第5章 通用输入/输出接口

通用输入/输出(General-Purpose Input/Output,GPIO)接口是处理器中最简单也是最重要的接口配置,通常也表示为I/O接口。即便如此,GPIO接口也有各种各样的类型和配置选项,即有输入、输出、上拉、下拉以及推挽功能等。软硬件工程师几乎天天和它们打交道,因此深入了解其中的配置,特别是它们的多功能配置和使用方法至关重要。GPIO接口最常见的用途是控制电子设备。例如,无论你期望构建自己的机械臂还是DIY气象站,GPIO接口都可以让你自定义信号,以便它们能正确地操控设备。从本章开始介绍ARM STM32系列处理器的典型外设GPIO接口。以STM32F103ZET6处理器为例,介绍其GPIO接口的工作模式、逻辑结构以及使用方法。

5.1 GPIO接口概述

在绝大多数嵌入式系统设计与应用中,都会涉及数字开关量的输入和输出,如常见的状态指示、报警输出、继电器闭合和断开信号输入判断、按钮的开或关状态的读入以及开关控制报警信息的输出等。所有这些开关量的输入和输出控制都必须通过处理器的GPIO接口来实现。随着大规模集成电路的快速发展,为了节约其输入/输出引脚数量,处理器中许多GPIO接口都有复用功能,用户可视需求进行不同的设置和取舍。本章仅介绍STM32F103ZET6处理器GPIO接口的基本输入/输出功能和设置。

STM32F103ZET6处理器拥有112根能承受TTL 5V电压的多功能双向快速引脚(也称为I/O引脚)。每16根引脚划分为一组,分别称为PA、PB、PC、PD和PE、PF、PG等GPIO接口。每组GPIO接口拥有两个32位的配置寄存器(GPIOx_CRL和GPIOx_CRH)、两个32位的数据寄存器(GPIOx_IDR和GPIOx_ODR)、一个32位的置位/复位寄存器(GPIOx_BSRR)、一个16位的复位寄存器(GPIOx_BRR)和一个32位的锁定寄存器(GPIOx_LCKR)来配置该接口的初始状态和功能。每组GPIO接口的每个引脚都有不同的工作模式,可通过软件配置相应的控制寄存器位,可设置成以下几种工作模式。

(1) 输入浮空模式:浮空模式Floating是设置处理器引脚为输入模式,它既不接高电平,也不接低电平。基于逻辑器件的内部结构,当引脚设置为输入引脚浮空模式时,等于引脚接了一个高电平。因为这种设置易受电磁干扰,所以通常不建议设置引脚为浮空模式。

(2) 输入上拉模式:上拉就是设置为高电平,即把引脚电压向上拉高,如拉到V_{cc}或V_{dd}。上拉就是将不确定的输入信号电平通过一个上拉电阻使引脚的电位钳位在高电平,同时电阻也起限流作用。使用中有强上拉和弱上拉之分,两者仅上拉电阻的阻值大小不同。

(3) 输入下拉模式:就是把引脚电压拉低到低电平,一般拉到接地GND。

(4) 模拟输入模式:模拟输入是指模拟量输入模式。数字输入是指输入以1或0表示的高或低电平信号。

(5) 开漏输出模式:输出端相当于三极管的集电极使用前悬空。要想得到确定的高电平状态,需要外接上拉电阻。此模式适用于电流型的驱动,其引脚灌流电流的能力较强。

（6）推挽式输出模式：推挽结构是指两个三极管分别受两个互补信号的控制，在一个三极管导通的时候另一个总是截止，且两个互补三极管工作在极限状态。它的输出电流较大，常用于驱动功率较大的器件。

（7）推挽式复用功能模式，引脚可设置为多功能引脚。

（8）开漏复用功能模式，引脚可设置为多功能引脚。

ARM STM32 处理器接口复用功能可以被理解为 GPIO 接口被用于第二功能时的情况，即它们可以被设置为不作为通用的 GPIO 接口使用。每个接口引脚可以由接口控制寄存器自由编程来设置。控制寄存器设置必须按 32 位字访问，不允许按半字或字节访问。如 GPIOx_BSRR 和 GPIOx_BRR 控制寄存器允许对任何 GPIO 寄存器的读/更改的独立访问，保证在读和更改访问之间产生中断 IRQ 时不会发生危险行为。ARM STM32 处理器单个 I/O 引脚的基本结构如图 5.1 所示，其基本结构包括以下几部分。

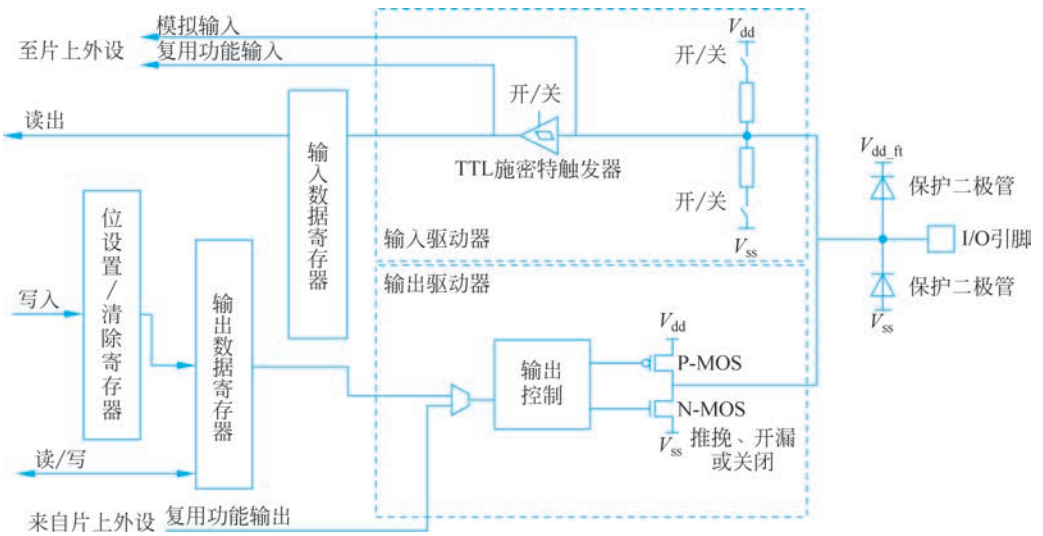

图 5.1　GPIO 接口引脚电路设计结构图

1. 输入通道

输入通道如图 5.1 中上面虚框部分所示。输入通道包括输入数据寄存器和输入驱动器。在 GPIO 接口引脚电路中设计了两个保护二极管。因为二极管导通电压降为 V_d，所以输入到输入驱动器的信号电压范围被钳位在：

$$V_{ss} - V_d < V_{in} < V_{dd} + V_d$$

由于 V_d 的导通压降不会超过 0.7V，若电源电压 V_{dd} 为 3.3V，则输入到输入驱动器的信号最低不会低于 −0.7V，最高不会高于 4V，该设计起到了对输入接口的保护作用。一般在输入信号为 0～3.3V 时，两个保护二极管都不会导通。输入驱动器中设计了上拉电阻和下拉电阻，通过开关接电源 V_{dd} 和地 V_{ss}。由控制寄存器控制此开关的动作，用来设置当 GPIO 接口引脚用作输入时，起到选择使用上拉电阻或者下拉电阻的作用。输入驱动器中的另一部件是 TTL 施密特触发器，当 GPIO 接口引脚用于开关量输入或者复用功能输入时，触发器用于对输入波形进行整形处理。

2. 输出通道

如图 5.1 所示，输出通道中包括位设置/清除寄存器、输出数据寄存器和输出驱动器。在输出开关量数据时，首先写入位设置/清除寄存器，通过读写命令进入输出数据寄存器，然后进入输出驱动的输出控制模块。如图 5.1 所示，输出控制模块可以接收开关量的输出和复用功能输出。信号通过由 P-MOS 和 N-MOS 场效应管推挽电路输出到 GPIO 接口引脚。根据需要可由软件设置控制寄存器，将 P-MOS 和 N-MOS 场效应管电路设置成推挽输出模式、开漏输出模式或关闭模式。

5.2 GPIO 接口基本功能

5.2.1 GPIO

在 GPIO 引脚设置为输出配置时，写到输出数据寄存器 GPIOx_ODR 的值输出到相应的引脚。它可以推挽模式或者开漏模式使用输出驱动器模块。在输入配置时，在每个 APB32 时钟周期捕捉并将 GPIO 接口引脚上的数据送到数据寄存器 GPIOx_IDR。每个 GPIO 接口引脚都有一个内部弱上拉和弱下拉电阻配置。在设置为输入时，内部上拉或下拉电阻可以被激活也可被断开。

当 JTAG 接口在复位期间和刚复位后，复用功能未开启，其 GPIO 接口引脚被设置成浮空输入模式。在复位后 JTAG 接口引脚被系统设置于输入上拉或下拉模式。其设置为：PA13 引脚-JTMS 置于上拉模式；PA14 引脚-JTCK 置于下拉模式；PA15 引脚-JTDI 置于上拉模式；PB4 引脚-JNTRST 置于上拉模式。

5.2.2 接口位设置或位清除

当需要对数据寄存器 GPIOx_ODR 的个别位编程时，软件不需要禁止中断。在单次 APB2 总线写操作中，可以只更改单个或多个位。接口位的置位/复位操作可通过对置位/复位寄存器 GPIOx_BSRR 和 GPIOx_BRR 中指定位直接设置实现。

5.2.3 外部中断/唤醒线

处理器的 GPIO 接口都有外部中断响应能力。在使用外部中断功能时，其接口引脚必须配置成输入模式。如何配置和控制接口中断，详见本书第 6 章内容。

5.2.4 接口复用功能及其配置

复用功能是除 GPIO 接口的基本输入/输出功能外的其他功能，在使用默认复用功能前，必须对配置寄存器的相关位进行配置和控制。对于复用功能的使用，应注意以下几点：

（1）复用输入功能，其接口引脚必须配置成输入模式，可配置为浮空、上拉或下拉，且输入引脚必须由外部驱动。

（2）复用输出功能，接口引脚必须配置成复用功能输出模式，可设置推挽模式或开漏模式。

（3）双向复用功能，接口引脚必须配置复用功能输出模式，可设置推挽模式或开漏模式。

当 GPIO 接口的某 I/O 引脚被配置为复用功能时，各模块处于以下状态：

（1）在开漏或推挽式配置中，输出缓冲器被打开。

（2）内置外设的信号驱动输出缓冲器，处于复用功能输出状态。

（3）施密特触发器的输入被激活。

（4）弱上拉和下拉电阻被禁止使用。

（5）在每个 APB2 总线时钟周期中，在 GPIO 引脚上的数据被采样到输入数据寄存器。

（6）在开漏模式下，读输入数据寄存器时可得到 GPIO 接口状态。

（7）在推挽模式下，读输出数据寄存器时可得到最后一次写入寄存器的数据值。

图 5.2 所示各 GPIO 接口引脚的复用功能配置。在读写复用功能寄存器时，用户可以把复用功能重新映射到不同的引脚使用。

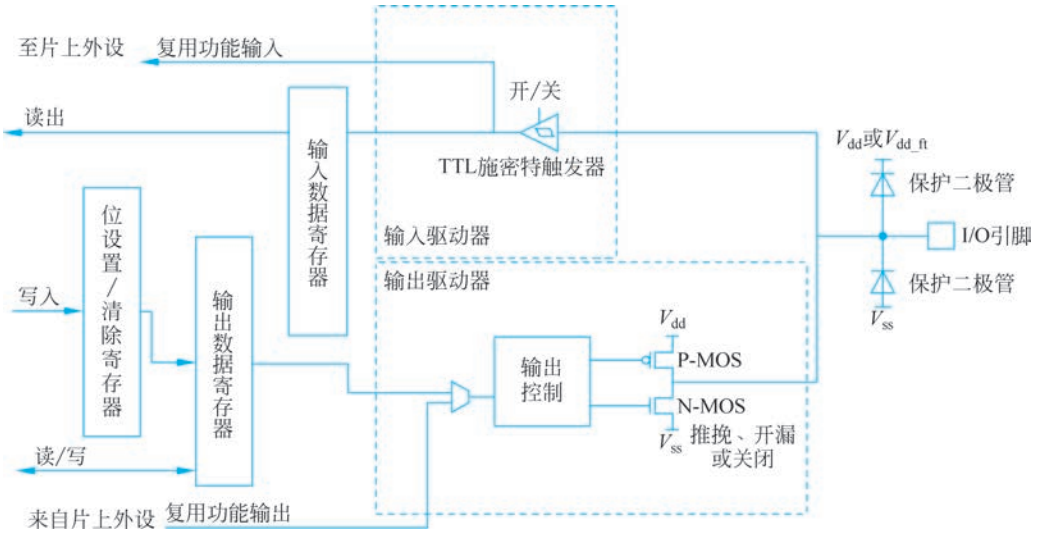

图 5.2　复用功能配置示意框图

5.2.5　软件重新映射 GPIO 复用功能

处理器设计厂商为使器件的外设 GPIO 接口复用功能的数量达到最优，允许用户通过软件配置 AFIO 寄存器灵活地把一些复用功能重新映射到器件的其他引脚上。此时，其复用功能就不再映射到它们的原始引脚上。

5.2.6　GPIO 接口的锁定机制

为防止应用程序跑飞可能引起的灾难性后果，ARM 处理器设计有锁定机制，即可以通过设置冻结 GPIO 接口的配置。即如果在一个接口位上执行了锁定 LOCK 程序，就可固定接口位模式。在重新复位之前，不能再改变接口位的配置。此功能主要用于对处理器的一些关键引脚的配置。

5.2.7 输入和输出配置

GPIO 接口引脚的输入配置框图如图 5.3 所示。当接口配置为输入模式时，其设置为：

（1）输出缓冲器被禁止使用。

（2）施密特触发输入被激活。

（3）根据输入配置（上拉、下拉或浮动）的不同，其弱上拉、下拉电阻被连接或断开。

（4）输入在引脚上的数据，在每个 APB2 总线时钟被采样送到输入数据寄存器。

（5）对输入数据寄存器的读取，可得到接口引脚接口状态。

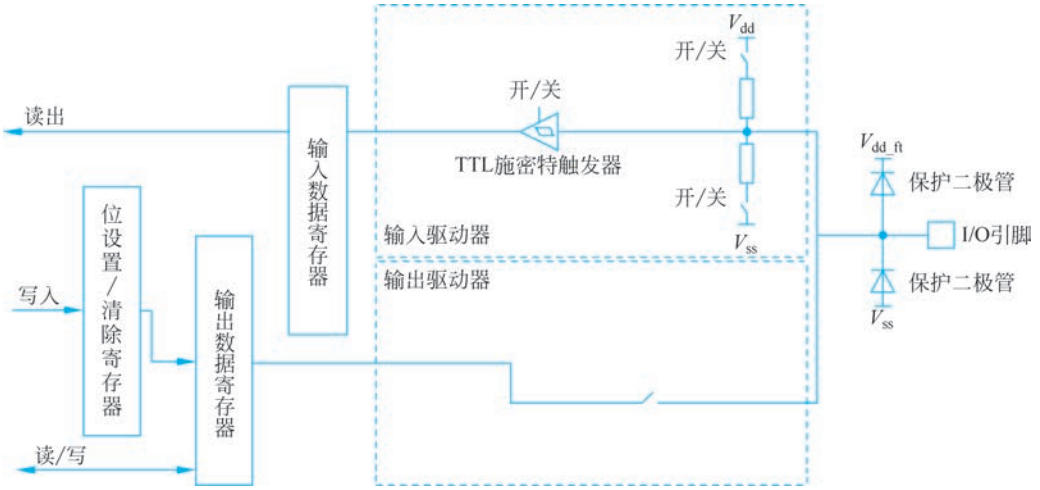

图 5.3 GPIO 接口引脚输入配置示意图

GPIO 接口引脚的输出配置框图如图 5.4 所示。当接口被配置为输出模式时，其设置和特点为：

（1）输出缓冲器被激活。

① 开漏模式：输出寄存器的数据 0 激活 N-MOS，而输出寄存器的数据 1 将接口置

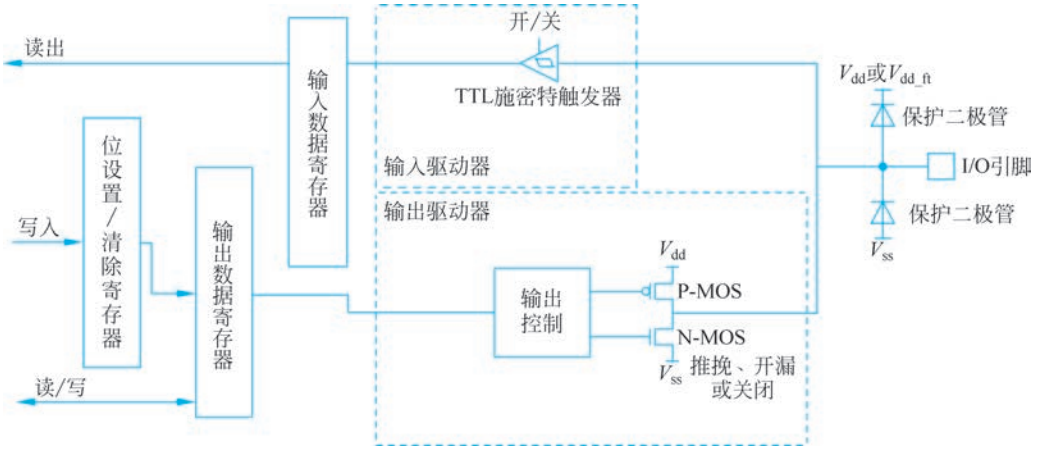

图 5.4 GPIO 接口引脚输出配置示意图

于高阻状态，即 P-MOS 从不被激活。

② 推挽模式：输出寄存器的数据 0 激活 N-MOS，而输出寄存器的数据 1 将激活 P-MOS。

（2）施密特触发器输入被激活。

（3）弱上拉和下拉电阻被禁止。

（4）出现在引脚上的数据在每个 APB2 总线时钟被采样到输入数据寄存器。

（5）在开漏模式下，对输入数据寄存器的读访问可得到 GPIO 状态。

（6）在推挽模式下，对输出数据寄存器的读访问获得最后一次写入的数据。

GPIO 接口引脚的高阻抗模拟输入配置框图如图 5.5 所示。当接口被配置为模拟输入时，其设置和特点为：

（1）输出缓冲器被禁止。

（2）禁止施密特触发器输入，使得模拟 GPIO 接口引脚上的功耗零消耗。施密特触发器输出值被强制为低电平 0。

（3）弱上拉和下拉电阻被禁止使用。

（4）在读取输入数据寄存器时，数据为低电平 0。

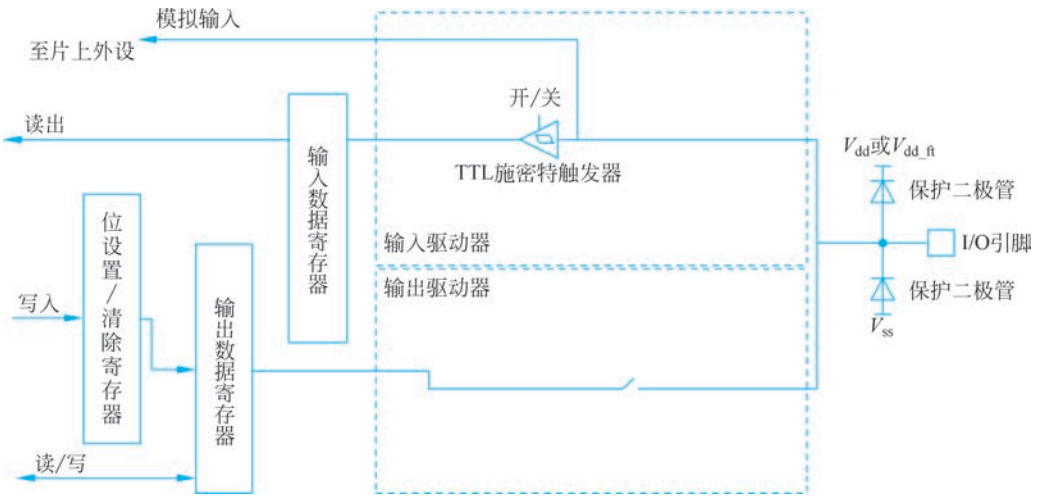

图 5.5　GPIO 接口引脚高阻抗模拟输入配置示意框图

5.3 GPIO 寄存器及其功能配置

使用 ARM STM32 处理器的应用开发，就是根据需要对多个处理器外设进行操作，实现其原始设计。学过单片机的用户都知道，对外设的控制都是通过操作该外设相关的多种寄存器来实现的。要操作和控制外设的寄存器前，必须了解此寄存器每个位的定义及功能。本章介绍 GPIO 接口相关寄存器控制位和功能。ARM STM32 处理器的存储器映射中，STM32F103ZET6 处理器的 7 个接口 PA、PB、PC、PD、PE、PF 和 PG 对应的地址范围如下：

PA 接口，0x4001 0800～0x4001 0BFF；PB 接口，0x4001 0C00～0x4001 0FFF

PC 接口，0x4001 1000～0x4001 13FF；PD 接口，0x4001 1400～0x4001 17FF

PE 接口，0x4001 1800～0x4001 1BFF；PF 接口，0x4001 1C00～0x4001 1FFF

PG 接口，0x4001 2000～0x4001 23FF

每个 GPIO 接口的第一个地址就是该接口的地址范围的首地址，如 PA 接口的首地址是 0x4001 0800，也可称为基地址。各个寄存器的地址都是以偏移量的方式给出，各寄存器的物理地址就是基地址加上其偏移量。

5.3.1 x 接口配置低寄存器 GPIOx_CRL

偏移地址：0x00，复位值：0x4444 4444。各位定义如下：

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

注意，其中 rw 说明该位是可读可写的。如果为 r，则说明该位是只读的；如果为 w，则说明该位是只写的。其他各位定义如下：

CNFy[1:0]——接口 x 配置位(y=0～7)。通过这些位配置相应的 I/O 接口工作模式。

MODEy[1:0]——接口 x 模式位(y=0～7)。通过这些位配置相应的 I/O 接口工作模式。

在输入模式下，MODEy[1:0]=00b(b 代表二进制数)时，可配置为：

00——模拟输入模式。　　01——浮空输入模式(复位后的状态)。

10——上拉/下拉输入模式。　　11——保留。

在输出模式下，MODEy[1:0]>00b 时，可配置为：

00——通用推挽输出模式。　　01——通用开漏输出模式。

10——复用功能推挽输出模式。　　11——复用功能开漏输出模式。

另外，MODEy[1:0]，对接口 x 的模式位(y=0～7)，可配置为：

00——输入模式(复位后的状态)。　　01——输出模式，最大速度为 10MHz。

10——输出模式，最大速度为 2MHz。　　11——输出模式，最大速度为 50MHz。

I/O 引脚工作模式配置如表 5.1 所示。

表 5.1　I/O 引脚工作模式配置表

配置模式		CNF1	CNF0	MODE1	MODE0	PxODR 寄存器
通用输出	推挽(Push-Pull)	0	0	01：最大输出速度为 10MHz 10：最大输出速度为 2MHz 11：最大输出速度为 50MHz		0 或 1
	开漏(Open-Drain)		1			0 或 1
复用功能输出	推挽(Push-Pull)	1	0			不使用
	开漏(Open-Drain)		1			不使用
输入	模拟输入	0	0	00		不使用
	浮空输入		1			不使用
	下拉输入	1	0			0
	上拉输入					1

5.3.2　x 接口配置高寄存器 GPIOx_CRH

偏移地址：0x04，复位值：0x4444 4444。各位定义如下：

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

CNFy[1:0]——接口 x 配置位(y=8～15)。

MODEy[1:0]——接口 x 模式位(y=8～15)。

除 y 的取值不同，接口配置高寄存器的各位定义和接口配置低寄存器类似。

5.3.3　x 接口输入/输出数据寄存器 GPIOx_IDR 和 GPIOx_ODR

GPIOx_IDR 偏移地址：0x08，复位值：0x0000 XXXX。各位定义如下：

位号	31～16	15	14	13	12	11	10	9	8
定义	保留	IDR15	IDR14	IDR13	IDR12	IDR11	IDR10	IDR9	IDR8
读写		r	r	r	r	r	r	r	r
位号	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**	
定义	IDR7	IDR6	IDR5	IDR4	IDR3	IDR2	IDR1	IDR0	
读写	r	r	r	r	r	r	r	r	

位[31:16]——保留，始终读为 0。

位[15:0]——IDRy[15:0]，接口输入数据(y=0～15)。所有位只读，并只能以字(16 位)方式的形式读出。读出的值为对应 I/O 接口引脚的状态。

GPIOx_ODR 偏移地址：0x0C，复位值：0x0000 0000。各位定义如下：

位号	31～16	15	14	13	12	11	10	9	8
定义	保留	ODR15	ODR14	ODR13	ODR12	ODR11	ODR10	ODR9	ODR8
读写		rw	rw	rw	rw	rw	rw	rw	rw
位号	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**	
定义	ODR7	ODR6	ODR5	ODR4	ODR3	ODR2	ODR1	ODR0	
读写	rw	rw	rw	rw	rw	rw	rw	rw	

位[31:16]——保留，始终读为0。

位[15:0]——ODRy[15：0]，接口输出数据(y=0～15)。

所有位可读可写，并只能以字(16位)的形式操作。

注意： 对GPIOx_BSRR(x=A,B,C,D,E,F,G)，可以分别地对各个ODR位进行独立的设置/清除。

5.3.4 接口位设置/清除寄存器 GPIOx_BSRR

地址偏移：0x10，复位值：0x0000 0000。各位定义如下：

位号	31	30	29	28	27	26	25	24
定义	BR15	BR14	BR13	BR12	BR11	BR10	BR9	BR8
读写	w	w	w	w	w	w	w	w
位号	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**
定义	BR7	BR6	BR5	BR4	BR3	BR2	BR1	BR0
读写	w	w	w	w	w	w	w	w
位号	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**
定义	BS15	BS14	BS13	BS12	BS11	BS10	BS9	BS8
读写	w	w	w	w	w	w	w	w
位号	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
定义	BS7	BS6	BS5	BS4	BS3	BS2	BS1	BS0
读写	w	w	w	w	w	w	w	w

位[31:16]——BRy，清除接口 x 的位 y(y=0～15)。所有位只能以字(16位)的形式写入。

清0表示对对应接口的ODRy位不产生影响。置1表示清除对应接口的ODRy位为0。

注意： 如果同时设置了BSy和BRy的对应位，BSy位起作用。

位[15:0]——BSy，设置接口 x 的位 y(y=0～15)。所有位只能以字(16位)的形式写入。

清0表示对对应接口的ODRy位不产生影响。置1表示设置对应接口的ODRy位为1。

5.3.5 接口位清除寄存器 GPIOx_BRR

偏移地址：0x14，复位值：0x0000 0000。各位定义如下：

位号	31～16	15	14	13	12	11	10	9	8
定义	保留	BR15	BR14	BR13	BR12	BR11	BR10	BR9	BR8
读写		w	w	w	w	w	w	w	w
位号	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**	
定义	BR7	BR6	BR5	BR4	BR3	BR2	BR1	BR0	
读写	w	w	w	w	w	w	w	w	

位[31:16]——保留。

位[15:0]——BRy,清除接口 x 的位 y(y=0～15)。所有位只能以字(16 位)的形式写入。

清 0 表示对对应的 ODRy 位不产生影响。置 1 表示清除对应的 ODRy 位为 0。

5.3.6 接口配置锁定寄存器 GPIOx_LCKR

该寄存器用来锁定接口位的配置。位[15:0]用于锁定 GPIO 接口的配置。在规定操作期间,不能改变 LCKR[15:0]。对相应的接口位执行 LOCK 序列后,在下次系统复位之前将不能再更改接口位的值。每个锁定位锁定控制寄存器(CRL,CRH)中相应的 4 位。

地址偏移:0x18,复位值:0x0000 0000。各位定义如下:

位号	31～17	16	15	14	13	12	11	10	9
定义	保留	LCKK16	LCK15	LCK14	LCK13	LCK12	LCK11	LCK10	LCK9
读写		rw	rw	rw	rw	rw	rw	rw	rw
位号	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
定义	LCK8	LCK7	LCK6	LCK5	LCK4	LCK3	LCK2	LCK1	LCK0
读写	rw	rw	rw	rw	rw	rw	rw	rw	rw

位[31:17]——保留。

位[16]——LCKK,锁键(Lock Key)。该位可随时读出,仅可通过锁键写入序列修改。

清 0 表示接口配置锁键未激活。置 1 表示接口配置锁键被激活,下次系统复位前 GPIOx_LCKR 寄存器被锁住。

锁键的写入序列:写 1→写 0→写 1→读 0→读 1。最后一个读可省略,但可以用来确认锁键已被激活。

注意:在操作锁键的写入序列时,不能改变 LCK[15:0]的值。操作锁键写入序列中的任何错误将不能激活锁键。

位[15:0]——LCKy,接口 x 的锁位 y(y=0～15)。所有位可读可写,但只能在 LCKK 位为 0 时写入。清 0 表示不锁定接口的配置,置 1 表示锁定接口的配置。

5.4 RCC 时钟模块寄存器

任何计算机硬件系统在复位后,首先都要进行初始化工作。其中时钟配置是初始化的一项重要任务。从第 3 章可看到,在 STM32F103ZET6 处理器的时钟结构中,如果要使用某个外设,就必须先使能该外设的时钟。时钟配置需要首先考虑系统时钟的来源,

即选择使用内部时钟、外部时钟或外部振荡器，以及是否需要锁相环(PLL)等。再考虑内部总线和外部总线，最后考虑外设的时钟信号。通常使用时应先倍频作为处理器的时钟，再使用由内向外分频的原则逐步设置。其时钟配置流程如图 5.6 所示。

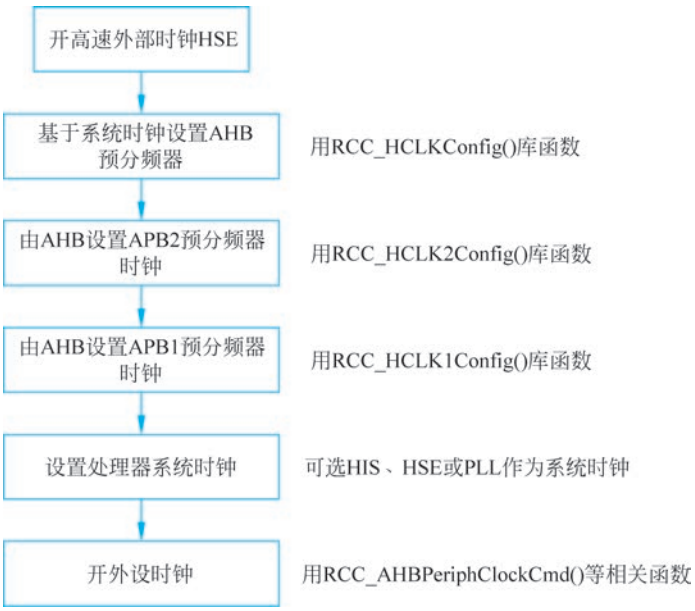

图 5.6　时钟配置流程图

时钟配置主要是对 RCC 时钟模块的各相关寄存器进行设置。RCC 时钟模块寄存器的首地址是 0x40021000，下面逐一详细介绍。

5.4.1　时钟控制和配置寄存器 RCC_CR 和 RCC_CFGR

1. 时钟控制寄存器 RCC_CR

偏移地址：0x00，复位值：0x000 XX83，X 代表未定义。

访问方式：无等待状态。字、半字和字节访问。各位定义如下：

位号	31～26	25	24	23～20	19	18	17	16
定义	保留	PLLRDY	PLLON	保留	CSSON	HSEBYP	HSERDY	HSEON
读写		r	rw		rw	rw	r	rw

位号	15～8	7	6	5	4	3	2	1	0
定义	HSICAL[7:0]	HSITRIM[4:0]					保留	HSIRDY	HSION
读写	r	rw	rw	rw	rw	rw		r	rw

位[31:26]——保留，始终读为 0。

位[25]——PLLRDY，PLL 时钟就绪标志，PLL 锁定后由硬件置 1。

0：PLL 未锁定；1：PLL 锁定。

位[24]——PLLON，PLL 使能。由软件置 1 或清 0。当进入待机和停止模式时，该位由硬件清 0。当 PLL 时钟被用作或被选择作为系统时钟时，该位不能被清 0。

0：PLL 关闭；1：PLL 使能。

位[23:20]——保留，始终读为 0。

位[19]——CSSON，时钟安全系统使能。由软件置 1 或清 0 以使能时钟监测器。

0：时钟监测器关闭；

1：如果外部 4～25MHz 振荡器就绪，时钟监测器开启。

位[18]——HSEBYP，外部高速时钟旁路。在调试模式下由软件置 1 或清 0 来旁路外部晶体振荡器。只有在外部 4～25MHz 振荡器关闭的情况下，才能写入该位。

0：外部 4～25MHz 振荡器没有旁路；

1：外部 4～25MHz 外部晶体振荡器被旁路。

位[17]——HSERDY，外部高速时钟就绪标志。由硬件置 1 来指示外部 4～25MHz 振荡器已经稳定。在 HSEON 位清 0 后，该位需要 6 个外部 4～25MHz 振荡器周期清零。

0：外部 4～25MHz 振荡器没有就绪；1：外部 4～25MHz 振荡器就绪。

位[16]——HSEON，外部高速时钟使能。由软件置 1 或清 0。当进入待机和停止模式时，该位由硬件清零，关闭 4～25MHz 外部振荡器。当外部 4～25MHz 振荡器被用作或被选择作为系统时钟时，该位不能被清 0。

0：HSE 振荡器关闭；1：HSE 振荡器开启。

位[15:8]——HSICAL[7:0]，内部高速时钟校准。在系统启动时，这些位被自动初始化。

位[7:3]——HSITRIM[4:0]，内部高速时钟调整。由软件写入来调整内部高速时钟，它们被叠加在 HSICAL[5:0]数值上。这些位在 HSICAL[7:0]的基础上，让用户可以输入一个调整数值，根据电压和文档的变化调整内部 HSI RC 振荡器的频率。默认数值为 16，可以把 HIS 调整到 8MHz±1%；每步 HSICAL 的变化调整约 40kHz。

位[2]——保留，始终读为 0。

位[1]——HSIRDY，内部高速时钟就绪标准。由硬件置 1 来指示内部 8MHz 振荡器已经稳定。

在 HSION 位清 0 后，该位需要 6 个内部 8MHz 振荡器周期清 0。

0：内部 8MHz 振荡器没有就绪；1：内部 8MHz 振荡器就绪。

位[0]——HSION，内部高速时钟使能。由软件置 1 或清 0。当从待机或停止模式返回或用作系统时钟的外部 4～16MHz 振荡器发生故障时，该位由硬件置 1 来启动内部 8MHz 的 RC 振荡器。当内部 8MHz 振荡器被直接或间接地用作或选择为系统时钟时，该位不能被清 0。

0：内部 8MHz 振荡器关闭；1：内部 8MHz 振荡器开启。

2. 时钟配置寄存器 RCC_CFGR

偏移地址：0x04，复位值：0x0000 0000。

访问方式：0～2 个等待周期，可字、半字、字节访问。

只有当访问发生在时钟切换时，才会插入 1 或 2 个等待周期。各位定义如下：

位号	31～27					26	25	24	23	22	21	20	19	18	17	16
定义	保留					MCO[2:0]			保留	USBPRE	PLLMUL[3:0]				PLLXTPRE	PLLSRC
读写						rw	rw	rw		rw	rw	rw	rw	rw	rw	rw
位号	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
定义	ADCPRE[1:0]		PPRE2[2:0]			PPRE1[2:0]			HPRE[3:0]				SWS[1:0]		SW[1:0]	
读写	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	r	r	rw	rw

位[31:27]——保留，始终读为0。

位[26:24]——MCO，微控制器时钟输出(Microcontroller Clock Output)。由软件置1或清0。

0xx：没有时钟输出；

100：系统时钟(SYSCLK)输出；

101：内部RC振荡器时钟(HSI，8MHz)输出；

110：外部振荡器时钟(HSE，4～25MHz)输出；

111：PLL时钟2分频后输出。

注意：该时钟输出在启动和切换MCO时钟源时可能会被截断。在系统时钟作为输出至MCO引脚时，请保证输出时钟频率不超过50MHz(I/O引脚最高频率)。

位[22]——USBPRE：USB预分频。由软件置1或清0来产生48MHz的USB时钟。在RCC_APB1ENR寄存器中使能USB时钟之前，必须保证该位已经有效。如果USB被使能，该位不能被清0。

0：PLL时钟1.5倍分频作为USB时钟；

1：PLL时钟直接作为USB时钟。

位[21:18]——PLLMUL，PLL倍频系数。由软件设置来确定PLL倍频系数。只有在PLL关闭的情况下才可被写入。注意PLL的输出频率不能超过72MHz。

0000：PLL2倍频输出；　　0001：PLL3倍频输出；

0010：PLL4倍频输出；　　0011：PLL5倍频输出；

0100：PLL6倍频输出；　　0101：PLL7倍频输出；

0110：PLL8倍频输出；　　0111：PLL9倍频输出；

1000：PLL10倍频输出；　　1001：PLL11倍频输出；

1010：PLL12倍频输出；　　1011：PLL13倍频输出；

1100：PLL14倍频输出；　　1101：PLL15倍频输出；

1110：PLL16倍频输出；　　1111：PLL16倍频输出。

位[17]——PLLXTPRE：HSE分频器作为PLL输入。由软件置1或清0来分频HSE后作为PLL输入时钟。只能在关闭PLL时才能写入此位。

0：HSE不分频；　　1：HSE 2分频。

位[16]——PLLSRC：PLL输入时钟源。由软件置1或清0来选择PLL输入时钟源。只能在关闭PLL时才能写入此位。

0：HSI 振荡器时钟经 2 分频后作为 PLL 输入时钟；

1：HSE 时钟作为 PLL 输入时钟。

位[15:14]——ADCPRE[1:0]，ADC 预分频由软件置 1 或清 0 来确定 ADC 时钟频率。

00：PCLK2 2 分频后作为 ADC 时钟；

01：PCLK2 4 分频后作为 ADC 时钟；

10：PCLK2 6 分频后作为 ADC 时钟；

11：PCLK2 8 分频后作为 ADC 时钟。

位[13:11]——PPRE2[2:0]，高速 APB 预分频(APB2)。由软件置 1 或清 0 来控制 APB2 时钟(PCLK2)的预分频系数。

0xx：HCLK 不分频；　　100：HCLK 2 分频；

101：HCLK 4 分频；　　110：HCLK 8 分频；

111：HCLK 16 分频。

位[10:8]——PPRE1[2:0]，低速 APB 预分频(APB1)。由软件置 1 或清 0 来控制低速 APB1 时钟(PCLK1)的预分频系数。注意：软件必须保证 APB1 时钟频率不超过 36MHz。

0xx：HCLK 不分频；　　100：HCLK 2 分频；

101：HCLK 4 分频；　　110：HCLK 8 分频；

111：HCLK 16 分频。

位[7:4]——HPRE[3:0]，AHB 预分频。由软件置 1 或清 0 来控制 AHB 时钟的预分频系数。

0xxx：SYSCLK 不分频；　　1000：SYSCLK 2 分频；

1001：SYSCLK 4 分频；　　1010：SYSCLK 8 分频；

1011：SYSCLK 16 分频；　　1100：SYSCLK 64 分频；

1101：SYSCLK 128 分频；　　1110：SYSCLK 256 分频；

1111：SYSCLK 512 分频。

位[3:2]——SWS[1:0]，系统时钟切换状态。由硬件置 1 或清 0 来选择系统时钟的时钟源。

00：HSI 作为系统时钟；　　01：HSE 作为系统时钟；

10：PLL 输出作为系统时钟；　　11：不可用。

位[1:0]——SW[1:0]，系统时钟切换。由软件置 1 或清 0 来选择系统时钟源。从停止或待机模式中返回时，或者直接或间接作为系统时钟的 HSE 在出现故障时，由硬件强制选择 HSI 作为系统时钟。

00：HSI 作为系统时钟；　　01：HSE 作为系统时钟；

10：PLL 输出作为系统时钟；　　11：不可用。

5.4.2　时钟中断寄存器 RCC_CIR

偏移地址：0x08，复位值：0x0000 0000。

访问方式：无等待周期。可字、半字、字节访问。各位定义如下：

位号	31～24							23	22	21	20	19	18	17	16
定义	保留							CSSC	保留		PLLRDYC	HSERDYC	HISRDYC	LSERDYC	LSIRDYC
读写								w			w	w	w	w	w
位号	15～13	12	11	10	9	8	7	6	5	4	3	2	1	0	
定义	保留	PLLRDYIE	HSERDYIE	HISRDYIE	LSERDYIE	LSIRDYIE	CSSF	保留		PLLRDYF	HSERDYF	HISRDYF	LSERDYF	LSIRDYF	
读写		rw	rw	rw	rw	rw	r			r	r	r	r	r	

位[31:24]——保留，始终读为0。

位[23]——CSSC(Clock Security System interrupt Clear)，清除始终安全系统中断。由软件置1来清除CSSF安全系统中断标志位CSSF。

0：无作用；　　1：清除CSSF安全系统中断标志位。

位[22:21]——保留，始终读为0。

位[20]——PLLRDYC，清除PLL就绪中断。由软件置1来清除PLL中断就绪标志位PLLRDYE。

0：无作用；　　1：清除PLL就绪中断标志位PLLRDYE。

位[19]——HSERDYC，清除HSE就绪中断。由软件置1来清除HSE就绪中断标志位HSERDYF。

0：无作用；　　1：清除HSE就绪中断标志位HSERDYF。

位[18]——HSIRDYC，清除HSI就绪中断。由软件置1来清除HSI就绪中断标志位HSIRDYF。

0：无作用；　　1：清除HSI就绪中断标志位HSIRDYF。

位[17]——LSERDYC，清除LSE就绪中断。由软件置1来清除LSE就绪中断标志位LSERDYF。

0：无作用；　　1：清除LSE就绪中断标志位LSERDYF。

位[16]——LSIRDYC，清除LSI就绪中断。由软件置1来清除LSI就绪中断标志位LSIRDYF。

0：无作用；　　1：清除LSI就绪中断标志位LSIRDYF。

位[15:13]——保留，始终读为0。

位[12]——PLLRDYIE，PLL就绪中断使能。由软件置1或清0来使能或关闭PLL就绪中断。

0：PLL就绪中断关闭；　　1：PLL就绪中断使能。

位[11]——HSERDYIE，HSE就绪中断使能。由软件置1或清0来使能或关闭外部4～16MHz振荡器就绪中断。

0：HSE就绪中断关闭；　　1：HSE就绪中断使能。

位[10]——HSIRDYIE，HSI就绪中断使能。由软件置1或清0来使能或关闭内部8MHz RC振荡器就绪中断。

0：HSI 就绪中断关闭；　　1：HSI 就绪中断使能。

位[9]——LSERDYIE，LSE 就绪中断使能。由软件置 1 或清 0 来使能或关闭外部 32kHz RC 振荡器就绪中断。

0：LSE 就绪中断关闭；　　1：LSE 就绪中断使能。

位[8]——LSIRDYIE，LSI 就绪中断使能。由软件置 1 或清 0 来使能或关闭内部 40kHz RC 振荡器就绪中断。

0：LSI 就绪中断关闭；　　1：LSI 就绪中断使能。

位[7]——CSSF，时钟安全系统中断标志。在外部 4～16MHz 振荡器始终出现故障时，由硬件置 1。由软件通过置 1CSSC 位来清除。

0：无 HSE 时钟失效产生的安全系统中断；

1：HSE 时钟失效导致了时钟安全系统中断。

位[6:5]——保留，始终读为 0。

位[4]——PLLRDYF，PLL 就绪中断标志。在 PLL 就绪且 PLLRDYIE 位被置 1 时，由硬件置 1。由软件通过置 1PLLRDYC 位来清除。

0：无 PLL 上锁产生的时钟就绪中断；

1：PLL 上锁导致时钟就绪中断。

位[3]——HSERDYF，HSE 就绪中断标志。在外部低速时钟就绪且 HSERDYIE 位被置 1 时，由硬件置 1。由软件通过置 1HSERDYC 位来清除。

0：无外部 4～16MHz 振荡器产生的时钟就绪中断；

1：外部 4～16MHz 振荡器导致时钟就绪中断。

位[2]——HSIRDYF，HSI 就绪中断标志。在内部高速时钟就绪且 HSIRDYIE 位被置 1 时，由硬件置 1。由软件通过将 HSIRDYC 位置 1 来清除。

0：无内部 8MHz RC 振荡器产生的时钟就绪中断；

1：内部 8MHz RC 振荡器导致时钟就绪中断。

位[1]——LSERDYF，LSE 就绪中断标志。在外部低速时钟就绪且 LSERDYIE 位被置 1 时，由硬件置 1。由软件通过置 1LSERDYC 位来清除。

0：无外部 32kHz 振荡器产生的时钟就绪中断；

1：外部 32kHz 振荡器导致的时钟就绪中断。

位[0]——LSIRDYF，LSI 就绪中断标志。在内部时钟就绪且 LSIRDYIE 位被置 1 时，由硬件置 1。由软件通过置 1LSIRDYC 位来清除。

0：无内部 40kHz RC 振荡器产生的时钟就绪中断；

1：内部 40kHz RC 振荡器导致的时钟就绪中断。

5.4.3　APB1/2 外设复位寄存器 RCC_APB1RSTR 和 RCC_APB2RSTR

1. 外设复位寄存器 RCC_APB1RSTR

偏移地址：0x10，复位值：0x0000 0000。

访问方式：无等待周期，可字、半字、字节访问。

所有可设置的位都由软件置1或清0。各位定义如下：

位号	31～30		29	28	27	26	25	24
定义	保留		DACRST	PWRRST	BKPRST	保留	CANRST	保留
读写			rw	rw	rw		rw	
位号	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**
定义	USBRST	I2C2RST	I2C1RST	UART5RST	UART4RST	USART3RST	USART2RST	保留
读写	rw	rw	rw	rw	rw	rw	rw	
位号	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**
定义	SPI3RST	SPI2RST	保留		WWDGRST	保留		
读写	rw	rw			rw			
位号	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
定义			TIM7RST	TIM6RST	TIM5RST	TIM4RST	TIM3RST	TIM2RST
读写			rw	rw	rw	rw	rw	rw

位[31:30]——保留，始终读为0。

位[29]——DACRST，DAC接口复位。0：无作用；1：复位DAC接口。

位[28]——PWRRST，电源接口复位。0：无作用；1：复位电源接口。

位[27]——BKPRST，备份接口复位。0：无作用；1：复位备份接口。

位[26]——保留，始终读为0。

位[25]——CANRST，CAN复位。0：无作用；1：复位CAN。

位[24]——保留，始终读为0。

位[23]——USBRST，USB复位。0：无作用；1：复位USB。

位[22]——I2C2RST，I2C 2复位。0：无作用；1：复位I2C 2。

位[21]——I2C1RST，I2C 1复位。0：无作用；1：复位I2C 1。

位[20]——UART5RST，UART5复位。0：无作用；1：复位UART5。

位[19]——UART4RST，UART4复位。0：无作用；1：复位UART4。

位[18]——USART3RST，USART3复位。0：无作用；1：复位USART3。

位[17]——USART2RST，USART2复位。0：无作用；1：复位USART2。

位[16]——保留，始终读为0。

位[15]——SPI3RST，SPI3复位。0：无作用；1：复位SPI3。

位[14]——SPI2RST，SPI2复位。0：无作用；1：复位SP12。

位[13:12]——保留，始终读为0。

位[11]——WWDGRST，窗口看门狗复位。0：无作用；1：复位窗口看门狗。

位[10:6]——保留，始终读为0。

位[5]——TIM7RST，定时器7复位。0：无作用；1：复位TIM7定时器。

位[4]——TIM6RST，定时器6复位。0：无作用；1：复位TIM6定时器。

位[3]——TIM5RST，定时器5复位。0：无作用；1：复位TIM5定时器。

位[2]——TIM4RST，定时器4复位。0：无作用；1：复位TIM4定时器。

位[1]——TIM3RST，定时器3复位。0：无作用；1：复位TIM3定时器。

位[0]——TIM2RST，定时器 2 复位。0：无作用；1：复位 TIM2 定时器。

2. 外设复位寄存器 RCC_APB2RSTR

偏移地址：0x0C，复位值：0x0000 0000。

访问方式：无等待周期，可字、半字和字节访问。

所有可设置的位都可由软件置 1 或清 0。各位定义如下：

位号	31～16	15	14	13	12	11	10	9	8
定义	保留	ADC3 RST	USART 1RST	TIM8 RST	SPI1 RST	TIM1 RST	ADC2 RST	ADC1 RST	IOPG RST
读写		rw	rw	rw	rw	rw	rw	rw	rw
位号	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**	
定义	IOPF RST	IOPE RST	IOPD RST	IOPC RST	IOPB RST	IOPA RST	保留	AFIO RST	
读写	rw	rw	rw	rw	rw	rw		rw	

位[31:16]——保留，始终读为 0。

位[15]——ADC3RST，ADC3 接口复位。0：无作用；1：复位 ADC3 接口。

位[14]——USART1RST，USART1 复位。0：无作用；1：复位 USART1。

位[13]——TIM8RST，TIM8 定时器复位。0：无作用；1：复位 TIM8 定时器。

位[12]——SPI1RST，SPI1 复位。0：无作用；1：复位 SPI1。

位[11]——TIM1RST，TIM1 定时器复位。0：无作用；1：复位 TIM1 定时器。

位[10]——ADC2RST，ADC2 接口复位。0：无作用；1：复位 ADC2 接口。

位[9]——ADC1RST，ADC1 接口复位。0：无作用；1：复位 ADC1 接口。

位[8]——IOPGRST，I/O 接口 G 复位。0：无作用；1：复位 I/O 接口 G。

位[7]——IOPFRST，I/O 接口 F 复位。0：无作用；1：复位 I/O 接口 F。

位[6]——IOPERST，I/O 接口 E 复位。0：无作用；1：复位 I/O 接口 E。

位[5]——IOPDRST，I/O 接口 D 复位。0：无作用；1：复位 I/O 接口 D。

位[4]——IOPCRST，I/O 接口 C 复位。0：无作用；1：复位 I/O 接口 C。

位[3]——IOPBRST，I/O 接口 B 复位。0：无作用；1：复位 I/O 接口 B。

位[2]——IOPARST，I/O 接口 A 复位。0：无作用；1：复位 I/O 接口 A。

位[1]——保留，始终读为 0。

位[0]——AFIORST，辅助功能 I/O 复位。0：无作用；1：复位辅助功能。

5.4.4 AHB 外设时钟使能寄存器 RCC_AHBENR

偏移地址：0x14，复位值：0x0000 0014。

访问方式：无等待周期，字、半字、字节访问。

所有可设置的位都由软件置 1 或清零。各位定义如下：

位号	31～11	10	9	8	7	6	5	4	3	2	1	0
定义	保留	SDIOEN	保留	FSMCEN	保留	CRCEN	保留	FLITFEN	保留	ISRAMEN	DMA2EN	DMA1EN
读写		rw		rw		rw		rw		rw	rw	rw

位[31:11]——保留，始终读为0。

位[10]——SDIOEN，SDIO时钟使能。0：SDIO时钟关闭；1：SDIO时钟开启。

位[9]——保留，始终读为0。

位[8]——FSMCEN，FSMC时钟使能。0：FSMC时钟关闭；1：FSMC时钟开启。

位[7]——保留，始终读为0。

位[6]——CRCEN，CRC时钟使能。0：CRC时钟关闭；1：CRC时钟开启。

位[5]——保留，始终读为0。

位[4]——FLITFEN，闪存接口电路时钟使能。

0：睡眠模式时闪存接口电路时钟关闭；

1：睡眠模式时闪存接口电路时钟开启。

位[3]——保留，始终读为0。

位[2]——SRAMEN，SRAM时钟使能。

0：睡眠模式时SRAM时钟关闭；　　1：睡眠模式时SRAM时钟开启。

位[1]——DMA2EN，DMA2时钟使能。

0：DMA2时钟关闭；　　1：DMA2时钟开启。

位[0]——DMA1EN，DMA1时钟使能。

0：DMA1时钟关闭；　　1：DMA1时钟开启。

5.4.5　APB1/2外设时钟使能寄存器RCC_APB1ENR和RCC_APB2ENR

1. 外设时钟使能寄存器RCC_APB1ENR

偏移地址：0x1C，复位值：0x0000 0000。

访问方式：通常无访问等待周期，字、半字、字节访问，但APB1总线上的外设被访问时，将插入等待状态直到APB1的外设访问结束。

所有可访问的位都可由软件置1或清0。各位定义如下：

位号	31～30		29	28	27	26	25	24	23	22	21	20	19	18	17	16
定义	保留		DACEN	PWREN	BKPEN	保留	CANEN	保留	USBEN	I2C2EN	I2C1EN	UART5EN	UART4EN	USART3EN	USART2EN	保留
读写			rw	rw	rw		rw		rw	rw	rw	rw	rw	rw	rw	
位号	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
定义	SPI3EN	SPI2EN	保留		WWDGEN	保留					TIM7EN	TIM6EN	TIM5EN	TIM4EN	TIM3EN	TIM2EN
读写	rw	rw			rw						rw	rw	rw	rw	rw	rw

位[31:30]——保留，始终读为0。

位[29]——DACEN，DAC接口时钟使能。

0：DAC 接口时钟关闭；　　1：DAC 接口时钟开启。

位[28]——PWREN，电源接口时钟使能。

0：电源接口时钟关闭；　　1：电源接口时钟开启。

位[27]——BKPEN，备份接口时钟使能。0：备份接口时钟关闭；1：备份接口时钟开启。

位[26]——保留，始终读为 0。

位[25]——CANEN，CAN 时钟使能。0：CAN 时钟关闭；1：CAN 时钟开启。

位[24]——保留，始终读为 0。

位[23]——USBEN，USB 时钟使能。0：USB 时钟关闭；1：USB 时钟开启。

位[22]——I2C2EN，I2C 2 时钟使能。0：I2C 2 时钟关闭；1：I2C 2 时钟开启。

位[21]——I2C1EN，I2C 1 时钟使能。0：I2C 1 时钟关闭；1：I2C 1 时钟开启。

位[20]——UART5EN，UART5 时钟使能。

0：UART5 时钟关闭；　　1：UART5 时钟开启。

位[19]——UART4EN，UART4 时钟使能。

0：UART4 时钟关闭；　　1：UART4 时钟开启。

位[18]——USART3EN，USART3 时钟使能。

0：USART3 时钟关闭；　　1：USART3 时钟开启。

位[17]——USART2EN，USART2 时钟使能。

0：USART2 时钟关闭；　　1：USART2 时钟开启。

位[16]——保留，始终读为 0。

位[15]——SPI3EN，SPI3 时钟使能。0：SPI3 时钟关闭；1：SPI3 时钟开启。

位[14]——SPI2EN，SPI2 时钟使能。0：SPI2 时钟关闭；1：SPI2 时钟开启。

位[13:12]——保留，始终读为 0。

位[11]——WWDGEN，窗口看门狗时钟使能。

0：窗口看门狗时钟关闭；　　1：窗口看门狗时钟开启。

位[10:6]——保留，始终读为 0。

位[5]——TIM7EN，定时器 7 时钟使能。0：定时器 7 时钟关闭；1：定时器 7 时钟开启。

位[4]——TIM6EN，定时器 6 时钟使能。0：定时器 6 时钟关闭；1：定时器 6 时钟开启。

位[3]——TIM5EN，定时器 5 时钟使能。0：定时器 5 时钟关闭；1：定时器 5 时钟开启。

位[2]——TIM4EN，定时器 4 时钟使能。0：定时器 4 时钟关闭；1：定时器 4 时钟开启。

位[1]——TIM3EN，定时器 3 时钟使能。0：定时器 3 时钟关闭；1：定时器 3 时钟开启。

位[0]——TIM2EN，定时器 2 时钟使能。0：定时器 2 时钟关闭；1：定时器 2 时钟开启。

2. 外设时钟使能寄存器 RCC_APB2ENR

偏移地址：0x18，复位值：0x0000 000。

访问方式：通常无访问等待周期，字、半字、字节访问，但 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。

所有可设置的位都由软件置 1 或清 0。各位定义如下：

位号	31～16	15	14	13	12	11	10	9	8
定义	保留	ADC3EN	USART1EN	TIM8EN	SPI1EN	TIM1EN	ADC2EN	ADC1EN	IOPGEN
读写		rw	rw	rw	rw	rw	rw	rw	rw
位号	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**	
定义	IOPFEN	IOPEEN	IOPDEN	IOPCEN	IOPBEN	IOPAEN	保留	AFIOEN	
读写	rw	rw	rw	rw	rw	rw		rw	

位[31:16]——保留，始终读为0。

位[15]——ADC3EN，ADC3接口时钟使能。

0：ADC3接口时钟关闭；　　1：ADC3接口时钟开启。

位[14]——USART1EN，USARTI时钟使能。

0：USART1时钟关闭；　　1：USART1时钟开启。

位[13]——TIM8EN，TIM8定时器时钟使能。

0：TIM8定时器时钟关闭；　　1：TIM8定时器时钟开启。

位[12]——SPI1EN，SPI1时钟使能。0：SPI1时钟关闭；1：SPI1时钟开启。

位[11]——TIM1EN，TIM1定时器时钟使能。

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。

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。

0：ADC1接口时钟关闭；　　1：ADC1接口时钟开启。

位[8]——IOPGEN，I/O接口G时钟使能。

0：I/O接口G时钟关闭；　　1：I/O接口G时钟开启。

位[7]——IOPFEN，I/O接口F时钟使能。

0：I/O接口F时钟关闭；　　1：I/O接口F时钟开启。

位[6]——IOPEEN，I/O接口E时钟使能。

0：I/O接口E时钟关闭；　　1：I/O接口E时钟开启。

位[5]——IOPDEN，I/O接口D时钟使能。

0：I/O接口D时钟关闭；　　1：I/O接口D时钟开启。

位[4]——IOPCEN，I/O接口C时钟使能。

0：I/O接口C时钟关闭；　　1：I/O接口C时钟开启。

位[3]——IOPBEN，I/O接口B时钟使能。

0：I/O接口B时钟关闭；　　1：I/O接口B时钟开启。

位[2]——IOPAEN，I/O接口A时钟使能。

0：I/O接口A时钟关闭；　　1：I/O接口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。

0：辅助功能 I/O 时钟关闭；　　1：辅助功能 I/O 时钟开启。

5.4.6 备份域控制寄存器 RCC_BDCR

偏移地址：0x20，复位值：0x00000000。它只能由备份域复位有效复位。

访问方式：需要 0～3 个等待周期，字、半字、字节访问。各位定义如下：

位号	31～17	16	15	14～10	9	8	7～3	2	1	0
定义	保留	BDRST	RTCEN	保留	RTCSEL[1:0]		保留	LSEBYP	LSERDY	LSEON
读写		rw	rw		rw	rw		rw	r	rw

位[31:17]——保留，始终读为 0。

位[16]——BDRST，备份域软件复位，由软件置 1 或清 0。

0：复位未激活；　　1：复位整个备份域。

位[15]——RTCEN，RTC 时钟使能，由软件置 1 或清 0。

0：RTC 时将关闭；　　1：RTC 时钟开启。

位[14:10]——保留，始终读为 0。

位[9:8]——RTCSEL[1:0]，RTC 时钟源选择。由软件设置来选择 RTC 时钟源。一旦 RTC 时钟源被选定，直到下次后备域被复位，它不能再被改变。可通过设置 BDRST 位来清除。

00：无时钟；　　01：LSE 振荡器作为 RTC 时钟；

10：LSI 振荡器作为 RTC 时钟；11：HSE 振荡器在 128 分频后作为 RTC 时钟。

位[7:3]——保留，始终读为 0。

位[2]——LSEBYP，外部低速时钟振荡器旁路。在调试模式下由软件置 1 或清 0 来旁路 LSE。只有在外部 32kHz 振荡器关闭时，才能写入该位。

0：LSE 时钟未被旁路；　　1：LSE 时钟被旁路。

位[1]——LSERDY，外部低速 LSE 就绪。由硬件置 1 或清零来指示是否外部 32kHz 振荡器就绪。在 LSEON 被清 0 后，该位需要 6 个外部低速振荡器的周期才被清 0。

0：外部 32kHz 振荡器未就绪；　　1：外部 32kHz 振荡器就绪。

位[0]——LSEON，外部低速振荡器使能。由软件置 1 或清 0。

0：外部 32kHz 振荡器关闭；　　1：外部 32kHz 振荡器开启。

5.4.7 控制/状态寄存器 RCC_CSR

偏移地址：0x24，复位值：0x0C00 0000，除复位标志外由系统复位清除，复位标志只能由电源复位清除。

访问方式：需要 0～3 个等待周期，字、半字、字节访问。当连续对该寄存器进行访问时，将插入等待状态。各位定义如下：

位号	31	30	29	28	27	26	25	24	23～2	1	0
定义	LPWR RSTF	WWDG RSTF	IWDG RSTF	SFT RSTF	POR RSTF	PIN RSTF	保留	RMVF	保留	LSIRDY	LSION
读写	rw	rw	rw	rw	rw	rw		rw		rw	rw

位[31]——LPWRRSTF，低功耗复位标志。发生低功耗复位时由硬件置1；由软件写RMVF位清除。

0：无低功耗管理复位发生；　　1：发生低功耗管理复位。

位[30]——WWDGRSTF，窗口看门狗复位标志。发生窗口看门狗复位时由硬件置1；由软件写RMVF位清除。

0：无窗口看门狗复位发生；　　1：发生窗口看门狗复位。

位[29]——IWDGRSTF，独立看门狗复位标志。发生独立看门狗复位时由硬件置1；由软件通过写RMVF位清除。

0：无独立看门狗复位发生；　　1：发生独立看门狗复位。

位[28]——SFTRSTF，软件复位标志。发生软件复位时由硬件置1；由软件写RMVF位清除。

0：无软件复位发生；　　1：发生软件复位。

位[27]——PORRSTF，上电/掉电复位标志。发生上电/掉电复位时由硬件置1；由软件写RMVF位清除。

0：无上电/掉电复位发生；　　1：发生上电/掉电复位。

位[26]——PINRSTF，NRST引脚复位标志。在NRST引脚发生复位时由硬件置1；由软件写RMVF位清除。

0：无NRST引脚复位发生；　　1：发生NRST引脚复位。

位[25]——保留，读操作返回0。

位[24]——RMVF，清除复位标志。由软件置1来清除复位标志。

0：无作用；　　1：清除复位标志。

位[23:2]——保留，读操作返回0。

位[1]——LSIRDY，内部低速振荡器就绪。由硬件置1或清0来指示内部40kHz RC振荡器是否就绪。在LSION清零后，3个内部40kHz RC振荡器的周期后LSIRDY被清0。

0：内部40kHz RC振荡器时钟未就绪；　1：内部40kHz RC振荡器时钟就绪。

位[0]——LSION，内部低速振荡器使能。由软件置1或清0。

0：内部40kHz RC振荡器关闭；　　1：内部40kHz RC振荡器开启。

5.5 通用输入输出GPIO接口使用

在使用处理器GPIO接口时，可按下面流程步骤进行：

(1) 配置系统时钟，打开GPIO接口时钟。

(2) 设置 GPIO 接口的工作模式。

(3) 使用 GPIO 接口进行输入或输出操作。

在第 4 章虽然介绍了汇编语言设计,但是现今用汇编语言编写程序已经很少了,仅作为入门学习用。本章将通过具体实例,详细介绍使用 C 语言操作寄存器和固件库函数两种方式来完成应用程序的设计。

5.5.1 利用 C 语言直接操作寄存器方法访问 GPIO 方法

【例 5-1】 编写 C 语言程序,利用 C 语言直接操作外设寄存器,利用第 3 章实验平台控制系统板上,详见图 3.17 和图 3.63。连接有 GPIO PE 和 PF 接口的发光二极管共 12 个。请编写控制程序使用 GPIO,控制 11 个 LED 二极管灯同时点灭,亮 0.5s,灭 0.5s。

解: 通过查看实验系统(图 3.17)和实验板(图 3.63),如要将发光二极管点亮,需要将 SW 开关拨到 ON,给发光二极管提供电源。经过分析当连接 LED0～LED11 的 GPIO PB 接口引脚输出低电平 0 时,发光二极管亮;输出高电平 1 时,发光二极管灭。

在使用 C 语言直接操作 STM32 处理器寄存器进行开发时,应避免进行重复的系统初始化操作,如设置堆栈和中断向量表等内容。其 C 语言文件的程序内容如下:

```
# include"stm32f10x. h"                    //包含 STM32F1 系列处理器的头文件
void delay_ ms(unsigned short int Number);  //延时子函数
# define LED0 (1 << 5)                     //LED 接口定义,LED0 连接 PE8
# define LED1 (1 << 6)
# define LED2 (1 << 0)
# define LED3 (1 << 1)
# define LED4 (1 << 2)
# define LED5 (1 << 3)
# define LED6 (1 << 4)
# define LED7 (1 << 5)
# define LED8 (1 << 6)
# define LED9 (1 << 7)
# define LED10 (1 << 8)
# define LED11 (1 << 12)
# define RCC_APB2Periph_GPIOE ((uint32_ t) 0x00000040)
# define RCC_APB2Periph_GPIOF ((uint32_ t) 0x00000080)
int main(void)
{
RCC - >APB2ENR | = RCC_ APB2Periph_GPIOE   //使能 GPIOE 时钟
GPIOE - > CRL& = 0XFF0FFFFF;
GPIOE - > CRL | = 0X00300000;              //推挽输出
GPIOE - > ODR | = 1 << 5;                  //输出高(下面同理)
GPIOE - > CRL& = 0XF0FFFFFF;
GPIOE - > CRL | = 0X03000000;
GPIOE - > ODR | = 1 << 6;
RCC - >APB2ENR | = RCC_ APB2Periph_GPIOF   //使能 GPIOF 时钟
GPIOF - > CRL& = 0XFFFFFFF0;
GPIOF - > CRL | = 0X00000003;
GPIOF - > ODR | = 1 << 0;
GPIOF - > CRL& = 0XFFFFFF0F;
```

```
GPIOF->CRL |= 0X00000030;
GPIOF->ODR |= 1<<1;
GPIOF->CRL&= 0XFFFFF0FF;
GPIOF->CRL |= 0X00000300;
GPIOF->ODR |= 1<<2;
GPIOF->CRL&= 0XFFFF0FFF;
GPIOF->CRL |= 0X00003000;
GPIOF->ODR |= 1<<3;
GPIOF->CRL&= 0XFFF0FFFF;
GPIOF->CRL |= 0X00030000;
GPIOF->ODR |= 1<<4;
GPIOF->CRL&= 0XFF0FFFFF;
GPIOF->CRL |= 0X00300000;
GPIOF->ODR |= 1<<5;
GPIOF->CRL&= 0XF0FFFFFF;
GPIOF->CRL |= 0X03000000
GPIOF->ODR |= 1<<6;
GPIOF->CRL&= 0X0FFFFFFF;
GPIOF->CRL |= 0X30000000;
GPIOF->ODR |= 1<<7;
GPIOF->CRH&= 0XFFFFFFF0;
GPIOF->CRH|= 0X00000003;
GPIOF->ODR|= 1<<8;
GPIOF->CRH&= 0XFFF0FFFF;
GPIOF->CRH|= 0X00030000;
GPIOF->ODR|= 1<<12;
While(1)
{
    GPIOE->ODR|=1<<5|1<<6;          //PE.5,PE.6 输出高
    Delay_ms(500) ;
    GPIOF->ODR|=1<<0|1<<1|1<<2|1<<3|1<<4|1<<5|1<<6|1<<7|1<<8|1<<12;
                                    //PF.0～PF8,PF.12 输出高
    Delay_ms(500) ;
    GPIOE->ODR&=~(1<<5);            //LED 输出低
    GPIOE->ODR&=~(1<<6);
    GPIOF->ODR&=~(1<<0);
    GPIOF->ODR&=~(1<<1);
    GPIOF->ODR&=~(1<<2);
    GPIOF->ODR&=~(1<<3);
    GPIOF->ODR&=~(1<<4);
    GPIOF->ODR&=~(1<<5);
    GPIOF->ODR&=~(1<<6);
    GPIOF->ODR&=~(1<<7);
    GPIOF->ODR&=~(1<<8);
    GPIOF->ODR&=~(1<<12);
    delay_ ms(500);
  }
}
void delay_ ms(unsigned short int Number )
```

```
{
    unsigned int i;
    while(Number-- ){
          i= 12000; while(i-- );
    }
  }
```

由于在 stm32f10x.h 文件中定义了 STM32F10 处理器所有的外设寄存器，因此在 C 语言程序中，只要包含了这个头文件，就可以省略外设寄存器的定义，直接在用户程序代码中使用即可。读者可能发现，在应用程序中没有进行系统时钟配置。其实系统时钟配置工作已经在 SystemInit()函数中实现了，并且默认的系统时钟配置为 72MHz。查看启动文件 startup_stm32f10x_hd.s 可以发现，SystemInit()函数是在 main()函数之前调用的。因此在 main()函数中，不再需要进行系统时钟配置工作。C 语言编程比汇编语言编程相对简单，但是需要开发者查阅相关的寄存器定义进行设置。

5.5.2 利用固件库函数方法访问 GPIO 接口方法

GPIO 接口固件库函数具有多个用途，包括引脚设置、单位设置/重置、锁定机制、从接口引脚读入或者向接口引脚写入数据等用途。

5.5.2.1 GPIO 接口寄存器结构

GPIO 接口寄存器结构 CPIO_TypeDef 和 AFIO_TypeDef，在文件 stm32f10x.h 中定义如下：

```
typedef struct
{
  vu32 CRL;              //接口配置低寄存器
  vu32 CRH;              //接口配置高寄存器
  vu32 IDR;              //接口输入数据寄存器
  vu32 ODR;              //接口输出数据寄存器
  vu32 BSRR;             //接口位设置/复位寄存器
  vu32 BRR;              //接口位复位寄存器
  vu32 LCKR;             //接口配置锁定寄存器
  } GPIO_TypeDef;
typedef struct
{
  vu32 EVCR;             //事件控制寄存器
  vu32 MAPR;             //复用重映射和调试 I/O 配置寄存器
  vu32 EXTICR[4];        //外部中断线路 0～15 配置寄存器
}AFIO_TypeDef;
```

在文件结构中列出了 GPIO 接口用所有的寄存器。其中 AFIO 与 GPIO 接口的中断和事件有关，如需要可参考第 6 章。另外，在 stm32f10x.h 文件中声明了 7 个 GPIO 接口外设：

```
...
# define PERIPH_BASE ((u32 )0x40000000)
```

```
# define APB1PERIPH_BASE PERIPH BASE
# define APB2PERIPH_BASE (PERIPH_BASE + 0x10000)
# define AHBPERIPH_BASE (PERIPH_BASE + 0x20000)
...
# define AFIO_BASE (APB2PERIPH_BASE + 0x0000)
# define GPIOA_BASE (APB2PERIPH_BASE + 0x0800)
# define GPIOB_BASE (APB2PERIPH_BASE + 0x0C00)
# define GPIOC_BASE (APB2PERIPH_BASE + 0x1000)
# define GPIOD_BASE (APB2PERIPH_BASE + 0x1400)
# define GPIOE_BASE (APB2PERIPH_BASE + 0x1800)
# define GPIOE_BASE (APB2PER2PH_BASE + 0x1C00)
# define GPIOE_BASE (APB2PER2PH_BASE + 0x4001 2000)
```

5.5.2.2 GPIO 接口库函数

ARM STM32 处理器的固件库提供的 GPIO 接口库函数包括：

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 EXT 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 中的每个参数按默认值填入。

GPIO_ReadInputDataBit()——读取指定接口引脚的输入。

GPIO_ReadInputData()——读取指定的 GPIO 接口输入。

GPIO_ReadOutputDataBit()——读取指定接口引脚的输出。

GPIO_ReadOutputData()——读取指定的 GPIO 接口输出。

GPIO_SetBits()——设置指定的数据接口位。

GPIO_ResetBits()——清除指定的数据接口位。

GPIO_WriteBit()——设置或者清除指定的数据接口位。

GPIO_Write()——向指定 GPIO 接口写入数据。

GPIO_PinLockConfig()——锁定 GPIO 引脚设置寄存器。

GPIO_EventOutputCmd()——使能或者除能事件输出。

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。

下面仅介绍与本例题相关的，与普通输入/输出相关的库函数。其他库函数的使用方法，请查阅相关手册。

1. 函数 GPIO_Init()

函数原型：

```
void GPIO_Init(GPIO_TypeDef * GPIOx, GPIO_InitTypeDef * GPI0_InitStruct);
```

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、E、F 和 G，用来选择外设 GPIO；

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，GPIO_InitTypeDef 在文件 stm32f10x_gpio.h 中定义。

```
typedef struct
{
    u16 GPIO_Pin;
    GPIOSpeed_TypeDef GPIO_Speed;
    GPIOMode_TypeDef GPIO_ Mode;
} GPIO InitTypeDef;
```

1）GPIO_Pin

GPIO_Pin 用于选择待设置的 GPIO 接口引脚，使用操作符“|”可以一次选中多个引脚。可以使用如表 5.2 所示的任意组合。

表 5.2　GPIO_Pin 可取值

GPIO_Pin 可取值	描　　述	GPIO_Pin 可取值	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

2）GPIO_Speed

GPIO_Speed 用于设置选中接口引脚的速率。GPIO_Speed 可选取的值如表 5.3 所示。

表 5.3　GPIO_Speed 可取值

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

3）GPIO_Mode

GPIO_Mode 用于设置选中接口引脚的工作模式。GPIO_Mode 可选取的值如表 5.4 所示。

表 5.4　GPIO_Mode 可取值

GPIO_Mode 可取的值	描　　述	GPIO_Mode 可取的值	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，若需设置 PE0 接口位模式为推挽输出，最大速率为 50MHz，则可以使用下面的代码：

```
GPIO_InitTypeDef GPIO_InitStructure;
GPIO_InitStructure.GPIO_Pin = GPIO Pin_0;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
GPIO_Init(GPIOE, &GPIO_InitStructure);
```

2. 函数 GPIO_SetBits()

函数原型：

```
void GPIO_SetBits(GPIO_TypeDef * GPIOx, ul6 GPIO_ Pin);
```

该函数用于将指定的接口位输出高电平。

输入参数 1：GPIOx，x 可以是 A、B、C、D、E、F 和 G，用来选择 GPIO 不同接口外设；

输入参数 2：GPIO_Pin，带设置的接口位。

例如，若需将 PE0 和 PE1 接口位设置输出高电平，则可以使用下面的代码：

```
GPIO_SetBits(GPIO_Pin_0|GPIO_Pin_1);
```

3. 函数 GPIO_ResetBits()

函数原型：

```
void GPIO_ResetBits(GPIO_TypeDef * GPIOx,u16GPIO_Pin);
```

该函数用于将指定的接口位输出低电平。

输入参数 1：GPIOx，x 可以是 A、B、C、D、E、F 和 G，用来选择 GPIO 不同接口外设；

输入参数 2：GPIO_Pin，待设置的接口位。

例如，若需将 PE0 接口位设置为输出低电平，则可以使用下面的代码：

```
GPIO_ResetBits(GPIOE,GPIO_Pin_0);
```

4. 函数 GPIO_WriteBit()

函数原型：

```
void GPIO_WriteBit(GPIO_TypeDef * GPIOx,u16 GPIO_Pin,BitAction BitVal);
```

该函数用于将指定的接口位设置为高电平或者低电平。

输入参数 1：GPIOx，x 可以是 A、B、C、D、E、F 和 G，用来选择 GPIO 不同接口外设；

输入参数 2：GPIO_Pin，待设置的接口位。

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 中的一个值。

Bit_RESET：将接口位设置为低电平；

Bit_SET：将接口位设置为高电平。

例如，若需将 PE8 设置为高电平，则可以使用下面的代码：

```
GPIO_WriteBit(GPIOE, GPIO_Pin_8,Bit_SET);
```

5. 函数 GPIO_Write()

函数原型：

```
void GPIO_Write(GPIO_TypeDef * GPIOx, u16 PortVal);
```

该函数用于向指定 GPIO 接口写入数据。

输入参数 1：GPIOx，x 可以是 A、B、C、D、E、F 和 G，用来选择 GPIO 不同接口外设；

输入参数 2：PortVal，待写入接口数据寄存器的值。

例如，若需向 PE 接口写入 0x1101，则可以使用下面的代码：

```
GPIO_ Write(GPIOE,0x1101);
```

6. 函数 GPIO_ReadInputDataBit()

函数原型：

```
Uint8_t GPIO_ReadInputDataBit(GPIO_TypeDef * GPIOx,u16 GPIO_Pin);
```

该函数用于读取指定接口引脚的输入。

输入参数 1：GPIOx，x 可以是 A、B、C、D、E、F 和 G，用来选择 GPIO 不同接口外设；

输入参数 2：GPIO_Pin，待读取的接口位。

返回值：输入接口引脚值。

例如，若需读取 PA0 的状态，则可以使用下面的代码：

```
Uint8_t ReadValue;
RedValue = GPIO_ReadInputDataBit(GPIOA,GPIO_Pin_0);
```

7. 函数 GPIO_ReadInputData()

函数原型：

```
u16 GPIO_ReadInputData(GPIO_TypeDef * GPIOx);
```

该函数用于读取指定的 GPIO 接口输入。

输入参数：GPIOx，x 可以是 A、B、C、D、E、F 和 G，用来选择 GPIO 不同接口外设；

返回值：GPIO 输入数据接口值。

例如，若需读取 GPIOA 接口值，则可以使用下面的代码：

```
u16 ReadValue;
ReadValue = GPIO_ReadInputData(GPIOA);
```

8. 函数 GPIO_ReadOutputDataBit()

函数原型：

```
u8 GPIO_ReadOutputDataBit(GPIO_TypeDef * GPIOx,u16 GPIO_Pin);
```

该函数用于读取指定接口引脚的输出。

输入参数 1：GPIOx，x 可以是 A、B、C、D、E、F 和 G，用来选择 GPIO 不同接口外设；

输入参数 2：GPIO_Pin，待读取的接口位。

返回值：输出接口引脚值。

例如，若需读取 PE7 接口位的输出，则可以使用下面的代码：

```
u8 ReadValue;
ReadValue = GPIO_ReadOutputDataBit(GPIOE,GPIO_Pin_7);
```

9. 函数 GPIO_ReadOutputData()

函数原型：

```
u16 GPIO_ReadOuputData(GPIO_TypeDef * GPIOx);
```

该函数用于读取指定的 GPIO 接口输出。

输入参数 1：GPIOx，x 可以是 A、B、C、D、E、F 和 G，用来选择 GPIO 不同接口外设。

返回值：GPIO 输出数据接口值。

例如，若需读取 GPIOE 的输出数据，则可以使用下面的代码：

```
u16 ReadValue;
ReadValue = GPIO_ReadOutputData(GPIOE);
```

10. 函数 GPIO_PinLockConfig()

函数原型：

```
void GPIO_PinLockConfig(GPIO_TypeDef * GPIOx, u16 GPIO_Pin);
```

该函数用于锁定 GPIO 引脚设置寄存器。

输入参数 1：GPIOx，x 可以是 A、B、C、D、E、F 和 G，用来选择 GPIO 外设；

输入参数 2：GPIO_Pin，待锁定的接口位。

例如，若需锁定 PA0 引脚，则可以使用下面的代码：

```
GPIO_PinLockConfig(GPIOA,GPIO_Pin_0);
```

下面使用固件库函数，解决例 5-1 的设计编程问题，完成同样的功能。

【例 5-2】 利用实验平台，可详见图 3.17 和图 3.63。连接有 GPIO PE 和 PF 接口的发光二极管 LED 共 12 个。请利用固件库函数编写控制程序使用 GPIO 接口，控制 11 个 LED 灯同时点灭：亮 0.5s 和灭 0.5s 闪烁。

解： 下面介绍利用固件库函数进行 GPIO 接口应用编程的过程。

按照上面描述的过程，在创建 Manage Run_Time Envionment 对话框时，选中 Device→Startup；选中 CMSIS→CORE。选中 Device→StdPeriph Drivers→GPIO，则屏幕下方出现如图 5.7 所示的提示信息。

根据提示，选中 StdPeriph Drivers 中的 Framework 和 RCC。选中这两项后，屏幕底部不再出现提示信息，说明选择能够满足要求。单击 OK 按钮进入开发界面。此时可以看到 Project 视图中包含了如图 5.8 所示的元素。

创建 C 语言文件 ex5-2.c，并加入到 Source Group 1 中，然后输入下面的代码：

```
#include "stm32f10x.h"                          //包含 STM32F1 系列处理器的头文件
Void delay_ms(unsigned short intNumber);        //延时子函数
Int main(void)
{
```

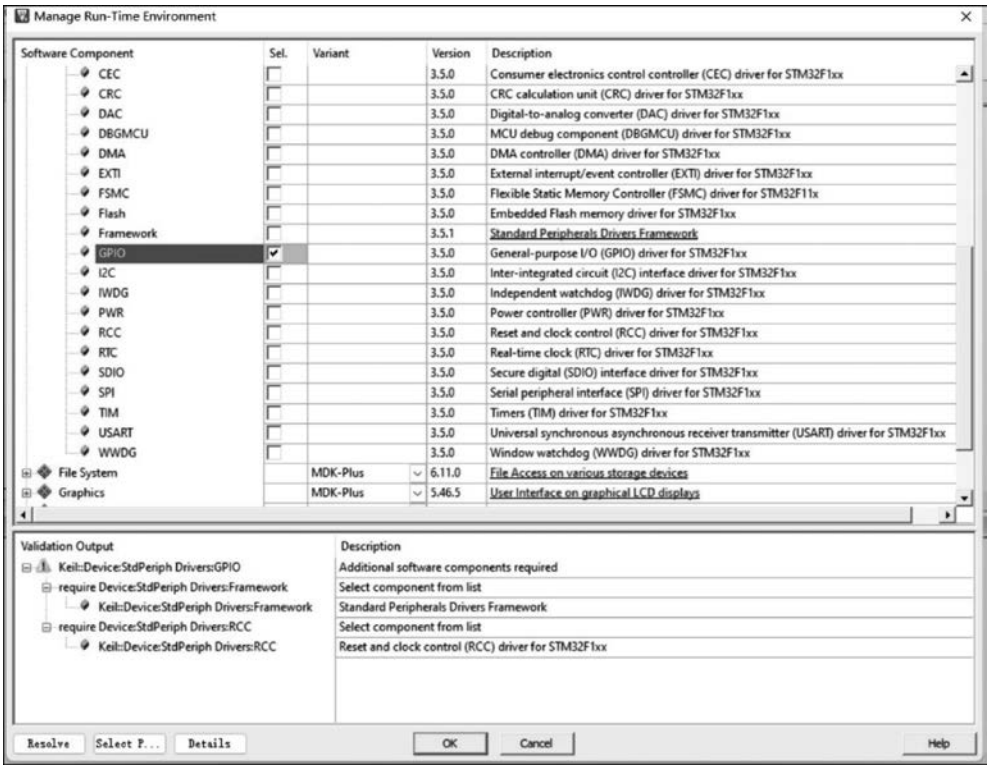

图 5.7　选中 Device→StdPeriph Drivers→GPIO 时屏幕底部的提示

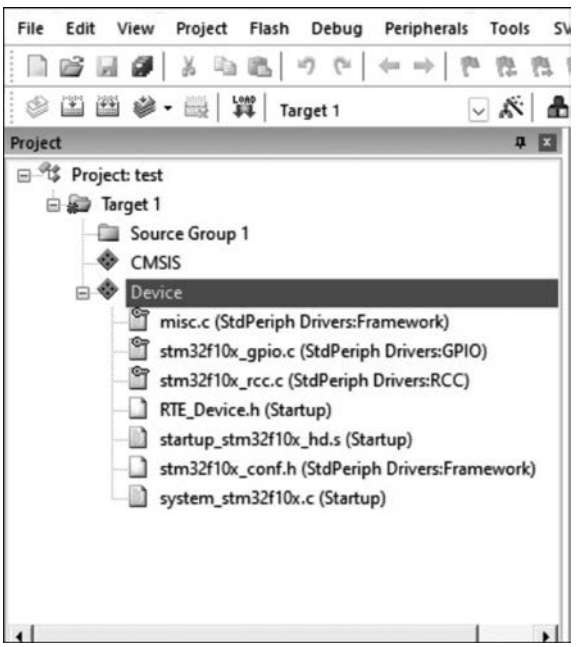

图 5.8　设置固件库函数支持后的 Project 视图

```
    GPIO_InitTypeDef GPIO_InitStructure;              //声明用于 GPIO 初始化的结构体
    RCC_APB2PeriphClockCmd(RCC_APBPeriph_GPIOE, ENABLE);     //使能 PE 接口时钟
    RCC_APB2PeriphClockCmd(RCC_APBPeriph_GPIOF, ENABLE);     //使能 PF 接口时钟
    //GPIOE//
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_5|GPIO_Pin_6;     //对 PE 的 LED 引脚进行设置
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;         //选择推挽输出模式
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;        //频率最高为 50MHz
    GPIO_Init( GPIOE, &GPIO_InitStructure) ;                 //对引脚进行配置
    //GPIOF//
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0|GPIO_Pin_1|GPIO_Pin_2|GPIO_Pin_3 GPIO_Pin_
4|GPIO_Pin_5|GPIO_Pin_6|GPIO_Pin_7|GPIO_Pin_8|GPIO_Pin_12;
    while(1)
    {
        GPIO_SetBits(GPIOE, GPIO_Pin_5|GPIO_Pin_6);          //输出高电平
        GPIO_SetBits(GPIOF, GPIO_Pin_0|GPIO_Pin_1|GPIO_Pin_2|GPIO_Pin_3|GPIO_Pin_4|
GPIO_Pin_5|GPIO_Pin_6|GPIO_Pin_7|GPIO_Pin_8|GPIO_Pin_12);
        delay_ms(500);
        GPIO_ResetBits(GPIOE, GPIO_Pin_5|GPIO_Pin_6);        //输出低电平
        GPIO_ResetBits(GPIOF, GPIO_Pin_0|GPIO_Pin_1|GPIO_Pin_2|GPIO_Pin_3|GPIO_Pin_4|
GPIO_Pin_5|GPIO_Pin_6|GPIO_Pin_7|GPIO_Pin_8|GPIO_Pin_12);
        delay_ms(500);
     }
}
void delay_ms(unsigned short int Number)
{
    unsigned int i;
    while(Number--){
    i = 12000;
    while(i--);
    }
}
```

由上述程序代码可以看出，使用固件库函数进行应用程序开发，可以让开发者不必记忆和查阅大量的寄存器名称和符号，编写的代码也更加符合人类的语言习惯，程序更加流畅，提高了应用开发效率。初学读者可以利用本书配套实验板进行反复验证。