2
神奇的流水灯

经过前面章节的讲述，可能读者对 STM32 处理器还是一头雾水，在本章中将向读者展示如何点亮一个 LED，以此来打开硬件开发和程序下载及调试的大门。就像软件开发中的“Hello World”一样，在硬件开发中一个简单的点亮 LED 背后隐藏着许多知识，有硬件方面的，也有软件方面的，只要能顺利点亮 LED，相信其他的硬件操作都是大同小异的事情。可能控制寄存器更复杂一些，也可能控制信号更多一点，等等，但本质上与点亮 LED 是很相似的。

通过本章的学习，希望读者能够熟练掌握 STM32 开发中编程、编译、烧写及程序调试的基本方法和步骤，能够结合 STM32 处理器对应的数据手册，掌握 GPIO 硬件的基本构成以及编程规范，进一步理解如何通过程序来控制硬件工作。此外，还需要读者初步理解 STM32 固件库的基本用法。

2.1 开发环境简介

本书使用的开发环境为 RealView MDK-ARM（Version 3.80a），即通常所说的 Keil uVersion3，其软件编程及调试界面如图 2-1 所示。

关于开发环境的搭建以及使用，本书不做赘述，对于 STM32 处理器的开发，开发环境设置时有些地方需要注意，但是这些对初学者而言，较为繁琐，所以不建议读者从一个新工程开始逐步的添加源文件，设置编译选项等，本着给读者提供一个平台的原则，笔者建议读者直接打开智造者科技有限公司开发板例程中的工程文件（购买开发板时，该公司会提供本书所用的源代码以及相应的实例工程文件）进行学习即可，读者经过一段时间的学习，基本了解了开发流程后，再尝试着根据具体项目需求去自行搭建开发环境。

按照一般的书籍，接下来需要讲解一些繁琐的知识点，例如，开发环境使用方法、STM32Fxx 系列处理器的寄存器，长篇大论其寄存器的控制方法，等等。

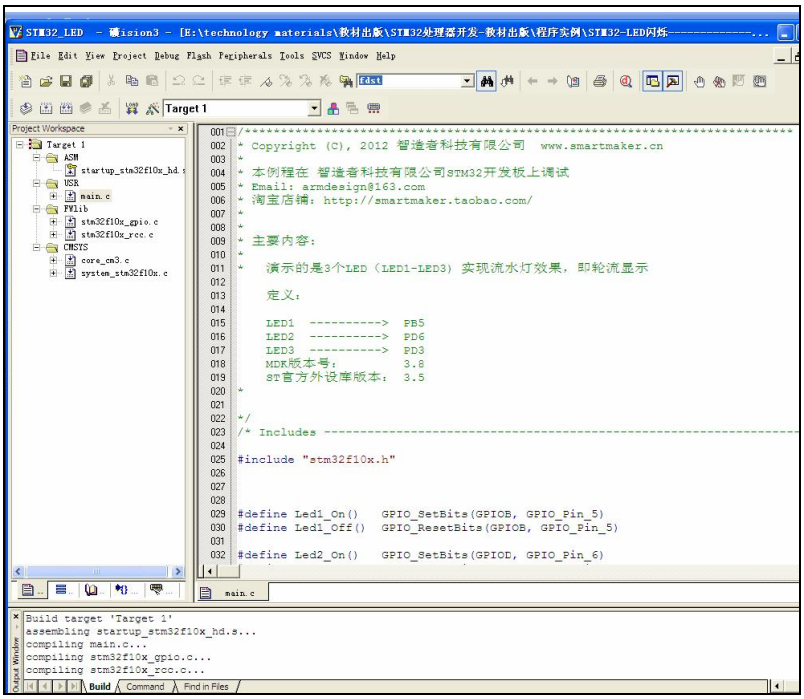

图 2-1　软件开发环境界面

但是，笔者不推荐这种学习方式，毕竟，一款处理器功能繁琐，但有些功能模块可能是用不到的，或者说只有具备一定水平以后，才会使用到，那么，在初学阶段，完全可以不去关注这些功能模块，不去学习这部分寄存器的操作。

文字描述的东西，未免使人觉得厌烦，没有什么比自己点亮流水灯更让人觉得兴奋了。的确，只要点亮了 LED，剩下的工作无非就是不断地学习其他硬件资源的控制方法。

费尽心思读长篇大论，最后不知所云，还不如经过一个简单的实验，看看效果，然后提出疑问，最后查阅资料，逐个解决这些疑问，当最后对实验原理了然于胸时，相信读者已经步入了 STM32 系列处理器开发的大门。

正所谓“人以鱼，不如授之以渔”，因此，接下来，笔者会介绍流水灯的实现方式以及遇到的各种问题及其解决方法。

2.2　流水灯

流水灯对应一款处理器的学习是至关重要的，只要 LED 点亮了，足以证明读者可以对 GPIO 口进行简单的操作了，对后续模块的学习以及程序调试也会起到一定的指示作用，例如当不知道程序是否执行到某个函数时，可以在该函数中点亮一个 LED，如果在程序执行过程中该 LED 点亮，则说明程序可以顺利的执行到该函数中。

例：STM32F103VET6-EV 开发板上共有 3 个 LED，其接口如图 2-2 所示。

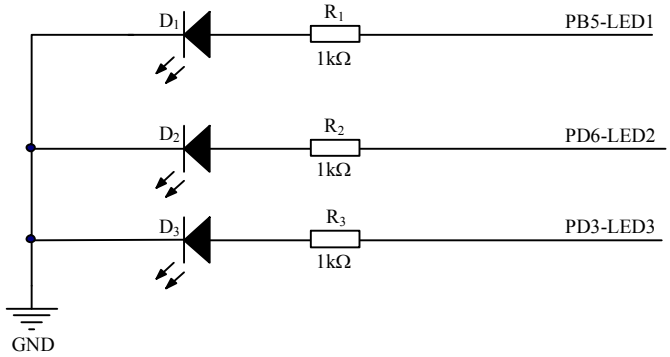

图 2-2　LED 接口电路

现在的问题是 LED 是怎么被点亮的呢？简单些说就是只要 LED 中流过足够的电流，它就发光。现在就是不知道这个足够的电流是多少。

在上述 LED 接口电路中，当 PB5 输出高电平时，电阻和 LED 两端的总电压为 3.3V，那么有初中欧姆定律的知识就可以很容易的判断流过 LED 的电流不会超过 3.3mA，因为 LED 是有内阻的。因此就可以判断，对于这种 LED 发光时的电流是小于 3.3mA 的，一般 1～2mA 就会发光。

现在最关键的问题是如何用程序控制 PB5 输出高电平？

相信学习 STM32 系列处理器的读者，会有一定的单片机开发基础，结合单片机的开发思路，控制 PB5 输出高电平的方法可以概述为：

第 1 步，找到 PB 口的控制寄存器，将其设置为输出模式；

第 2 步，找到 PB 口的数据寄存器，将该寄存器中控制 PB5 的控制位的值设置为 1；

第 3 步，PB5 管脚就会输出高电平，相应的 LED 就会点亮。

上述过程如图 2-3 所示。

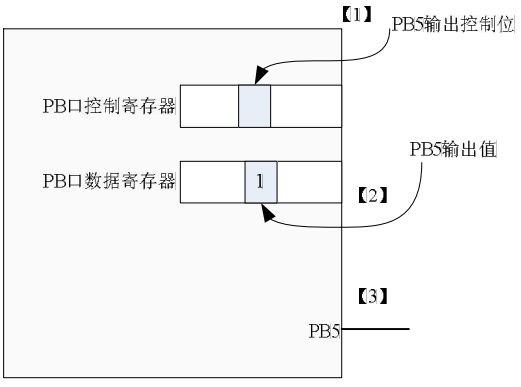

图 2-3　PB5 输出高电平示意图

接下来，就需要查看具体的 PB 口控制寄存器、PB 口数据寄存器的具体地址以及使用方法，但是，不同的处理器操作方法都是相似的，不同的寄存器名字可能有所不同，各个寄存器的地址可能有所不同，但是，这个过程，对于开发人员来讲，是必不可少且较为繁琐的，那么有没有什么改进的措施呢？

回答这个问题之前，先回顾一下上面的例子，上面的例子中之所以繁琐，是因为每次都要查看 PB 口控制寄存器和数据寄存器的地址，然后才能操作这些寄存器。如果提前在内存中分配一块存储区域，并对其进行初始化，然后将初始化的值直接拷贝到 PB 口的对应寄存器中，那么该过程将变得较为简单，如图 2-4 所示。

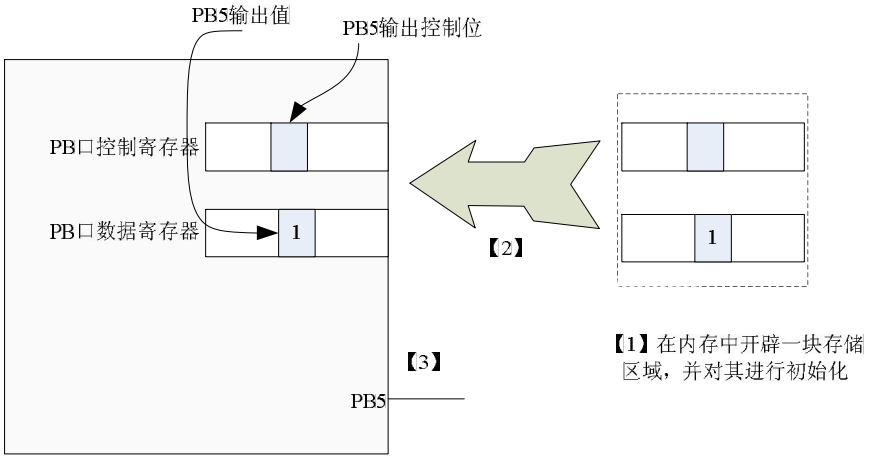

图 2-4　改进型寄存器初始化方法示意图

在上图中，首先在内存中开辟一块存储区域，并对其进行初始化，注意初始化的值应该和初始化 PB 口寄存器的值是相同的，然后，借助某一个函数，直接将这些值复制到 PB 口对应的寄存器中，那么同样可以实现对 PB 口寄存器的初始化。请读者注意观察，此过程中，第 2 步所用到的拷贝函数尤其重要，只要解决了这个拷贝函数，那么，上述问题将得到解决。

前文提到拷贝函数的问题，很高兴的告诉读者，这个函数是由 ST 公司工程师提供给广大开发用户的，这就是“传说中”的固件库（或者说，该函数就是 ST 公司固件库中提供给用户的一个函数库），使用固件库，读者只需要了解固件库提供的函数，然后调用即可，随着使用频率的增多，读者将会渐渐熟悉固件库中提供的函数，因此，无需强制性的记忆。

使用固件库开发，主要是理解其实现思想，该函数的具体实现方式，可以有选择的去学习，这也是固件库设计的初衷，尽最大程度，降低开发人员对底层寄存器级操作的依赖性，从而更好的关注系统级应用程序的实现。所谓“站在巨人的肩上”也就是这个道理吧。

2.2.1　认识固件库

讲解固件库之前，先给读者展示一个具体的例子。在 C 语言开发中，经常会用到 printf()函数

（该函数在 C 语言标准库 stdio.h 中），该函数可以很容易的实现格式化输出，请读者注意：在使用该函数时，是否关注过它的具体实现呢？相信很多读者都不会去关注其具体的实现细节。

类似的道理，使用 ST 公司的固件库时，前期学习阶段，应当把重点放在熟悉固件库提供的函数上，着重分析几个函数的实现即可，先熟悉函数，然后才是关注其具体的实现方式。

下面给出流水灯所使用到的函数。

```
1．GPIO_InitTypeDef  GPIO_Structure;
2．GPIO_Structure.GPIO_Pin = GPIO_Pin_5;
3．GPIO_Structure.GPIO_Mode = GPIO_Mode_Out_PP;
4．GPIO_Init(GPIOB, &GPIO_Structure);
5．GPIO_SetBits(GPIOB, GPIO_Pin_5);
6．GPIO_ResetBits(GPIOB, GPIO_Pin_5);
```

上述 6 行代码基本可以实现点亮一个 LED 了！就是这么简单，下面对其进行分析，着重讲解其实现原理及设计方法。

第 1 行，首先定义了一个结构体变量 GPIO_Structure，其类型为 GPIO_InitTypeDef。为什么定义一个结构体呢？

一般而言，GPIO 口的寄存器都是一组，例如输入输出控制寄存器、数据寄存器、上拉控制寄存器、输出置位复位寄存器等，通常这些寄存器地址都是连续的，所以，请读者记住：GPIO_InitTypeDef 结构体中的各个变量是用来初始化 GPIO 口有关的寄存器，因此，只需要通过 GPIO_Structure 结构体变量，就可以“间接的”访问 GPIO 口的所有寄存器了。

什么？什么？可能有读者会有疑问了，什么叫“间接的”的访问呢？

第 2 行，对结构体变量 GPIO_Structure 中的 GPIO_Pin 赋值为 GPIO_Pin_5，其中，GPIO_Pin_5 是一个宏定义。

第 3 行，对结构体变量 GPIO_Structure 中的 GPIO_Mode 赋值为 GPIO_Mode_Out_PP，其中，GPIO_Mode_Out_PP 是一个宏定义。其实从字面意思看，该变量控制 GPIO 口的输入、输出模式。

第 4 行，调用 GPIO_Init()函数，那么该函数完成什么工作呢？可能有读者会想这个函数是如何实现的呢？还是注意转换思路，这里的问题是，尽快弄清楚这个函数完成什么事情就行了，其他的工作都可以适当延后来完成。

前文讲到，第一步是在内存中开辟一块存储区域，并对其进行初始化，按照什么原则来开辟这块内存区域呢？

考虑生活中的场景，邮寄快递时，一般是快递公司会有相应的包装盒，用户只需要根据邮寄的物品选择相应的包装盒即可，然后快递公司就会将物品运送到目的地。

这里的 GPIO_Structure 结构体类似于一个包装盒，只要将该结构体里面的变量初始化，然后调用 GPIO_Init()函数，即可实现对相应寄存器的初始化工作，此时 GPIO_Init()函数就类似于快递公司了，用户只需要按照该函数的调用方法调用即可，无需关心其具体的实现。这也就是固件库开发的便利之处。从这个层面上说，ST 公司的固件库就像个大的快递公司，根据不同的需求，提供了各种“快递”函数供用户调用，来实现自己的功能，如图 2-5 所示。

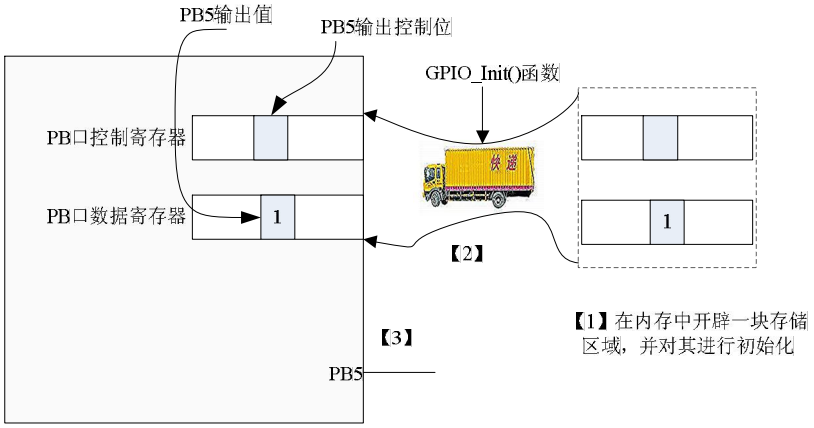

图 2-5　寄存器初始化形象示意图

第 5 行，GPIO_SetBits()函数，实现对某个 IO 口的置位操作，即使其输出高电平。

第 6 行，GPIO_ResetBits()函数，实现对某个 IO 口的复位操作，即使其输出低电平。

经过前面的讲解，可以完成如下例子中的程序。

例：已知 LED2 接在 PD6 引脚，当 PD6 引脚输出高电平时，点亮 LED2，请对比前面的例子，写出具体的程序。

下面分析下其具体的步骤。首先需要找个恰当的“快递盒子”，也就是定义一个结构体变量，其类型为 GPIO_InitTypeDef，然后对其初始化，可以参考上面的例子。

程序代码如下：

```
GPIO_InitTypeDef   GPIO_Structure;
GPIO_Structure.GPIO_Pin = GPIO_Pin_6;
GPIO_Structure.GPIO_Mode = GPIO_Mode_Out_PP;
```

然后，将这个“快递盒子”交给快递公司，即调用 GPIO_Init()函数，此时，GPIO 口的初始化工作结束，GPIO_Init()函数会完成 PD 口的相应寄存器初始化工作。

```
GPIO_Init(GPIOD, &GPIO_Structure);
```

最后，就可以调用 IO 口的置位和复位函数来实现相应 IO 输出高低电平了。程序代码如下：

```
GPIO_SetBits(GPIOD, GPIO_Pin_6);
GPIO_ResetBits(GPIOD, GPIO_Pin_6);
```

还有一个小问题，上述函数都是 ST 公司固件库中的函数，既然是使用了 ST 公司的固件库，那么就需要包含该固件库的头文件，例如：#include "stm32f10x.h"，stm32f10x.h 文件中包含了 STM32F10xx 系列处理器所使用到的大量库函数以及寄存器定义，读者在以后的学习中会经常用到该文件里面的函数，在此提及一下。

尽管笔者尽力使该部分知识简单化，但是，还是有各种各样的问题，例如，GPIO_Init()函数是如何调用的呢？可能还有其他类似的问题，不迷惑才怪！

毕竟这是第一次使用固件库开发嘛，迷惑是正常的。但是，问题多了是好事情，正是这些繁杂

的、琐碎的、难理解的各种问题，才构成了广大读者学习的动力，促使广大读者从一个入门级的菜鸟逐步蜕变为 STM32 处理器应用级大师，就当这些问题是你卓越工程师成长之路上的一块块铺路石吧，问题越多，以后的路越长，当然，发展空间越大。

2.2.2 流水灯程序分析

尽管还有很多问题尚未解决，尽管还有很多固件库函数读者还不熟悉，但是笔者还是建议读者读完下面的代码，不求通篇理解，只求看一下库函数使用的具体例子即可，在后面章节会对该程序段中使用到的函数进行具体的分析。

流水灯工程可以从成都智造者科技有限公司的开发板文档里面找到，打开该工程后，会看到如下代码：

```
1.  #include "stm32f10x.h"

2.  #define Led1_On()       GPIO_SetBits(GPIOB, GPIO_Pin_5)
3.  #define Led1_Off()      GPIO_ResetBits(GPIOB, GPIO_Pin_5)

4.  #define Led2_On()       GPIO_SetBits(GPIOD, GPIO_Pin_6)
5.  #define Led2_Off()      GPIO_ResetBits(GPIOD, GPIO_Pin_6)

6.  #define Led3_On()       GPIO_SetBits(GPIOD, GPIO_Pin_3)
7.  #define Led3_Off()      GPIO_ResetBits(GPIOD, GPIO_Pin_3)
```

第 1 行，既然使用了固件库中的函数，那么就需要包含库函数的头文件了。

第 2～3 行，实现对 PB5 引脚的置位和复位操作，也就是点亮 LED，熄灭 LED。

第 4～7 行，实现对 PD6、PD3 引脚的置位和复位操作。

```
8.  void LED_Init(void);
9.  void Delay(__IO uint32_t nCount);
```

第 8～9 行声明了 2 个函数。

```
void Delay(unsigned int nCount)
{
    for(; nCount != 0; nCount--);
}
```

上述函数是个延时函数，实现一定时间的延时。

```
void LED_Init(void)
{
10.  GPIO_InitTypeDef   GPIO_Structure;

11.  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB , ENABLE);
12.  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOD , ENABLE);

13.  GPIO_Structure.GPIO_Pin = GPIO_Pin_5;
14.  GPIO_Structure.GPIO_Mode = GPIO_Mode_Out_PP;
15.  GPIO_Structure.GPIO_Speed = GPIO_Speed_50MHz;
```

```
16.    GPIO_Init(GPIOB, &GPIO_Structure);

17.    GPIO_Structure.GPIO_Pin = GPIO_Pin_6;
18.    GPIO_Structure.GPIO_Pin = GPIO_Pin_3;
19.    GPIO_Init(GPIOD, &GPIO_Structure);
}
```

在上述函数中，只有 3 条语句没有讲解，其他的语句相信读者有种似曾相识的感觉。

第 10 行，定义了一个 GPIO_InitTypeDef 类型的结构体变量 GPIO_Structure，即通过该语句找到了“快递盒子”。

第 13、14、15 行，对结构体变量 GPIO_Structure 中的值进行了初始化，可以理解为向“盒子”里面填东西，当然填哪些东西是需要注意的，具体能填哪些东西，都在固件库中定义好了，随着后续内容的学习，读者将会渐渐的熟悉这部分内容。

第 16 行，调用 GPIO_Init()函数进行“快递”，此时相当于“发送快递”了，剩下的工作交给该函数来完成，用户无需参与即可。

第 17、19 行，类似于第 13～16 行。

第 16 行，这行程序的意思是初始化 IO 口的翻转速度，通常而言，一个 IO 口可以实现一会儿输出高电平，一会儿输出低电平，但是，这里涉及一个最大切换速度的问题，这个参数就是定义了 IO 口的最大切换速度，在此可以暂时不予关注。

此外，对比第 16、19 两行可以发现，GPIO_Init()函数第一个参数决定了“快递”的目的地址，这类似于快递盒子是一样的，但是接收方可以不同，示意图如图 2-6 所示。

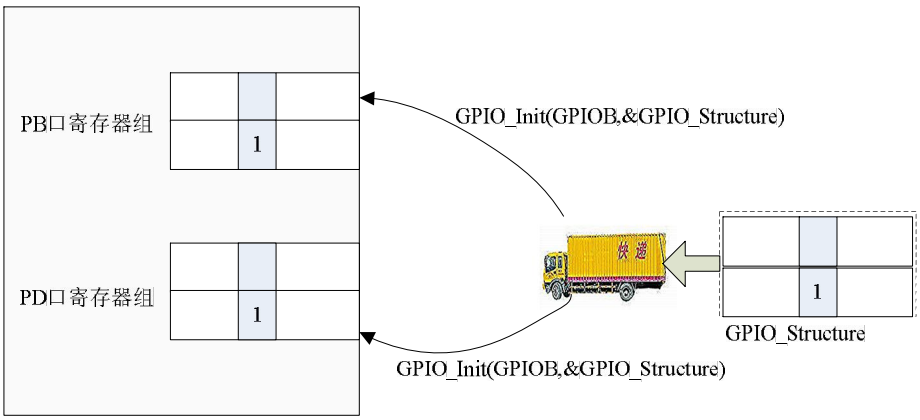

图 2-6　GPIO_Init()函数示意图

```
int main(void)
{
20.    SystemInit();
21.    LED_Init();
while (1)
```

```
{
22.    Led1_On(); Led2_Off(); Led3_Off();
23.    Delay(0x8FFFF);
24.    Led1_Off(); Led2_On(); Led3_Off();
25.
26.    Delay(0x8FFFF);
27.    Led1_Off(); Led2_Off(); Led3_On();
28.    Delay(0x8FFFF);
}
}
```

第 20 行，调用 SystemInit()函数，有读者可能又有疑问了，这是个什么函数呢？这个函数也是固件库中的函数，主要实现对处理器时钟的初始化工作，但是，请读者不需要把注意力集中在这里，一般在主函数中需要首先调用该函数，只需要记住这一点就可以了。

第 21 行，调用 LED_Init()函数，进行初始化。

第 22～28 行，才是真正实现了流水灯效果。

2.2.3 流水灯测试

经过上述分析，基本讲清楚了程序的大概脉络，虽然有个别函数没有深入讲解，但是当前的主要任务是看到流水灯效果，只要效果出来了，然后再慢慢分析程序才会更有动力，一般学习新知识都是这样的规律，看不到效果时感觉特别枯燥无味，一旦看到了具体的效果，再回头看这部分知识时，发现很多知识点已变的渐渐熟悉，熟悉的东西一般是不会感觉难理解的，就像这条语句 GPIO_InitTypeDef GPIO_Structure，相信读者会有似曾相识的感觉。

打开工程以后，单击左边的 Build target 按钮，即可进行工程的编译与连接，如图 2-7 所示，等待编译完成后，即可单击 Download to Flash Memory 进行程序的下载，如图 2-8 所示。

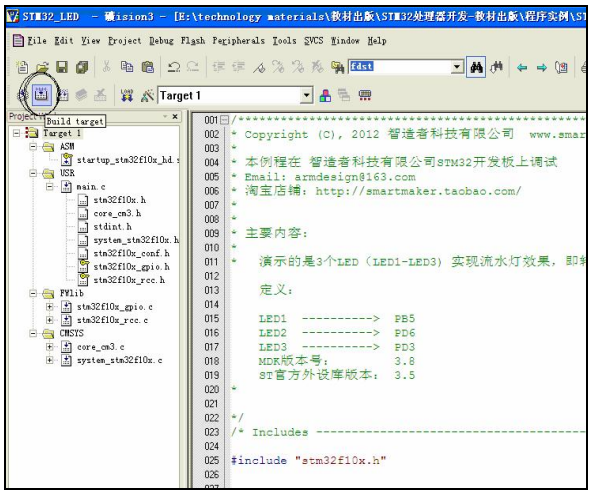

图 2-7 程序编译界面

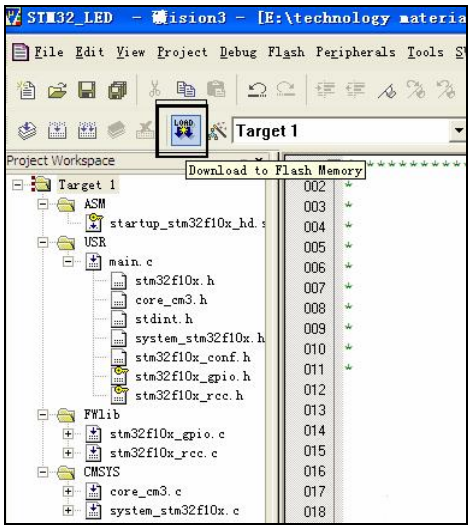

图 2-8　程序下载界面

等待程序下载完成后，即可看到开发板上 3 个 LED 已经轮流点亮了。

读者阅读到这里，也许还是有云里雾里的感觉，但是，要尽最大努力点亮 3 个 LED，那么本章的目的也就达到了，剩下的问题，慢慢来解决。

2.2.4　流水灯引发的思考

经过不懈努力，LED 是点亮了，也许不同的读者会有这样或者那样的疑问，那么下面这些问题又该如何解释呢？

- 固件库中都有哪些函数呢，哪些是初学阶段需要掌握的，哪些是需要重点学习的呢？
- 到底程序从哪里执行的呢，第一条指令在哪里呢？
- 系统上电后是如何一步一步运行到 main 函数的呢？
- 一般编写 C 语言函数都用 main 作为主函数名，这里为什么主函数名用 Main 呢，为什么不用 main 呢，什么情况下才可以用 main 做主函数名呢？

相信读者可能会有更多的问题，不错，正是这些问题困扰着初学者，疯狂的在论坛发帖求助，搜索资料，但是最终却还是有些问题搞不懂……

基于对上述问题的解决，构成了本书的编写原始动力。

给初学者的一点建议：要敢于提出问题，并且要将问题记录下来，只要问题提出来了，总会有解决问题的方法，但是如果不把问题提出来，这个问题可能会一直困扰着你，甚至腐蚀掉你学习 ARM 的积极性。在本书的其他章节中，编者会尽自己的最大努力向读者展现出解决上述问题的思路和方法，必要的时候会涉及到部分经典书籍，同样也会推荐给读者。

当然在这里初学者可能对大段的汇编代码感到无从下手，没有关系，正是这些汇编代码构成了传说中的启动代码，在本书中会有专门的章节讲解启动代码。

在基于 ARM 处理器的嵌入式系统开发中，应用程序大多采用 C 或者 C++等高级语言编写，在运行应用程序之前，需要对系统进行初始化，因此在系统上电后需要有一段引导程序完成对系统资源的初始化，为用户程序建立基本的运行环境。因此，启动代码主要是完成对 ARM 处理器的初始化，使其能够正常工作。为了给读者留下一个印象，下面给读者展示启动代码到底做了哪些事情：

- 建立异常中断向量表；
- 初始化各模式的堆栈；
- 初始化硬件；
- 最后跳转到主应用程序。

看到很多书上都这么写，那么具体是怎么实现的呢？在本书后面章节将会揭开启动代码的神秘面纱。

2.3 本章小结

本章通过一个流水灯实例对 STM32F103VET6 处理器开发进行了具体讲解，可能有的地方显得比较繁琐，但考虑到初学者面临的各种问题，还是将具体细节展现出来了。当然，在上面这些例子中，唯一的缺点可能是程序代码功能太单一，不过本章主要是想给读者展现出 STM32 处理器开发的基本方法，以及固件库的概念及使用方法，着重讲解了 ST 公司固件库提供的函数 GPIO_Init()所承担的“快递公司”的角色，至于其他的问题在目前情况下都是次要的。

2.4 附录 1—流水灯源程序

```
#include "stm32f10x.h"

#define Led1_On()       GPIO_SetBits(GPIOB, GPIO_Pin_5)
#define Led1_Off()      GPIO_ResetBits(GPIOB, GPIO_Pin_5)

#define Led2_On()       GPIO_SetBits(GPIOD, GPIO_Pin_6)
#define Led2_Off()      GPIO_ResetBits(GPIOD, GPIO_Pin_6)

#define Led3_On()       GPIO_SetBits(GPIOD, GPIO_Pin_3)
#define Led3_Off()      GPIO_ResetBits(GPIOD, GPIO_Pin_3)


void LED_Init(void);
void Delay(__IO uint32_t nCount);


void LED_Init(void)
{
```

```
    GPIO_InitTypeDef GPIO_Structure;

    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB , ENABLE);
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOD , ENABLE);
    GPIO_Structure.GPIO_Pin = GPIO_Pin_5;
    GPIO_Structure.GPIO_Mode = GPIO_Mode_Out_PP;
    GPIO_Structure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOB, &GPIO_Structure);

    GPIO_Structure.GPIO_Pin = GPIO_Pin_6|GPIO_Pin_3;
    GPIO_Init(GPIOD, &GPIO_Structure);
}

int main(void)
{
    SystemInit();                    //系统时钟配置
    LED_Init();                      //LED 控制配置
    while (1)
    {
        Led1_On(); Led2_Off(); Led3_Off();
        Delay(0x8FFFF);
        Led1_Off(); Led2_On(); Led3_Off();

        Delay(0x8FFFF);
        Led1_Off(); Led2_Off(); Led3_On();
        Delay(0x8FFFF);
    }
}

void Delay(unsigned int nCount)
{
    for(; nCount != 0; nCount--);
}
```


2.5 附录 2—开发环境搭建

使用 Keil 进行 ARM 程序开发时需要遵循一定的操作步骤，首先需要建立工程，然后向工程中添加源文件，最后编译源程序生成可执行文件，本节将讲述上述内容。

建立工程时，可以选择 Project 按钮，在弹出的下拉菜单中选择 New uVision Project 菜单项，如图 2-9 所示。

接下来输入工程名然后单击保存即可。此时会出现选择处理器型号窗口如图 2-10 所示。

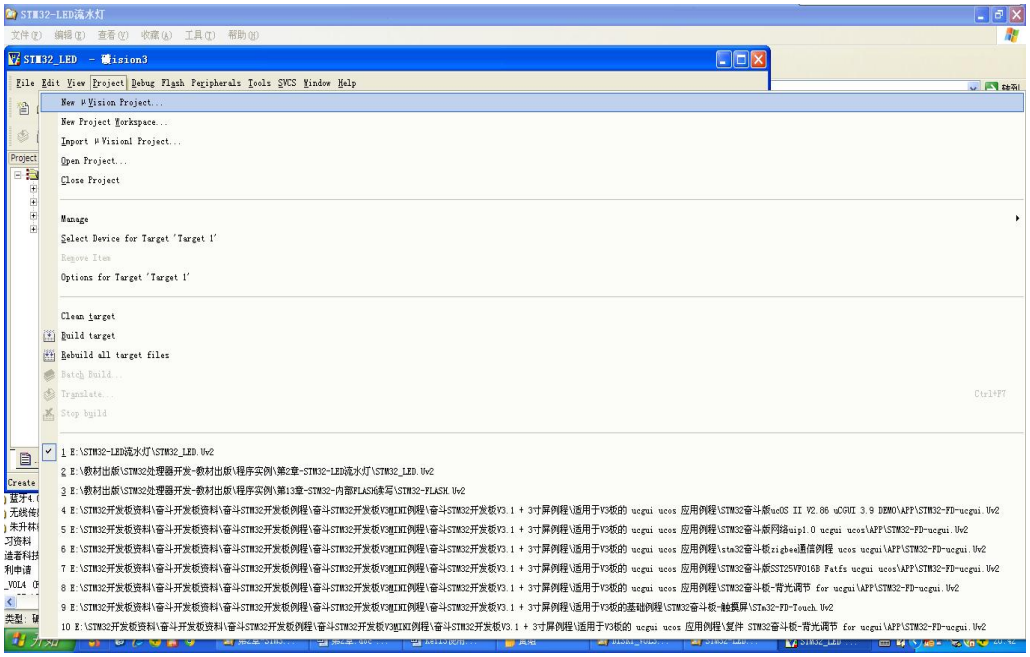

图 2-9　新建工程

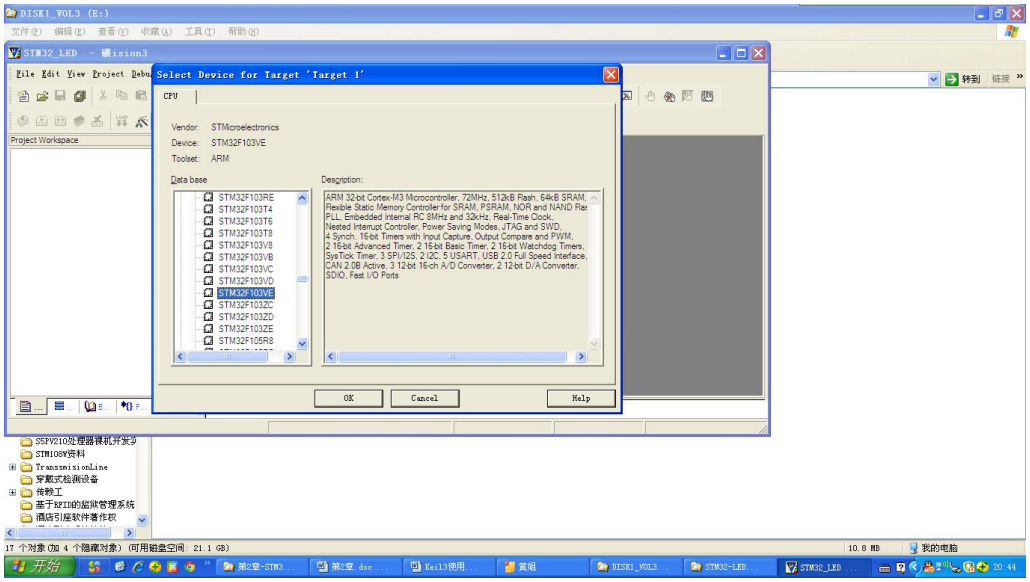

图 2-10　选择处理器型号窗口

选择 STM32F103VE 单击 OK 按钮，弹出是否添加启动代码的窗口，如图 2-11 所示。单击“否”按钮即可（ST 公司以及提供了官方的启动代码，建立工程以后可以添加 ST 公司提供的启动代码）。

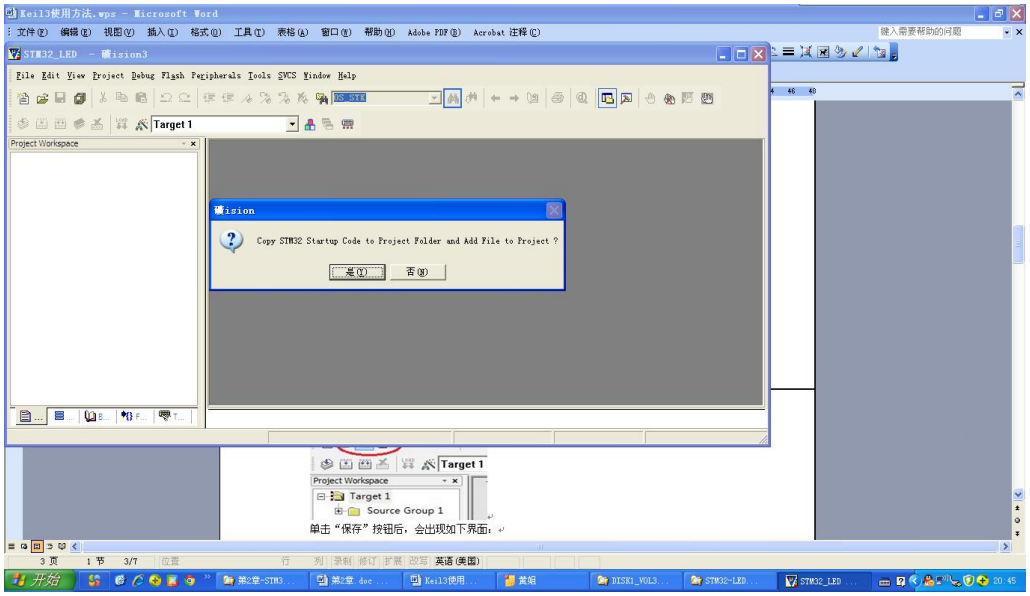

图 2-11　添加启动代码选择按钮

工程建立以后，可以新建源程序文件并将其添加到工程中。新建源程序文件的操作如图 2-12 所示。

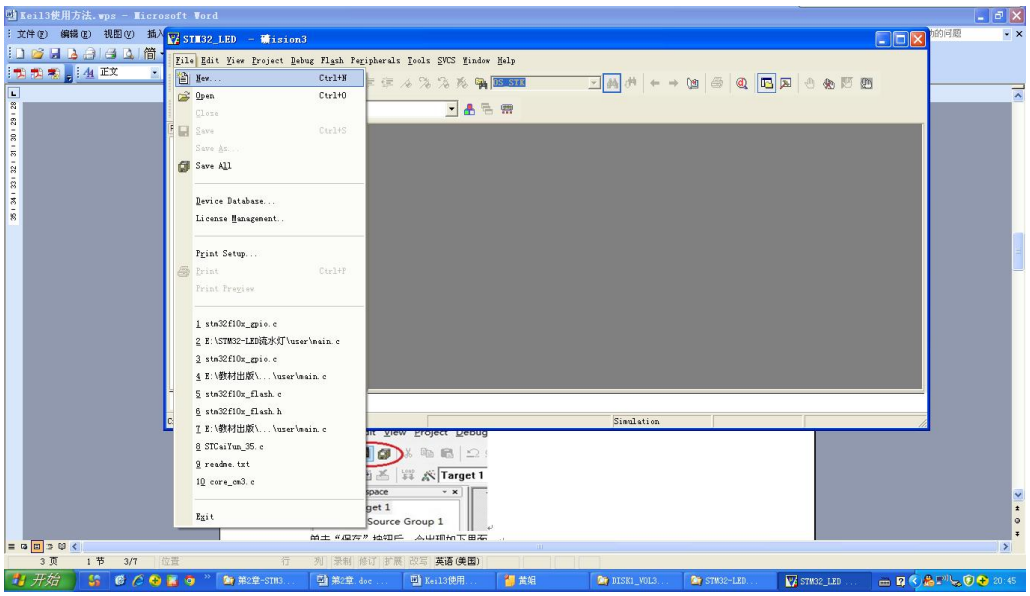

图 2-12　添加源文件

建立了文件后，单击“保存”按钮。

最后需要将源程序文件加入工程中，选中 Source group1，然后单击右键，此时将会出现下拉列表框如图 2-13 所示。

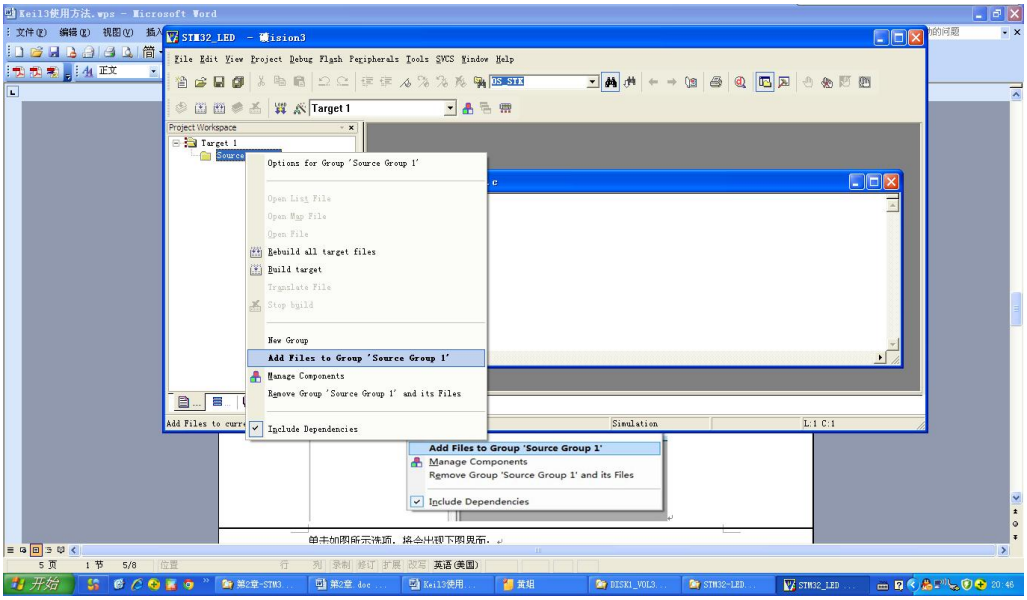

图 2-13　将源文件加入工程

选择 Add Files to Group ‘Source Group 1’，然后选择要添加的源文件即可。