第 5 章

LED 流水灯与 SysTick 定时器

本章要点

- 基于库函数的开发方法
- GPIO 输出相关库函数
- 库函数版 LED 流水灯程序设计
- SysTick 定时器概述、寄存器和应用方法
- SysTick 定时器毫秒和微秒延时函数
- SysTick 函数嵌入到 LED 流水灯项目中实现精确延时

第 4 章章介绍了 STM32 的 GPIO 并给出了通过操作 GPIO 寄存器的 LED 灯闪烁程序，使大家对 STM32 程序设计有一定的了解，本章将首先介绍 STM32 的库函数开发方式，并详细给出库函数版的 LED 流水灯程序设计方法。Cortex-M3 处理器内部包含了一个简单的定时器，因为所有的 Cortex-M3 芯片都带有这个定时器，软件在不同 Cortex-M3 器件间的移植工作得以简化。本章将进一步介绍利用 SysTick 定时器编写延时函数方法，并将其嵌入到流水灯程序当中，为其提供精确的 1s 延时程序。

LED 流水灯

5.1 库函数开发方法

从第 4 章的分析可以看出，对寄存器操作虽然简单、直接、高效，但是需要对 STM32 硬件有非常好的理解，并且要记住所有相关寄存器的名称、位定义以及操作方式，这对于绝大部分初学者来说相当不容易。另外，基于寄存器编写出来的程序可读性比较差，不便于系统维护和程序员之间的交流，所以对于初学者和普通程序开发人员，本书推荐另外一种程序开发方法，即基于库函数的开发方法。

库函数对于程序设计人员并不陌生，我们在学习 C 语言时，经常会使用到 stdio.h 库所提供的标准输入输出库函数 scanf()和 printf()。类似地，为了简化编程开发难度，意法半导体公司向用户提供了 STM32 标准库函数，又称为 STM32 固件库，它包括所有标准外设

的驱动程序，可以极大地方便用户使用 STM32 微控制器的片上外设。

STM32 固件库是由 ST 公司针对 STM32 微控制器为用户开发提供的 API（APPlication Program Interface，应用程序接口）。实际上，STM32 固件库是位于寄存器和用户之间的预定义代码，它由程序、数据结构和各种宏定义组成。它向下实现与寄存器的直接相关操作，向上为用户提供配置寄存器的标准接口。通过使用固件库的标准函数，无须深入掌握底层硬件细节，开发者就可以轻松应用每一个外设。就像学习 C 语言编程使用库函数 printf()和 scanf()时，只是学习它们的调用方法，并没有去研究它们的源代码实现一样。

显而易见，相比于寄存器开发方式，基于库函数的开发方式具有容易学习、便于阅读和维护成本低等优点，降低了开发难度和门槛，缩短了开发周期。标准库函数由于考虑到软件通用性，需要面面俱到，为提高软件的鲁棒性，需要对软件参数进行检测，这些都会使得库函数方式生成的代码较直接配置寄存器方式要大一些。但是由于 STM32 拥有充足的硬件资源，权衡利弊，绝大多数情况下，宁愿牺牲一点资源而选择库函数开发。通常，只有在对代码运行时间要求极其苛刻的场合，例如，频繁调用的异常服务程序，才会选择寄存器方式编写程序。随着意法半导体官方固件库的不断丰富和完善，库函数方式目前已经成为 STM32 嵌入式开发的首选。

5.2　GPIO 输出库函数

由 LED 流水灯控制电路可知，需要配置 PC 口为输出方式，并设置 PC0～PC7 的电平状态，以点亮或是熄灭 LED 指示灯。现将涉及的库函数一一详解如下，因为这是本书第一次介绍库函数，所以讲解要详尽一些。

5.2.1　函数 RCC_APB2PeriphClockCmd

表 5-1 描述了函数 RCC_APB2PeriphClockCmd。

表 5-1　函数 RCC_APB2PeriphClockCmd

函数名	RCC_APB2PeriphClockCmd
函数原型	void RCC_APB2PeriphClockCmd(u32 RCC_APB2Periph，FunctionalState NewState)
功能描述	使能或者失能 APB2 外设时钟
输入参数 1	RCC_APB2Periph：门控 APB2 外设时钟
	参阅 Section：RCC_APB2Periph，查阅更多该参数允许取值范围
输入参数 2	New State：指定外设时钟的新状态
	这个参数可以取：ENABLE 或者 DISABLE
输出参数	无
返回值	无
先决条件	无
被调用函数	无

RCC_APB2Periph 参数：

该参数被门控的 APB2 外设时钟，可以取下表的一个或者多个取值的组合作为该参数的值。

表 5-2　RCC_AHB2Periph 值

RCC_AHB2Periph	描　　述
RCC_APB2Periph_AFIO	功能复用 I/O 时钟
RCC_APB2Periph_GPIOA	GPIOA 时钟
RCC_APB2Periph_GPIOB	GPIOB 时钟
RCC_APB2Periph_GPIOC	GPIOC 时钟
RCC_APB2Periph_GPIOD	GPIOD 时钟
RCC_APB2Periph_GPIOE	GPIOE 时钟
RCC_APB2Periph_ADC1	ADC1 时钟
RCC_APB2Periph_ADC2	ADC2 时钟
RCC_APB2Periph_TIM1	TIM1 时钟
RCC_APB2Periph_SPI1	SPI1 时钟
RCC_APB2Periph_USART1	USART1 时钟
RCC_APB2Periph_ALL	全部 APB2 外设时钟

例如：

```
/* Enable GPIOA, GPIOB and SPI1 clocks */
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA | RCC_APB2Periph_GPIOB |
RCC_APB2Periph_SPI1, ENABLE);
```

此函数用于打开挂接在 APB2 总线上的外设时钟，要打开哪一个设备只要将其名称作为参数填入到函数中即可，如果是要打开多个设备的时钟，多个设备的名称用“|”号连接。

例如，对于 LED 流水灯控制来说，因为 LED 的阴极由 STM32 单片机的 GPIOC 口控制的，所以需要调用该函数打开 GPIO 时钟，其语句为：

```
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC, ENABLE);
```

5.2.2　函数 GPIO_Init

表 5-3 描述了函数 GPIO_Init。

表 5-3　函数 GPIO_Init

函数名	GPIO_Init
函数原型	void GPIO_Init(GPIO_TypeDef * GPIOx, GPIO_InitTypeDef * GPIO_Init Struct)
功能描述	根据 GPIO_InitStruct 中指定的参数初始化外设 GPIOx 寄存器
输入参数 1	GPIOx：x 可以是 A、B、C、D 或者 E，用来选择 GPIO 外设

续表

输入参数 2	GPIO_InitStruct：指向结构 GPIO_InitTypeDef 的指针，包含了外设 GPIO 的配置信息 参阅 Section：GPIO_InitTypeDef，查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

GPIO_InitTypeDef structure

GPIO_InitTypeDef 定义于文件“stm32f10x_gpio.h”：

```
typedef struct
{
u16  GPIO_Pin;
GPIOSpeed_TypeDef  GPIO_Speed;
GPIOMode_TypeDef  GPIO_Mode;
}  GPIO_InitTypeDef;
```

GPIO_Pin

该参数选择待设置的 GPIO 引脚，使用操作符“|”可以一次选中多个引脚。可以使用表 5-4 中的任意组合。

表 5-4　GPIO_Pin 值

GPIO_Pin	描　述	GPIO_Pin	描　述
GPIO_Pin_None	无引脚被选中	GPIO_Pin_8	选中引脚 8
GPIO_Pin_0	选中引脚 0	GPIO_Pin_9	选中引脚 9
GPIO_Pin_1	选中引脚 1	GPIO_Pin_10	选中引脚 10
GPIO_Pin_2	选中引脚 2	GPIO_Pin_11	选中引脚 11
GPIO_Pin_3	选中引脚 3	GPIO_Pin_12	选中引脚 12
GPIO_Pin_4	选中引脚 4	GPIO_Pin_13	选中引脚 13
GPIO_Pin_5	选中引脚 5	GPIO_Pin_14	选中引脚 14
GPIO_Pin_6	选中引脚 6	GPIO_Pin_15	选中引脚 15
GPIO_Pin_7	选中引脚 7	GPIO_Pin_All	选中全部引脚

GPIO_Speed

GPIO_Speed 用以设置选中引脚的速率。表 5-5 给出了该参数可取的值。

表 5-5　GPIO_Speed 值

GPIO_Speed	描　述	GPIO_Speed	描　述
GPIO_Speed_10MHz	最高输出速率 10MHz	GPIO_Speed_50MHz	最高输出速率 50MHz
GPIO_Speed_2MHz	最高输出速率 2MHz		

GPIO_Mode

GPIO_Mode 用以设置选中引脚的工作状态。表 5-6 给出了该参数可取的值。

表 5-6 GPIO_Mode 值

GPIO_Speed	描 述	GPIO_Speed	描 述
GPIO_Mode_AIN	模拟输入	GPIO_Mode_Out_OD	开漏输出
GPIO_Mode_IN_FLOATING	浮空输入	GPIO_Mode_Out_PP	推挽输出
GPIO_Mode_IPD	下拉输入	GPIO_Mode_AF_OD	复用开漏输出
GPIO_Mode_IPU	上拉输入	GPIO_Mode_AF_PP	复用推挽输出

例如：

```
/* Configure all the GPIOA in Input Floating mode */
GPIO_InitTypeDef GPIO_InitStructure;
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_All;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_10MHz;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
GPIO_Init(GPIOA, &GPIO_InitStructure);
```

对于 LED 流水灯控制，因为既需要输出高电平，又需要输出低电平，所以需要将 GPIOC 配置为推挽输出模式；而对输出速度并没有特殊要求，配置成 2MHz 即可。引脚可以选全部也可选 GPIO_Pin_0～GPIO_Pin_7。所以其参考初始化程序如下：

```
/* Configure all the GPIOC in Output Push-Pull mode */
GPIO_InitTypeDef GPIO_InitStructure;
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_All;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
GPIO_Init(GPIOC, &GPIO_InitStructure);
```

5.2.3 函数 GPIO_Write

表 5-7 描述了 GPIO_Write。

表 5-7 GPIO_Write

函数名	GPIO_Write
函数原型	void GPIO_Write(GPIO_TypeDef * GPIOx, u16 PortVal)
功能描述	向指定 GPIO 数据端口写入数据
输入参数 1	GPIOx：x 可以是 A、B、C、D 或者 E，用来选择 GPIO 外设
输入参数 2	PortVal：待写入端口数据寄存器的值
输出参数	无

续表

返回值	无
先决条件	无
被调用函数	无

例如：

```
/* Write data to GPIOA data port */
GPIO_Write(GPIOA, 0x1101);
```

在 LED 流水灯控制中，由于采用共阳接法，GPIOC 的 I/O 口输出低电平点亮，如果只需要点亮 L1，只需要 PC0 输出低电平，其余 I/O 口输出高电平，对应 GPIO 输出数据为 0xFE，其对应的控制语句为：

```
GPIO_Write(GPIOC, 0xFE);
```

5.2.4　函数 GPIO_SetBits

表 5-8 描述了 GPIO_SetBits。

表 5-8　GPIO_SetBits

函数名	GPIO_SetBits
函数原型	void GPIO_SetBits(GPIO_Type Def * GPIOx, u16 GPIO_Pin)
功能描述	设置指定的数据端口位
输入参数 1	GPIOx：x 可以是 A、B、C、D 或者 E，用来选择 GPIO 外设
输入参数 2	GPIO_Pin：待设置的端口位 该参数可以取 GPIO_Pin_x(x 可以是 0～15)的任意组合 参阅 Section：GPIO_Pin，查阅更多该参数允许取值范围
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例如：

```
/* Set the GPIOA port pin 10 and pin 15 */
GPIO_Set Bits(GPIOA, GPIO_Pin_10 | GPIO_Pin_15);
```

5.2.5　函数 GPIO_ResetBits

表 5-9 描述了 GPIO_ResetBits。

表 5-9 GPIO_ResetBits

函数名	GPIO_ResetBits
函数原型	void GPIO_ResetBits(GPIO_TypeDef * GPIOx, u16 GPIO_Pin)
功能描述	清除指定的数据端口位
输入参数 1	GPIOx：x 可以是 A、B、C、D 或者 E，来选择 GPIO 外设
输入参数 2	GPIO_Pin：待清除的端口位该参数可以取 GPIO_Pin_x(x 可以是 0～15)的任意组合
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例如：

```
/* Clears the GPIOA port pin 10 and pin 15 */
GPIO_ResetBits(GPIOA, GPIO_Pin_10 | GPIO_Pin_15);
```

5.2.6 函数 GPIO_WriteBit

表 5-10 描述了 GPIO_WriteBit。

表 5-10 GPIO_WriteBit

函数名	GPIO_WriteBit
函数原型	void GPIO_WriteBit(GPIO_TypeDef * GPIOx, u16 GPIO_Pin, BitAction BitVal)
功能描述	设置或者清除指定的数据端口位
输入参数 1	GPIOx：x 可以是 A、B、C、D 或者 E，来选择 GPIO 外设
输入参数 2	GPIO_Pin：待设置或者清除指的端口位 该参数可以取 GPIO_Pin_x(x 可以是 0～15)的任意组合 参阅 Section：GPIO_Pin，查阅更多该参数允许取值范围
输入参数 3	BitVal：该参数指定了待写入的值，该参数必须取枚举 BitAction 的其中一个值，Bit_RESET：清除数据端口位 Bit_SET：设置数据端口位
输出参数	无
返回值	无
先决条件	无
被调用函数	无

例如：

```
/* Set the GPIOA port pin 15 */
GPIO_WriteBit(GPIOA, GPIO_Pin_15, Bit_SET);
```

5.3　LED 流水灯控制

已知开发板 LED 流水灯电路原理图如图 5-1 所示。由图可知，如需实现 LED 流水灯控制只需要依次点亮 L1～L8，即需依次设置 PC0～PC8 为低电平即可，对应 GPIOC 端口写入数据分别为 0xFE、0xFD、0xFB、0XF7、0xEF、0xDF、0xBF、0x7F。

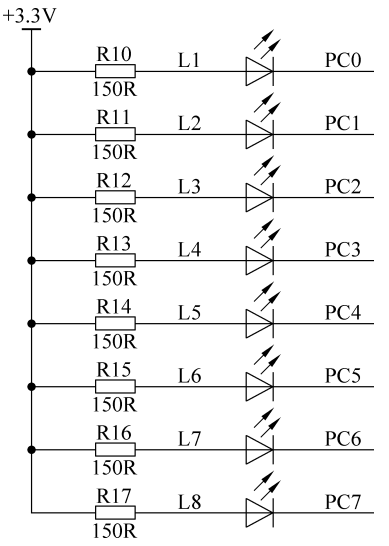

图 5-1　LED 流水灯电路原理图

项目具体实施步骤为：

第一步：复制第 3 章创建的工程模板文件夹到桌面（其他文件夹路径也可以，只是桌面操作起来更方便），并将文件夹改名为“2 LED 流水灯”（其他名称完全可以，只是命名需要遵循一定原则，以便于项目积累）。

第二步：将原工程模板编译一下，直到没有错误和警告为止。新建两个文件，将其改名为 LED. C 和 LED. H 并保存到工程模板下的 APP 文件中。并将 LED. C 文件添加到 APP 项目组下，并再次编译一下。

第三步：在 LED. C 文件中输入如下源程序，在程序中首先包含 LED. H 头文件，然后创建三个函数，分别是延时函数 void delay(u32 i)，LED 流水灯初始化函数 void LEDInit()，以及流水灯显示函数 void LEDdisplay()。

```
#include "LED.h"
/*****************************************************************************
* 函 数 名          : delay
* 函数功能          : 延时函数,delay(6000000)延时约 1s
* 输    入          : i
* 输    出          : 无
*****************************************************************************/
void delay(u32 i)
{
    while(i--);
}

/*****************************************************************************
* 函 数 名          : LEDInit
* 函数功能          : LED 初始化函数
* 输    入          : 无
* 输    出          : 无
*****************************************************************************/
```

```
void LEDInit()
{
    GPIO_InitTypeDef GPIO_InitStructure;
    SystemInit();
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC , ENABLE);

    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_All;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP ;
    GPIO_Init(GPIOC, &GPIO_InitStructure);
}

/*********************************************************************
* 函 数 名          : LEDdisplay
* 函数功能          : LED 显示函数  LED 闪烁
* 输    入          : 无
* 输    出          : 无
*********************************************************************/
void LEDdisplay()
{
    while(1)
    {
        GPIO_Write(GPIOC, 0xfe);
        delay(6000000);                    //延时约为 1s
        GPIO_Write(GPIOC, 0xfd);
        delay(6000000);                    //延时约为 1s
        GPIO_Write(GPIOC, 0xfb);
        delay(6000000);                    //延时约为 1s
        GPIO_Write(GPIOC, 0xf7);
        delay(6000000);                    //延时约为 1s
        GPIO_Write(GPIOC, 0xef);
        delay(6000000);                    //延时约为 1s
        GPIO_Write(GPIOC, 0xdf);
        delay(6000000);                    //延时约为 1s
        GPIO_Write(GPIOC, 0xbf);
        delay(6000000);                    //延时约为 1s
        GPIO_Write(GPIOC, 0x7f);
        delay(6000000);                    //延时约为 1s
    }
}
```

第四步：在 LED. H 文件中输入如下源程序，其中条件编译和包含 STM32 头文件内容可以参考第 2 章工程模板创建时编写 public. h 的编写方法，其实一般情况下是直接复制 public. h 文件内容到 LED. H 文件中来，然后再进行修改即可，此处需要将我们在 LED. C 创建的函数声明加进来。

```
#ifndef _public_H
#define _public_H
#include "stm32f10x.h"
void delay(u32 i);
void LEDInit(void);
void LEDdisplay(void);
#endif
```

第五步：在 public.h 文件的中间部分添加#include "LED.H"语句，即包含 LED.H 头文件，此处要记住，任何时候程序中需要使用某一源文件中函数，必须先包含其头文件，否则编译是不能通过的。Public.h 文件的源代码如下。

```
#ifndef _public_H
#define _public_H
#include "stm32f10x.h"
#include "LED.H"
#endif
```

第六步：在 main.c 文件中输入如下源程序，其中 main 函数就是两条语句，分别调用 LEDInit()函数对 GPIO 引脚进行初始化，调用 LEDdisplay()函数进行流水灯显示，由于在 LEDdisplay()函数已经包含无限循环结构，此处不需要重复构建。

```
#include "public.h"
int main()
{
    LEDInit();
    LEDdisplay();
}
```

第七步：编译工程，如没有错误，则会在 output 文件夹中生成“工程模板.hex”文件，如有错误则修改源程序直至没有错误为止。

第八步：将生成的目标文件通过 ISP 软件下载到开发板微控制器的 Flash 存储器当中，复位运行，检查实验效果。

5.4　SysTick 定时器

5.4　SysTick 定时器

5.4.1　SysTick 定时器概述

在以前，大多操作系统需要一个硬件定时器来产生操作系统需要的滴答中断，作为整个系统的时基。例如，为多个任务许以不同数目的时间片，确保没有一个任务能霸占系统；或者把每个定时器周期的某个时间范围赐予特定的任务等，还有操作系统提供的各种定时功

能，都与这个滴答定时器有关。因此，需要一个定时器来产生周期性的中断，而且最好还让用户程序不能随意访问它的寄存器，以维持操作系统“心跳”的节律。

Cortex-M3 处理器内部包含了一个简单的定时器。因为所有的 Cortex-M3 处理器都带有这个定时器，软件在不同 Cortex-M3 处理器间的移植工作得以化简。该定时器的时钟源可以是内部时钟（FCLK，Cortex-M3 处理器上的自由运行时钟），或者是外部时钟（Cortex-M3 处理器上的 STCLK 信号）。不过，STCLK 的具体来源由芯片设计者决定，因此不同产品之间的时钟频率可能会大不相同，需要检视芯片的器件手册来决定选择什么作为时钟源。

SysTick 定时器能产生中断，Cortex-M3 处理器为它专门开出一个异常类型，并且在向量表中有它的一席之地。它使操作系统和其他系统软件在 Cortex-M3 处理器间的移植变得简单多了，因为在所有 CM3 产品间对其处理都是相同的。

SysTick 定时器除了能服务于操作系统之外，还能用于其他目的：如作为一个闹钟，用于测量时间等。需要注意的是，当处理器在调试期间被喊停（halt）时，则 SysTick 定时器亦将暂停运作。

5.4.2 SysTick 定时器寄存器

有 4 个寄存器控制 SysTick 定时器，如表 5-11～表 5-14 所示。

表 5-11 SysTick 控制及状态寄存器 STK_CSR（0xE000_E010）

位段	名　称	类型	复位值	描　述
16	COUNTFLAG	R	0	如果在上次读取本寄存器后，Sys Tick 已经数到了 0，则该位为 1。如果读取该位，该位将自动清零
2	CLKSOURCE	R/W	0	0＝外部时钟源（STCLK） 1＝内核时钟（FCLK）
1	TICKINT	R/W	0	1＝Sys Tick 倒数到 0 时产生 Sys Tick 异常请求 0＝数到 0 时无动作
0	ENABLE	R/W	0	Sys Tick 定时器的使能位

表 5-12 SysTick 重装载数值寄存器 STK_LOAD（0xE000_E014）

位段	名　称	类型	复位值	描　述
23:0	RELOAD	R/W	0	当倒数至零时，将被重装载的值

表 5-13 SysTick 当前数值寄存器 STK_VAL（0xE000_E018）

位段	名　称	类型	复位值	描　述
23:0	CURRENT	R/Wc	0	读取时返回当前倒计数的值，写它则使之清零，同时还会清除在 Sys Tick 控制及状态寄存器中的 COUNTFLAG 标志

表 5-14　SysTick 校准数值寄存器 STK_CALRB(0xE000_E01C)

位段	名　称	类型	复位值	描　述
31	NOREF	R	—	1=没有外部参考时钟(STCLK 不可用) 0=外部参考时钟可用
30	SKEW	R	—	1=校准值不是准确的 10ms 0=校准值是准确的 10ms
23:0	TENMS	R/W	0	10ms 的时间内倒计数的格数。芯片设计者应该通过 Cortex-M3 的输入信号提供该数值。若该值读回零,则表示无法使用校准功能

5.4.3　SysTick 定时器应用

上一节 LED 流水灯控制程序中延时程序是通过软件延时的方法来实现的,这个时间很不精确,只能大概估计。根据上述分析,本节利用 SysTick 定时器可以分别编写 delay_us( )延时函数和 delay_ms( )延时函数,供流水灯控制程序调用,实现精确延时。本项目具体操作步骤如下:

第一步:复制上一节项目文件夹到桌面,并将文件夹改名为"3 SysTick 定时器"。

第二步:将原工程模板编译一下,直到没有错误和警告为止。新建两个文件,将其改名为 systick.c 和 systick.h 并保存到工程模板下的 APP 文件夹中。并将 systick.c 文件添加到 APP 项目组下,并再次编译一下。

第三步:在 systick.c 文件中输入如下源程序,在程序中首先包含 systick.h 头文件,然后创建两个延时函数,一个是微秒延时函数:delay_us(u32 i),另一个是毫秒延时函数:delay_ms(u32 i)。

```
#include "systick.h"
/*************************************************************************
* 函 数 名          : delay_us
* 函数功能          : 延时函数,延时 μs
* 输    入          : i
* 输    出          : 无
*************************************************************************/
void delay_us(u32 i)
{
    u32 temp;
    SysTick->LOAD = 9*i;                    //设置重装数值, 72MHz 时
    SysTick->CTRL = 0X01;                   //使能,减到零是无动作,采用外部时钟源
    SysTick->VAL = 0;                       //清零计数器
    do
    {
        temp = SysTick->CTRL;               //读取当前倒计数值
```

```
    }
    while((temp&0x01)&&(!(temp&(1 << 16))));        //等待时间到达
    SysTick -> CTRL = 0;                            //关闭计数器
    SysTick -> VAL = 0;                             //清空计数器
}
/ ********************************************************************
 * 函 数 名          : delay_ms
 * 函数功能          : 延时函数,延时 ms
 * 输    入          : i
 * 输    出          : 无
 ********************************************************************/
void delay_ms(u32 i)
{
    u32 temp;
    SysTick -> LOAD = 9000 * i;                     //设置重装数值, 72MHz 时
    SysTick -> CTRL = 0X01;                         //使能,减到零是无动作,采用外部时钟源
    SysTick -> VAL = 0;                             //清零计数器
    do
    {
        temp = SysTick -> CTRL;                     //读取当前倒计数值
    }
    while((temp&0x01)&&(!(temp&(1 << 16))));        //等待时间到达
    SysTick -> CTRL = 0;                            //关闭计数器
    SysTick -> VAL = 0;                             //清空计数器
}
```

第四步：在 systick.h 文件中输入如下源程序，其中条件编译格式不变，只要更改一下预定义变量名称即可，需要将刚定义的两个延时函数的声明加到头文件当中。

```
#ifndef _systick_H
#define _systick_H
#include < stm32f10x.h >
void delay_us(u32 i);
void delay_ms(u32 i);
#endif
```

第五步：在 LED 流水灯头文件 LED.H 中包含 systick.h，其源程序如下：

```
#ifndef _LED_H
#define _LED_H
#include "stm32f10x.h"
#include "systick.h"
void delay(u32 i);
void LEDInit(void);
void LEDdisplay(void);
#endif
```

第六步：修改 LED 流水灯源文件 LED.C 中的延时函数，将原来的“delay(6000000);”修改为“delay_ms(1000);”。

```
#include "LED.h"
/*****************************************************************
* 函 数 名          : LEDInit
* 函数功能          : LED 初始化函数
* 输    入          : 无
* 输    出          : 无
*****************************************************************/
void LEDInit()
{

    GPIO_InitTypeDef GPIO_InitStructure;
    SystemInit();
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC , ENABLE);

    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_All;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP ;
    GPIO_Init(GPIOC, &GPIO_InitStructure);
}
/*****************************************************************
* 函 数 名          : LEDdisplay
* 函数功能          : LED 显示函数  LED 闪烁
* 输    入          : 无
* 输    出          : 无
*****************************************************************/
void LEDdisplay()
{
    while(1)
    {
        GPIO_Write(GPIOC, 0xfe);
        delay_ms(1000);              //SysTick Timer Delay 1s
        GPIO_Write(GPIOC, 0xfd);
        delay_ms(1000);              // SysTick Timer Delay 1s
        GPIO_Write(GPIOC, 0xfb);
        delay_ms(1000);              // SysTick Timer Delay 1s
        GPIO_Write(GPIOC, 0xf7);
        delay_ms(1000);              // SysTick Timer Delay 1s
        GPIO_Write(GPIOC, 0xef);
        delay_ms(1000);              // SysTick Timer Delay 1s
        GPIO_Write(GPIOC, 0xdf);
        delay_ms(1000);              // SysTick Timer Delay 1s
        GPIO_Write(GPIOC, 0xbf);
```

```
            delay_ms(1000);         // SysTick Timer Delay 1s
            GPIO_Write(GPIOC, 0x7f);
            delay_ms(1000);         // SysTick Timer Delay 1s
        }
    }
```

第七步：编译工程，如没有错误则会在 output 文件中生成“工程模板. hex”文件，如有错误则修改源程序直至没有错误为止。

第八步：将生成的目标文件通过 ISP 软件下载到开发板 CPU 的 Flash 相存储器当中，复位运行，检查实验效果。

本节创建的 SysTick 延时函数还可以供后续章节所介绍项目调用，使用时需要包含其头文件 systick. h，给涉及时间控制的项目带来很大便利。

本章小结

本章首先介绍了库函数开方方式原理、特点及应用场合，并和寄存器开发方式进行了对比，指出基于库函数的开发方式是 STM32 嵌入式应用的首选。随后介绍了第一个基于库函数的嵌入式开发实例，即 LED 流水灯控制程序设计。由于读者前面没有接触过基于固件库的开发方式，所本部分内容介绍较为详尽，包括 GPIO 所有输出库函数的功能、参数和应用方法，并详细地给出了 LED 流水灯控制程序设计的步骤。SysTick 定时器是所有基于 ARM Cortex-M3 内核微控制器都具有的一个简单定时器，为应用程序在具有 Cortex-M3 内核的微控器之间移植提供了极大的方便。本章最后又介绍了 SysTick 定时器的功能、原理和控制寄存器，并编写了毫秒和微秒延时函数，用该延时函数重新改写 LED 流水灯控制程序，使其获得精确延时，为使用 SysTick 定时器进行时间控制应用程序提供了范例。

思考与扩展

1. 基于库函数开发方式的特点有哪些？
2. 函数 RCC_APB2PeriphClockCmd 的功能有哪些？
3. 函数 GPIO_Init 的功能有哪些？有哪些参数？
4. 函数 GPIO_Write 的功能是什么，有哪些参数？
5. 简述函数 GPIO_Write、GPIO_SetBits 和 GPIO_ResetBits 的异同点。
6. 简要说明 SysTick 定时器的概况以及使用该定时器的好处。
7. SysTick 定时器相关的控制寄存器有哪些？
8. SysTick 定时器常用的延时函数有哪两个？