第 1 章　STM32 MCU 简介

STM32 系列 32 位 Flash 微控制器基于 ARM Cortex-M 系列处理器，旨在为 MCU 用户提供新的开发自由度。它包括一系列 32 位产品，具有高性能、实时功能、数字信号处理、低功耗与低电压操作特性，同时还保持了集成度高和易于开发的特点。无可比拟且品种齐全的 STM32 产品基于行业标准内核，提供了大量工具和软件选项，使该系列产品成为小型项目和完整平台的理想选择。

作为一个主流的微控制器系列，STM32 满足工业、医疗和消费电子市场的各种应用需求。凭借这个产品系列，ST 在全球的 ARM Cortex-M 微控制器中处于领先地位，同时树立了嵌入式应用的里程碑。该系列最大化地集成了高性能与一流外设和低功耗、低电压工作特性，在可以接受的价格范围内提供简单的架构和易用的工具。

该系列包含 5 个产品线，它们之间引脚、外设和软件相互兼容：

- 基本型系列 STM32F101：36MHz 最高主频，具有高达 1MB 的片上闪存
- USB 基本型系列 STM32F102：48MHz 最高主频，具有全速 USB 模块
- 增强型系列 STM32F103：72MHz 最高主频，具有高达 1MB 的片上闪存，集成电机控制、USB 和 CAN 模块
- 互联型系列 STM32F105/107：72MHz 最高主频，具有以太网 MAC、CAN 及 USB 2.0 OTG 功能

本书以增强型系列 STM32F103 为核心，介绍 STM32 MCU 的设计应用。

1.1　STM32 MCU 结构

STM32 MCU 由控制单元、从属单元和总线矩阵三大部分组成，控制单元和从属单元通过总线矩阵相连接，如图 1.1 所示。

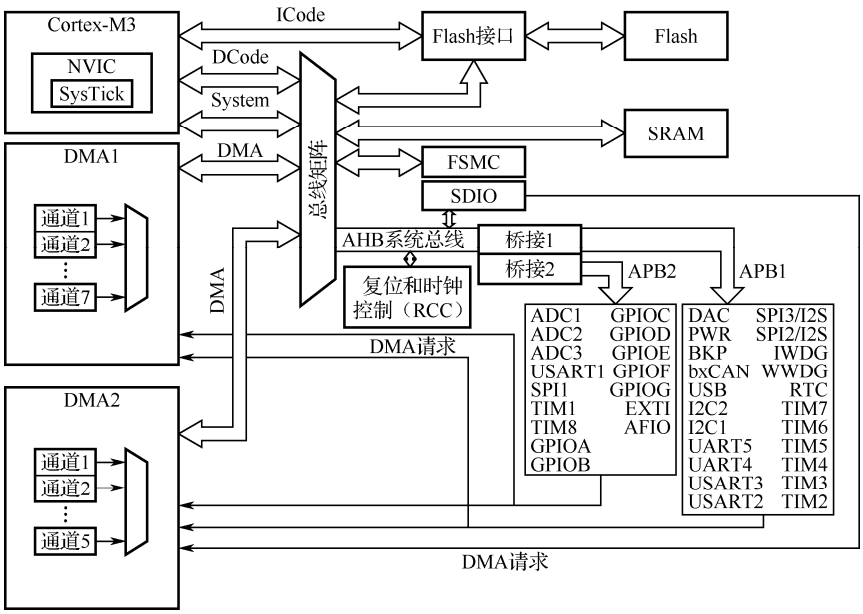


图 1.1　STM32 MCU 结构

控制单元包括 Cortex-M3 内核和两个 DMA 控制器（DMA1 和 DMA2）。其中 Cortex-M3 内核通过指令总线 ICode 从 Flash 中读指令，通过数据总线 DCode 与存储器交换数据，通过系统总线 System（设备总线）、高性能系统总线 AHB 和高级设备总线 APB 与设备交换数据。

从属单元包括存储器（Flash 和 SRAM 等）和设备（连接片外设备的接口和片内设备）。其中设备通过 AHB-APB 桥接器和总线矩阵与控制单元相连接，与 APB1 相连的是低速设备（最高频率 36MHz），与 APB2 相连的是高速设备（最高频率 72MHz）。

连接片外设备的接口有并行接口和串行接口两种，并行接口即通用 I/O 接口 GPIO，串行接口有通用同步/异步收发器接口 USART、串行设备接口 SPI、内部集成电路总线接口 I^2C、通用串行总线接口 USB 和控制器局域网络接口 CAN 等。

片内设备有定时器 TIM、模数转换器 ADC 和数模转换器 DAC 等，其中定时器包括高级控制定时器 TIM1/8、通用定时器 TIM2-5、基本定时器 TIM6/7、实时钟 RTC、独立看门狗 IWDG 和窗口看门狗 WWDG 等。

系统复位后，除 Flash 接口和 SRAM 时钟开启外，所有设备都被关闭，使用前必须设置时钟使能寄存器（RCC_APBENR）开启设备时钟。

1.2 STM32 MCU 存储器映像

STM32 MCU 的程序存储器、数据存储器和输入/输出端口寄存器被组织在同一个 4GB 的线性地址空间内，存储器映像如表 1.1 所示。

表 1.1 STM32 MCU 存储器映像表

地 址 范 围		设 备 名 称	备 注
0xE000 0000～0xE00FFFFF（1MB）		内核设备	
内核设备	0xE000E100～0xE000E4EF	NVIC（嵌套矢量中断控制）	详见表 8.2
	0xE000E010～0xE000E01F	SysTick（系统滴答定时器）	详见表 1.9
0x4000 0000～0x5FFFFFFF（512MB）		片上设备	
AHB	0x5000 0000～0x5003 FFFF	USB OTG 全速	
	0x4002 8000～0x4002 9FFF	以太网	
	0x4002 3000～0x4002 33FF	CRC	
	0x4002 2000～0x4002 23FF	Flash 接口	
	0x4002 1000～0x4002 13FF	RCC（复位和时钟控制）	详见表 1.2
	0x4002 0400～0x4002 07FF	DMA2	
	0x4002 0000～0x4002 03FF	DMA1	详见表 9.2
	0x4001 8000～0x4001 83FF	SDIO	
APB2	0x4001 3C00～0x4001 3FFF	ADC3	
	0x4001 3800～0x4001 3BFF	USART1	详见表 3.3
	0x4001 3400～0x4001 37FF	TIM8	
	0x4001 3000～0x4001 33FF	SPI1	详见表 4.2
	0x4001 2C00～0x4001 2FFF	TIM1	详见表 6.2
	0x4001 2800～0x4001 2BFF	ADC2	详见表 7.2
	0x4001 2400～0x4001 27FF	ADC1	详见表 7.2

续表

地 址 范 围		设 备 名 称	备　注
APB2	0x4001 2000～0x4001 23FF	GPIOG	
	0x4001 1C00～0x4001 1FFF	GPIOF	
	0x4001 1800～0x4001 1BFF	GPIOE	
	0x4001 1400～0x4001 17FF	GPIOD	
	0x4001 1000～0x4001 13FF	GPIOC	详见表 2.1
	0x4001 0C00～0x4001 0FFF	GPIOB	详见表 2.1
	0x4001 0800～0x4001 0BFF	GPIOA	详见表 2.1
	0x4001 0400～0x4001 07FF	EXTI	详见表 8.6
	0x4001 0000～0x4001 03FF	AFIO	
APB1	0x4000 7400～0x4000 77FF	DAC	
	0x4000 7000～0x4000 73FF	PWR（电源控制）	
	0x4000 6C00～0x4000 6FFF	BKP （后备寄存器）	
	0x4000 6800～0x4000 6BFF	bxCAN2	
	0x4000 6400～0x4000 67FF	bxCAN1	
	0x4000 6000～0x4000 63FF	USB/CAN 共享的 512B SRAM	
	0x4000 5C00～0x4000 5FFF	USB 全速设备寄存器	
	0x4000 5800～0x4000 5BFF	I2C2	详见表 5.2
	0x4000 5400～0x4000 57FF	I2C1	详见表 5.2
	0x4000 5000～0x4000 53FF	UART5	
	0x4000 4C00～0x4000 4FFF	UART4	
	0x4000 4800～0x4000 4BFF	USART3	
	0x4000 4400～0x4000 47FF	USART2	
	0x4000 4000～0x4000 43FF	保留	
	0x4000 3C00～0x4000 3FFF	SPI3/I2S3	
	0x4000 3800～0x4000 3BFF	SPI2/I2S2	
	0x4000 3400～0x4000 37FF	保留	
	0x4000 3000～0x4000 33FF	IWDG （独立看门狗）	
	0x4000 2C00～0x4000 2FFF	WWDG （窗口看门狗）	
	0x4000 2800～0x4000 2BFF	RTC	详见表 6.20
	0x4000 1400～0x4000 17FF	TIM7	
	0x4000 1000～0x4000 13FF	TIM6	
	0x4000 0C00～0x4000 0FFF	TIM5	
	0x4000 0800～0x4000 0BFF	TIM4	
	0x4000 0400～0x4000 07FF	TIM3	
	0x4000 0000～0x4000 03FF	TIM2	
0x2000 0000～0x3FFF FFFF (512MB)		SRAM	
0x00000000～0x1FFF FFFF (512MB)		FLASH	
FLASH	0x1FFF F800～0x1FFF F80F	选择字节	
	0x1FFF F000～0x1FFF F7FF	系统存储器	
	0x0800 0000～0x0801 FFFF	主存储器	

存储器映像在 stm32f10x_map.h（V2.0.1）或 stm32f10x.h（V3.5.0）中定义。两者的主要区别

是数据类型定义不同，如寄存器类型定义前者使用 VU32，后者使用_IO uint32_t。

1.3 STM32 MCU 系统时钟树

STM32 MCU 系统时钟树由系统时钟源、系统时钟 SYSCLK 和设备时钟等部分组成，如图 1.2 所示。

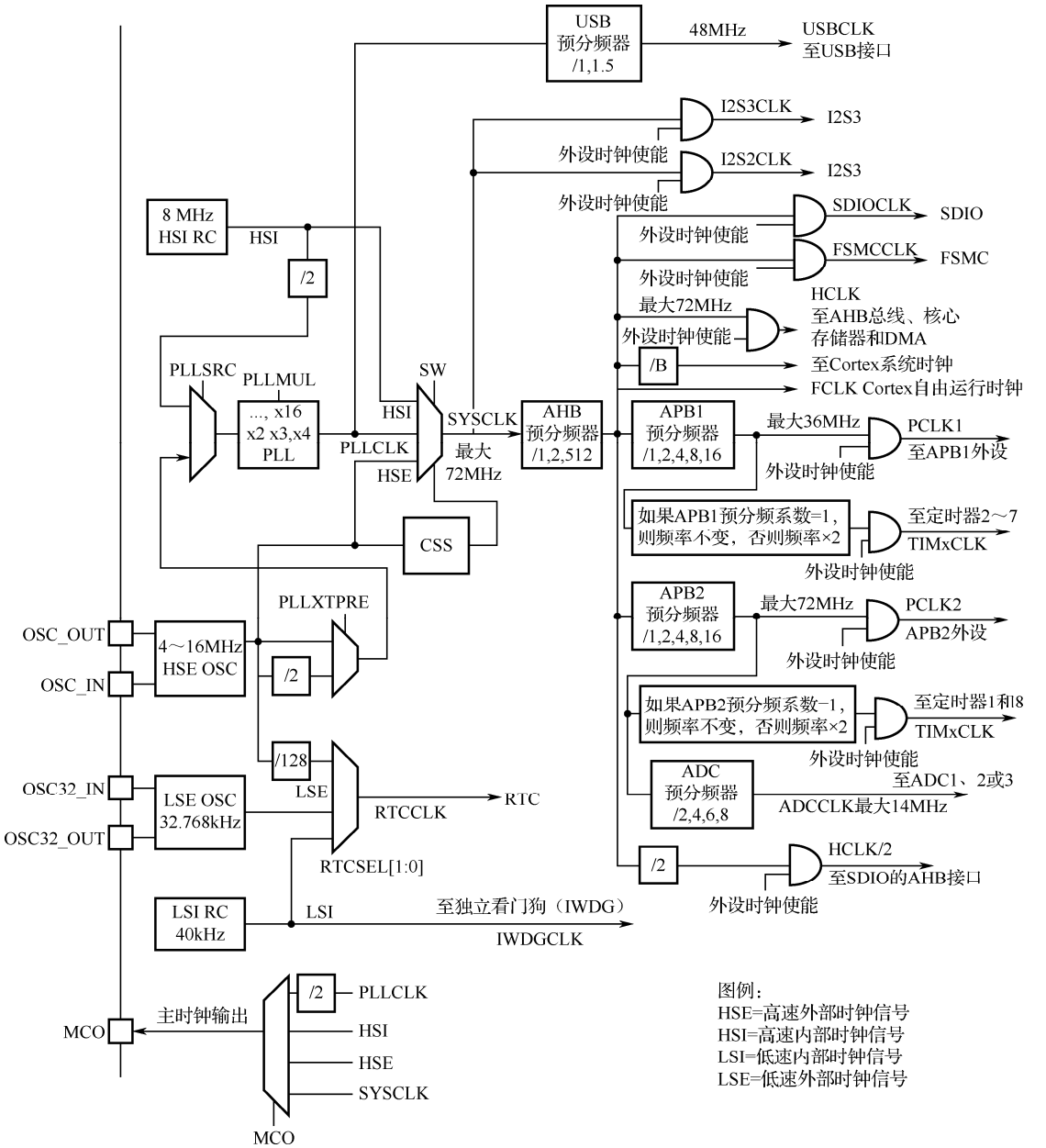


图 1.2　STM32 MCU 系统时钟树

系统时钟源有 4 个：高速外部时钟 HSE（4～16MHz）、低速外部时钟 LSE（32.768kHz）、高速内部时钟 HSI（8MHz）和低速内部时钟 LSI（40kHz），其中外部时钟用晶体振荡器 OSC 实现，内部时钟用 RC 振荡器实现。

系统时钟 SYSCLK（最大 72MHz）可以是 HSE 或 HSI，也可以是 HSE 或 HSI 通过锁相环 2～16 倍频后的锁相环时钟 PLLCLK。系统复位后的系统时钟为 HSI，这就意味着即使没有 HSE 系统也能正常工作，只是 HSI 的精度没有 HSE 高。

SYSCLK 经 AHB 预分频器分频后得到 AHB 总线时钟 HCLK（最大 72MHz），HCLK 经 APB1/APB2 预分频器分频后得到 APB1/APB2 总线时钟 PCLK1（最大 36MHz）和 PCLK2（最大 72MHz），PCLK1 和 PCLK2 分别为相连的设备提供设备时钟。

系统时钟树中的时钟选择、预分频值和外设时钟使能等都可以通过对复位和时钟控制（RCC）寄存器编程实现，复位和时钟控制（RCC）寄存器如表 1.2 所示（RCC 的基地址是 0x4002 1000）。

表 1.2　复位和时钟控制（RCC）寄存器

偏移地址	名　称	类　型	复 位 值	说　明
0x00	CR	读/写	0x0000 XX83	时钟控制寄存器（HSIRDY=1，HSION=1，详见表 1.3）
0x04	CFGR	读/写	0x0000 0000	时钟配置寄存器（SYSCLK=HSI，AHB、APB1 和 APB2 均不分频，即频率均为 8MHz，定时器时钟频率也为 8MHz，ADC 时钟为 APB2/2，即频率为 4MHz，详见表 1.4）
0x08	CIR	读/写	0x0000 0000	时钟中断寄存器（禁止所有中断）
0x0C	APB2RSTR	读/写	0x0000 0000	APB2 设备复位寄存器
0x10	APB1RSTR	读/写	0x0000 0000	APB1 设备复位寄存器
0x14	AHBENR	读/写	0x0000 0014	AHB 设备时钟使能寄存器（开启 Flash 接口和 SRAM 时钟）
0x18	APB2ENR	读/写	0x0000 0000	APB2 设备时钟使能寄存器（关闭所有 APB2 设备时钟，详见表 1.5）
0x1C	APB1ENR	读/写	0x0000 0000	APB1 设备时钟使能寄存器（关闭所有 APB1 设备时钟，详见表 1.6）
0x20	BDCR	读/写	0x0000 0000	备份域控制寄存器（详见表 1.7）
0x24	CSR	读/写	0x0C00 0000	控制状态寄存器（上电复位，NRST 引脚复位，详见表 1.8）

复位和时钟控制（RCC）寄存器结构体在 stm32f10x_map.h（V2.0.1）中定义如下：

```
typedef struct
{
  vu32 CR;                    // 时钟控制寄存器
  vu32 CFGR;                  // 时钟配置寄存器
  vu32 CIR;                   // 时钟中断寄存器
  vu32 APB2RSTR;              // APB2 设备复位寄存器
  vu32 APB1RSTR;              // APB1 设备复位寄存器
  vu32 AHBENR;                // AHB 设备时钟使能寄存器
  vu32 APB2ENR;               // APB2 设备时钟使能寄存器
  vu32 APB1ENR;               // APB1 设备时钟使能寄存器
  vu32 BDCR;                  // 备份域控制寄存器
  vu32 CSR;                   // 控制状态寄存器
} RCC_TypeDef;
```

1.3.1　时钟控制

时钟控制主要包括 HSI 使能、HSE 使能和 PLL 使能等，时钟控制寄存器（CR）如表 1.3 所示（保留位未列出）。

表 1.3　时钟控制寄存器（CR）

位	名　称	类　型	复 位 值	说　明
0	HSION	读/写	1	高速内部时钟使能：0—关闭时钟，1—开启时钟
1	HSIRDY	读	1	高速内部时钟就绪：0—时钟未就绪，1—时钟就绪
7:3	HSITRIM[4:0]	读/写	10000	高速内部时钟调整
15:8	HSICAL[7:0]	读	XXXXXXXX	高速内部时钟校准
16	HSEON	读/写	0	高速外部时钟使能：0—关闭时钟，1—开启时钟
17	HSERDY	读	0	高速外部时钟就绪：0—时钟未就绪，1—时钟就绪
18	HSEBYP	读/写	0	高速外部时钟旁路：0—时钟未旁路，1—时钟旁路
19	CSSON	读/写	0	时钟安全系统使能：0—关闭检测，1—开启检测
24	PLLON	读/写	0	PLL 使能：0—关闭 PLL，1—开启 PLL
25	PLLRDY	读	0	PLL 就绪：0—PLL 未就绪，1—PLL 就绪

常用与时钟控制有关的 RCC 库函数在 stm32f10x_rcc.h（V2.0.1）中声明如下：

```
void RCC_HSEConfig(u32 RCC_HSE);
ErrorStatus RCC_WaitForHSEStartUp(void);
void RCC_HSICmd(FunctionalState NewState);
void RCC_PLLCmd(FunctionalState NewState);
```

1）配置 HSE

```
void RCC_HSEConfig(u32 RCC_HSE);
```

参数说明：

★ RCC_HSE：HSE 配置，在 stm32f10x_rcc.h 中定义如下：

```
#define RCC_HSE_ON                    ((u32)0x00010000)   // 开启 HSE
#define RCC_HSE_Bypass                ((u32)0x00040000)   // 旁路 HSE
```

RCC_HSEConfig()函数的核心语句是：

```
RCC->CR &= CR_HSEON_Reset;
RCC->CR &= CR_HSEBYP_Reset;
RCC->CR |= CR_HSEON_Set;
RCC->CR |= CR_HSEBYP_Set | CR_HSEON_Set;
```

CR_HSEON_Reset、CR_HSEBYP_Reset、CR_HSEON_Set 和 CR_HSEBYP_Set 在 stm32f10x_rcc.c 中定义如下：

```
#define CR_HSEBYP_Reset               ((u32)0xFFFBFFFF)   // 旁路 HSE 复位
#define CR_HSEBYP_Set                 ((u32)0x00040000)   // 旁路 HSE 置位
#define CR_HSEON_Reset                ((u32)0xFFFEFFFF)   // 开启 HSE 复位
#define CR_HSEON_Set                  ((u32)0x00010000)   // 开启 HSE 置位
```

2）等待 HSE 启动

```
ErrorStatus RCC_WaitForHSEStartUp(void);
```

返回值：SUCCESS—HSE 就绪，ERROR—HSE 未就绪

RCC_WaitForHSEStartUp()函数的核心语句是：

```
HSEStatus = RCC_GetFlagStatus(RCC_FLAG_HSERDY);
```

RCC_GetFlagStatus()函数的说明参加 1.3.6（2）。

3）使能 HSI

```
void RCC_HSICmd(FunctionalState NewState);
```

参数说明：

★ NewState：HSI 新状态，ENABLE（1）—允许，DISABLE（0）—禁止

4）使能 PLL

```
void RCC_PLLCmd(FunctionalState NewState);
```

参数说明：

★ NewState：HSI 新状态，ENABLE（1）—允许，DISABLE（0）—禁止

1.3.2 时钟配置

时钟配置主要包括系统时钟切换、AHB 预分频、APB1 预分频、APB2 预分频、ADC 预分频、PLL 倍频和时钟输出选择等，时钟配置寄存器（CFGR）如表 1.4 所示。

表 1.4 时钟配置寄存器（CFGR）

位	名 称	类 型	复 位 值	说 明
1:0	SW[1:0]	读/写	00	系统时钟切换：00—HSI，01—HSE，10—PLLCLK，11—不可用
3:2	SWS[1:0]	读	00	系统时钟切换状态：00—HSI，01—HSE，10—PLLCLK，11—不可用
7:4	HPRE[3:0]	读/写	0000	AHB 预分频：0XXX—SYSCLK，1000—SYSCLK/2，1001—SYSCLK/4，1010－SYSCLK/8，1011－SYSCLK/16，1100－SYSCLK/64，1101—SYSCLK/128，1110—SYSCLK/256，1111—SYSCLK/512
10:8	PPRE1[2:0]	读/写	000	APB1 预分频：0XX—HCLK，100—HCLK/2，101—HCLK/4，110—HCLK/8，111—HCLK/16
13:11	PPRE2[2:0]	读/写	000	APB2 预分频：0XX—HCLK，100—HCLK/2，101—HCLK/4，110—HCLK/8，111—HCLK/16
15:14	ADCPRE[1:0]	读/写	00	ADC 预分频：00—PCLK2/2，01—PCLK2/4，10—PCLK2/6，11—PCLK2/8
16	PLLSRC	读/写	0	PLL 输入时钟源：0—HSI/2，1—HSE 或 HSE/2
17	PLLXTPRE	读/写	0	PLL 外部输入分频：0—HSE，1—HSE/2
21:18	PLLMUL[3:0]	读/写	0000	PLL 倍频：0000~1110—2~16 倍频，1111—16 倍频
22	USBPRE	读/写	0	USB 预分频：0—PLLCLK/1.5，1—PLLCLK
26:24	MCO[2:0]	读/写	000	时钟输出选择：0XX—不输出，100—SYSCLK，101—HSI，110—HSE，111—PLLCLK/2

常用与时钟配置有关的 RCC 库函数在 stm32f10x_rcc.h（V2.0.1）中声明如下：

```
void RCC_PLLConfig(u32 RCC_PLLSource, u32 RCC_PLLMul);
void RCC_SYSCLKConfig(u32 RCC_SYSCLKSource);
u8 RCC_GetSYSCLKSource(void);
void RCC_HCLKConfig(u32 RCC_SYSCLK);
```

```
void RCC_PCLK1Config(u32 RCC_HCLK);
void RCC_PCLK2Config(u32 RCC_HCLK);
void RCC_ADCCLKConfig(u32 RCC_PCLK2);
void RCC_GetClocksFreq(RCC_ClocksTypeDef* RCC_Clocks);
void RCC_MCOConfig(u8 RCC_MCO);
```

1）配置 PLL

```
void RCC_PLLConfig(u32 RCC_PLLSource, u32 RCC_PLLMul);
```

参数说明：

★ RCC_PLLSource：PLL 时钟源，在 stm32f10x_rcc.h 中定义如下：

```
#define RCC_PLLSource_HSI_Div2      ((u32)0x00000000)   // HSI 分频 2
#define RCC_PLLSource_HSE_Div1      ((u32)0x00010000)   // HSE 分频 1
#define RCC_PLLSource_HSE_Div2      ((u32)0x00030000)   // HSE 分频 2
```

★ RCC_PLLMul：PLL 倍频，在 stm32f10x_rcc.h 中定义如下：

```
#define RCC_PLLMul_2                ((u32)0x00000000)   // PLL 倍频 2
#define RCC_PLLMul_3                ((u32)0x00040000)   // PLL 倍频 3
#define RCC_PLLMul_4                ((u32)0x00080000)   // PLL 倍频 4
#define RCC_PLLMul_5                ((u32)0x000C0000)   // PLL 倍频 5
#define RCC_PLLMul_6                ((u32)0x00100000)   // PLL 倍频 6
#define RCC_PLLMul_7                ((u32)0x00140000)   // PLL 倍频 7
#define RCC_PLLMul_8                ((u32)0x00180000)   // PLL 倍频 8
#define RCC_PLLMul_9                ((u32)0x001C0000)   // PLL 倍频 9
#define RCC_PLLMul_10               ((u32)0x00200000)   // PLL 倍频 10
#define RCC_PLLMul_11               ((u32)0x00240000)   // PLL 倍频 11
#define RCC_PLLMul_12               ((u32)0x00280000)   // PLL 倍频 12
#define RCC_PLLMul_13               ((u32)0x002C0000)   // PLL 倍频 13
#define RCC_PLLMul_14               ((u32)0x00300000)   // PLL 倍频 14
#define RCC_PLLMul_15               ((u32)0x00340000)   // PLL 倍频 15
#define RCC_PLLMul_16               ((u32)0x00380000)   // PLL 倍频 16
```

RCC_PLLConfig()函数的核心语句是：

```
tmpreg |= RCC_PLLSource | RCC_PLLMul;
RCC->CFGR = tmpreg;
```

2）配置 SYSCLK

```
void RCC_SYSCLKConfig(u32 RCC_SYSCLKSource);
```

参数说明：

★ RCC_SYSCLKSource：SYSCLK 时钟源，在 stm32f10x_rcc.h 中定义如下：

```
#define RCC_SYSCLKSource_HSI     ((u32)0x00000000)      // SYSCLK=HSI
#define RCC_SYSCLKSource_HSE     ((u32)0x00000001)      // SYSCLK=HSE
#define RCC_SYSCLKSource_PLLCLK  ((u32)0x00000002)      // SYSCLK=PLLCLK
```

RCC_SYSCLKConfig()函数的核心语句是：

```
tmpreg |= RCC_SYSCLKSource;
RCC->CFGR = tmpreg;
```

3）获取 SYSCLK 时钟源

```
u8 RCC_GetSYSCLKSource(void);
```

返回值：SYSCLK 时钟源，0—HSI，4—HSE，8—PLLCLK
RCC_GetSYSCLKSource()函数的核心语句是：

```
return ((u8)(RCC->CFGR & CFGR_SWS_Mask));
```

CFGR_SWS_Mask 在 stm32f10x_rcc.c 中定义如下：

```
#define CFGR_SWS_Mask                ((u32)0x0000000C)   // SYSCLK 屏蔽
```

4）配置 HCLK

```
void RCC_HCLKConfig(u32 RCC_SYSCLK);
```

参数说明：
★ RCC_SYSCLK：SYSCLK 分频，在 stm32f10x_rcc.h 中定义如下：

```
#define RCC_SYSCLK_Div1          ((u32)0x00000000)   // SYSCLK 分频 1
#define RCC_SYSCLK_Div2          ((u32)0x00000080)   // SYSCLK 分频 2
#define RCC_SYSCLK_Div4          ((u32)0x00000090)   // SYSCLK 分频 4
#define RCC_SYSCLK_Div8          ((u32)0x000000A0)   // SYSCLK 分频 8
#define RCC_SYSCLK_Div16         ((u32)0x000000B0)   // SYSCLK 分频 16
#define RCC_SYSCLK_Div64         ((u32)0x000000C0)   // SYSCLK 分频 64
#define RCC_SYSCLK_Div128        ((u32)0x000000D0)   // SYSCLK 分频 128
#define RCC_SYSCLK_Div256        ((u32)0x000000E0)   // SYSCLK 分频 256
#define RCC_SYSCLK_Div512        ((u32)0x000000F0)   // SYSCLK 分频 512
```

RCC_HCLKConfig()函数的核心语句是：

```
tmpreg |= RCC_SYSCLK;
RCC->CFGR = tmpreg;
```

5）配置 PCLK1

```
void RCC_PCLK1Config(u32 RCC_HCLK);
```

参数说明：
★ RCC_HCLK：HCLK 分频，在 stm32f10x_rcc.h 中定义如下：

```
#define RCC_HCLK_Div1            ((u32)0x00000000)   // HCLK 分频 1
#define RCC_HCLK_Div2            ((u32)0x00000400)   // HCLK 分频 2
#define RCC_HCLK_Div4            ((u32)0x00000500)   // HCLK 分频 4
#define RCC_HCLK_Div8            ((u32)0x00000600)   // HCLK 分频 8
#define RCC_HCLK_Div16           ((u32)0x00000700)   // HCLK 分频 16
```

RCC_PCLK1Config()函数的核心语句是：

```
tmpreg |= RCC_HCLK;
RCC->CFGR = tmpreg;
```

6）配置 PCLK2

```
void RCC_PCLK2Config(u32 RCC_HCLK);
```

参数说明：

★ RCC_HCLK：HCLK 分频，在 stm32f10x_rcc.h 中定义如下：

```
#define RCC_HCLK_Div1              ((u32)0x00000000)   // HCLK 分频 1
#define RCC_HCLK_Div2              ((u32)0x00000400)   // HCLK 分频 2
#define RCC_HCLK_Div4              ((u32)0x00000500)   // HCLK 分频 4
#define RCC_HCLK_Div8              ((u32)0x00000600)   // HCLK 分频 8
#define RCC_HCLK_Div16             ((u32)0x00000700)   // HCLK 分频 16
```

RCC_PCLK2Config()函数的核心语句是：

```
tmpreg |= RCC_HCLK<< 3;
RCC->CFGR = tmpreg;
```

7）配置 ADCCLK

```
void RCC_ADCCLKConfig(u32 RCC_PCLK2);
```

参数说明：

★ RCC_PCLK2：PCLK2 分频，在 stm32f10x_rcc.h 中定义如下：

```
#define RCC_PCLK2_Div2             ((u32)0x00000000)   // PCLK2 分频 2
#define RCC_PCLK2_Div4             ((u32)0x00004000)   // PCLK2 分频 4
#define RCC_PCLK2_Div6             ((u32)0x00008000)   // PCLK2 分频 6
#define RCC_PCLK2_Div8             ((u32)0x0000C000)   // PCLK2 分频 8
```

RCC_ADCCLKConfig()函数的核心语句是：

```
tmpreg |= RCC_PCLK2;
RCC->CFGR = tmpreg;
```

8）获取时钟频率

```
void RCC_GetClocksFreq(RCC_ClocksTypeDef* RCC_Clocks);
```

参数说明：

★ RCC_Clocks：时钟结构体指针，时钟结构体在 stm32f10x_rcc.h 中定义如下：

```
typedef struct
{
  u32 SYSCLK_Frequency;          // SYSCLK 频率
  u32 HCLK_Frequency;          // HCLK 频率
  u32 PCLK1_Frequency;           // PCLK1 频率
  u32 PCLK2_Frequency;           // PCLK2 频率
  u32 ADCCLK_Frequency;          // ADCCLK 频率
} RCC_ClocksTypeDef;
```

9）配置时钟输出

```
void RCC_MCOConfig(u8 RCC_MCO);
```

参数说明：

★ RCC_MCO：时钟输出选择，在 stm32f10x_rcc.h 中定义如下：

```
#define RCC_MCO_NoClock              ((u8)0x00)      // 无输出
#define RCC_MCO_SYSCLK               ((u8)0x04)      // 输出 SYSCLK
#define RCC_MCO_HSI                  ((u8)0x05)      // 输出 HSI
#define RCC_MCO_HSE                  ((u8)0x06)      // 输出 HSE
#define RCC_MCO_PLLCLK_Div2          ((u8)0x07)      // 输出 PLLCLK/2
```

1.3.3 APB2 设备时钟使能

APB2 设备时钟使能主要包括 GPIO 时钟使能、USART 时钟使能、SPI 时钟使能、TIM 时钟使能和 ADC 时钟使能等，APB2 设备时钟使能寄存器（APB2ENR）如表 1.5 所示（保留位未列出）。

表 1.5 APB2 设备时钟使能寄存器（APB2ENR）

位	名 称	类 型	复 位 值	说 明
0	AFIOEN	读/写	0	AFIO 时钟使能：0—关闭时钟，1—开启时钟
2	GPIOAEN	读/写	0	GPIOA 时钟使能：0—关闭时钟，1—开启时钟
3	GPIOBEN	读/写	0	GPIOB 时钟使能：0—关闭时钟，1—开启时钟
4	GPIOCEN	读/写	0	GPIOC 时钟使能：0—关闭时钟，1—开启时钟
5	GPIODEN	读/写	0	GPIOD 时钟使能：0—关闭时钟，1—开启时钟
6	GPIOEEN	读/写	0	GPIOE 时钟使能：0—关闭时钟，1—开启时钟
7	GPIOFEN	读/写	0	GPIOF 时钟使能：0—关闭时钟，1—开启时钟
8	GPIOGEN	读/写	0	GPIOG 时钟使能：0—关闭时钟，1—开启时钟
9	ADC1EN	读/写	0	ADC1 时钟使能：0—关闭时钟，1—开启时钟
10	ADC2EN	读/写	0	ADC2 时钟使能：0—关闭时钟，1—开启时钟
11	TIM1EN	读/写	0	TIM1 定时器时钟使能：0—关闭时钟，1—开启时钟
12	SPI1EN	读/写	0	SPI1 时钟使能：0—关闭时钟，1—开启时钟
13	TIM8EN	读/写	0	TIM8 定时器时钟使能：0—关闭时钟，1—开启时钟
14	USART1EN	读/写	0	USART1 时钟使能：0—关闭时钟，1—开启时钟
15	ADC3EN	读/写	0	ADC3 时钟使能：0—关闭时钟，1—开启时钟

使能 APB2 设备时钟的库函数在 stm32f10x_rcc.h 中声明如下：

```
void RCC_APB2PeriphClockCmd(u32 RCC_APB2Periph, FunctionalStateNewState)
```

功能：使能 APB2 设备时钟

参数说明：

★ RCC_APB2Periph：APB2 设备，在 stm32f10x_rcc.h 中定义如下：

```
#define RCC_APB2Periph_AFIO          ((u32)0x00000001)   // AFIO 设备
#define RCC_APB2Periph_GPIOA         ((u32)0x00000004)   // GPIOA 设备
#define RCC_APB2Periph_GPIOB         ((u32)0x00000008)   // GPIOB 设备
#define RCC_APB2Periph_GPIOC         ((u32)0x00000010)   // GPIOC 设备
#define RCC_APB2Periph_GPIOD         ((u32)0x00000020)   // GPIOD 设备
#define RCC_APB2Periph_GPIOE         ((u32)0x00000040)   // GPIOE 设备
```

```
#define RCC_APB2Periph_GPIOF          ((u32)0x00000080)   // GPIOF 设备
#define RCC_APB2Periph_GPIOG          ((u32)0x00000100)   // GPIOG 设备
#define RCC_APB2Periph_ADC1           ((u32)0x00000200)   // ADC1 设备
#define RCC_APB2Periph_ADC2           ((u32)0x00000400)   // ADC2 设备
#define RCC_APB2Periph_TIM1           ((u32)0x00000800)   // TIM1 设备
#define RCC_APB2Periph_SPI1           ((u32)0x00001000)   // SPI1 设备
#define RCC_APB2Periph_TIM8           ((u32)0x00002000)   // TIM8 设备
#define RCC_APB2Periph_USART1         ((u32)0x00004000)   // USART1 设备
#define RCC_APB2Periph_ADC3           ((u32)0x00008000)   // ADC3 设备
#define RCC_APB2Periph_ALL            ((u32)0x0000FFFD)   // 所有设备
```

★ NewState：时钟新状态，ENABLE（1）—允许，DISABLE（0）—禁止

RCC_APB2PeriphClockCmd()函数的核心语句是：

```
RCC->APB2ENR |= RCC_APB2Periph;
RCC->APB2ENR &= ~RCC_APB2Periph;
```

1.3.4 APB1 设备时钟使能

APB1 设备时钟使能主要包括 USART 时钟使能、SPI 时钟使能、I^2C 时钟使能和 TIM 时钟使能等，APB1 设备时钟使能寄存器（APB1ENR）如表 1.6 所示（保留位未列出）。

表 1.6 APB1 设备时钟使能寄存器（APB1ENR）

位	名 称	类 型	复 位 值	说 明
0	TIM2EN	读/写	0	TIM2 时钟使能：0—关闭时钟，1—开启时钟
1	TIM3EN	读/写	0	TIM3 时钟使能：0—关闭时钟，1—开启时钟
2	TIM4EN	读/写	0	TIM4 时钟使能：0—关闭时钟，1—开启时钟
3	TIM5EN	读/写	0	TIM5 时钟使能：0—关闭时钟，1—开启时钟
4	TIM6EN	读/写	0	TIM6 时钟使能：0—关闭时钟，1—开启时钟
5	TIM7EN	读/写	0	TIM7 时钟使能：0—关闭时钟，1—开启时钟
11	WWDGEN	读/写	0	WWDG 时钟使能：0—关闭时钟，1—开启时钟
14	SPI2EN	读/写	0	SPI2 时钟使能：0—关闭时钟，1—开启时钟
15	SPI3EN	读/写	0	SPI3 时钟使能：0—关闭时钟，1—开启时钟
17	USART2EN	读/写	0	USART2 时钟使能：0—关闭时钟，1—开启时钟
18	USART3EN	读/写	0	USART3 时钟使能：0—关闭时钟，1—开启时钟
19	UART4EN	读/写	0	UART4 时钟使能：0—关闭时钟，1—开启时钟
20	UART5EN	读/写	0	UART5 钟使能：0—关闭时钟，1—开启时钟
21	I2C1EN	读/写	0	I2C1 时钟使能：0—关闭时钟，1—开启时钟
22	I2C2EN	读/写	0	I2C2 时钟使能：0—关闭时钟，1—开启时钟
23	USBEN	读/写	0	USB 时钟使能：0—关闭时钟，1—开启时钟
25	CANEN	读/写	0	CAN 时钟使能：0—关闭时钟，1—开启时钟
27	BKPEN	读/写	0	BKP 时钟使能：0—关闭时钟，1—开启时钟
28	PWREN	读/写	0	PWR 时钟使能：0—关闭时钟，1—开启时钟
29	DACEN	读/写	0	DAC 时钟使能：0—关闭时钟，1—开启时钟

使能 APB1 设备时钟的库函数在 stm32f10x_rcc.h 中声明如下：

```
void RCC_APB1PeriphClockCmd(u32 RCC_APB1Periph, FunctionalStateNewState)
```

功能：使能 APB1 设备时钟

参数说明：

★ RCC_APB1Periph：APB1 设备，在 stm32f10x_rcc.h 中定义如下：

```
#define RCC_APB1Periph_TIM2          ((u32)0x00000001)   // TIM2 设备
#define RCC_APB1Periph_TIM3          ((u32)0x00000002)   // TIM3 设备
#define RCC_APB1Periph_TIM4          ((u32)0x00000004)   // TIM4 设备
#define RCC_APB1Periph_TIM5          ((u32)0x00000008)   // TIM5 设备
#define RCC_APB1Periph_TIM6          ((u32)0x00000010)   // TIM6 设备
#define RCC_APB1Periph_TIM7          ((u32)0x00000020)   // TIM7 设备
#define RCC_APB1Periph_WWDG          ((u32)0x00000800)   // WWDG 设备
#define RCC_APB1Periph_SPI2          ((u32)0x00004000)   // SPI2 设备
#define RCC_APB1Periph_SPI3          ((u32)0x00008000)   // SPI3 设备
#define RCC_APB1Periph_USART2        ((u32)0x00020000)   // USART2 设备
#define RCC_APB1Periph_USART3        ((u32)0x00040000)   // USART3 设备
#define RCC_APB1Periph_UART4         ((u32)0x00080000)   // UART4 设备
#define RCC_APB1Periph_UART5         ((u32)0x00100000)   // UART5 设备
#define RCC_APB1Periph_I2C1          ((u32)0x00200000)   // I2C1 设备
#define RCC_APB1Periph_I2C2          ((u32)0x00400000)   // I2C2 设备
#define RCC_APB1Periph_USB           ((u32)0x00800000)   // USB 设备
#define RCC_APB1Periph_CAN           ((u32)0x02000000)   // CAN 设备
#define RCC_APB1Periph_BKP           ((u32)0x08000000)   // BKP 设备
#define RCC_APB1Periph_PWR           ((u32)0x10000000)   // PWR 设备
#define RCC_APB1Periph_DAC           ((u32)0x20000000)   // DAC 设备
#define RCC_APB1Periph_ALL           ((u32)0x3AFEC83F)   // 所有设备
```

★ NewState：时钟新状态，ENABLE（1）—允许，DISABLE（0）—禁止

RCC_APB1PeriphClockCmd()函数的核心语句是：

```
RCC->APB1ENR |= RCC_APB1Periph;
RCC->APB1ENR &= ~RCC_APB1Periph;
```

1.3.5 备份域控制

备份域控制主要包括 LSE 使能、RTC 使能和备份域软复位等，备份域控制寄存器（BDCR）如表 1.7 所示（保留位未列出）。

表 1.7 备份域控制寄存器（BDCR）

位	名 称	类 型	复 位 值	说 明
0	LSEON	读/写	0	低速外部时钟使能：0—关闭时钟，1—开启时钟
1	LSERDY	读	0	低速外部时钟就绪：0—时钟未就绪，1—时钟就绪
2	LSEBYP	读/写	0	低速外部时钟旁路：0—时钟未旁路，1—时钟旁路
9:8	RTCSEL[1:0]	读/写	00	RTC 时钟源选择：00—无时钟，01—LSE，10—LSI，11—HSE/128
15	RTCEN	读/写	0	RTC 时钟使能：0—关闭时钟，1—开启时钟
16	BDRST	读/写	0	备份域软复位：0—不复位备份域，1—复位备份域

与备份域控制有关的 RCC 库函数在 stm32f10x_rcc.h 中声明如下：

```
void RCC_LSEConfig(u8 RCC_LSE);
void RCC_RTCCLKConfig(u32 RCC_RTCCLKSource);
void RCC_RTCCLKCmd(FunctionalState NewState);
void RCC_BackupResetCmd(FunctionalState NewState);
```

1）配置 LSE

```
void RCC_LSEConfig(u8 RCC_LSE);
```

参数说明：

★ RCC_LSE：LSE 配置，在 stm32f10x_rcc.h 中定义如下：

```
#define RCC_LSE_OFF                    ((u8)0x00)      // 关闭 LSE
#define RCC_LSE_ON                     ((u8)0x01)      // 开启 LSE
#define RCC_LSE_Bypass                 ((u8)0x04)      // 旁路 LSE
```

2）配置 RTCCLK

```
void RCC_RTCCLKConfig(u32 RCC_RTCCLKSource);
```

参数说明：

★ RCC_RTCCLKSource：RTC 时钟源，在 stm32f10x_rcc.h 中定义如下：

```
#define RCC_RTCCLKSource_LSE         ((u32)0x00000100)
#define RCC_RTCCLKSource_LSI         ((u32)0x00000200)
#define RCC_RTCCLKSource_HSE_Div128  ((u32)0x00000300)
```

3）使能 RTCCLK

```
void RCC_RTCCLKCmd(FunctionalState NewState);
```

参数说明：

★ NewState：RTCCLK 新状态，ENABLE（1）—允许，DISABLE（0）—禁止

4）使能备份域软复位

```
void RCC_BackupResetCmd(FunctionalState NewState);
```

参数说明：

★ NewState：备份域软复位新状态，ENABLE（1）—允许，DISABLE（0）—禁止

注意：备份域控制寄存器中的 LSEON、LSEBYP、RTCSEL 和 RTCEN 位处于备份域，因此这些位在复位后处于写保护状态，只有在电源控制寄存器（PWR_CR）中的 DBP（取消备份域写保护）位置位后才能对这些位进行操作，这些位只能由备份域软复位清除，任何内部或外部复位都不会影响这些位。

5）使能备份域访问

```
void PWR_BackupAccessCmd(FunctionalState NewState);
```

参数说明：

★ NewState：备份域访问新状态，ENABLE（1）—允许，DISABLE（0）—禁止

注意：使能备份域访问时必须开启 PWR 和 BKP 时钟。

1.3.6 控制状态

控制状态主要包括 LSI 使能和复位标志等，控制状态寄存器（CSR）如表 1.8 所示（保留位未列出）。

表 1.8 控制状态寄存器（CSR）

位	名 称	类 型	复 位 值	说 明
0	LSION	读/写	0	低速内部时钟使能：0—关闭时钟，1—开启时钟
1	LSIRDY	读	0	低速内部时钟就绪：0—时钟未就绪，1—时钟就绪
24	RMVF	读/写	0	清除复位标志
26	PINPSTF	读/写	1	NRST 引脚复位标志
27	PORRSTF	读/写	1	上电/掉电复位标志
28	SFTRSTF	读/写	0	软件复位标志
29	IWDGRSTF	读/写	0	独立看门狗复位标志
30	WWDGRSTF	读/写	0	窗口看门狗复位标志
31	LPWRRSTF	读/写	0	低功耗复位标志

与控制状态有关的 RCC 库函数在 stm32f10x_rcc.h 中声明如下：

```
void RCC_LSICmd(FunctionalState NewState);
FlagStatus RCC_GetFlagStatus(u8 RCC_FLAG);
void RCC_ClearFlag(void);
```

1）使能 LSI

```
void RCC_LSICmd(FunctionalState NewState);
```

参数说明：

★ NewState：LSI 新状态，ENABLE（1）—允许，DISABLE（0）—禁止

2）获取 RCC 标志状态

```
FlagStatus RCC_GetFlagStatus(u8 RCC_FLAG);
```

参数说明：

★ RCC_FLAG：RCC 标志，在 stm32f10x_rcc.h 中定义如下：

```
#define RCC_FLAG_HSIRDY          ((u8)0x20)          // HSI 就绪
#define RCC_FLAG_HSERDY          ((u8)0x31)          // HSE 就绪
#define RCC_FLAG_PLLRDY          ((u8)0x39)          // PLL 就绪
#define RCC_FLAG_LSERDY          ((u8)0x41)          // LSE 就绪
#define RCC_FLAG_LSIRDY          ((u8)0x61)          // LSI 就绪
#define RCC_FLAG_PINRST          ((u8)0x7A)          // 引脚复位
#define RCC_FLAG_PORRST          ((u8)0x7B)          // 上电复位
#define RCC_FLAG_SFTRST          ((u8)0x7C)          // 软件复位
#define RCC_FLAG_IWDGRST         ((u8)0x7D)          // 独立看门狗复位
#define RCC_FLAG_WWDGRST         ((u8)0x7E)          // 窗口看门狗复位
```

```
#define RCC_FLAG_LPWRRST                ((u8)0x7F)          // 低功耗复位
```

返回值：RCC 标志状态，SET（1）—置位，RESET（0）—复位

3）清除 RCC 标志

```
void RCC_ClearFlag(void);
```

RCC_ClearFlag()的核心语句是：

```
RCC->CSR |= CSR_RMVF_Set;
```

CSR_RMVF_Set 在 stm32f10x_rcc.c 中定义如下：

```
#defineCSR_RMVF_Set                ((u32)0x01000000)
```

在 Keil 的调试界面选择“Peripherals”（设备）菜单下的“Power, Reset and Clock Control”（电源、复位和时钟控制）子菜单可以打开如图 1.3 所示对话框，其中包含复位和时钟控制寄存器的复位值。

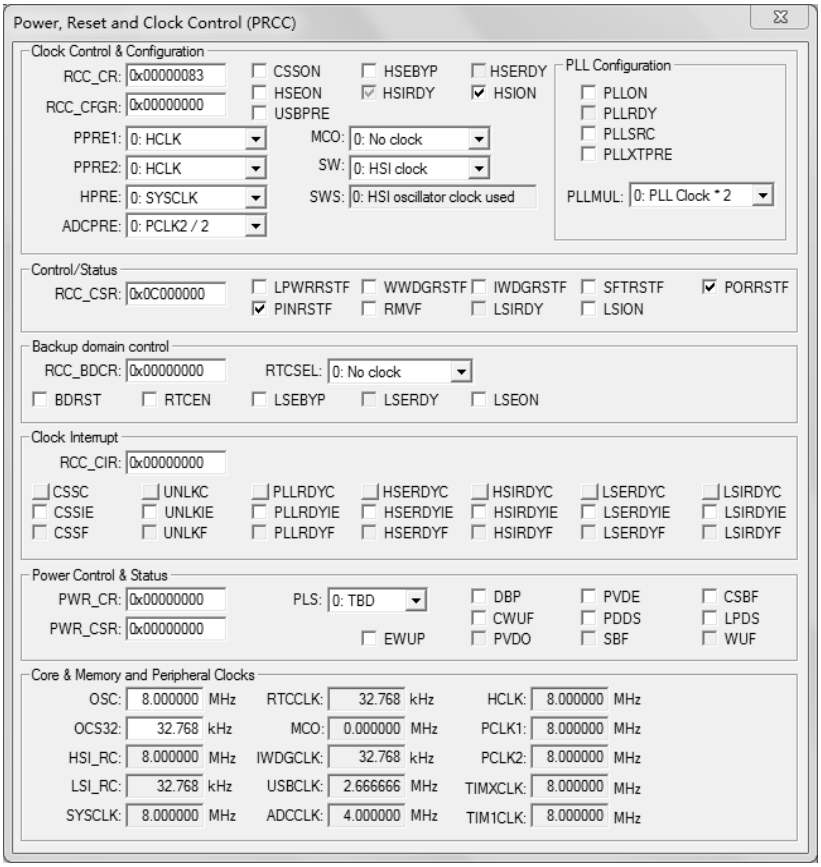
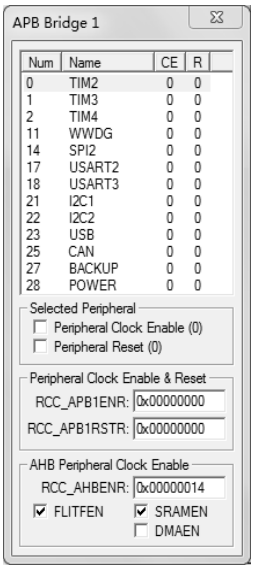
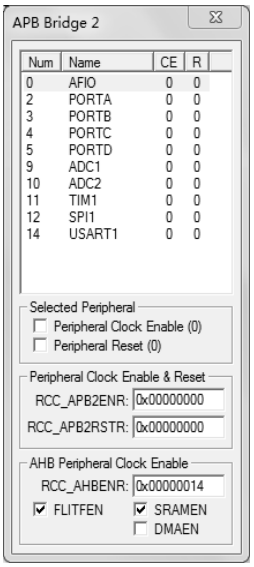


图 1.3　电源、复位和时钟控制对话框

选择“Peripherals”（设备）菜单下的“APB Bridge”（APB 桥）子菜单可以打开“APB Bridge 1”（APB 桥 1）和“APB Bridge 2”（APB 桥 2）对话框，如图 1.4 所示，其中分别包含 APB1 和 APB2 设备时钟使能寄存器的复位值。



（a）APB 桥 1



（b）APB 桥 2

图 1.4　APB 桥对话框

1.4　Cortex-M3 简介

Cortex-M3 采用哈佛结构的 32 位处理器内核，拥有独立的指令总线和数据总线，两者共享同一个 4GB 存储器空间。

Cortex-M3 内建一个嵌套向量中断控制器（NVIC，Nested Vectored Interrupt Controller），支持可嵌套中断、向量中断和动态优先级等，详见第 8 章。

Cortex-M3 内部还包含一个系统滴答定时器 SysTick，结构如图 1.5 所示。

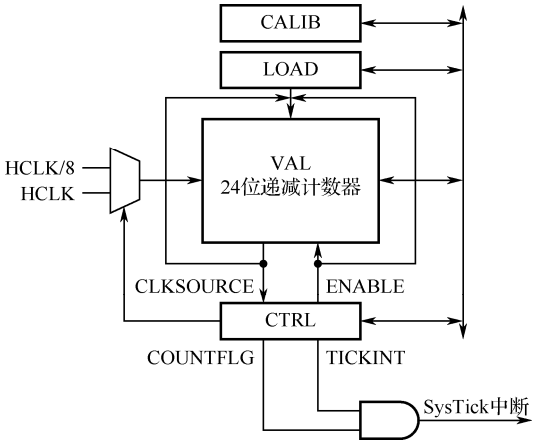


图 1.5　SysTick 结构图

SysTick 的核心是 1 个 24 位递减计数器，使用时根据需要设置初值（LOAD），启动（ENABLE=1）后在系统时钟（HCLK 或 HCLK/8）的作用下递减，减到 0 时置计数标志位（COUNTFLAG）并重装初值。系统可以查询计数标志位，也可以在中断允许（TICKINT=1）时产生 SysTick 中断。

SysTick 通过 4 个 32 位寄存器进行操作，如表 1.9 所示。

表 1.9　SysTick 寄存器

地　址	名　称	类　型	复 位 值	说　明
0xE000 E010	CTRL	读/写	0	控制状态寄存器（详见表 1.10）
0xE000 E014	LOAD	读/写	—	重装值寄存器（24 位），计数到 0 时重装到 VAL
0xE000 E018	VAL	读/写清除	—	当前值寄存器（24 位），写清除，同时清除计数标志
0xE000 E01C	CALIB	读	—	校准寄存器

SysTick 寄存器结构体在 stm32f10x_map.h（V2.0.1）中定义如下：

```
typedef struct
{
  vu32 CTRL;                    // 控制状态寄存器
  vu32 LOAD;                    // 重装值寄存器
  vu32 VAL;                     // 当前值寄存器
  vuc32 CALIB;                  // 校准寄存器
} SysTick_TypeDef;
```

控制状态寄存器（CTRL）有 3 个控制位和 1 个状态位，如表 1.10 所示。

表 1.10　SysTick 控制状态寄存器（CTRL）

位	名　称	类　型	复 位 值	说　明
0	ENABLE	读/写	0	定时器允许：0—停止定时器，1—启动定时器
1	TICKINT	读/写	0	中断允许：0—计数到 0 时不中断，1—计数到 0 时中断
2	CLKSOURCE	读/写	0	时钟源选择：0—时钟源为 HCLK/8，1—时钟源为 HCLK
16	COUNTFLAG	读	0	计数标志：SysTick 计数到 0 时置 1，读取后自动清零

SysTick 相关的库函数在 stm32f10x_systick.h（V2.0.1）中声明如下：

```
void SysTick_CLKSourceConfig(u32 SysTick_CLKSource);
void SysTick_SetReload(u32 Reload);
void SysTick_CounterCmd(u32 SysTick_Counter);
void SysTick_ITConfig(FunctionalState NewState);
u32 SysTick_GetCounter(void);
FlagStatus SysTick_GetFlagStatus(u8 SysTick_FLAG);
```

1）配置 SysTick 时钟源

```
void SysTick_CLKSourceConfig(u32 SysTick_CLKSource);
```

参数说明：

★ SysTick_CLKSource：SysTick 时钟源，在 stm32f10x_systick.h 中定义如下：

```
#define SysTick_CLKSource_HCLK_Div8  ((u32)0xFFFFFFFB)   // HCLK/8
#define SysTick_CLKSource_HCLK       ((u32)0x00000004)   // HCLK
```

SysTick_CLKSourceConfig()函数的代码在 stm32f10x_systick.c 中，核心语句如下：

```
SysTick->CTRL |= SysTick_CLKSource_HCLK;
SysTick->CTRL &= SysTick_CLKSource_HCLK_Div8;
```

2）设置 SysTick 重装值

```
void SysTick_SetReload(u32 Reload);
```

参数说明：

★ Reload：SysTick 重装值，取值范围是 1~0xFFFFFF（24 位），重装值=定时值*时钟频率，例如，时钟频率为 8MHz，定时值为 1s，重装值为 8000000

SysTick_SetReload()函数的代码在 stm32f10x_systick.c 中，核心语句如下：

```
SysTick->LOAD = Reload;
```

3）使能 SysTick

```
void SysTick_CounterCmd(u32 SysTick_Counter);
```

参数说明：

★ SysTick_Counter：SysTick 新状态，在 stm32f10x_systick.h 中定义如下：

```
#define SysTick_Counter_Disable      ((u32)0xFFFFFFFE)   // 禁止计数
#define SysTick_Counter_Enable       ((u32)0x00000001)   // 允许计数
#define SysTick_Counter_Clear        ((u32)0x00000000)   // 清除计数
```

SysTick_CounterCmd()函数的代码在 stm32f10x_systick.c 中，核心语句如下：

```
SysTick->CTRL |= SysTick_Counter_Enable;
SysTick->CTRL &= SysTick_Counter_Disable;
SysTick->VAL = SysTick_Counter_Clear;
```

4）使能 SysTick 中断

```
void SysTick_ITConfig(FunctionalState NewState);
```

参数说明：

★ NewState：中断新状态，ENABLE（1）—允许，DISABLE（0）—禁止

SysTick_ITConfig()函数的代码在 stm32f10x_systick.c 中，核心语句如下：

```
SysTick->CTRL |= CTRL_TICKINT_Set;
SysTick->CTRL &= CTRL_TICKINT_Reset;
```

CTRL_TICKINT_Set 和 CTRL_TICKINT_Reset 在 stm32f10x_systick.c 中定义如下：

```
#define CTRL_TICKINT_Set             ((u32)0x00000002)
#define CTRL_TICKINT_Reset           ((u32)0xFFFFFFFD)
```

5）获取 SysTick 计数值

```
u32 SysTick_GetCounter(void);
```

返回值：SysTick 计数值

SysTick_GetCounter()函数的代码在 stm32f10x_systick.c 中，核心语句如下：

```
return(SysTick->VAL);
```

6）获取 SysTick 标志状态

```
FlagStatus SysTick_GetFlagStatus(u8 SysTick_FLAG);
```

参数说明：

★ SysTick_FLAG：SysTick 标志，在 stm32f10x_systick.h 中定义如下：

```
#define SysTick_FLAG_COUNT          ((u32)0x00000010)
#define SysTick_FLAG_SKEW           ((u32)0x0000001E)
#define SysTick_FLAG_NOREF          ((u32)0x0000001F)
```

返回值：SysTick 标志状态，SET（1）—置位，RESET（0）—复位

SysTick_GetFlagStatus()函数的代码在 stm32f10x_systick.c 中，核心语句如下：

```
statusreg = SysTick->CTRL;
statusreg = SysTick->CALIB;
```

在 Keil 的调试界面选择“Peripherals”（设备）菜单下“CorePeripherals”（内核设备）子菜单中的“System Tick Timer”（系统滴答定时器），可以打开如图 1.6 所示对话框。

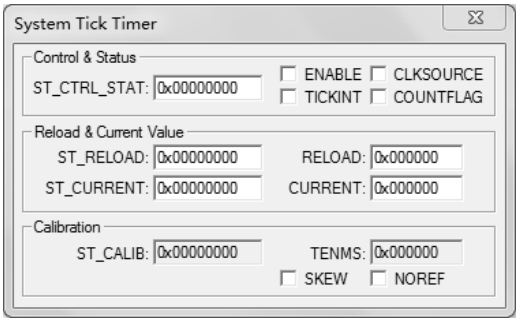


图 1.6　系统滴答定时器对话框