第3章 基本I/O端口控制

CHAPTER 3

3.1 本章导读

从基本I/O操作开始学习，读者可以了解到以下重要信息：

（1）通过流水灯的例子进一步强化通过MDK建立工程的方法，配置的详细步骤。

（2）掌握直接寄存器控制I/O的步骤和方法，认识时钟树的概念，对STM32的时钟系统有初步的了解。

（3）认识I/O端口，掌握I/O端口的8种工作模式和各种模式对应应用的场合，以及如何采用寄存器配置各种模式。

（4）了解库函数，通过库函数操作流水灯，总结寄存器方法和库函数方法的区别和共同点。

（5）通过数码管和简单按键操作两个典型案例，进一步掌握库函数编程的方法和I/O端口控制的步骤。

3.2 新建工程进阶

单击桌面Keil μVision5图标，启动软件。如果是第一次使用，会打开一个自带的工程文件，可以通过菜单栏Project→Close Project选项把它关掉。

新建的工程文件保存在一个文件夹中。首先新建一个名为“流水灯”的文件夹，把第2章建立好的工程复制到该文件夹下，可以参考书中附带的例子，从例子中添加这些代码，后续会陆续解释其含义。“流水灯”工程的子文件夹如图3-1所示。

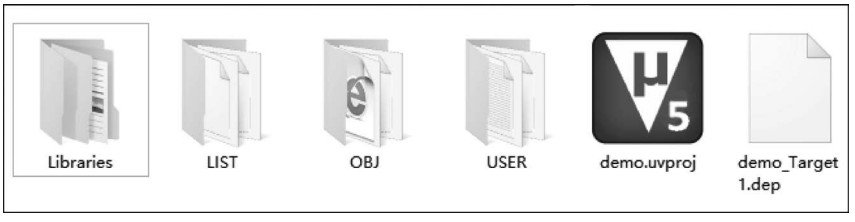

图3-1 “流水灯”工程的子文件夹

Libraries文件夹用来存放ST库里最核心的文件，其中包含两个子文件夹：STM32F10x_

StdPeriph_Driver 和 CMSIS。

- STM32F10x_StdPeriph_Driver 文件夹用来存放 STM32 库里面芯片上的所有驱动。inc 和 src 两个文件夹也是直接从 ST 的库里面复制过来的。
  - ✓ inc 文件夹里面是 ST 片上资源的驱动的头文件，如要用到某个资源，则必须把相应的头文件包含进来。
  - ✓ src 文件夹里面是 ST 片上资源的驱动的源文件，这些驱动文件涉及了大量的 C 语言的知识，是学习库函数的重点。
- CMSIS 文件夹用来存放库自带的启动文件和一些 M3 系列通用的文件。CMSIS 里面存放的文件适用于任何 M3 内核的单片机。CMSIS 的全称为 Cortex Microcontroller Software Interface Standard，是 ARM Cortex 微控制器软件接口标准，是 ARM 公司为芯片厂商提供的一套通用的且独立于芯片厂商的处理器软件接口。

LIST 文件夹用来保存编译后生成的链接列表清单文件。

OBJ 文件夹用来存放软件编译后输出的文件和编译过程中产生的文件。

USER 文件夹用来存放用户写的驱动文件，里面的 readme. txt 文件可以作为说明文档。

用户开始使用该软件时，建立工程相对困难，可以直接将 demo. uvproj 改为“流水灯. uvproj”。打开后重新编译，可以正常使用，这样就避免了工程名称都是 demo 的情况，便于管理。

打开工程后，可将 Target 1 工程名改为“流水灯”，这样更方便管理和阅读，如图 3-2 所示，以后类似工程的建立都可采用此方法，不用反复建立工程。

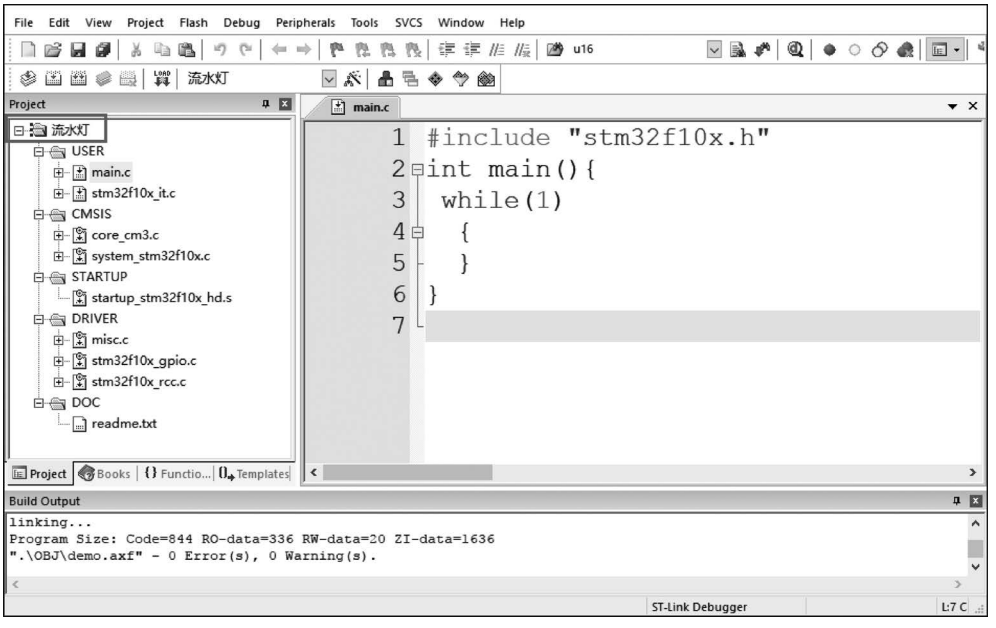

图 3-2　修改工程

3.3 MDK 工程配置

MDK 工程配置步骤如下。

(1) 单击工具栏中的魔术棒按钮，弹出配置菜单，可以看到配置菜单关于 Device、Target、Output、Listing、User、C/C++、Asm、Linker、Debug 和 Utilities 选项的设置。

(2) Device 在新建工程时已经选定了器件，单击 Target 选项卡，勾选微库，这样是为了后面的串口例程可以使用 printf 函数，如图 3-3 所示。

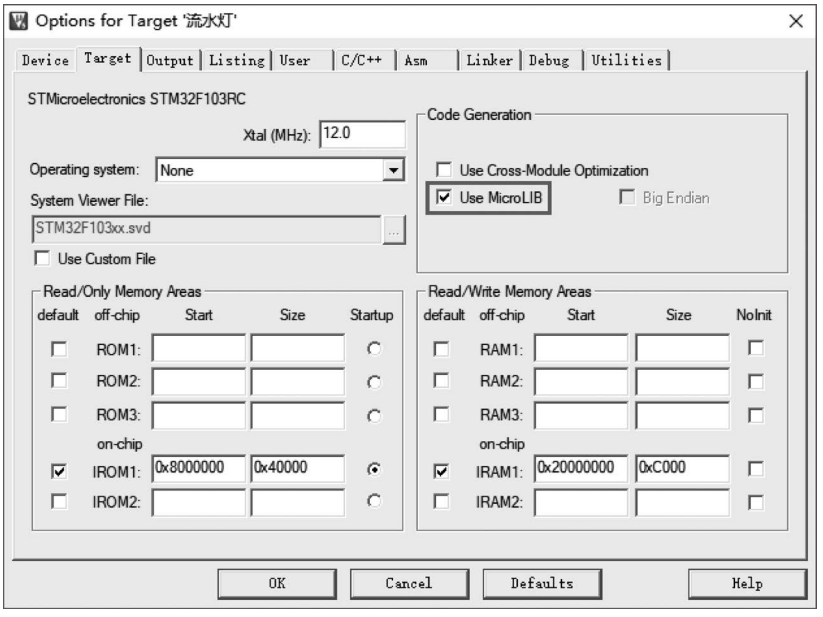

图 3-3 Target 选项卡

(3) 单击 Output 选项卡，再单击 Select Folder for Objects…按钮，设置编译后输出文件保存的位置。同时勾选 Debug Information、Create HEX File 和 Browse Information 复选框，如图 3-4 所示。

(4) 在 Listing 选项卡中，单击 Select Folder for Listings…按钮，定位到模板中的 Listing 文件夹，如图 3-5 所示。

(5) 在 C/C++选项卡上需要设置的比较多。

① 在 Define 里输入添加 STM32F10X_HD、USE_STDPERIPH_DRIVER 两个宏。添加 USE_STDPERIPH_DRIVER 是为了屏蔽编译器的默认搜索路径，转而使用添加到工程中的 ST 的库，添加 STM32F10X_HD 是因为用的芯片是大容量的，添加了 STM32F10X_HD 宏之后，库文件为大容量定义的寄存器就可以用了。芯片是小或中容量时，宏要换成 STM32F10X_LD 或者 STM32F10X_MD。

② 在 Include Paths 栏添加库文件的搜索路径，就可以屏蔽掉默认的搜索路径。

③ 当编译器在指定的路径下搜索不到时，还是会回到标准目录去搜索，如 ANSIC C 的库文件，例如 stdin.h、stdio.h。

库文件路径修改成功之后，如图 3-6 所示。

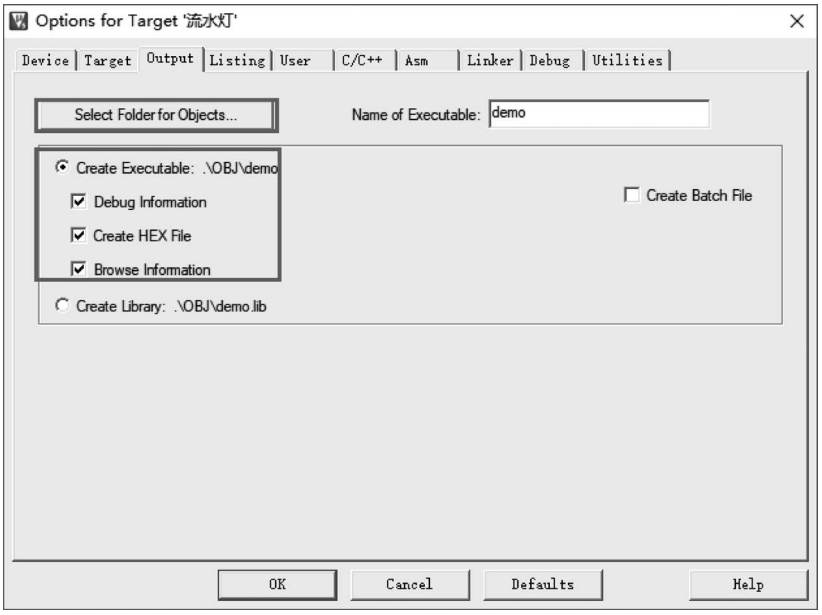

图 3-4　Output 选项卡

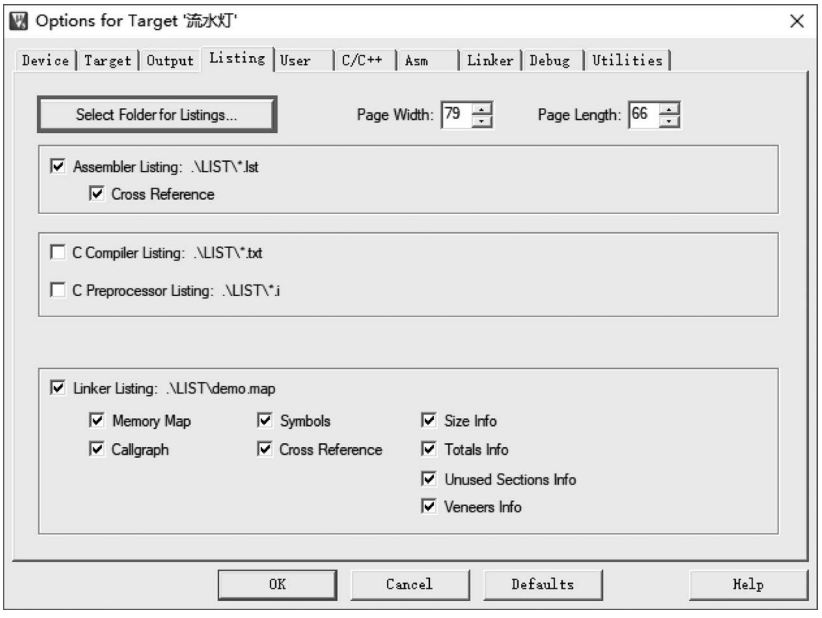

图 3-5　Listing 选项卡

(6) 单击 Debug 选项卡,选择 Use Simulator,软件仿真设置完成,如图 3-7 所示。

(7) 单击菜单栏中的编译按钮,对该工程进行编译,弹出编译信息,编译成功。

```
Build target '流水灯'
compiling main.c...
compiling stm32f10x_it.c...
compiling core_cm3.c...
compiling system_stm32f10x.c...
assembling startup_stm32f10x_hd.s...
```

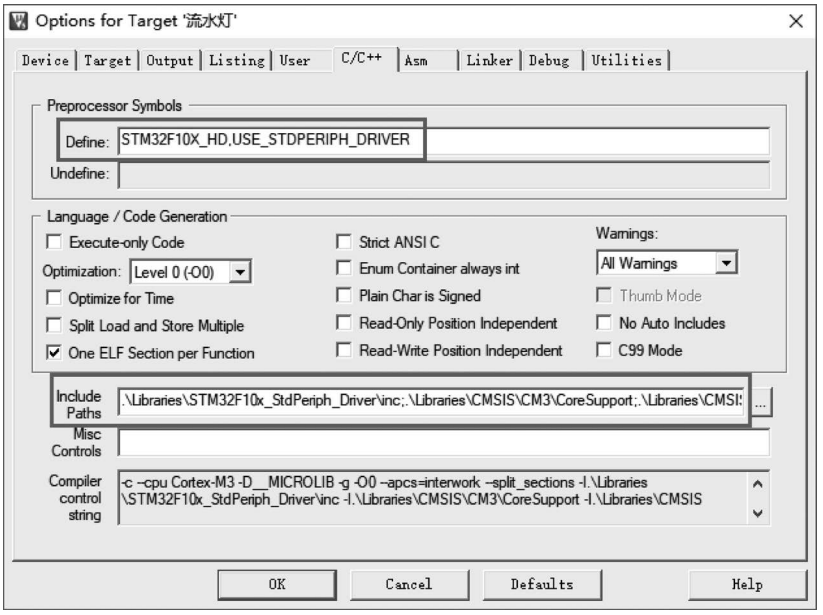

图 3-6　C/C++选项卡

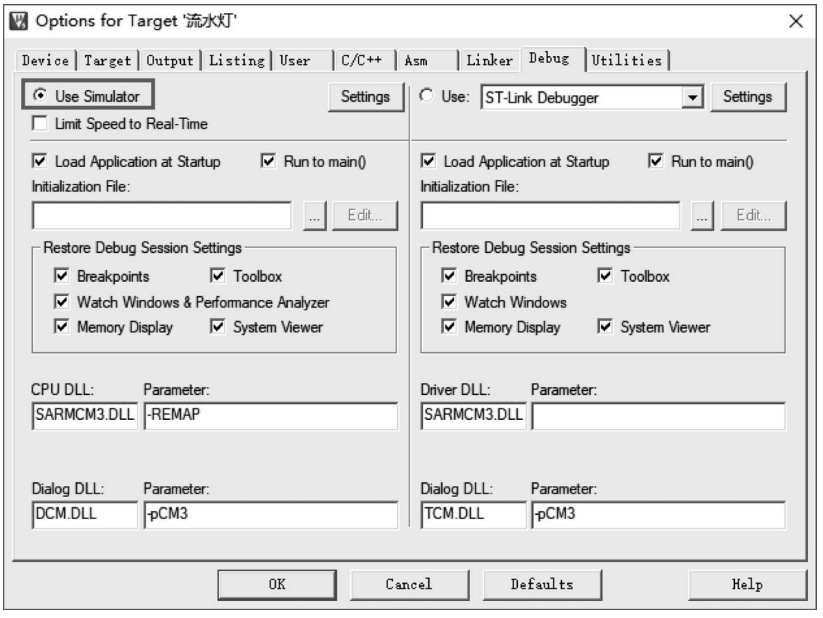

图 3-7　Debug 选项卡

```
compiling misc.c...
compiling stm32f10x_gpio.c...
compiling stm32f10x_rcc.c...
linking...
Program Size: Code = 484 RO - data = 320 RW - data = 0 ZI - data = 1024
FromELF: creating hex file...
".\OBJ\demo.axf" - 0 Error(s), 0 Warning(s).
```

3.4　寄存器操作

STM32 编程主要有两种方法：寄存器法和库函数法。寄存器法类似于单片机内部寄存器直接控制单片机，库函数法是通过调用 ST 公司提供的标准库控制单片机，两种方法各有千秋。以流水灯设计为例，其中发光二极管接到 STM32 的 GPIOB 口的 8～15 号引脚，电路如图 3-8 所示。

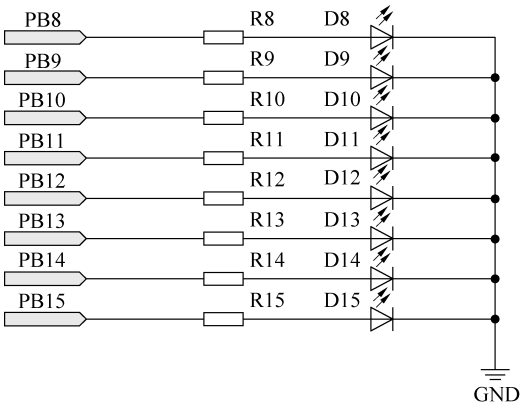

图 3-8　流水灯电路

在 main.c 文件里输入以下代码，实现流水灯的闪烁现象。有单片机编程经验的人很容易看懂这段代码，它主要涉及几个寄存器：RCC-> APB2ENR，GPIOB-> CRH 和 GPIOB-> ODR。STM32 的普通 I/O 端口的使用配置过程为：

① 先开启对应 I/O 端口时钟(RCC-> APB2ENR)(代码第 5 行)；

② 配置 I/O 端口(GPIOB-> CRH)(代码第 6 行)；

③ 给 I/O 端口赋值(GPIOB-> ODR)(代码第 9 行和第 11 行)。

这 3 步完成一个 I/O 端口的基本操作，代码如下(后面会详述)。

```
1.  #include "stm32f10x.h"
2.  void Delay(__IO u32 nCount);
3.  int main(void)
4.  {
5.  RCC->APB2ENR |= (1<<3);
6.  GPIOB->CRH = 0X22222222;
7.  while (1)
8.  {
9.  GPIOB->ODR = 0X0000;
10. Delay(0x0FFFEF);
11. GPIOB->ODR = 0XFFFF;
12. Delay(0x0FFFEF);
13. }
14. }
15. void Delay(__IO u32 nCount)
16. {
17. for(; nCount != 0; nCount--);
18. }
```

视频讲解

3.5 时钟配置

STM32的时钟系统功能完善，布局清晰，针对性更强。传统的51单片机外接晶振后，其他寄存器就可以使用，但STM32针对不同的功能要相应地设置其时钟。

3.5.1 时钟树

在使用51单片机时，时钟速度取决于外部晶振或内部RC振荡电路的频率，是不可以改变的。而ARM的出现打破了这个传统，可以通过软件随意改变时钟速度。这让设计更加灵活，但也给设计增加了复杂性。在使用某一功能前，要先对其时钟进行初始化。图3-9是它的时钟树，不同的外设对应不同的时钟，在STM32中有5个时钟源，分别为HSI、HSE、LSI、LSE、PLL。PLL是由锁相环电路倍频得到PLL时钟。

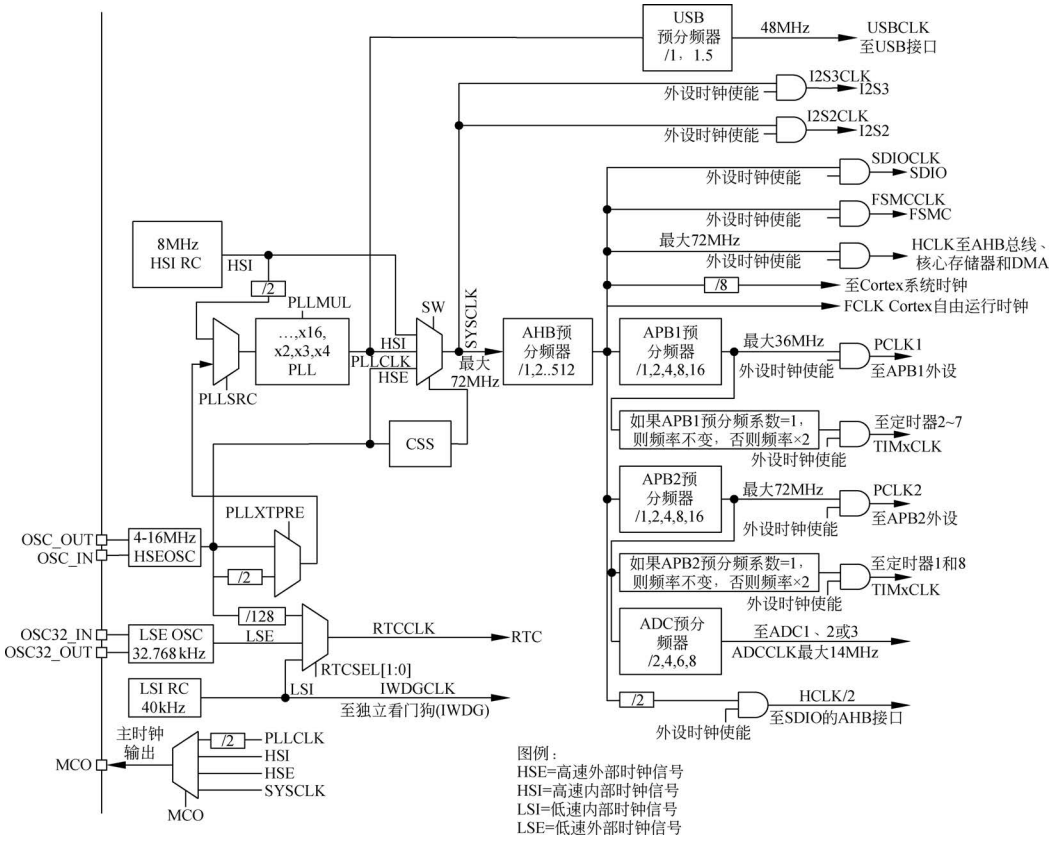

图3-9 时钟树

(1) HSI是高速内部时钟，RC振荡器，频率为8MHz。

(2) HSE是高速外部时钟，可接石英/陶瓷谐振器，或者接外部时钟源，频率范围为4～16MHz。

(3) LSI是低速内部时钟，RC振荡器，频率为40kHz。

(4) LSE是低速外部时钟，接频率为32.768kHz的石英晶体。

(5) PLL 为锁相环倍频输出,其时钟输入源可选择为 HSI/2、HSE 或者 HSE/2。倍频可选择为 2～16 倍,但是其输出频率最高不得超过 72MHz。

其中,40kHz 的 LSI 供独立看门狗 IWDG 使用,另外它还可以选择为实时时钟 RTC 的时钟源。另外,实时时钟 RTC 的时钟源还可以选择 LSE,或者是 HSE 的 128 分频。RTC 的时钟源通过 RTCSEL[1:0]来选择。

STM32 中有一个全速功能的 USB 模块,其串行接口引擎需要一个频率为 48MHz 的时钟源。该时钟源只能从 PLL 输出端获取,可以选择为 1.5 分频或者 1 分频。也就是说,当需要使用 USB 模块时,PLL 必须使能,并且时钟频率配置为 48MHz 或 72MHz。

另外,STM32 还可以选择一个时钟信号输出到 MCO 引脚(PA8)上,可以选择为 PLL 输出的 2 分频、HSI、HSE 或者系统时钟。

3.5.2　时钟源

系统时钟 SYSCLK 是供 STM32 中绝大部分功能工作的时钟源。系统时钟可选择为 PLL 输出、HSI 或者 HSE。系统时钟最大频率为 72MHz,通过 AHB 分频器分频后送给各模块使用,AHB 分频器可选择 1、2、4、8、16、64、128、256、512 分频。其中,AHB 分频器输出的时钟送给 5 大模块使用。

(1) 送给 AHB 总线、内核、内存和 DMA 使用的 HCLK 时钟。

(2) 通过 8 分频后送给 Cortex 的系统定时器时钟。

(3) 直接送给 Cortex 的空闲运行时钟 FCLK。

(4) 送给 APB1 分频器。APB1 分频器可选择 1、2、4、8、16 分频,其输出一路供 APB1 外设使用(PCLK1,最大频率 36MHz),另一路送给定时器 2、3、4 倍频器使用。该倍频器可选择 1 或者 2 倍频,时钟输出供定时器 2、3、4 使用。

(5) 送给 APB2 分频器。APB2 分频器可选择 1、2、4、8、16 分频,其输出一路供 APB2 外设使用(PCLK2,最大频率 72MHz),另一路送给定时器 1 倍频器使用。该倍频器可选择 1 或者 2 倍频,时钟输出供定时器 1 使用。另外,APB2 分频器还有一路输出供 ADC 分频器使用,分频后送给 ADC 模块使用。ADC 分频器可选择为 2、4、6、8 分频。

在以上的时钟输出中,有很多是带使能控制的,例如 AHB 总线时钟、内核时钟、各种 APB1 外设、APB2 外设等。当需要使用某模块时,一定要先使能对应的时钟。

需要注意定时器的倍频器,当 APB 的分频为 1 时,它的倍频值为 1;否则它的倍频值就为 2。

连接在 APB1(低速外设)上的设备有电源接口、备份接口、CAN、USB、I2C1、I2C2、UART2、UART3、SPI2、窗口看门狗、Timer2、Timer3、Timer4。注意,USB 模块虽然需要一个单独的 48MHz 时钟信号,但它不是供 USB 模块工作的时钟,只是提供给串行接口引擎(SIE)使用的时钟。USB 模块工作的时钟应该是由 APB1 提供的。

连接在 APB2(高速外设)上的设备有 UART1、SPI1、Timer1、ADC1、ADC2、所有普通 I/O 端口(PA～PE)、第二功能 I/O 端口。

通过对时钟树的简单了解,明确了不同的功能对应不同的时钟关系。知道了普通 I/O 端口连接在 APB2 设备上,需要初始化 APB2 的时钟,即时钟控制(RCC)的 APB2 的对应使能寄存器。


视频讲解

3.5.3 APB2 外设时钟使能寄存器(RCC_APB2ENR)

外设通常无访问等待周期。但在 APB2 总线上的外设被访问时，将插入等待状态直到 APB2 的外设访问结束。它的寄存器格式如图 3-10 所示。各位对应含义如表 3-1 所示。

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
保留															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
保留	USART1 EN	保留	SPI1 EN	TIM1 EN	ADC2 EN	ADC1 EN	保留		IOPE EN	IOPD EN	IOPC EN	IOPB EN	IOPA EN	保留	AFIO EN
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

图 3-10　APB2 外设时钟使能寄存器格式

表 3-1　使能寄存器位置功能表

位	描　　述
位 31:15	保留，始终读为 0
位 14 USART1EN	USART1 时钟使能，由软件置 1 或清 0 0：USART1 时钟关闭 1：USART1 时钟开启
位 13	保留，始终读为 0
位 12 SPI1EN	SPI1 时钟使能，由软件置 1 或清 0 0：SPI1 时钟关闭 1：SPI1 时钟开启
位 11 TIM1EN	TIM1 定时器时钟使能，由软件置 1 或清 0 0：TIM1 定时器时钟关闭 1：TIM1 定时器时钟开启
位 10 ADC2EN	ADC2 接口时钟使能，由软件置 1 或清 0 0：ADC2 接口时钟关闭 1：ADC2 接口时钟开启
位 9 ADC1EN	ADC1 接口时钟使能，由软件置 1 或清 0 0：ADC1 接口时钟关闭 1：ADC1 接口时钟开启
位 8:7	保留，始终读为 0
位 6 IOPEEN	I/O 端口 E 时钟使能，由软件置 1 或清 0 0：I/O 端口 E 时钟关闭 1：I/O 端口 E 时钟开启
位 5 IOPDEN	I/O 端口 D 时钟使能，由软件置 1 或清 0 0：I/O 端口 D 时钟关闭 1：I/O 端口 D 时钟开启
位 4 IOPCEN	I/O 端口 C 时钟使能，由软件置 1 或清 0 0：I/O 端口 C 时钟关闭 1：I/O 端口 C 时钟开启

续表

位	描　　述
位 3 IOPBEN	I/O 端口 B 时钟使能，由软件置 1 或清 0 0：I/O 端口 B 时钟关闭 1：I/O 端口 B 时钟开启
位 2 IOPAEN	I/O 端口 A 时钟使能，由软件置 1 或清 0 0：I/O 端口 A 时钟关闭 1：I/O 端口 A 时钟开启
位 1	保留，始终读为 0
位 0 AFIOEN	辅助功能 I/O 时钟使能，由软件置 1 或清 0 0：辅助功能 I/O 时钟关闭 1：辅助功能 I/O 时钟开启

例如，开启 PB 口时钟的寄存器操作为

```
RCC->APB2ENR |= (1<<3); //开启 PB 口的时钟
```

请读者思考采用“|=”赋值的原因。

3.6　I/O 端口配置


视频讲解

除配置时钟，I/O 在使用之前也需要进行配置，通过 PB 口的使用，说明它的一般配置过程。

3.6.1　I/O 基本情况

每个 GPIO 端口有：

(1) 两个 32 位配置寄存器(GPIOx_CRL，GPIOx_CRH)；

(2) 两个 32 位数据寄存器(GPIOx_IDR，GPIOx_ODR)；

(3) 一个 32 位置位/复位寄存器(GPIOx_BSRR)；

(4) 一个 16 位复位寄存器(GPIOx_BRR)；

(5) 一个 32 位锁定寄存器(GPIOx_LCKR)。

根据数据手册中列出的每个 I/O 端口的特定硬件特征，GPIO 端口的每个位可以由软件分别配置成多种模式。

STM32 的 I/O 端口可以由软件配置成如下 8 种模式，包括 4 种输入模式和 4 种输出模式。

(1) 浮空输入。

(2) 上拉输入。

(3) 下拉输入。

(4) 模拟输入。

(5) 开漏输出。

(6) 推挽输出。

(7) 复用开漏输出。

(8) 复用推挽输出。

每个 I/O 端口可以自由编程，但 I/O 端口寄存器必须要按 32 位访问。STM32 的很多

I/O端口都是兼容5V的，这些I/O端口在与5V电压的外设连接时很有优势，具体哪些I/O端口是兼容5V的，可以从该芯片的数据手册引脚描述章节查到。

STM32的每个I/O端口都有7个寄存器来控制。常用的I/O端口寄存器有4个，分别为CRL、CRH、IDR、ODR。CRL和CRH控制着每个I/O端口的模式及输出速率。STM32的I/O端口位配置如表3-2所示。

表3-2 STM32的I/O端口位配置表

配置模式		CNF1	CNF0	MODE1	MODE0	PxODR寄存器
通用输出	推挽式输出	0	0	01(最大输出速率10MHz) 10(最大输出速率2MHz) 11(最大输出速率50MHz)		0或1
	开漏输出	0	1			0或1
复用功能输出	推挽式输出	1	0			不使用
	开漏输出	1	1			不使用
输入	模拟输入	0	0	00(保留)		不使用
	浮空输入	0	1			不使用
	下拉输入	1	0			0
	上拉输入	1	0			1

为了方便，在对寄存器的描述中使用了下列缩写，具体含义如下。

- rw(read/write)：软件能读写此位。
- r(read-only)：软件只能读此位。
- w(write-only)：软件只能写此位，读此位将返回复位值。
- rc_w1(read/clear)：软件可以读此位，也可以通过写“1”清除此位，写“0”对此位无影响。
- rc_w0(read/clear)：软件可以读此位，也可以通过写“0”清除此位，写“1”对此位无影响。
- rc_r(read/clear by read)：软件可以读此位；读此位将自动地将它清除为“0”，写“0”对此位无影响。
- rs(read/set)：软件可以读也可以设置此位，写“0”对此位无影响。
- rt_w(read-only write trigger)：软件可以读此位；写“0”或“1”触发一个事件，但对此位数值没有影响。
- t(toggle)：软件只能通过写“1”来翻转此位，写“0”对此位无影响。
- Res.(Reserved)：保留位，必须保持默认值不变。

3.6.2 GPIO配置寄存器描述

(1) 端口配置低寄存器(GPIOx_CRL)(x=A,B,…,E)，如图3-11所示。各位对应关系如表3-3所示。

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CNF7[1:0]		MODE7[1:0]		CNF6[1:0]		MODE6[1:0]		CNF5[1:0]		MODE5[1:0]		CNF4[1:0]		MODE4[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CNF3[1:0]		MODE3[1:0]		CNF2[1:0]		MODE2[1:0]		CNF1[1:0]		MODE1[1:0]		CNF0[1:0]		MODE0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

图3-11 端口配置低寄存器格式

表 3-3　GPIO 端口配置低寄存器配置方式

位	描　　述	
位	**CNFy[1:0]**：端口 x 配置位(y = 0,1,…,7)	
31:30	软件通过这些位配置相应的 I/O 端口	
27:26	在输入模式(MODE[1:0]=00)：	在输出模式(MODE[1:0]>00)：
23:22	00：模拟输入模式	00：通用推挽输出模式
19:18	01：浮空输入模式(复位后的状态)	01：通用开漏输出模式
15:14	10：上拉/下拉输入模式	10：复用功能推挽输出模式
11:10	11：保留	11：复用功能开漏输出模式
7:6		
3:2		
位	**MODEy[1:0]**：端口 x 的模式位(y=0,1,…,7)	
29:28	软件通过这些位配置相应的 I/O 端口	
25:24	00：输入模式(复位后的状态)	
21:20	01：输出模式,最大速度 10MHz	
17:16	10：输出模式,最大速度 2MHz	
13:12	11：输出模式,最大速度 50MHz	
9:8		
5:4		
1:0		

(2) 端口配置高寄存器(GPIOx_CRH)(x=A,B,…,E),如图 3-12 所示。各位对应关系如表 3-4 所示。

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CNF15[1:0]		MODE15[1:0]		CNF14[1:0]		MODE14[1:0]		CNF13[1:0]		MODE13[1:0]		CNF12[1:0]		MODE12[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CNF11[1:0]		MODE11[1:0]		CNF10[1:0]		MODE10[1:0]		CNF9[1:0]		MODE9[1:0]		CNF8[1:0]		MODE8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

图 3-12　端口配置高寄存器格式

表 3-4　GPIO 端口配置高寄存器配置方式

位	描　　述	
位	**CNFy[1:0]**：端口 x 配置位(y=8,9,…,15)	
31:30	软件通过这些位配置相应的 I/O 端口	
27:26	在输入模式(MODE[1:0]=00)：	在输出模式(MODE[1:0]>00)：
23:22	00：模拟输入模式	00：通用推挽输出模式
19:18	01：浮空输入模式(复位后的状态)	01：通用开漏输出模式
15:14	10：上拉/下拉输入模式	10：复用功能推挽输出模式
11:10	11：保留	11：复用功能开漏输出模式
7:6		
3:2		

续表

位	描　　述
位	**MODEy[1:0]**：端口 x 的模式位(y=8,9,…,15)
29:28	软件通过这些位配置相应的 I/O 端口
25:24	00：输入模式(复位后的状态)
21:20	01：输出模式,最大速度 10MHz
17:16	10：输出模式,最大速度 2MHz
13:12	11：输出模式,最大速度 50MHz
9:8	
5:4	
1:0	

例如,控制的是 LED 小灯,可以选择通用推挽输出模式,设置速度为 2MHz,实现代码为

```
GPIOB->CRH = 0X22222222;
```

3.6.3 端口输出数据寄存器

端口输出数据寄存器(GPIOx_ODR)(x=A,B,…,E),如图 3-13 所示。各位对应关系如表 3-5 所示。

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
保留															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ODR15	ODR14	ODR13	ODR12	ODR11	ODR10	ODR9	ODR8	ODR7	ODR6	ODR5	ODR4	ODR3	ODR2	ODR1	ODR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

图 3-13　端口输出数据寄存器格式

表 3-5　端口输出数据寄存器各位含义

位	描　　述
位 31:16	保留,始终读为 0
位 15:0	ODRy[15:0]：端口输出数据(y=0,1,…,15),这些位可读可写并只能以字节(16 位)的形式操作 注：对 GPIOx_BSRR(x=A,B,…,E),可以分别对各个 ODR 位进行独立地设置/清除

例如：

```
GPIOB->ODR = 0X0000;  //灯灭
GPIOB->ODR = 0XFFFF;  //灯亮
```


视频讲解

3.7 库函数操作

采用库函数控制流水灯,程序的可读性增强,代码维护方便,且操作简便。在主程序中输入以下代码：

```
1.  #include "stm32f10x.h"
2.  void Delay(__IO u32 nCount);
3.  int main(void)
4.  {
```

```
5.  GPIO_InitTypeDef GPIO_InitStructure;
6.  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);
7.  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_All;
8.  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
9.  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
10. GPIO_Init(GPIOB, &GPIO_InitStructure);
11. while (1)
12. {
13. GPIO_Write(GPIOA, 0xFFFF);
14. Delay(0x0FFFEF);
15. GPIO_Write(GPIOA, 0x0000);
16. Delay(0x0FFFEF);
17. }
18. }
19. void Delay(__IO u32 nCount)
20. {
21. for(; nCount != 0; nCount--);
22. }
```

这段代码在完成了 LED 初始化后实现了小灯的闪烁。LED 初始化实际是库函数操作的核心，采用库函数来配置时钟、工作模式等与寄存器操作比较，库函数法的可读性增强了。

3.7.1　GPIO_Init 函数

在 main 文件中，第 10 行代码调用了 GPIO_Init 函数。通过《STM32F101xx 和 STM32F103xx 固件函数库》手册找到该库函数的原型，表 3-6 为函数 GPIO_Init 的具体说明。

表 3-6　函数 GPIO_Init

函　数　名	GPIO_Init
函数原型	Void GPIO_Init(GPIO_TypeDef * GPIOx,GPIO_InitTypeDef * GPIO_InitStruct)
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
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
输出参数	无
返回值	无
先决条件	无
被调用函数	无

第 5 行代码利用库定义了一个 GPIO_InitStructure 的结构体，结构体的类型为 GPIO_InitTypeDef；它是利用 typedef 定义的新类型。追踪其定义原型，知道它位于 stm32f10x_gpio.h 文件中，代码为

```
typedef struct
{
  uint16_t GPIO_Pin;
  GPIOSpeed_TypeDef GPIO_Speed;
  GPIOMode_TypeDef GPIO_Mode;
}GPIO_InitTypeDef;
```

通过这段代码可知，GPIO_InitTypeDef 类型的结构体有 3 个成员，分别为 uint16_t 类

型的 GPIO_Pin、GPIOSpeed_TypeDef 类型的 GPIO_Speed 及 GPIOMode_TypeDef 类型的 GPIO_Mode，下面分别解释这 3 个成员的含义。

1. 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 3-7 中的数据任意组合。

表 3-7 GPIO_Pin 值

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

这些宏的值，就是允许给结构体成员 GPIO_Pin 赋的值，例如给 GPIO_Pin 赋值为宏 GPIO_Pin_0，表示选择了 GPIO 端口的第 0 个引脚，在后面会通过一个函数把这些宏的值进行处理，设置相应的寄存器，实现对 GPIO 端口的配置。

2. GPIOSpeed

GPIOSpeed_TypeDef 库定义的新类型，GPIOSpeed_TypeDef 原型如下：

```
typedef enum
{
  GPIO_Speed_10MHz = 1,
  GPIO_Speed_2MHz,
  GPIO_Speed_50MHz
}GPIOSpeed_TypeDef;
```

这是一个枚举类型，定义了 3 个枚举常量，GPIO_Speed 值如表 3-8 所示。

表 3-8 GPIO_Speed 值

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

这些常量可用于标识 GPIO 引脚配置成的各自最高速度，所以在为结构体中的 GPIO_

Speed 赋值时，就可以直接用这些含义清晰地枚举标识符。

3. GPIOMode

GPIOMode_TypeDef 也是一个枚举类型定义符，具体含义如表 3-9 所示，其原型如下：

```
typedef enum
{
GPIO_Mode_AIN = 0x0,
  GPIO_Mode_IN_FLOATING = 0x04,
  GPIO_Mode_IPD = 0x28,
  GPIO_Mode_IPU = 0x48,
  GPIO_Mode_Out_OD = 0x14,
  GPIO_Mode_Out_PP = 0x10,
  GPIO_Mode_AF_OD = 0x1C,
  GPIO_Mode_AF_PP = 0x18
}GPIOMode_TypeDef;
```

表 3-9　GPIO_Mode 值

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

这个枚举类型也定义了很多含义清晰的枚举常量，用来帮助配置 GPIO 引脚的模式，例如，GPIO_Mode_AIN 为模拟输入，GPIO_Mode_IN_FLOATING 为浮空输入模式。可以明白 GPIO_InitTypeDef 类型结构体的作用，整个结构体包含 GPIO_Pin、GPIO_Speed、GPIO_Mode 3 个成员，这 3 个成员赋予不同的数值可以对 GPIO 端口进行不同的配置，这些可配置的数值已经由 ST 的库文件封装成见名知义的枚举常量，这使编写代码变得非常简便。

3.7.2　RCC_APB2PeriphClockCmd

GPIO 所用的时钟 PCLK2 采用的默认值为 72MHz。采用默认值可以不修改分频器，但外设时钟默认处在关闭状态，所以外设时钟一般会在初始化外设时设置为开启，开启和关闭外设时钟采用封装好的库函数 RCC_APB2PeriphClockCmd()，该函数如表 3-10 所示。

表 3-10　RCC_APB2PeriphClockCmd()库函数

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
输入参数 2	NewState：指定外设时钟的新状态 这个参数可以取 ENABLE 或者 DISABLE

续表

函　数　名	RCC_APB2PeriphClockCmd()
输出参数	无
返回值	无
先决条件	无
被调用函数	无

该参数为门控的 APB2 外设时钟，可以取表 3-11 中的一个或者多个值的组合作为该参数的值。

表 3-11　APB2 外设时钟的取值参数

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

例如，使能 GPIOA、GPIOB 和 SPI1 时钟，代码为

```
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA|RCC_APB2Periph_GPIOB |RCC_APB2Periph_SPI1,
ENABLE);//开放对应功能的时钟
```

3.7.3　控制 I/O 输出电平

前面选择好了引脚，配置了其功能及开启了相应的时钟，终于可以正式控制 I/O 端口的电平高低，从而实现控制 LED 灯的亮与灭。

要控制 GPIO 引脚的电平高低，只要在 GPIOx_BSRR 寄存器相应的位写入控制参数即可。ST 库也提供了具有这样功能的函数，可以分别用 GPIO_SetBits()(见表 3-12)控制输出高电平和 GPIO_ResetBits()(见表 3-13)控制输出低电平。GPIO_Write()可以向指定 GPIO 数据端口写入数据，如表 3-14 所示。

表 3-12　函数 GPIO_SetBits

函　数　名	GPIO_SetBits
函数原型	void GPIO_SetBits(GPIO_TypeDef * 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

续表

函　数　名	GPIO_SetBits
输出参数	无
返回值	无
先决条件	无
被调用函数	无

表 3-13　函数 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，用来选择 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

表 3-14　函数 GPIO_Write

函　数　名	GPIO_Write
函数原形	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
返回值	无
先决条件	无
被调用函数	无

例如，设置 GPIOA 端口引脚 10 和引脚 15 为高电平，代码为

```
GPIO_SetBits(GPIOA, GPIO_Pin_10 | GPIO_Pin_15);
```

例如，设置 GPIOA 端口引脚 10 和引脚 15 为低电平，代码为

```
GPIO_ResetBits(GPIOA,GPIO_Pin_10 | GPIO_Pin_15);
```

例如，设置 GPIOA 口，代码为

```
GPIO_Write(GPIOA, 0x1101);
```

3.8　数码管操作实例


视频讲解

通过数码管的例子，进一步说明 I/O 的使用方法。常用 GPIO 库函数，如表 3-15 所示，这些函数在以后的 I/O 控制中会陆续使用到。

表 3-15 GPIO 库函数

函 数 名	描 述
GPIO_DeInit	将外设 GPIOx 寄存器重设为默认值
GPIO_AFIODeInit	将复用功能(重映射事件控制和 EXTI 设置)重设为默认值
GPIO_Init	根据 GPIO_InitStruct 中指定的参数初始化外设 GPIOx 寄存器
GPIO_StructInit	把 GPIO_InitStruct 中的每一个参数按默认值填入
GPIO_ReadInputDataBit	读取指定端口引脚的输入
GPIO_ReadInputData	读取指定的 GPIO 端口输入
GPIO_ReadOutputDataBit	读取指定端口引脚的输出
GPIO_ReadOutputData	读取指定的 GPIO 端口输出
GPIO_SetBits	设置指定的数据端口位
GPIO_ResetBits	清除指定的数据端口位
GPIO_WriteBit	设置或者清除指定的数据端口位
GPIO_Write	向指定 GPIO 数据端口写入数据
GPIO_PinLockConfig	锁定 GPIO 引脚设置寄存器
GPIO_EventOutputConfig	选择 GPIO 引脚用作事件输出
GPIO_EventOutputCmd	使能或者失能事件输出
GPIO_PinRemapConfig	改变指定引脚的映射
GPIO_EXTILineConfig	选择 GPIO 引脚用作外部中断线路

3.8.1 数码管基础知识

(1) 一个数码管有 8 段,分别为 A、B、C、D、E、F、G、DP,即由 8 个发光二极管组成,如图 3-14 所示。

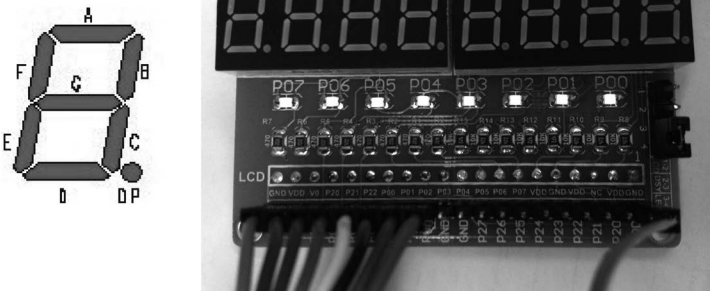

图 3-14 数码管示意图

(2) 发光二极管导通的方向是一定的(导通电压一般取 1.7V),这 8 个发光二极管的公共端有两种,可以接+5V(即为共阳极数码管)或接地(即为共阴极数码管)。

(3) 可分共阳极(公共端接高电平或+5V 电压)和共阴极(共低电平或接地)两种数码管。

① 共阳极数码管编码表。位选为高电平(即 1)选中数码管,各段选为低电平选中各数码段亮,由 0 到 f 的编码为

```
uchar code table[ ] = {0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,
```

```
                0xf8,0x80,0x90,0x88,0x83,0xc6,0xa1,
                0x86,0x8e};
```

② 共阴极数码管编码表。位选为低电平(即 0)选中数码管,各段选为高电平选中各数码段亮,由 0 到 f 的编码为

```
uchar code table[ ] = {0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,
                0x07,0x7f,0x6f,0x77,0x7c,0x39,0x5e,
                0x79,0x71};
```

(4) 其中,每个段均有 0(不导通)和 1(导通发光)两种状态,但共阳极数码管和共阴极数码管显然是不同的。

(5) 它在程序中的应用是用一个 8 位二进制数表示,A 为最低位,……,F 为最高位(第 8 位)。

3.8.2　硬件电路设计

如图 3-15 所示的电路使用共阳极数码管,段选端接到 PB0～PB7,位选端接到 PB8～PB15,通过 PNP 三极管实现电流的放大,增加驱动能力,提高显示的亮度,当位选端为低电平时,三极管导通,段选端输入正确的数据,数码管就能实现正确的显示。

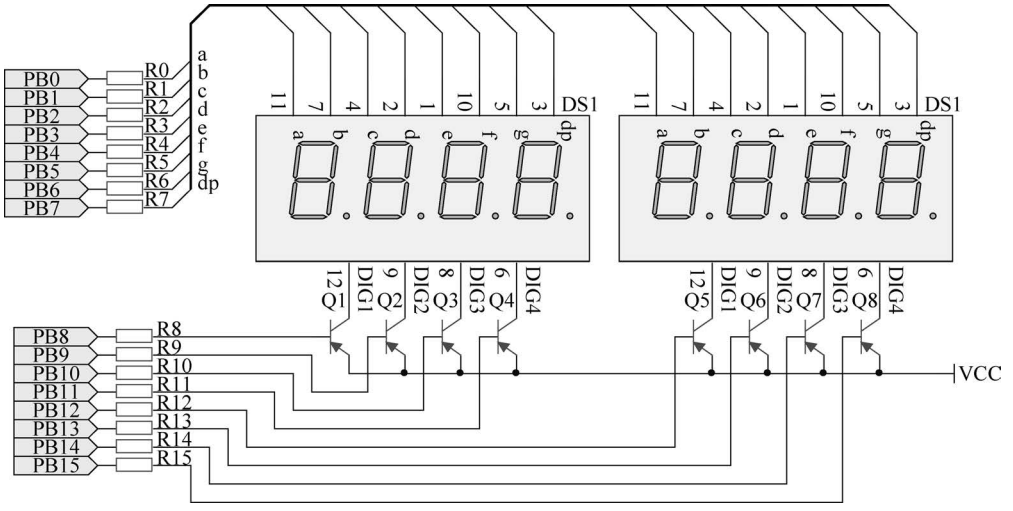

图 3-15　数码管接口电路

通过这个例子进一步了解 I/O 端口的库函数操作方法,以及 I/O 复用端口的设置问题。

3.8.3　软件说明

下面这段代码简单实现了数码管从 1～F 的亮灭,这里使用 RCC_APB2Periph_AFIO、GPIO_PinRemapConfig 的配置。下面对程序进行解释和说明。

STM32F10x 系列的 MCU 复位后,PA13/14/15 和 PB3/4 默认配置为 JTAG 功能。有时为了充分利用 STM32 I/O 端口的资源,会把这些端口设置为普通 I/O 端口。STM32 的 PB3、PB4 分别是 JTAG 的 JTDO 和 NJTRST 引脚,在没关闭 JTAG 功能之前,程序中配置不了这些引脚的功能。要配置这些引脚,首先要开启 AFIO 时钟,然后在 AFIO 中设置释放

这些引脚。

```
1. #include "stm32f10x.h"
2. void Delay(__IO u32 nCount);
3. u8 table[] = {0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,\
4. 0x80,0x90,0x88,0x83,0xc6,0xa1,0x86,0x8e};
5. u8 i;
6. int main(void)
7. {
8.
9. GPIO_InitTypeDef GPIO_InitStructure;
10. RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO | RCC_APB2Periph_GPIOB , ENABLE);
11. GPIO_PinRemapConfig(GPIO_Remap_SWJ_Disable,ENABLE);
12. GPIO_InitStructure.GPIO_Pin = GPIO_Pin_All;
13. GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
14. GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
15. GPIO_Init(GPIOB, &GPIO_InitStructure);
16.while (1)
17.{
18.for(i = 0;i < 16;i++)
19.{
20. GPIO_Write(GPIOB, table[i]);
21. Delay(0x0FFFEF5);
22.if(i == 15) i = 0;
23.}
24.}
25.}
26. void Delay(__IO u32 nCount)
27.{
28.for(;nCount != 0; nCount--);
29.}
30. void Delay(__IO u32 nCount)
31. {
32. for(; nCount != 0; nCount--);
33. }
```

第 10 行代码：开启 AFIO 时钟和 GPIOB 的时钟。

第 11 行代码：禁用 JTAG 功能，重新映射为普通的 I/O 端口。

为了优化 64 引脚或 100 引脚封装的外设数目，可以把一些复用功能重新映射到其他引脚上。设置复用重映射和调试 I/O 配置寄存器（AFIO_MAPR）可实现引脚的重新映射。这时，复用功能不再映射到它们的原始分配上。GPIO_PinRemapConfig 函数如表 3-16 所示。

表 3-16　函数 GPIO_PinRemapConfig

函　数　名	GPIO_PinRemapConfig
函数原型	void GPIO_PinRemapConfig(u32 GPIO_Remap, FunctionalState NewState)
功能描述	改变指定引脚的映射
输入参数 1	GPIO_Remap：选择重映射的引脚
输入参数 2	NewState：引脚重映射的新状态 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

第 12～15 行：设置 I/O 的工作为推挽输出工作模式，速度为 50MHz。

第 26～33 行：延时子程序。

第 20 行：把数码管对应的赋值给 PB0～PB7 口。注意：这里赋值是 8 位数据，PB8～PB15 并没有给赋值，相当于高 8 位为低电平，根据图 3-15 可知，代码运行 8 个数码管，同时显示相同的数字。

参数 GPIO_Remap 用以选择用作事件输出的 GPIO 端口。表 3-17 给出了该参数可取的值。

表 3-17　GPIO_Remap

GPIO_Remap	描　　述
GPIO_Remap_SPI1	SPI1 复用功能映射
GPIO_Remap_I2C1	I2C1 复用功能映射
GPIO_Remap_USART1	USART1 复用功能映射
GPIO_PartialRemap_USART3	USART2 复用功能映射
GPIO_FullRemap_USART3	USART3 复用功能完全映射
GPIO_PartialRemap_TIM1	USART3 复用功能部分映射
GPIO_FullRemap_TIM1	TIM1 复用功能完全映射
GPIO_PartialRemap1_TIM2	TIM2 复用功能部分映射 1
GPIO_PartialRemap2_TIM2	TIM2 复用功能部分映射 2
GPIO_FullRemap_TIM2	TIM2 复用功能完全映射
GPIO_PartialRemap_TIM3	TIM3 复用功能部分映射
GPIO_FullRemap_TIM3	TIM3 复用功能完全映射
GPIO_Remap_TIM4	TIM4 复用功能映射
GPIO_Remap1_CAN	CAN 复用功能映射 1
GPIO_Remap2_CAN	CAN 复用功能映射 2
GPIO_Remap_PD01	PD01 复用功能映射
GPIO_Remap_SWJ_NoJTRST	除 JTRST 外 SWJ 完全使能(JTAG+SW-DP)
GPIO_Remap_SWJ_JTAGDisable	JTAG-DP 失能+SW-DP 使能
GPIO_Remap_SWJ_Disable	SWJ 完全失能(JTAG+SW-DP)

3.9　简单按键操作实例


视频讲解

显示模块 I/O 端口都是作为输出使用控制显示模块，本节通过简单按键控制，配合 LED 小灯，了解 I/O 端口的输入使用方法。按键被按下，LED 小灯熄灭。图 3-16 为按键电路图。

主程序如下：

```
1.  #include "stm32f10x.h"
2.  #include "led.h"
3.  #include "key.h"
4.  int main(void)
5.  {
6.      LED_GPIO_Config();
7.      LED5(ON);
```

```
8.      Key_GPIO_Config();
9.      while(1)
10.     {
11.         if(Key_Scan(GPIOA,GPIO_Pin_0) == KEY_ON)
12.         {
13.             LED5(OFF);
14.         }
15.     }
16. }
```

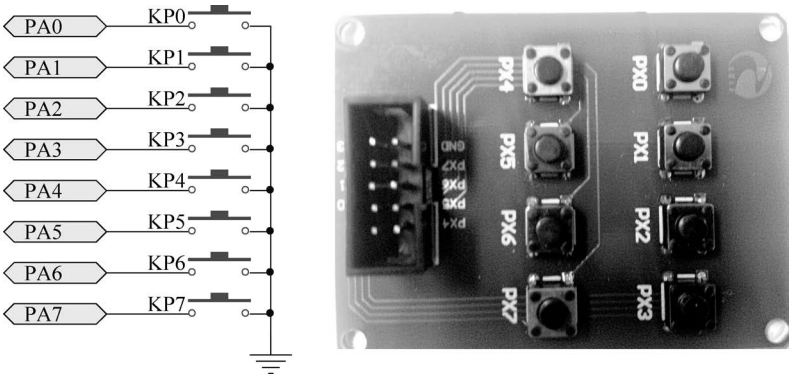

图 3-16　按键电路图

第 6 行代码配置 LED，和 3.7 节使用方法一致。

第 7 行代码是点亮 LED，采用对 I/O 端口的位操作函数实现，这里封装成一个函数 LED5()，就增强了程序的可读性。该函数定义在编写的 led.h 文件中，led.h 文件中代码如下：

```
17.  #ifndef __LED_H
18.  #define  __LED_H

19.  #include "stm32f10x.h"
20.  #define ON 0
21.  #define OFF 1
22.  #define LED5(a)   if (a)   \
23.             GPIO_SetBits(GPIOB,GPIO_Pin_5);\
24.             else     \
25.             GPIO_ResetBits(GPIOB,GPIO_Pin_5)
26.  #define LED6(a)   if (a)   \
27.             GPIO_SetBits(GPIOB,GPIO_Pin_6);\
28.             else     \
29.             GPIO_ResetBits(GPIOB,GPIO_Pin_6)
30.  #define LED7(a)   if (a)   \
31.             GPIO_SetBits(GPIOB,GPIO_Pin_7);\
32.             else     \
33.             GPIO_ResetBits(GPIOB,GPIO_Pin_7)
34.  void LED_GPIO_Config(void);
35.  #endif
```

第 23～25 行代码实现了 LED 小灯的亮灭，是通过位操作函数 GPIO_SetBits(GPIOB，GPIO_Pin_5) 和 GPIO_ResetBits(GPIOB,GPIO_Pin_5)实现的。同理，第 26～33 行实现

了另外 2 个 LED 小灯的控制，这么做的好处是在主程序中代码可读性强，程序也便于维护。

第 8 行代码初始化按键操作子程序，具体代码为第 36～44 行。

第 11 行代码按键识别程序，检测 PA0 是否被按下，如果被按下，LED5 小灯熄灭。具体实现过程为第 45～60 行代码。

按键配置代码如下：

```
36.  void Key_GPIO_Config(void)
37.  {
38.      GPIO_InitTypeDef GPIO_InitStructure;
39.      RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA,ENABLE);
40.      GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0;
41.      GPIO_InitStructure.GPIO_Speed = GPIO_Speed_10MHz;
42.      GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPU;
43.      GPIO_Init(GPIOA, &GPIO_InitStructure);
44.  }
```

按键初始化与 LED 初始化类似，其中第 42 行代码设置 PA 口为上拉输入模式。

按键识别代码如下：

```
45.  uint8_t Key_Scan(GPIO_TypeDef * GPIOx,u16 GPIO_Pin)
46.  {
47.      if(GPIO_ReadInputDataBit(GPIOx,GPIO_Pin) == KEY_ON)
48.      {
49.        Delay(10000);
50.          if(GPIO_ReadInputDataBit(GPIOx,GPIO_Pin) == KEY_ON)
51.          {
52.             while(GPIO_ReadInputDataBit(GPIOx,GPIO_Pin) == KEY_ON);
53.                return KEY_ON;
54.          }
55.          else
56.             return KEY_OFF;
57.      }
58.      else
59.           return KEY_OFF;
60.  }
```

第 47 行代码利用 GPIO_ReadInputDataBit()函数读取输入数据，若从相应引脚读取的数据等于(KEY_ON)，为低电平，表明可能有按键按下，调用延时函数；否则返回 KEY_OFF，表示按键没有被按下。

第 50 行代码，延时之后再次利用 GPIO_ReadInputDataBit()函数读取输入数据，若依然为低电平，则表明确实有按键被按下；否则返回 KEY_OFF，表示没有按键被按下。

第 52 行代码，循环调用 GPIO_ReadInputDataBit()函数(见表 3-18)，一直检测按键的电平，直至按键被释放。释放后，返回表示按键被按下的标志 KEY_ON。

表 3-18　函数 GPIO_ReadInputDataBit

函　数　名	GPIO_ReadInputDataBit
函数原型	u8 GPIO_ReadInputDataBit(GPIO_TypeDef * GPIOx, u16 GPIO_Pin)
功能描述	读取指定端口引脚的输入

续表

函　数　名	GPIO_ReadInputDataBit
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：待读取的端口位
输出参数	无
返回值	输入端口引脚值
先决条件	无
被调用函数	无

3.10 本章小结

本章通过流水灯的例子进一步强化通过 MDK 建立工程的方法，配置的详细步骤。采用两种方法编程：寄存器法和库函数法。两种方法控制 I/O 的步骤和方法一致，寄存器法有助于帮助学习者更好地了解 STM32 的内部结构，而库函数法能让学习者更方便地编程。不管哪种方法，包括后面的应用，始终要遵循配置 I/O 的如下步骤。

（1）配置时钟。

（2）配置 I/O 工作模式，常用的工作模式为推挽输出模式和上拉输入模式。

（3）操作 I/O 端口（赋值或者读入数据）。

后期的串口操作，定时器操作，DMA 操作等部分内容，均是在这三步的基础上的扩展。这三步是配置的核心。

配置时钟，核心是了解时钟树，常用的功能主要用 APB2 和 APB1 时钟树相关联的库函数，常规操作 I/O 端口工作模式均用推挽输出模式和上拉输入模式，涉及总线操作用浮空模式，涉及模拟输入采用模拟输入模式，对于 I/O 端口的操作和一般微处理器操作没有不同，通过数码管操作实例和简单按键操作实例进一步强化 I/O 的使用步骤和方法。

3.11 习题

（1）简述通过 MDK 建立工程的方法。

（2）直接库函数操作和寄存器操作有哪些区别？

（3）分析 STM32 的时钟数结构。

（4）通用 GPIO 的初始化过程是什么？

（5）采用查询法编写按键识别代码。

（6）编写复杂流水灯程序，灯光闪烁的频率为一首歌曲的频率。

（7）利用数码管编写程序，动态显示 12345678。

（8）利用数码管和简单按键操作实例编写万年历代码，如 23-11-30，表示 2023 年 11 月 30 日，通过按键可以切换显示 10-30-45，表示 10 点 30 分 45 秒，可以通过按键修改时钟。