第 3 章

CHAPTER 3

C51 编程基础

计算机程序运行离不开硬件结构和软件代码。本章主要介绍单片机的结构、C51 语法基础、基本结构编程思路，以及掌握 Keil C 软件开发环境的安装、器件库、建工程、配置、编译等，此外，还深入剖析了流水灯经典案例。

3.1 单片机内部结构

3.1.1 MCS51 结构

MCS51 单片机的内部结构如图 3-1 所示。若除去图中的存储器电路和 I/O 部件，剩下

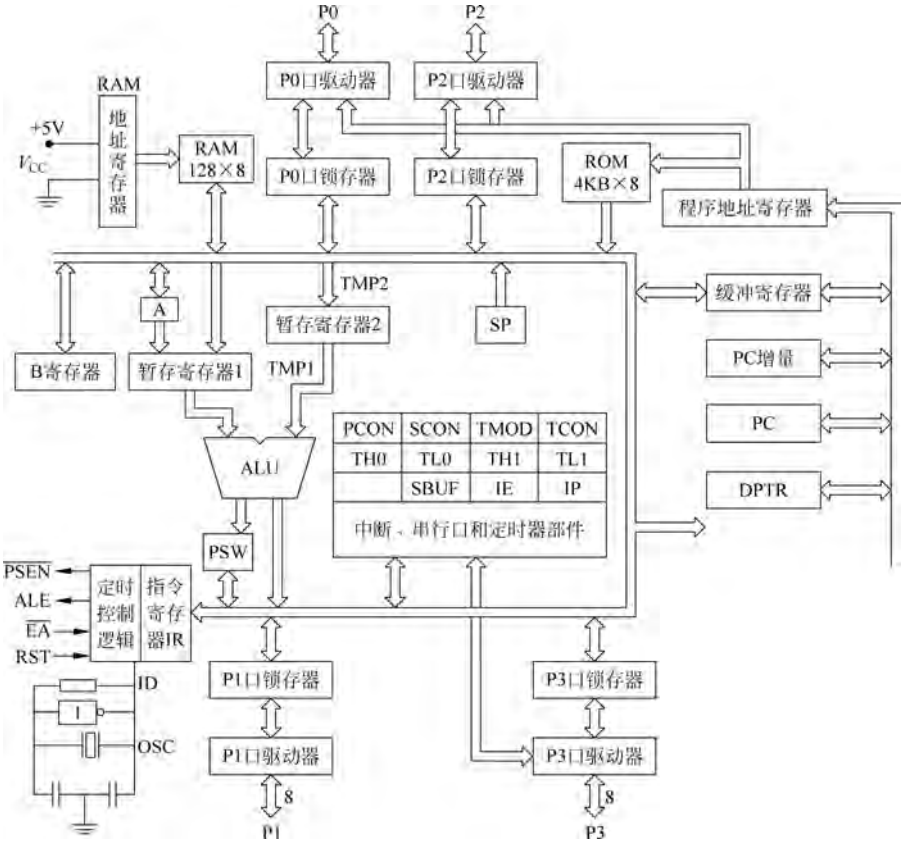

图 3-1　MCS51 的结构框图

的便是 CPU。它可以分为运算器和控制器两部分。运算器功能部件包括算术逻辑运算单元 ALU、累加器 A/ACC、寄存器 B、暂存寄存器 TMP1、TMP2、程序状态字寄存器 PSW 等。控制器功能部件包括程序计数器 PC、指令寄存器 IR、指令译码器 ID、定时控制逻辑电路 CU、数据指针寄存器 DPTR、堆栈指针 SP 及时钟电路等。

3.1.2 STC15 结构

STC-B 学习板上使用的单片机型号为 IAP15F2K61S2，这个单片机是宏晶公司生产的 STC15F2K60S2 系列单片机中的一种。STC15F2K60S2 系列单片机的内部结构图如图 3-2 所示。STC15F2K60S2 系列单片机中包含中央处理器(CPU)、程序存储器(Flash)、数据存储器(SRAM)、定时器、I/O 口、高速 A/D 转换、看门狗、UART 超高速异步串行通信口 1/串行通信口 2，CCP/PWM/PCA，1 组高速同步串行端口 SPI，片内高精度 R/C 时钟及高可靠复位等模块。

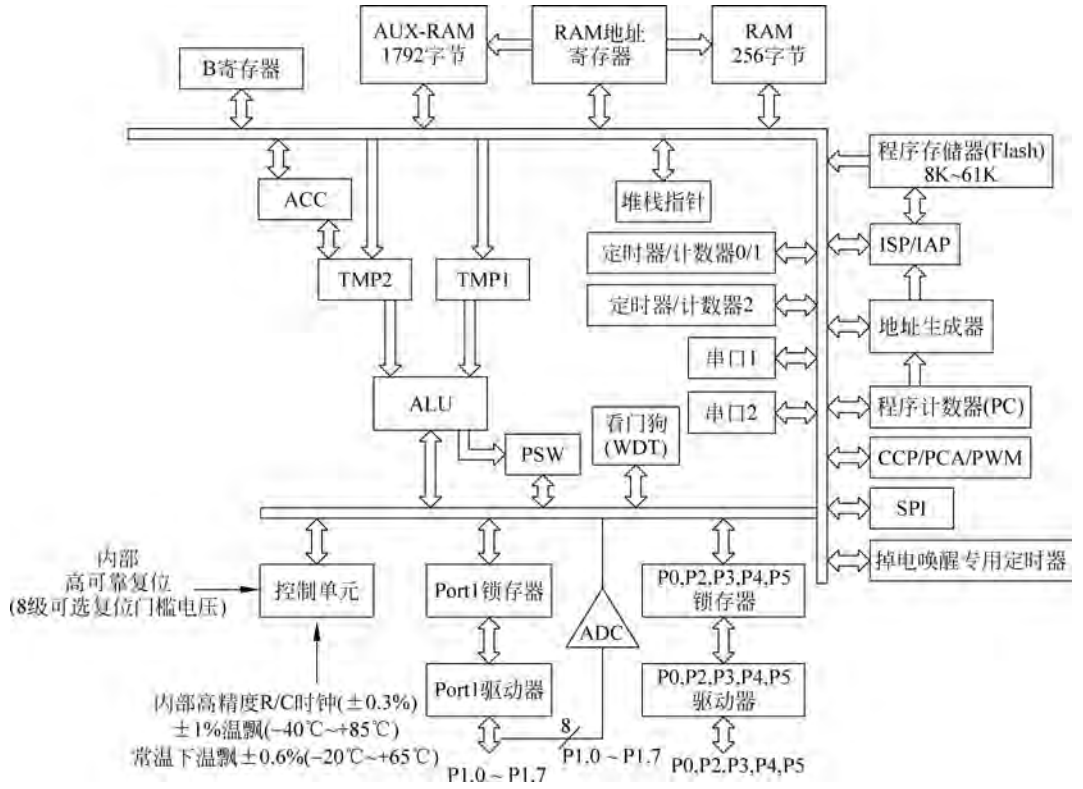

图 3-2 STC15F2K60S2 系列内部结构框图

3.2 Keil C51 基础

3.2.1 数据类型

每写一个程序，总离不开数据的应用，在学习 C51 语言的过程中理解和掌握数据类型是很关键的。在标准 C 语言中基本的数据类型为 char、int、short、long、float 和 double，而在 C51 编译器中 int 和 short 相同，float 和 double 相同，这里就不列出说明了。表 3-1 中列

出了 Keil C51 中 C 语言编译器所支持的数据类型。

表 3-1 Keil C51 编译器支持的数据类型

数据类型	长　度	值　域
unsigned char	单字节	0～255
signed char	单字节	－128～＋127
unsigned int	双字节	0～65 535
signed int	双字节	－32 768～＋32 767
unsigned long	4 字节	0～4 294 967 295
signed long	4 字节	－2 147 483 648～＋2 147 483 647
float	4 字节	±1.175 494E－38～±3.402 823E＋38
*	1～3 字节	对象的地址
bit	位	0 或 1
sfr	单字节	0～255
sfr16	双字节	0～65 535
sbit	位	0 或 1

下面来看看它们的具体定义。

1）char 字符类型

char 类型的长度是一个字节，通常用于定义处理字符数据的变量或常量。分无符号字符类型 unsigned char 和有符号字符类型 signed char，默认值为 signed char 类型。unsigned char 类型用字节中所有的位来表示数值，所能表达的数值范围是 0～255。signed char 类型用字节中最高位字节表示数据的符号，“0”表示正数，“1”表示负数，负数用补码表示。所能表示的数值范围是－128～＋127。unsigned char 常用于处理 ASCII 字符或用于处理小于或等于 255 的整型数。

2）int 整型

int 整型长度为两个字节，用于存放一个双字节数据。分有符号整型数 signed int 和无符号整型数 unsigned int，默认值为 signed int 类型。signed int 表示的数值范围是－32 768～＋32 767，字节中最高位表示数据的符号，“0”表示正数，“1”表示负数。unsigned int 表示的数值范围是 0～65 535。

3）long 长整型

long 长整型长度为四个字节，用于存放一个四字节数据。分有符号长整型 signed long 和无符号长整型 unsigned long，默认值为 signed long 类型。signed long 表示的数值范围是－2 147 483 648～＋2 147 483 647，字节中最高位表示数据的符号，“0”表示正数，“1”表示负数。unsigned long 表示的数值范围是 0～4 294 967 295。

4）float 浮点型

float 浮点型在十进制中具有 7 位有效数字，是符合 IEEE-754 标准的单精度浮点型数据，占用四个字节。

5）指针型

指针型本身就是一个变量，在这个变量中存放的指向另一个数据的地址。这个指针变量要占据一定的内存单元，对不一样的处理器长度也不尽相同，在 C51 中它的长度一般为 1～3 个字节。

6）bit 位标量

bit 位标量是 C51 编译器的一种扩充数据类型，利用它可定义一个位标量，但不能定义位指针，也不能定义位数组。它的值是一个二进制位，不是 0 就是 1，类似一些高级语言中的 Boolean 类型中的 True 和 False。

7）sfr 特殊功能寄存器

sfr 也是一种扩充数据类型，占用一个内存单元，值域为 0～255。利用它能访问 51 单片机内部的所有特殊功能寄存器。如用 sfr P1 =0x90 这一句定义 P1 为 P1 端口在片内的寄存器，在后面的语句中可以用 P1=255(对 P1 端口的所有引脚置高电平)之类的语句来操作特殊功能寄存器。

8）sfr16 16 位特殊功能寄存器

sfr16 占用两个内存单元，值域为 0～65535。sfr16 和 sfr 一样用于操作特殊功能寄存器，所不一样的是它用于操作占两个字节的寄存器，如定时器 T0 和 T1。

9）sbit 可寻址位

sbit 同样是单片机 C 语言中的一种扩充数据类型，利用它能访问芯片内部的 RAM 中的可寻址位或特殊功能寄存器中的可寻址位。

3.2.2 运算符

运算符就是完成某种特定运算的符号。运算符按其表达式中与运算符的关系可分为单目运算符、双目运算符和三目运算符。单目就是指需要有一个运算对象，双目就要求有两个运算对象，三目则要三个运算对象。表达式则是由运算及运算对象所组成的具有特定含义的式子。C 是一种表达式语言，表达式后面加“;”号就构成了一个表达式语句。

Keil C51 中的运算符主要有如下几种。

1）赋值运算符

对于“=”这个符号大家不会陌生的，功能是给变量赋值，称之为赋值运算符。它的作用就是把数据赋给变量，如：x=10；由此可见利用赋值运算符将一个变量与一个表达式连接起来的式子为赋值表达式，在表达式后面加“;”便构成了赋值语句。使用“=”的赋值语句格式如下：

```
变量 = 表达式;
```

示例如下：

```
a = 0xFF;                    //将常数十六进制数 FF 赋予变量 a
b = c = 33;                  //同时赋值给变量 b,c
d = e;                       //将变量 e 的值赋予变量 d
f = a+b;                     //将变量 a+b 的值赋予变量 f
```

由上面的例子可以知道赋值语句的意义就是先计算出“=”右边的表达式的值，然后将得到的值赋给左边的变量。

2）算术、增减量运算符

对于 a+b，a/b 这样的表达式大家都很熟悉，用在 C 语言中+，/就是算术运算符。C51 的算术运算符有如下几个，其中只有取正值和取负值运算符是单目运算符，其他则都是双目运算符：

- +加或取正值运算符

- －减或取负值运算符
- ＊乘运算符
- /除运算符
- ％取余运算符

算术表达式的形式为：

表达式 1　算术运算符　表达式 2

如：

```
a+b* (10-a), (x+9)/(y-a)
```

除法运算符和一般的算术运算规则有所不同，如果是两个浮点数相除，其结果为浮点数，如 10.0/20.0 所得值为 0.5，而两个整数相除时，所得值就是整数，如 7/3，值为 2。像别的语言一样 C 的运算符与有优先级和结合性，同样可用括号“()”来改变优先级。

＋＋增量运算符

－－减量运算符

这两个运算符是 C 语言中特有的一种运算符。在 VB，PASCAL 等都是没有的。作用就是对运算对象作加 1 和减 1 运算。要注意的是运算对象在符号前或后，其含义都是不同的，虽然同是加 1 或减 1。如：I＋＋，＋＋I，I－－，－－I。

I＋＋(或 I－－)是先使用 I 的值，再执行 I＋1(或 I－1)

＋＋I(或－－I)是先执行 I＋1(或 I－1)，再使用 I 的值。

增减量运算符只允许用于变量的运算中，不能用于常数或表达式。

3) 关系运算符

对于关系运算符，在 C 中有六种关系运算符：＞(大于)，＜(小于)，＞＝(大于或等于)，＜＝(小于或等于)，＝＝(等于)，!＝(不等于)。

计算机的语言也不过是人类语言的一种扩展，这里的运算符同样有着优先级别。前四个具有相同的优先级，后两个也具有相同的优先级，但是前四个的优先级要高于后两个的。

当两个表达式用关系运算符连接起来时，这时就是关系表达式。关系表达式通常是用来判别某个条件是否满足。要注意的是用关系运算符的运算结果只有 0 和 1 两种，也就是逻辑的真与假，当指定的条件满足时结果为 1，不满足时结果为 0。

关系表达式的形式为：

表达式 1　关系运算符　表达式 2

如：

```
I<J,I= =J,(I=4)>(J=3),J+I>J
```

4) 逻辑运算符

关系运算符所能反映的是两个表达式之间的大小等于关系，那逻辑运算符则是用于求条件式的逻辑值，用逻辑运算符将关系表达式或逻辑量连接起来就是逻辑表达式了。也许你会对为什么“逻辑运算符将关系表达式连接起来就是逻辑表达式了”这一个描述有疑惑的地方。其实之前说过“要注意的是用关系运算符的运算结果只有 0 和 1 两种，也就是逻辑的真与假”，换句话说也就是逻辑量，而逻辑运算符就用于对逻辑量运算的表达。逻辑表达式

的一般形式为：

- 逻辑与：条件式 1 && 条件式 2
- 逻辑或：条件式 1 ||条件式 2
- 逻辑非：！条件式 2

逻辑与，就是当条件式 1“与”条件式 2 都为真时结果为真（非 0 值），否则为假（0 值）。也就是说运算会先对条件式 1 进行判断，如果为真（非 0 值），则继续对条件式 2 进行判断，当结果也为真时，逻辑运算的结果为真（值为 1），如果结果不为真时，逻辑运算的结果为假（0 值）。如果在判断条件式 1 时就不为真的话，就不用再判断条件式 2 了，而直接给出运算结果为假。

逻辑或是指只要两个运算条件中有一个为真时，运算结果就为真，只有当条件式都不为真时，逻辑运算结果才为假。

逻辑非是把逻辑运算结果值取反，意思是如果条件式的运算值为真，进行逻辑非运算后则结果变为假，条件式运算值为假时最后逻辑结果为真。

同样逻辑运算符也有优先级别，!（逻辑非）→&&（逻辑与）→||（逻辑或），逻辑非的优先级最高。

5）位运算符

位运算符的作用是按位对变量进行运算，但是并不改变参与运算的变量的值。如果要求按位改变变量的值，则要利用相应的赋值运算。还有就是位运算符是不能用来对浮点型数据进行操作的。C51 中共有 6 种位运算符。位运算一般的表达形式如下：

变量 1　位运算符　变量 2

位运算符也有优先级，从高到低依次是：“～”（按位取反）→“≪”（左移）→“≫”（右移）→“&”（按位与）→“^”（按位异或）→“|”（按位或）。

6）复合赋值运算符

复合赋值运算符就是在赋值运算符“＝”的前面加上其他运算符。以下是 C 语言中的复合赋值运算符：

＋＝加法赋值　　≫＝右移位赋值
－＝减法赋值　　&＝逻辑与赋值
＊＝乘法赋值　　|＝逻辑或赋值
/＝除法赋值　　^＝逻辑异或赋值
％＝取模赋值　　～＝逻辑非赋值
≪＝左移位赋值

复合运算的一般形式为：

变量　复合赋值运算符　表达式

其含义就是变量与表达式先进行运算符所要求的运算，再把运算结果赋值给参与运算的变量。其实这是 C 语言中一种简化程序的一种方法，凡是二目运算都可以用复合赋值运算符去简化表达。

例如：

a＋＝56 等价于 a＝a＋56　y/＝x＋9 等价于 y＝y/(x＋9)

很明显采用复合赋值运算符会降低程序的可读性，但这样却可以使程序代码简单化，并能提高编译的效率。对于初学 C 语言的朋友在编程时最好还是根据自己的理解力和习惯

去使用程序表达的方式，不要一味追求程序代码的短小。

7）逗号运算符

C 语言中逗号还是一种特殊的运算符，也就是逗号运算符，可以用它将两个或多个表达式连接起来，形成逗号表达式。逗号表达式的一般形式为：

```
表达式 1,表达式 2,表达式 3,…,表达式 n
```

这样用逗号运算符组成的表达式在程序运行时，是从左到右计算出各个表达式的值，而整个用逗号运算符组成的表达式的值等于最右边表达式的值，就是“表达式 n”的值。在实际应用中，大部分情况下，使用逗号表达式的目的只是为了分别得到多个表达式的值，而并不一定要得到和使用整个逗号表达式的值。要注意的还有，并不是在程序的任何位置出现的逗号，都可以认为是逗号运算符，如函数中的参数，同类型变量的定义中的逗号只是用来间隔之用而不是逗号运算符。

8）条件运算符

C 语言中有一个三目运算符，它是“?：”条件运算符，它要求有三个运算对象。它可以把三个表达式连接构成一个条件表达式。条件表达式的一般形式如下：

```
逻辑表达式?表达式 1 :表达式 2
```

条件运算符的作用简单来说就是根据逻辑表达式的值选择使用表达式的值。当逻辑表达式的值为真时（非 0 值）时，整个表达式的值为表达式 1 的值；当逻辑表达式的值为假（值为 0）时，整个表达式的值为表达式 2 的值。要注意的是条件表达式中逻辑表达式的类型可以与表达式 1 和表达式 2 的类型不一样。下面是一个逻辑表达式的例子。

如有 a＝1，b＝2 时，我们的要求是取 ab 两数中的较小的值放入 min 变量中，可以写为：

```
if (a < b) min = a;
else       min = b;
```

当 a 用条件运算符去构成条件，表达式就变得简单明了，如果用条件运算符来描述，则应该写成

```
min = ( a < b) ? a : b ;
```

很明显它的结果和含意都和上面的一段程序是一样的，但是代码却比上一段程序精简很多，编译的效率也相对要高，但有着和复合赋值表达式一样的缺点就是可读性相对效差。

9）指针和地址运算符

指针是 C 语言中一个十分重要的概念，也是学习 C 语言中的一个难点。在这里我们先来了解一下 C 语言中提供的两个专门用于指针和地址的运算符：＊取内容，&. 取地址。

取内容和地址的一般形式分别为：

```
变量 = * 指针变量
指针变量 = & 目标变量
```

取内容运算是将指针变量所指向的目标变量的值赋给左边的变量；取地址运算是将目标变量的地址赋给左边的变量。要注意的是：指针变量中只能存放地址（也就是指针型数

据),一般情况下不要将非指针类型的数据赋值给一个指针变量。

10) sizeof 运算符

sizeof 是用来求数据类型、变量或是表达式的字节数的一个运算符,但它并不像“=”之类运算符那样在程序执行后才能计算出结果,它是直接在编译时产生结果的。它的语法如下:

```
sizeof (数据类型)  或  sizeof (表达式)
```

下面是两句应用例句:

```
printf("char 是多少个字节? %bd 字节\n",sizeof(char));
printf("long 是多少个字节? %bd 字节\n",sizeof(long));
```

结果是:

```
char 是多少个字节? 1 字节
long 是多少个字节? 4 字节
```

3.2.3 条件与循环语句

1. 条件语句

C 语言中的条件语句分为 if…else 条件语句和 switch…case 条件语句两种。

1) if…else 条件语句

if…else 条件语句又被称为分支语句,其关键字是由 if 构成。C 语言提供了 3 种形式的条件语句:

(1) if (条件表达式)语句

当条件表达式的结果为真时,就执行语句,否则就跳过。

如: if (a==b) a++; 当 a 等于 b 时,a 就加 1

(2) if (条件表达式)语句 1 else 语句 2

当条件表达式成立时,就执行语句 1,否则就执行语句 2。

如 if (a==b)a++; else a--; 当 a 等于 b 时,a 加 1,否则 a-1。

(3) if (条件表达式 1)语句 1

else if (条件表达式 2)语句 2

else if (条件表达式 3)语句 3

……

else if (条件表达式 m)语句 n

else 语句 m

这是由 if else 语句组成的嵌套,用来实现多方向条件分支,使用应注意 if 和 else 的配对使用,要是少了一个就会语法出错,else 总是与最临近的 if 相配对。

2) switch…case 语句

多个条件语句可以实现多方向条件分支,但是可以发现使用过多的条件语句实现多方向分支会使条件语句嵌套过多,程序冗长,这样读起来也很不好读。这时使用开关语句同样可以达到处理多分支选择的目的,又可以使程序结构清晰。

它的语法如下：

```
switch (表达式)
 {
    case 常量表达式 1: 语句 1; break;
    case 常量表达式 2: 语句 2; break;
    case 常量表达式 3: 语句 3; break;
    case 常量表达式 n: 语句 n; break;
    default: 语句 }
```

运行中 switch 后面的表达式的值将会作为条件，与 case 后面的各个常量表达式的值相对比，如果相等则执行后面的语句，再执行 break(间断语句)语句，跳出 switch 语句。如果 case 没有和条件相等的值时就执行 default 后的语句。当要求没有符合的条件时不做任何处理，则可以不写 default 语句。

2. 循环语句

C 语言中的循环语句分为三种形式，分别是：while 循环、do…while 循环和 for 循环。这三种循环语句在功能上存在细微的差别，但共同特点是实现一个循环体，可以使程序反复执行一段代码。

1) while 循环

用 while 语句的一般形式如下：

```
while(表达式)语句
```

其中“语句”就是循环体。其中循环体只能是一个语句，可以是一个简单语句，也可以是一个复合语句(用花括号括起来的语句)。“表达式”也称循环条件表达式，用于控制循环体执行的次数。如果表达式为“真”，就执行循环体；为“假”，就不执行循环体。

用 while 语句可简单的记为：只要当循环条件表达式为“真”，就执行循环体语句。

while 循环的特点是：先判断，后执行！

2) do…while 循环

do…while 循环语句的一般形式：

```
do
    语句
while(表达式);
```

其中语句就是循环体。do…while 的执行过程是：先执行循环体，再检查判断条件是否成立，若成立，再执行循环体。do…while 和 while 循环语句的区别是：前者是至少执行一次，后者是可以一次也不执行。

3) for 循环

for 语句的一般形式为：

```
for( 表达式 1;表达式 2;表达式 3)
    语句
```

三个表达式的主要作用是：

表达式 1：设置初始条件，只执行一次。可以为零个，一个或多个变量设置初值。

表达式 2：是循环条件表达式，用来判定是否执行循环。在每次执行循环体前先执行该

表达式，决定是否继续执行循环。

表达式 3：作为循环的调整，执行完循环体后才执行的。

3.3 Keil 开发环境

Keil C51 是美国 Keil Software 公司出品的 51 系列兼容单片机 C 语言软件开发系统，与汇编相比，C 语言在功能、结构性、可读性、可维护性上有明显的优势，因而易学易用。Keil 提供了包括 C 编译器、宏汇编、链接器、库管理和一个功能强大的仿真调试器等在内的完整开发方案，通过一个集成开发环境（μVision）将这些部分组合在一起。运行 Keil 软件需要 Windows NT、Windows 7、Window XP 等操作系统。如果你使用 C 语言编程，那么 Keil 几乎就是你的不二之选，即使不使用 C 语言而仅用汇编语言编程，其方便易用的集成环境、强大的软件仿真调试工具也会令开发事半功倍。

本节主要介绍 Keil C51 的用法，包括安装 Keil 软件，安装 STC 数据库文件，Keil 建立工程、工程配置、工程的编译及下载等内容。在使用 Keil 软件之前，要保证在用户的计算机上装有一套稳定可靠的 Keil C51 版本进行学习。

3.3.1 Keil C51 软件安装

要使用 Keil C51 进行编程，首选需要安装该软件。

Keil C51 的安装过程是很简单，首先打开 Keil4 安装文件夹并双击 c51_v9.51a 应用程序启动安装。

之后单击 Next 按钮，会弹出一个 License Agreement 对话框，此时选择 I agree to…复选框并单击 Next 按钮，会出现一个选择安装路径的对话框，单击 Browse 按钮选择想安装的目录。

之后需要用户填写一些个人信息，这里 4 个框可以随意填写。再单击 Next 按钮，出现安装界面之后等待片刻，软件就会安装完毕。

3.3.2 添加 STC 系列单片机数据库

因为 Keil C51 软件中不带 STC 系列单片机的数据库，在使用 STC 系列单片机之前，需要用 STC 公司所提供的 STC-ISP 在线编程软件将 STC 系列单片机的数据库添加到 Keil C51 软件设备库中，操作流程如下：

（1）运行 STC-ISP 在线编程软件，目前最新的版本是 V6.86C。选择“Keil 仿真设置”标签。

（2）单击“添加型号和头文件到 Keil 中，添加 STC 仿真器驱动到 Keil 中”按钮，如图 3-3 所示。在浏览文件夹中选择 Keil 的安装目录，如图 3-4 所示，单击“确定”按钮即完成添加工作。

图 3-3 Keil 仿真设置

3.3.3 Keil 工程的建立

首先我们要养成一个习惯：最好先建立一个空文件夹，把工程文件放到里面，以避免和其他文件混合。

如图 3-5 所示先创建了一个名为“流水灯[1]”的文件夹。

图 3-4　浏览文件夹

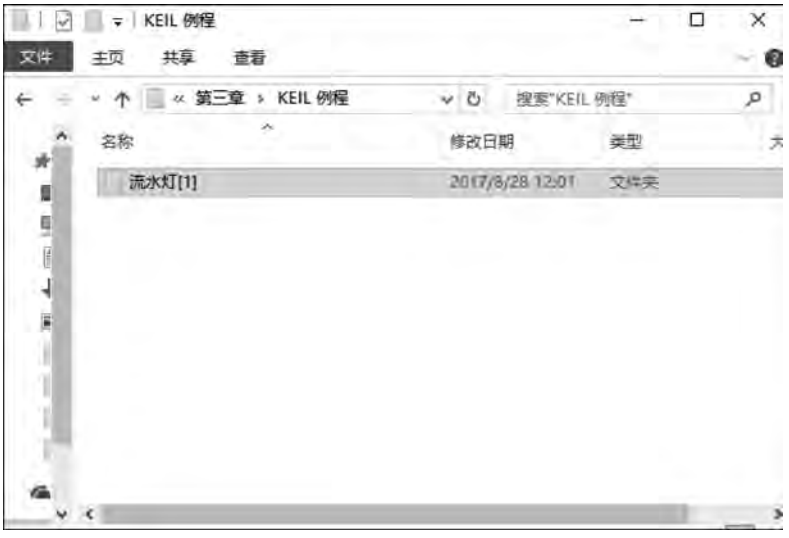

图 3-5　建立工程文件夹

双击桌面上的 Keil uVision4 图标，出现软件启动画面，紧接着出现编辑界面，如图 3-6 所示。

建立一个新的工程，单击 Project 菜单中的 New μVision Project 选项，如图 3-7 所示。

选择工程要保存的路径，并输入工程文件名。Keil 的一个工程里通常含有很多小文件，为了方便管理，通常我们将一个工程放在一个单独的文件夹下。比如，在此新建一个名

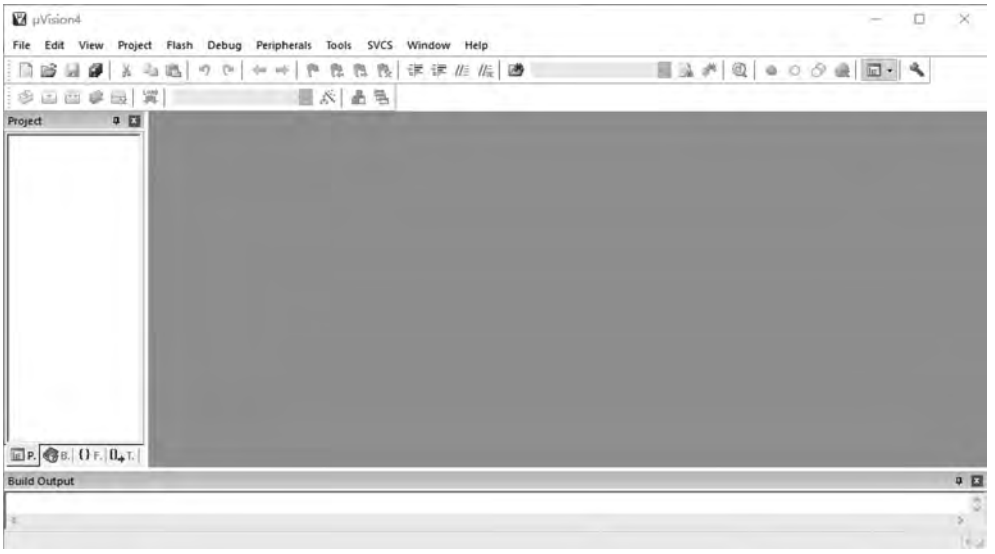

图 3-6 Keil 软件编辑界面

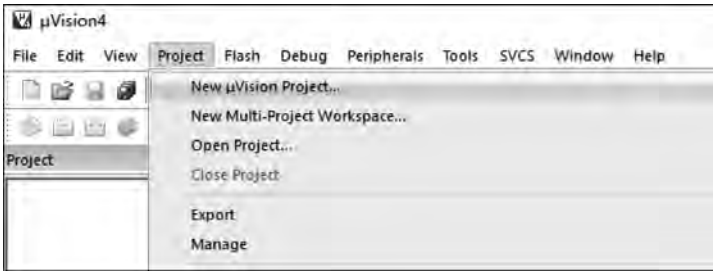

图 3-7 新建工程

为“led”的工程并放在工程文件夹“流水灯[1]”下。单击“保存”按钮后的文件扩展名为uvproj,这是 Keil uVision4 项目文件扩展名,以后我们可以直接单击此文件以打开之前做的项目,如图 3-8 所示。

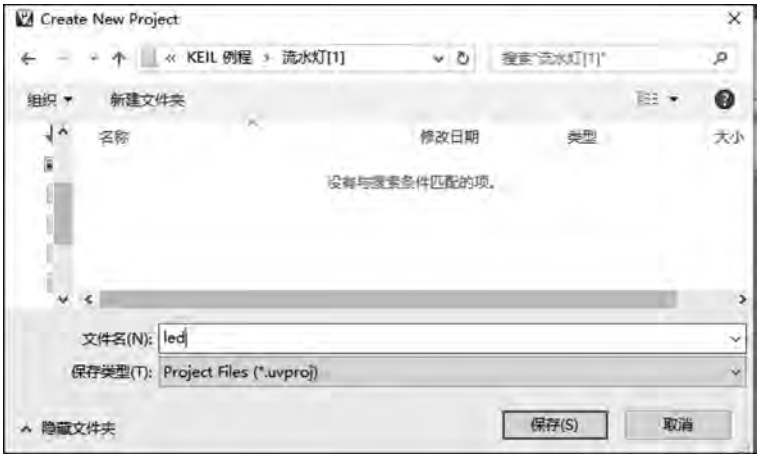

图 3-8 保存工程

上一步单击“保存”按钮还会弹出一个对话框，要求选择 CPU 数据库文件，下拉选择 STC MCU Database，并单击 OK 按钮确定，如图 3-9 所示。

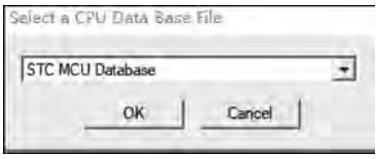

图 3-9　选择 CPU 数据库文件

紧接着还会弹出一个对话框，如图 3-10 所示，要求用户选择单片机型号，用户可以根据实际使用的单片机型号来选择。我们所使用的 STC-B 学习板可选择 STC15F2K60S2 芯片。选择完 STC15F2K60S2 之后，右边 Description 栏中是对应型号单片机的基本说明，主要介绍其功能特点，然后单击 OK 按钮，会弹出询问选择是否添加启动文件，一般根据需要单击 OK 按钮确定返回。

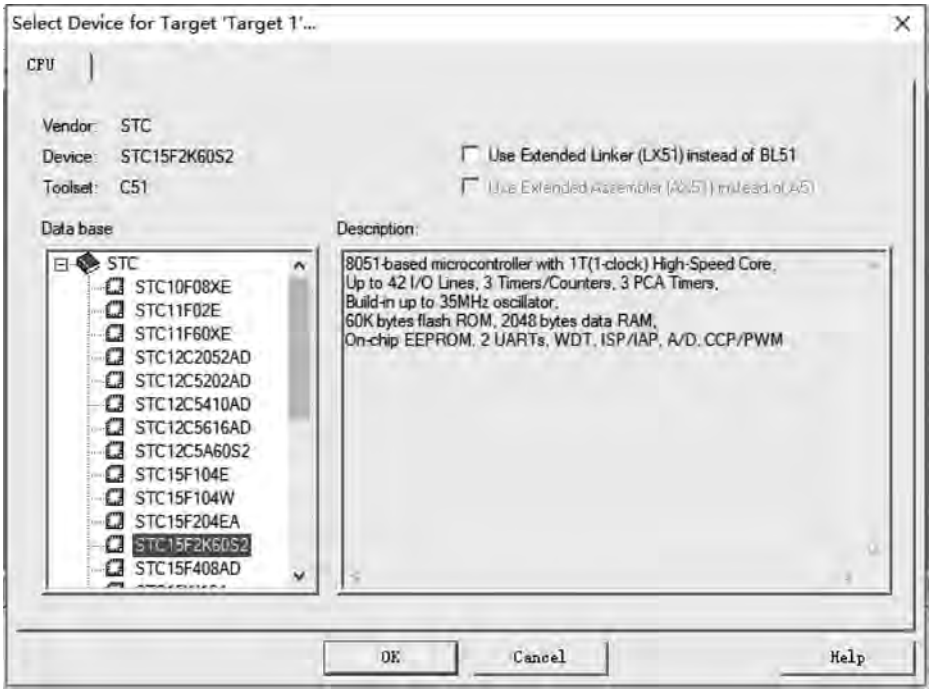

图 3-10　选择单片机型号

完成以上步骤后，其实还没有建立好一个完整的工程，虽然工程名有了，但工程中还没有任何文件及代码，接下来要做的是添加文件及代码。以添加 C 程序文件为例：单击 File 菜单中的 New...菜单项，或单击界面上的快捷图标来新建一个文本，如图 3-11 所示。

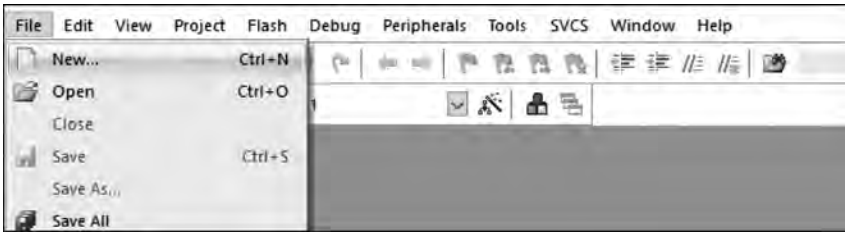

图 3-11　添加文件

此时可以在文本中输入用户程序，这里只复制、粘贴实例 1 中的源代码，暂时不需理会代码的具体含义，之后单击保存图标，如图 3-12 所示。在出现窗口界面中，如图 3-13 所示，在“文件名：”编辑框中输入要保存的文件名，同时必须输入正确的扩展名。注意，如果用 C 语言编写程序则扩展名必须为“.c”，如果用会汇编语言编写，则扩展名必须为“.asm”。这里的文件名不一定要和工程名相同，用户可以随意填写文件名，然后单击“保存”按钮完成并返回。

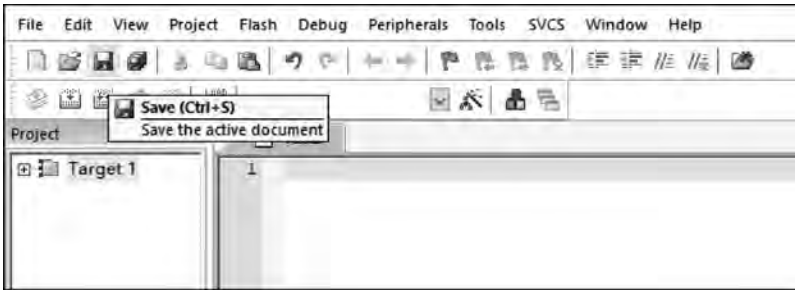

图 3-12 保存文件

图 3-13 加后缀名保存文件

接下来需要把刚创建的“led.c”源程序文件加入到工程项目文件中。回到编辑界面，单击 Project 窗口中的 Target 1 前面的“+”号展开，然后在 Source Group 1 选项上单单击鼠标右键，弹出的快捷菜单中选择 Add Existing File to Group ‘Source Group 1’... 选项，如图 3-14 所示。

选中“led.c”文件，单击 Add 按钮，再单击 Close 按钮，即添加文件成功，如图 3-15 所示。

然后再单击 Project 窗口左侧栏中 Sourse Group 1 前面的“+”号展开，屏幕窗口如图 3-16 所示。这时注意到 Sourse Group 1 文件夹中多了一个子项“led.c”，当一个工程中有多个代码文件时，都要加在这个文件夹下，源代码文件就与工程关联起来了。

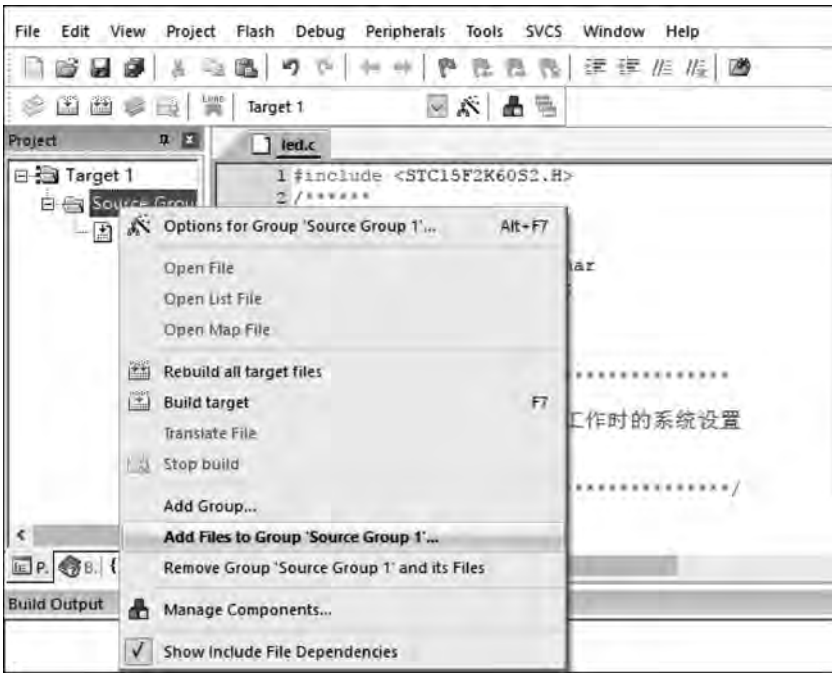

图 3-14　将文件加入工程的菜单

图 3-15　选中文件后的对话框

一个完整的工程，通常还需添加头文件。通常是把头文件“STC15F2K60S2. H”从已有案例文件夹中复制添加到工程文件夹“流水灯[1]”下，然后同“led. c”的添加方式一样，将“STC15F2K60S2. H”添加到工程目录文件中，如图 3-17 所示。为了减少重复工作，也可将文件“STC15F2K60S2. H”放在 Keil/C51/INC 类库路径下。

完成代码的编写、修改、保存后。在编译生成“. hex”的可执行文件之前，还要进行如下的设置：在 Project 菜单下单击 Options for Target 'Target 1'... 或直接在程序上方单击 均可，弹出对话框 Options for Target ‘Target 1’。在 Output 标签中勾选上“Create HEX File”，使编

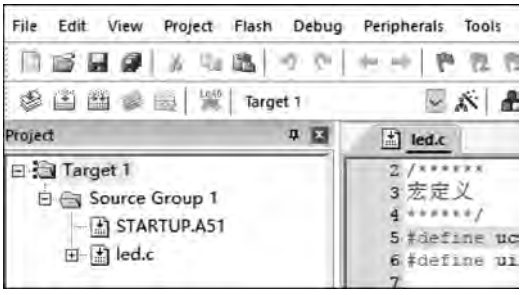

图 3-16 将文件加入工程后的屏幕窗口

图 3-17 将头文件加入工程文件目录下

译器输出单片机需要的 HEX-80 文件，如图 3-18 所示。

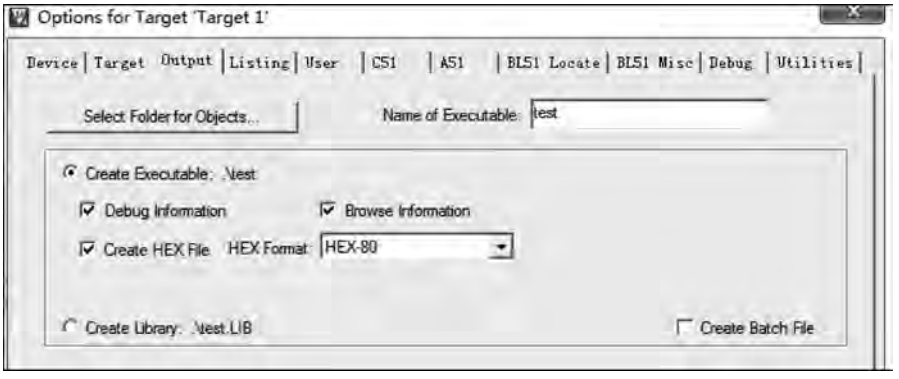

图 3-18 选择生成 HEX 文件

在此补充一点，单片机只能下载 HEX 文件或 BIN 文件，HEX 文件是十六进制文件，BIN 是二进制文件，这两种文件可以通过软件互相转换，其实际内容都是一样的。同时也可以将选项 Browse Information 选中，选中后在程序中某处调用函数的地方单击右键选择打开函数后，可直接跳转到该函数体内，这个功能在编写比较大的程序中会经常用到。

工程项目创建和设置全部完成，最后单击 按钮进行编译，生成. hex 文件，如图 3-19 所示。

完成上述步骤，编译成功后便可将程序下载到 STC-B 学习板上，进行观察并测试。

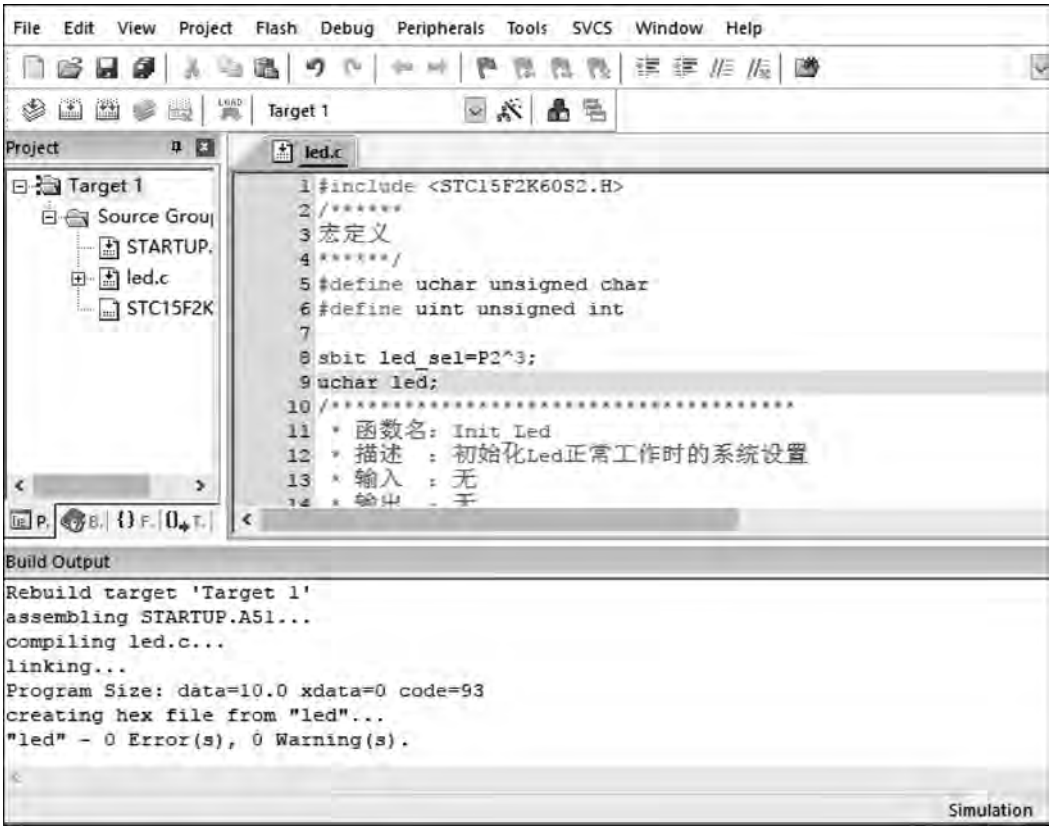

图 3-19　编译成功信息

3.4　任务　流水灯

流水灯是测试发光二极管是否正常工作和学习使用发光二极管的经典案例。用 C 语言编写流水灯的程序，理解并掌握它，也就意味着踏入了单片机学习的第一道门槛。

3.4.1　发光二极管电路

发光二极管电路工作原理如图 3-20，虚线框内是 LED 灯部分电路，P0 口的 8 位输出分别连接了 8 个发光二极管 L0～L7 的阳极，P2.3 经过一个反相器连接到 8 个发光二极管 L0～L7 的阴极(共阴极)。根据二极管的单向导通性，即当阳极为高(对应 P0 口位为 1)、阴极为低时，二极管导通，否则不导通。若 P2.3 输出信号为低电平“0”，则二极管的阴极都为高电平，此时无论 P0 输出的是“1”还是“0”，二极管都不会导通，也就不会发光。因此想要发光二极管导通，必须先设置 P2.3 输出信号为“1”，再通过设置 P0，点亮想要点亮的发光二极管。

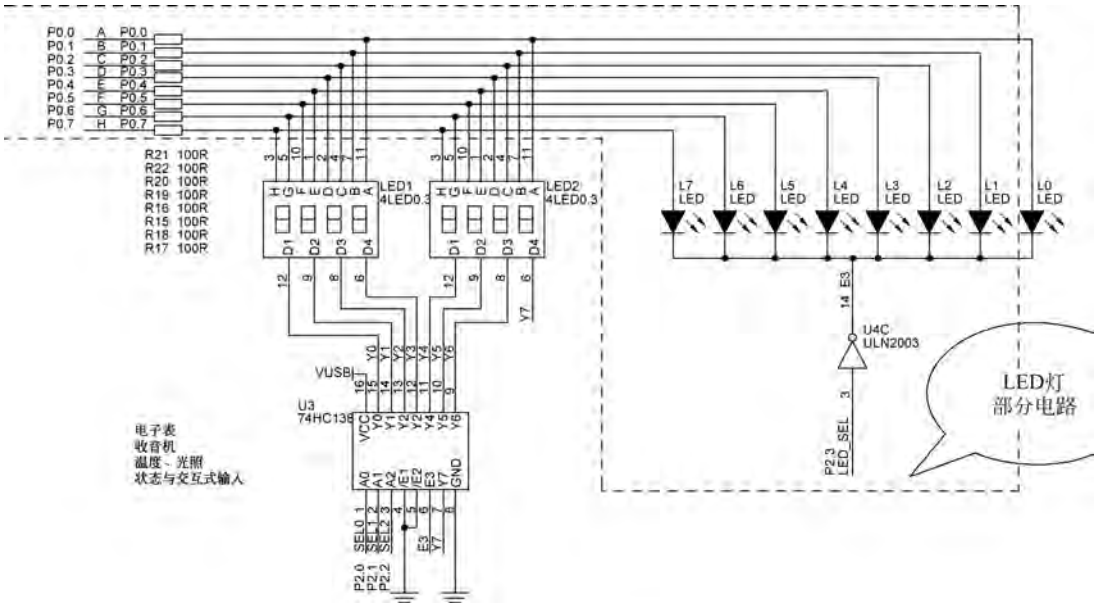

图 3-20 数码管与发光二极管硬件电路图

3.4.2 规划设计

目标：本节通过一个简单的点亮流水灯的案例，初步了解如何编写单片机的程序。

资源：STC-B 学习板、PC、Keil 4 软件、STC-ISP 软件(V6.8 以上)。

任务：

(1) 再次下载本工程 Hex 文件，并对照测试结果仔细观察将实现的功能。

(2) 利用 C51 编程实现任务功能。

功能：学习板上的 8 个 LED 灯将循环地从右往左依次点亮，时间间隔为 200ms。

测试结果：将程序下载到 STC 学习板上后，如图 3-21 所示，观察到 8 个 LED 灯一开始仅最右边的 L0 亮，过了小段时间，L1 亮 L0 灭，又过了会 L2 亮 L1 灭……如此反复实现 LED 灯从右往左流动的效果。

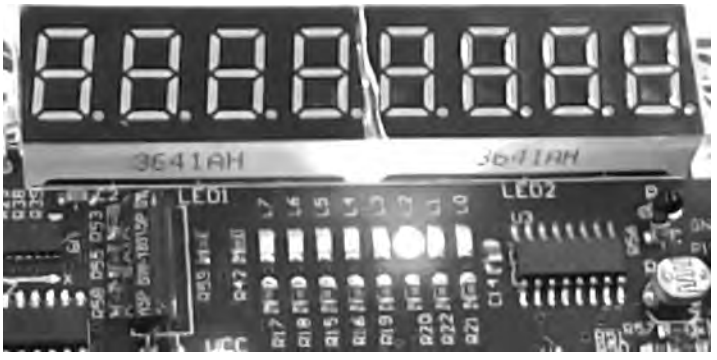

图 3-21 案例测试结果

3.4.3 实现步骤

1. 参考代码

```
#include <STC15F2K60S2.H>                    //STC15F2K60S2 单片机芯片头文件
#define uchar unsigned char
#define uint unsigned int
sbit led_sel = P2^3;                         //声明单片机 P2.3 口
uchar led;
//初始函数,初始化 Led 正常工作时的系统设置
void init()
{
    P0M1 = 0x00;                             //P0 设推挽
    P0M0 = 0xff;
    P2M1 = 0x00;                             //P2.3 设推挽
    P2M0 = 0x08;
    led_sel = 1;                             //使能发光二极管电路
     led = 0x01;                             //Led 初始为 L0 点亮
}
//延时函数,通过传入参数 n,设置延时时长为 n 毫秒
void delay_ms(uint n)
{
    while(n){
    uchar i, j;
    i = 11;
    j = 190;
    do
    {
        while ( -- j);
    } while ( -- i);
    n -- ;  }
}
//主函数
void main()
{
    init();
    while(1)
    {
        P0 = led;                            //点亮第一个灯
        delay_ms(200);                       //延时 200ms
        if(led == 0x80)                      //再从头开始亮灯
            led = 0x01;
        else
            led = led << 1;                  //左移一位
    }
}
```

2. 分析说明

在输入源代码时一定要将输入法切换为全英文状态,保证编辑时的符号为英文格式,否

则编译器会产生编译错误。

1）STC15F2K60S2.H 头文件的作用

在代码中引用头文件，其实际意义就是将这个头文件中的全部内容放到引用头文件的位置处，免去每次编写同类型程序都要将头文件中的语句重复编写。

在代码中加入头文件有两种方式，分别为#include <STC15F2K60S2.H>和#include "STC15F2K60S2.H"，包含头文件时都不需要在后面加分号，两种书写的区别在于当使用<>包含头文件时，编译器先进入到软件安装文件夹处开始搜索这个头文件，也就是 Keil/C51/INC 这个文件夹下，当使用双引号""包含头文件时，编译器先进入到当前工程所在文件夹处开始搜索该头文件，若找不到该头文件，还会去类库路径里查找，再找不到编译器将报错。

STC15F2K60 系列头文件部分如下所示。

```
#ifndef __STC15F2K60S2_H_
#define __STC15F2K60S2_H_

//////////////////////////////////////////////////
//注意: STC15W4K32S4 系列的芯片,上电后所有与 PWM 相关的 I/O 口均为
// 高阻态,需将这些口设置为准双向口或强推挽模式方可正常使用
//相关 IO: P0.6/P0.7/P1.6/P1.7/P2.1/P2.2
// P2.3/P2.7/P3.7/P4.2/P4.4/P4.5
//////////////////////////////////////////////////
//包含本头文件后,不用另外再包含"REG51.H"
//内核特殊功能寄存器                          //复位值 描述
sfr ACC         =       0xE0;               //0000,0000 累加器 Accumulator
sfr B           =       0xF0;               //0000,0000 B 寄存器
sfr PSW         =       0xD0;               //0000,0000 程序状态字
sbit CY         =       PSW^7;
sbit AC         =       PSW^6;
sbit F0         =       PSW^5;
sbit RS1        =       PSW^4;
sbit RS0        =       PSW^3;
sbit OV         =       PSW^2;
sbit P          =       PSW^0;
sfr SP          =       0x81;               //0000,0111 堆栈指针
sfr DPL         =       0x82;               //0000,0000 数据指针低字节
sfr DPH         =       0x83;               //0000,0000 数据指针高字节
//I/O 口特殊功能寄存器
sfr P0          =       0x80;               //1111,1111 端口 0
sbit P00        =       P0^0;
sbit P01        =       P0^1;
sbit P02        =       P0^2;
sbit P03        =       P0^3;
sbit P04        =       P0^4;
sbit P05        =       P0^5;
sbit P06        =       P0^6;
sbit P07        =       P0^7;
…
```

```
//系统管理特殊功能寄存器
sfr PCON         =       0x87;                  //0001,0000 电源控制寄存器
sfr AUXR         =       0x8E;                  //0000,0000 辅助寄存器
sfr AUXR1        =       0xA2;                  //0000,0000 辅助寄存器 1
sfr P_SW1        =       0xA2;                  //0000,0000 外设端口切换寄存器 1
sfr CLK_DIV      =       0x97;                  //0000,0000 时钟分频控制寄存器
sfr BUS_SPEED    =       0xA1;                  //xx10,x011 总线速度控制寄存器
sfr P1ASF        =       0x9D;                  //0000,0000 端口 1 模拟功能配置寄存器
sfr P_SW2        =       0xBA;                  //0xxx,x000 外设端口切换寄存器

//中断特殊功能寄存器
sfr IE           =       0xA8;                  //0000,0000 中断控制寄存器
sbit EA          =       IE^7;
sbit ELVD        =       IE^6;
sbit EADC        =       IE^5;
sbit ES          =       IE^4;
sbit ET1         =       IE^3;
sbit EX1         =       IE^2;
sbit ET0         =       IE^1;
sbit EX0         =       IE^0;
…
//定时器特殊功能寄存器
sfr TCON         =       0x88;                  //0000,0000 T0/T1 控制寄存器
sbit TF1         =       TCON^7;
sbit TR1         =       TCON^6;
sbit TF0         =       TCON^5;
sbit TR0         =       TCON^4;
sbit IE1         =       TCON^3;
sbit IT1         =       TCON^2;
…
//串行口特殊功能寄存器
sfr SCON         =       0x98;                  //0000,0000 串口 1 控制寄存器
sbit SM0         =       SCON^7;
sbit SM1         =       SCON^6;
sbit SM2         =       SCON^5;
…
//ADC 特殊功能寄存器
sfr ADC_CONTR    =       0xBC;                  //0000,0000 A/D 转换控制寄存器
sfr ADC_RES      =       0xBD;                  //0000,0000 A/D 转换结果高 8 位
sfr ADC_RESL     =       0xBE;                  //0000,0000 A/D 转换结果低 2 位
//SPI 特殊功能寄存器
sfr SPSTAT       =       0xCD;                  //00xx,xxxx SPI 状态寄存器
sfr SPCTL        =       0xCE;                  //0000,0100 SPI 控制寄存器
sfr SPDAT        =       0xCF;                  //0000,0000 SPI 数据寄存器
//IAP/ISP 特殊功能寄存器
sfr IAP_DATA     =       0xC2;                  //0000,0000 EEPROM 数据寄存器
sfr IAP_ADDRH    =       0xC3;                  //0000,0000 EEPROM 地址高字节
sfr IAP_ADDRL    =       0xC4;                  //0000,0000 EEPROM 地址低字节
```

```
sfr IAP_CMD         =       0xC5;                   //xxxx,xx00 EEPROM 命令寄存器
sfr IAP_TRIG        =       0xC6;                   //0000,0000 EEPRPM 命令触发寄存器
sfr IAP_CONTR       =       0xC7;                   //0000,x000 EEPROM 控制寄存器
//PCA/PWM 特殊功能寄存器
sfr CCON            =       0xD8;                   //00xx,xx00 PCA 控制寄存器
sbit CF             =       CCON^7;
sbit CR             =       CCON^6;
sbit CCF2           =       CCON^2;
sbit CCF1           =       CCON^1;
sbit CCF0           =       CCON^0;
…
//比较器特殊功能寄存器
sfr CMPCR1          =       0xE6;                   //0000,0000 比较器控制寄存器 1
sfr CMPCR2          =       0xE7;                   //0000,0000 比较器控制寄存器 2
//增强型 PWM 波形发生器特殊功能寄存器
sfr PWMCFG          =       0xf1;                   //x000,0000 PWM 配置寄存器
sfr PWMCR           =       0xf5;                   //0000,0000 PWM 控制寄存器
sfr PWMIF           =       0xf6;                   //x000,0000 PWM 中断标志寄存器
sfr PWMFDCR         =       0xf7;                   //xx00,0000 PWM 外部异常检测控制寄
存器
//如下特殊功能寄存器位于扩展 RAM 区域
//访问这些寄存器,需先将 P_SW2 的 BIT7 设置为 1,才可正常读写
#define PWMC        (*(unsigned int volatile xdata *)0xfff0)
#define PWMCH       (*(unsigned char volatile xdata *)0xfff0)
#define PWMCL       (*(unsigned char volatile xdata *)0xfff1)
…
#endif
```

2) 关键代码设计说明

本程序主要由 init()、delay_ms()、main()三个函数组成。

(1) void init()。函数 void init()主要是对发光二极管电路进行初始化设置,只要将 P0 口和 P2.3 工作模式设置为推挽输出(P0 口的具体工作原理请见后续第 5 章内容),同时将 P2.3 置“1”,使能发光二极管电路即可。其中对 P0 口和 P2 口的工作模式设置,可通过设置对应的 P0 口模式配置寄存器和 P2 口模式配置寄存器来实现。

(2) voiddelay_ms(uint n)。函数 delay_ms(uint n)实现的是延时 n 毫秒的功能,但是延时的时间可能不是很准确。单片机工作时,是在统一的时钟脉冲控制下有序进行的,这个脉冲是由单片机控制器中的时钟电路产生的。时钟电路由振荡器和分频器组成,如图 3-22 所示,振荡器产生基本的振荡信号,然后进行分频得到相应的时钟。振荡电路通常有内部振荡和外部振荡两种方式。STC15F2K60S2 单片机内部集成高精度 R/C 时钟,工作时钟可以使用内部振荡器或者外部晶体振荡器产生的时钟。外部振荡信号通过内部时钟电路,经过分频,得到相应的时钟信号。

振荡周期:晶体振荡器的周期。

状态周期:振荡信号经二分频后形成的时钟脉冲信号,用 S 表示。一个状态周期的两个振荡周期作为两个节拍分别称为节拍 P1 和节拍 P2。P1 有效时,通常完成算术逻辑操作;P2 有效时,一般进行内部寄存器之间的传输。一组节拍 P1 和节拍 P2 也可称为一个时

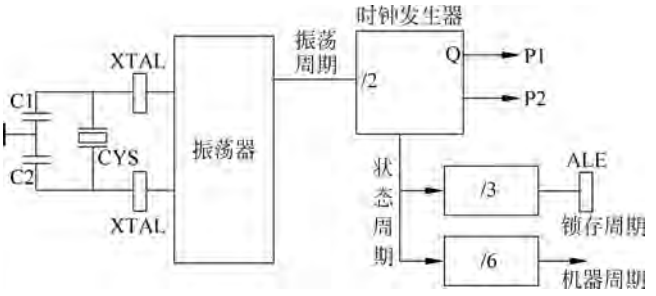

图 3-22　外部振荡模式

钟周期。

机器周期：完成一个基本操作所需的时间称为机器周期。如图 3-23 所示，一个机器周期包含 6 个状态周期，用 S1、S2、…、S6 表示；共 12 个节拍，依次可表示为 S1P1、S1P2、S2P1、S2P2、…、S6P1、S6P2。

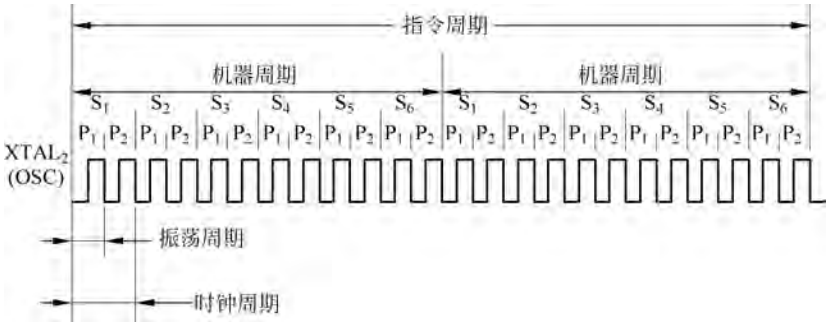

图 3-23　各种周期的相互关系示意

指令周期：CPU 执行一条指令所需要的时间。CPU 执行指令是在时钟脉冲控制下一步一步进行的，由于指令的功能和长短各不相同，因此，指令执行所需的时间也不一样。一个指令周期通常含有 1～4 个机器周期。

例如 MCS-51 单片机外接晶振为 12MHz 时，则单片机的四个周期的具体值为：

振荡周期＝1/12MHz＝1/12μs＝0.0833μs

时钟周期＝1/6μs＝0.167μs

机器周期＝1μs

指令周期＝1～4μs

单片机晶体振荡器 M 的频率可以在 4～48MHz 之间选择，典型值是 11.0592MHz(因为使用这个频率的晶振可以准确地得到 9600b/s 和 19200b/s 的波特率)。根据指令执行的时间，可计算出 1ms 可以相应执行多少条指令，函数中可通过循环执行空指令来达到延时 1ms 的效果。

当然，在此示例程序中也可直接在 STC-ISP 软件中通过“软件延时计算器”功能生成 1 毫秒的延时程序，如图 3-24 所示，然后循环 200 次达到延时 200 毫秒的效果，此外还可以在 STC-ISP 中自动生成指定延时 200 毫秒的延时函数代码。

(3) void main()。每个程序都是从主函数 main()开始执行，在主函数中，我们首先要

调用函数 init()对电路进行初始化，然后对 P0 口赋值 0x01，这时学习板上最右边的 LED 会被点亮，然后 while(1)循环中用“≪”操作符使 P0 端口的数据不断左移，当 P0 口数据变成 0x80 时，表示点亮的 LED 灯已经移动到最左边。此时将 P0 口数据重新赋值为 0x01，再重复上述过程，这样就能看到不断地被循环点亮的 LED 流水灯了。

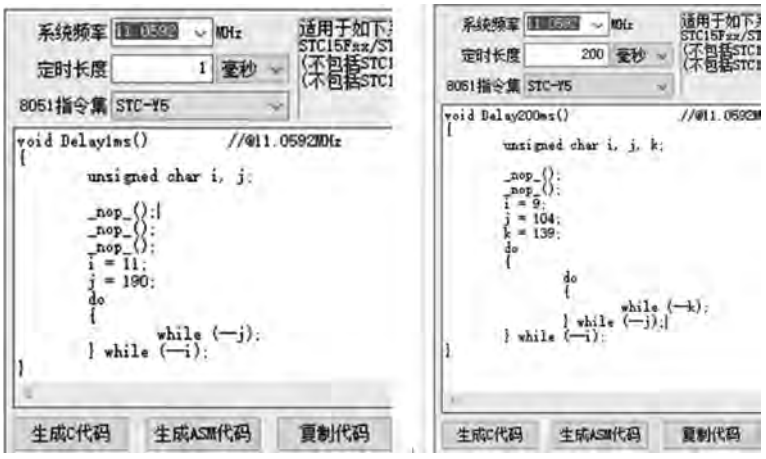

图 3-24 延时计算

3.5 思考题

1. C51 编译器设置的不同优化策略对结果是否有影响？

2. 示例程序中将延时 1ms 的程序循环 200 次达到延时 200ms 的方式是否延时准确？

3. 设计：除了逻辑运算实现法，还可以利用 C51 库自带的移位函数来实现流水灯。打开 Keil 软件安装文件夹，定位到 Keil/C51/Hlp 文件夹下的 c51tools 文件，这是 C51 自带库函数帮助文件，在索引栏找到_crol_函数，双击便会找到函数使用介绍。自己动手试试用库函数实现流水灯案例。