过程举例

① 完成前面的设置以后，选中项目中的一个源文件（如. c、. cpp、. cc、. s、. asm、msa），选择 Project→Compile 命令，或者单击工具栏中的图标按钮，对源文件进行编译。如果有错误，请根据提示的出错信息，将错误修正以后重新编译。

② 保证所有的源文件都编译通过以后，选择 Project→Make 命令，或者单击工具栏中的图标按钮，对源文件进行创建连接。如果有错误，请根据提示的出错信息，将错误修正以后重新创建连接。

③ 创建通过以后，就可以进入调试阶段了。选择 Project→Debuger 命令，或者单击工具栏中的图标按钮 就可以进入调试界面了。图 4－18 是编辑界面整个项目创建连接成功以后的截图；图 4－19 是进入调试界面以后的截图。

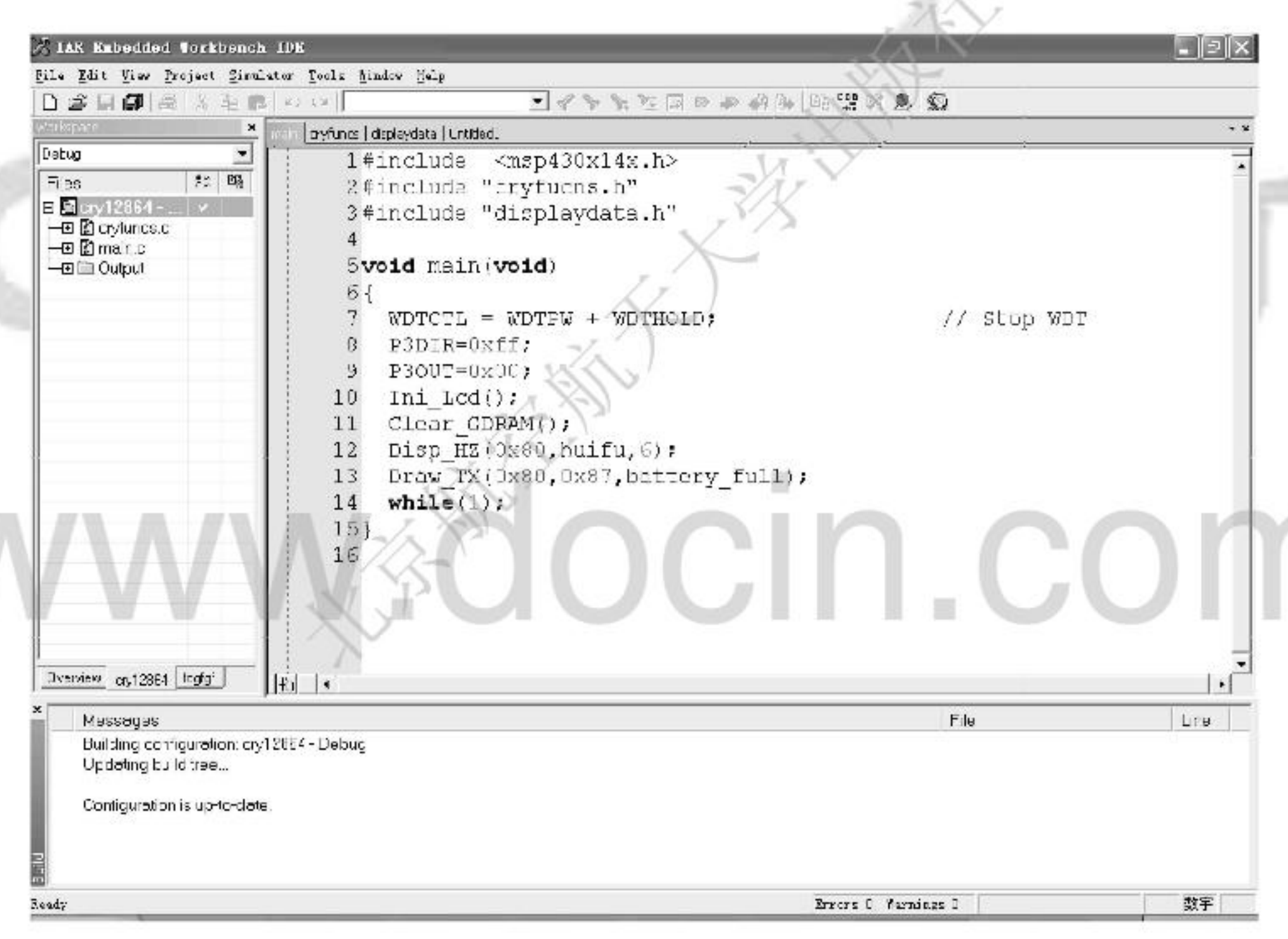


图 4－18 IAR for MSP430 界面

④ 在调试界面用户可以看到有一行被箭头选中，这表示程序计数器指向了此行的程序。将光标放在程序中的某一行，单击图标按钮 可以在这里设置一个断点，当程序运行到此处时会自动停止，用户可以观察某些变量；另外，也可以单击图标按钮 ，程序将自动运行到当