第 1 章　RCC—复位时钟控制

本章参考资料：《STM32 中文参考手册》RCC 章节。

学习本章时，配合《STM32 中文参考手册》RCC 章节一起阅读，效果会更佳，特别是涉及到寄存器说明的部分。

RCC ：reset clock control 复位和时钟控制器。本章我们主要讲解时钟部分，特别是要着重理解时钟树，理解了时钟树，STM32 的一切时钟的来龙去脉都会了如指掌。

1.1 RCC 主要作用—时钟部分

设置系统时钟 SYSCLK、设置 AHB 分频因子（决定 HCLK 等于多少）、设置 APB2 分频因子（决定 PCLK2 等于多少）、设置 APB1 分频因子（决定 PCLK1 等于多少）、设置各个外设的分频因子；控制 AHB、APB2 和 APB1 这三条总线时钟的开启、控制每个外设的时钟的开启。对于 SYSCLK、HCLK、PCLK2、PCLK1 这四个时钟的配置一般是：PCLK2 = HCLK = SYSCLK=PLLCLK = 72M，PCLK1=HCLK/2 = 36M。这个时钟配置也是库函数的标准配置，我们用的最多的就是这个。

1.2 RCC 框图剖析—时钟部分

时钟树单纯讲理论的话会比较枯燥，如果选取一条主线，并辅以代码，先主后次讲解的话会很容易，而且记忆还更深刻。我们这里选取库函数时钟系统时钟函数：SetSysClockTo72(); 以这个函数的编写流程来讲解时钟树，这个函数也是我们用库的时候默认的系统时钟设置函数。该函数的功能是利用 HSE 把时钟设置为：PCLK2 = HCLK = SYSCLK = 72M，PCLK1=HCLK/2 = 36M。下面我们就以这个代码的流程为主线，来分析时钟树，对应的是图中的黄色部分，代码流程在时钟树中以数字的大小顺序标识。

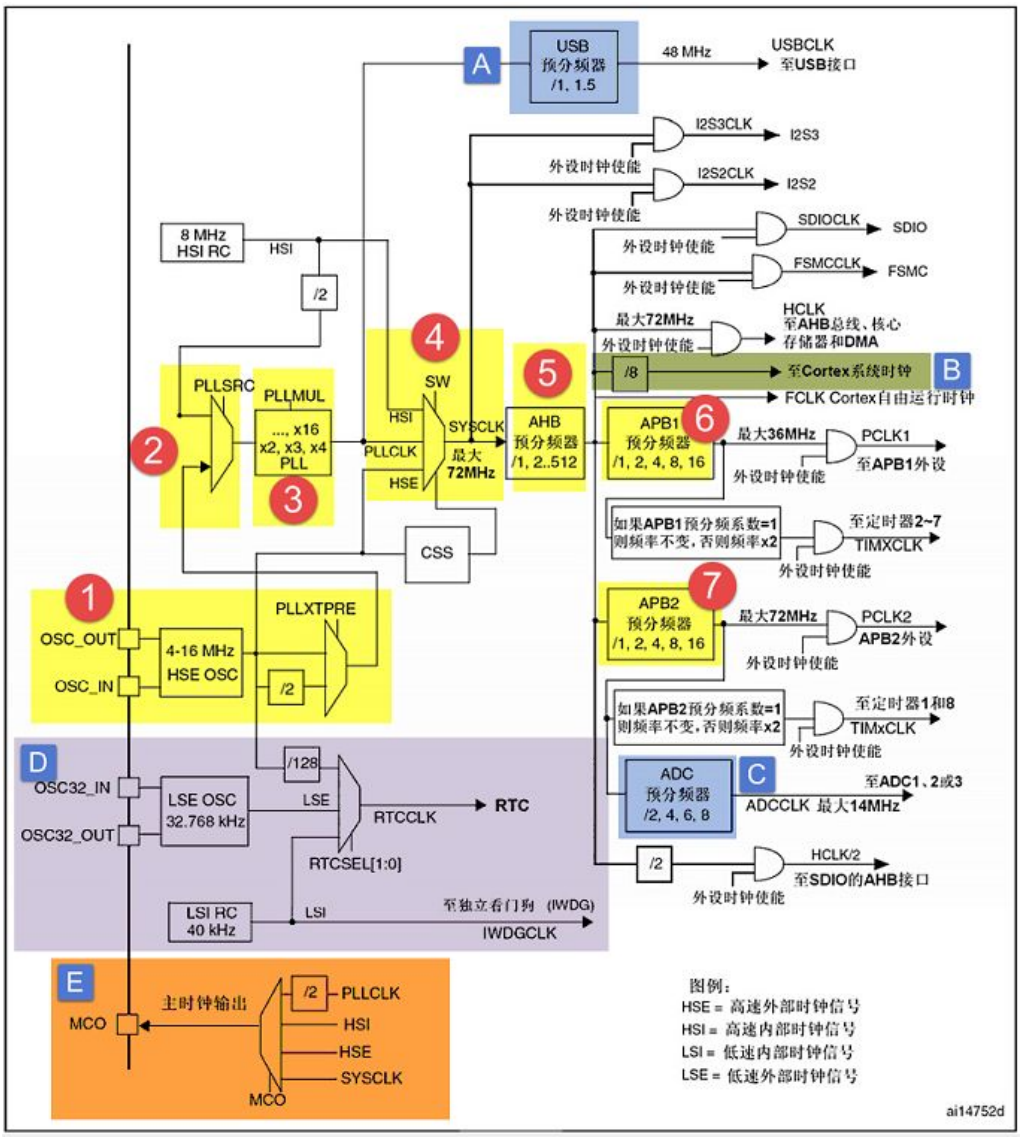


图 1 STM32 时钟树

1.2.1 系统时钟

1. ①HSE 高速外部时钟信号

HSE 是高速的外部时钟信号，可以由有源晶振或者无源晶振提供，频率从 4-16MHZ 不等。当使用有源晶振时，时钟从 OSC_IN 引脚进入，OSC_OUT 引脚悬空，当选用无源晶振时，时钟从 OSC_IN 和 OSC_OUT 进入，并且要配谐振电容。

HSE 最常使用的就是 8M 的无源晶振。当确定 PLL 时钟来源的时候，HSE 可以不分频或者 2 分频，这个由时钟配置寄存器 CFGR 的位 17：PLLXTPRE 设置，我们设置为 HSE 不分频。

2. ②PLL 时钟源

PLL 时钟来源可以有两个，一个来自 HSE，另外一个是 HSI/2，具体用哪个由时钟配置寄存器 CFGR 的位 16：PLLSRC 设置。HSI 是内部低速的时钟信号，频率为 8M，根据温度和环境的情况频率会有漂移，一般不作为 PLL 的时钟来源。这里我们选 HSE 作为 PLL 的时钟来源。

3. ③PLL 时钟 PLLCLK

通过设置 PLL 的倍频因子，可以对 PLL 的时钟来源进行倍频，倍频因子可以是:[2,3,4,5,6,7,8,9,10,11,12,13,14,15,16]，具体设置成多少，由时钟配置寄存器 CFGR 的位 21-18：PLLMUL[3:0]设置。我们这里设置为 9 倍频，因为上一步我们设置 PLL 的时钟来源为 HSE=8M，所以经过 PLL 倍频之后的 PLL 时钟：PLLCLK = 8M *9 = 72M。72M 是 ST 官方推荐的稳定运行时钟，如果你想超频的话，增大倍频因子即可，最高为 128M。我们这里设置 PLL 时钟：PLLCLK = 8M *9 = 72M。

4. ④系统时钟 SYSCLK

系统时钟来源可以是：HSI、PLLCLK、HSE，具体的时钟配置寄存器 CFGR 的位 1-0：SW[1:0]设置。我们这里设置系统时钟：SYSCLK = PLLCLK = 72M。

5. ⑤AHB 总线时钟 HCLK

系统时钟 SYSCLK 经过 AHB 预分频器分频之后得到时钟叫 APB 总线时钟，即 HCLK，分频因子可以是:[1,2,4，8，16，64，128，256，512]，具体的由时钟配置寄存器 CFGR 的位 7-4 ：HPRE[3:0]设置。片上大部分外设的时钟都是经过 PCLK 分频得到，至于 AHB 总线上的外设的时钟设置为多少，得等到我们使用该外设的时候才设置，我们这里只需粗线条的设置好 APB 的时钟即可。我们这里设置为 1 分频，即 HCLK=SYSCLK=72M。

6. ⑥APB2 总线时钟 HCLK2

APB2 总线时钟 PCLK2 由 HCLK 经过高速 APB2 预分频器得到，分频因子可以是:[1,2,4，8，16]，具体由时钟配置寄存器 CFGR 的位 13-11：PPRE2[2:0]决定。HCLK2 属于高速的总线时钟，片上高速的外设就挂载到这条总线上，比如全部的 GPIO、USART1、SPI1 等。至于 APB2 总线上的外设的时钟设置为多少，得等到我们使用该外设的时候才设置，我们这里只需粗线条的设置好 APB2 的时钟即可。我们这里设置为 1 分频，即 PCLK2 = HCLK = 72M。

7. ⑦APB1 总线时钟 HCLK1

APB1 总线时钟 PCLK1 由 HCLK 经过低速 APB 预分频器得到，分频因子可以是:[1,2,4，8，16]，具体的由时钟配置寄存器 CFGR 的位 10-8：PRRE1[2:0]决定。HCLK1 属于低速的总线时钟，最高为 36M，片上低速的外设就挂载到这条总线上，比如 USART2/3/4/5、SPI2/3，I2C1/2 等。至于 APB1 总线上的外设的时钟设置为多少，得等到我们使用该外设的时候才设置，我们这里只需粗线条的设置好 APB1 的时钟即可。我们这里设置为 2 分频，即 PCLK1 = HCLK/2 = 36M。

8. 设置系统时钟库函数

上面的 7 个步骤对应的设置系统时钟库函数如下，为了方便阅读，我把互联型相关的代码删掉，把英文注释翻译成了中文，并把代码标上了序号，总共七个步骤。该函数是直接操作寄存器的，有关寄存器部分请参考数据手册的 RCC 的寄存器描述部分。

代码 1 设置系统时钟库函数

```
1  static void SetSysClockTo72(void)
2  {
3      __IO uint32_t StartUpCounter = 0, HSEStatus = 0;
4
5      /1 ①使能 HSE，并等待 HSE 稳定
6      RCC->CR |= ((uint32_t)RCC_CR_HSEON);
7
8      // 等待 HSE 启动稳定，并做超时处理
9      do {
10         HSEStatus = RCC->CR & RCC_CR_HSERDY;
11         StartUpCounter++;
12     } while ((HSEStatus == 0) &&(StartUpCounter !=SE_STARTUP_TIMEOUT));
13
14     if ((RCC->CR & RCC_CR_HSERDY) != RESET) {
15         HSEStatus = (uint32_t)0x01;
16     } else {
17         HSEStatus = (uint32_t)0x00;
18     }
19     // HSE 启动成功，则继续往下处理
20     if (HSEStatus == (uint32_t)0x01) {
21         // 这部分是操作 FLASH 闪存用到的，如果不操作 FLASH，这两个注释掉也没影响
22         //------------------------------------------------------------
23         // 使能 FLASH 预存取缓冲区 */
24         FLASH->ACR |= FLASH_ACR_PRFTBE;
25
26         // SYSCLK 周期与闪存访问时间的比例设置，这里统一设置成 2
27         // 设置成 2 的时候，SYSCLK 低于 48M 也可以工作，如果设置成 0 或者 1 的时候，
28         // 如果配置的 SYSCLK 超出了范围的话，则会进入硬件错误，程序就死了
29         // 0: 0 < SYSCLK <= 24M
30         // 1: 24< SYSCLK <= 48M
31         // 2: 48< SYSCLK <= 72M */
32         FLASH->ACR &= (uint32_t)((uint32_t)~FLASH_ACR_LATENCY);
33         FLASH->ACR |= (uint32_t)FLASH_ACR_LATENCY_2;
34         //------------------------------------------------------------
35
36         // ②设置 AHB、APB2、APB1 预分频因子
37         // HCLK = SYSCLK
```

```
38          RCC->CFGR |= (uint32_t)RCC_CFGR_HPRE_DIV1;
39          //PCLK2 = HCLK
40          RCC->CFGR |= (uint32_t)RCC_CFGR_PPRE2_DIV1;
41          //PCLK1 = HCLK/2
42          RCC->CFGR |= (uint32_t)RCC_CFGR_PPRE1_DIV2;
43
44          //③设置 PLL 时钟来源，设置 PLL 倍频因子，PLLCLK = HSE * 9 = 72 MHz
45          RCC->CFGR &= (uint32_t)((uint32_t)
46                                  ~(RCC_CFGR_PLLSRC
47                                    | RCC_CFGR_PLLXTPRE
48                                    | RCC_CFGR_PLLMULL));
49          RCC->CFGR |= (uint32_t)(RCC_CFGR_PLLSRC_HSE
50                                  | RCC_CFGR_PLLMULL9);
51
52          // ④使能 PLL
53          RCC->CR |= RCC_CR_PLLON;
54
55          // ⑤等待 PLL 稳定
56          while ((RCC->CR & RCC_CR_PLLRDY) == 0) {
57          }
58
59          // ⑥选择 PLL 作为系统时钟来源
60          RCC->CFGR &= (uint32_t)((uint32_t)~(RCC_CFGR_SW));
61          RCC->CFGR |= (uint32_t)RCC_CFGR_SW_PLL;
62
63          // ⑦读取时钟切换状态位，确保 PLLCLK 被选为系统时钟
64          while ((RCC->CFGR&(uint32_t)RCC_CFGR_SWS) != (uint32_t)0x08){
65          }
66      } else {// 如果 HSE 启动失败，用户可以在这里添加错误代码出来
67      }
68  }
```

1.2.2 其他时钟

通过对系统时钟设置的讲解，整个时钟树我们已经把握的有六七成，剩下的时钟部分我们讲解几个重要的。

1. A、USB 时钟

USB 时钟是由 PLLCLK 经过 USB 预分频器得到，分频因子可以是：[1,1.5]，具体的由时钟配置寄存器 CFGR 的位 22：USBPRE 配置。USB 的时钟最高是 48M，根据分频因子反推过来算，PLLCLK 只能是 48M 或者是 72M。一般我们设置 PLLCLK=72M，USBCLK=48M。USB 对时钟要求比较高，所以 PLLCLK 只能是由 HSE 倍频得到，不能使用 HSI 倍频。

2. B、Cortex 系统时钟

Cortex 系统时钟由 HCLK 8 分频得到，等于 9M，Cortex 系统时钟用来驱动内核的定时器 SysTick，SysTick 一般用于操作系统的时钟节拍，也可以用做普通的定时。

3. C、ADC 时钟

ADC 时钟由 PCLK2 经过 ADC 预分频器得到，分频因子可以是[2,4,6,8]，具体的由时钟配置寄存器 CFGR 的位 15-14：ADCPRE[1:0]决定。很奇怪的是怎么没有 1 分频。ADC 时钟最高只能是 14M，如果采样周期设置成最短的 1.5 个周期的话，ADC 的转换时间可以达到最短的 1us。如果真要达到最短的转换时间 1us 的话，那 ADC 的时钟就得是 14M，反推 PCLK2 的时钟只能是：28M、56M、84M、112M，鉴于 PCLK2 最高是 72M，所以只能取 28M 和 56M。

4. D、RTC 时钟、独立看门狗时钟

RTC 时钟可由 HSE/128 分频得到，也可由低速外部时钟信号 LSE 提供，频率为 32.768KHZ，也可由低速内部时钟信号 HSI 提供，具体选用哪个时钟由备份域控制寄存器 BDCR 的位 9-8：RTCSEL[1:0]配置。独立看门狗的时钟由 LSI 提供，且只能是由 HSI 提供，LSI 是低速的内部时钟信号，频率为 30~60KHZ 直接不等，一般取 40KHZ。

5. E、MCO 时钟输出

MCO 是 microcontroller clock output 的缩写，是微控制器时钟输出引脚，在 STM32 F1 系列中 由 PA8 复用所得，主要作用是可以对外提供时钟，相当于一个有源晶振。MCO 的时钟来源可以是：PLLCLK/2、HSI、HSE、SYSCLK，具体选哪个由时钟配置寄存器 FGR 的位 26-24：MCO[2:0]决定。除了对外提供时钟这个作用之外，我们还可以通过示波器监控 MCO 引脚的时钟输出来验证我们的系统时钟配置是否正确。

1.3 自定义系统时钟实验

1.3.1 使用 HSE

一般情况下，我们都是使用 HSE，然后 HSE 经过 PLL 倍频之后作为系统时钟。通常的配置是：HSE=8M，PLL 的倍频因子为：9，系统时钟就设置成:SYSCLK = 8M * 9 = 72M。使用 HSE，系统时钟 SYSCLK 最高是 128M。我们使用的库函数就是这么干的， 当程序来到 main 函数之前，启动文件：statup_stm32f10x_hd.s 已经调用 SystemInit()函数把系统时钟初始化成 72MHZ，SystemInit()在库文件：system_stm32f10x.c 中定义。如果我们想把系统时钟设置低一点或者超频的话，可以修改底层的库文件，但是为了维持库的完整性，我们可以根据时钟树的流程自行写一个。

1.3.2 使用 HSI

当 HSE 故障的时候，因为 PLL 的时钟来源是 HSE，所以当 HSE 故障的时候，不仅 HSE 不能使用，连 PLL 也会被关闭，这个时候系统会自动切换 HSI 作为系统时钟，此时 SYSCLK=HSI=8M，如果没有开启 CSS 和 CSS 中断的话，那么整个系统就只能在低速率运行，这是系统跟瘫痪没什么两样。如果开启了 CSS 功能的话，那么可以当 HSE 故障时，在 CSS 中断里面采取补救措施，使用 HSI，并把系统时钟设置为更高的频率，最高是 64M，64M 的频率足够一般的外设使用，如：ADC 、SPI、I2C 等。但是这里就又有一个问题了，原来 SYSCLK=72M，现在因为故障改成 64M，那么那些外设的时钟肯定被改变了，那么外设工作就会被打乱，那我们是不是在设置 HSI 时钟的时候，也重新调整外设总线的分频因子，即 AHB，APB2 和 APB1 的分频因子，使外设的时钟达到跟 HSE 没有故障之前一样。但是这个也不是最保障的办法，毕竟不能一直使用 HSI，所以当 HSE 故障时还是要采取报警措施。

还有一种情况是，有些用户不想用 HSE，想用 HSI，但是又不知道怎么用 HSI 来设置系统时钟，因为调用库函数都是使用 HSE，下面我们给出个使用 HSI 配置系统时钟例子，起个抛砖引玉的作用。

1.3.3 硬件设计

1、RCC

2、LED 一个

RCC 是单片机内部资源，不需要外部电路。通过 LED 闪烁的频率来直观的判断不同系统时钟频率对软件延时的效果。

1.3.4 软件设计

我们编写两个 RCC 驱动文件，bsp_clkconfig.h 和 bsp_clkconfig.c，用来存放 RCC 系统时钟配置函数。

1. 编程要点

编程要点对应着时钟树图中的序号。

1、开启 HSE/HSI ，并等待 HSE/HSI 稳定

2、设置 AHB、APB2、APB1 的预分频因子

3、设置 PLL 的时钟来源，和 PLL 的倍频因子，设置各种频率主要就是在这里设置

4、开启 PLL，并等待 PLL 稳定

5、把 PLLCK 切换为系统时钟 SYSCLK

6、读取时钟切换状态位，确保 PLLCLK 被选为系统时钟

2. 代码分析

这里只讲解核心的部分代码，有些变量的设置，头文件的包含等并没有涉及到，完整的代码请参考本章配套的工程。

使用 HSE 配置系统时钟

代码 2 HSE 作为系统时钟来源

```
1  void HSE_SetSysClock(uint32_t pllmul)
2  {
3      __IO uint32_t StartUpCounter = 0, HSEStartUpStatus = 0;
4
5      // 把 RCC 外设初始化成复位状态
6      RCC_DeInit();
7
8      //使能 HSE，开启外部晶振，秉火开发板用的是 8M
9      RCC_HSEConfig(RCC_HSE_ON);
10
11     // 等待 HSE 启动稳定
12     HSEStartUpStatus = RCC_WaitForHSEStartUp();
13
14     // 只有 HSE 稳定之后则继续往下执行
15     if (HSEStartUpStatus == SUCCESS) {
16 //-------------------------------------------------------------------//
17         // 这两句是操作 FLASH 闪存用到的，如果不操作 FLASH，这两个注释掉也没影响
18         // 使能 FLASH 预存取缓冲区
19         FLASH_PrefetchBufferCmd(FLASH_PrefetchBuffer_Enable);
20
21         // SYSCLK 周期与闪存访问时间的比例设置，这里统一设置成 2
22         // 设置成 2 的时候，SYSCLK 低于 48M 也可以工作，如果设置成 0 或者 1 的时候，
23         // 如果配置的 SYSCLK 超出了范围的话，则会进入硬件错误，程序就死了
24         // 0: 0 < SYSCLK <= 24M
25         // 1: 24< SYSCLK <= 48M
26         // 2: 48< SYSCLK <= 72M
27         FLASH_SetLatency(FLASH_Latency_2);
28 //-------------------------------------------------------------------//
29
30         // AHB 预分频因子设置为 1 分频，HCLK = SYSCLK
31         RCC_HCLKConfig(RCC_SYSCLK_Div1);
32
33         // APB2 预分频因子设置为 1 分频，PCLK2 = HCLK
34         RCC_PCLK2Config(RCC_HCLK_Div1);
35
36         // APB1 预分频因子设置为 1 分频，PCLK1 = HCLK/2
37         RCC_PCLK1Config(RCC_HCLK_Div2);
38
39 //-----------------设置各种频率主要就是在这里设置------------------//
40         // 设置 PLL 时钟来源为 HSE，设置 PLL 倍频因子
41         // PLLCLK = 8MHz * pllmul
42         RCC_PLLConfig(RCC_PLLSource_HSE_Div1, pllmul);
43 //-------------------------------------------------------------------//
```

```
44
45          // 开启 PLL
46          RCC_PLLCmd(ENABLE);
47
48          // 等待 PLL 稳定
49          while (RCC_GetFlagStatus(RCC_FLAG_PLLRDY) == RESET) {
50          }
51
52          // 当 PLL 稳定之后，把 PLL 时钟切换为系统时钟 SYSCLK
53          RCC_SYSCLKConfig(RCC_SYSCLKSource_PLLCLK);
54
55          // 读取时钟切换状态位，确保 PLLCLK 被选为系统时钟
56          while (RCC_GetSYSCLKSource() != 0x08) {
57          }
58      } else {
59          // 如果 HSE 开启失败，那么程序就会来到这里，用户可在这里添加出错的代码处理
60          // 当 HSE 开启失败或者故障的时候，单片机会自动把 HSI 设置为系统时钟，
61          // HSI 是内部的高速时钟，8MHZ
62          while (1) {
63          }
64      }
65  }
```

这个函数采用库函数编写， 函数有个形参 pllmul，pllmul 用来设置 PLL 的倍频因子，在调用的时候形参可以是：RCC_PLLMul_x , x:[2,3,...16]，这些宏来源于库函数的定义，宏展开是一些 32 位的十六进制数，具体功能是配置了时钟配置寄存器 CFGR 的位 21-18 PLLMUL[3:0]，预先定义好倍频因子，方便调用。

函数调用举例：HSE_SetSysClock(RCC_PLLMul_9); 则设置系统时钟为：8MHZ * 9 = 72MHZ。 HSE_SetSysClock(RCC_PLLMul_16); 则设置系统时钟为：8MHZ * 16 = 128MHZ 超频慎用。

使用 HSI 配置系统时钟

```
1  void HSI_SetSysClock(uint32_t pllmul)
2  {
3      __IO uint32_t HSIStartUpStatus = 0;
4
5      // 把 RCC 外设初始化成复位状态
6      RCC_DeInit();
7
8      //使能 HSI
9      RCC_HSICmd(ENABLE);
10
11     // 等待 HSI 就绪
12     HSIStartUpStatus = RCC->CR & RCC_CR_HSIRDY;
13
14     // 只有 HSI 就绪之后则继续往下执行
15     if (HSIStartUpStatus == RCC_CR_HSIRDY) {
16     //------------------------------------------------------------//
17         // 这两句是操作 FLASH 闪存用到的，如果不操作 FLASH，这两个注释掉也没影响
18         // 使能 FLASH 预存取缓冲区
19         FLASH_PrefetchBufferCmd(FLASH_PrefetchBuffer_Enable);
20
21         // SYSCLK 周期与闪存访问时间的比例设置，这里统一设置成 2
22         // 设置成 2 的时候，SYSCLK 低于 48M 也可以工作，如果设置成 0 或者 1 的时候，
23         // 如果配置的 SYSCLK 超出了范围的话，则会进入硬件错误，程序就死了
24         // 0: 0 < SYSCLK <= 24M
25         // 1: 24< SYSCLK <= 48M
26         // 2: 48< SYSCLK <= 72M
```

```
        FLASH_SetLatency(FLASH_Latency_2);
    //-------------------------------------------------------------//

        // AHB 预分频因子设置为 1 分频，HCLK = SYSCLK
        RCC_HCLKConfig(RCC_SYSCLK_Div1);

        // APB2 预分频因子设置为 1 分频，PCLK2 = HCLK
        RCC_PCLK2Config(RCC_HCLK_Div1);

        // APB1 预分频因子设置为 1 分频，PCLK1 = HCLK/2
        RCC_PCLK1Config(RCC_HCLK_Div2);

        //-----------设置各种频率主要就是在这里设置-------------------//
        // 设置 PLL 时钟来源为 HSE，设置 PLL 倍频因子
        // PLLCLK = 4MHz * pllmul
        RCC_PLLConfig(RCC_PLLSource_HSI_Div2, pllmul);
       //-- ----------------------------------------------------//

        // 开启 PLL
        RCC_PLLCmd(ENABLE);

        // 等待 PLL 稳定
        while (RCC_GetFlagStatus(RCC_FLAG_PLLRDY) == RESET) {
        }

        // 当 PLL 稳定之后，把 PLL 时钟切换为系统时钟 SYSCLK
        RCC_SYSCLKConfig(RCC_SYSCLKSource_PLLCLK);

        // 读取时钟切换状态位，确保 PLLCLK 被选为系统时钟
        while (RCC_GetSYSCLKSource() != 0x08) {
        }
    } else {
        // 如果 HSI 开启失败，那么程序就会来到这里，用户可在这里添加出错的代码处理
        // 当 HSE 开启失败或者故障的时候，单片机会自动把 HSI 设置为系统时钟，
        // HSI 是内部的高速时钟，8MHZ
        while (1) {
        }
    }
}
```

HSI 设置系统时钟函数跟 HSE 设置系统时钟函数在原理上是一样的，有一个区别的地方就是，HSI 必须 2 分频之后才能作为 PLL 的时钟来源，所以使用 HSI 时，最大的系统时钟 SYSCLK 只能是 HSI/2*16=4*16=64MHZ。

函数调用举例：HSI_SetSysClock(RCC_PLLMul_9); 则设置系统时钟为：4MHZ * 9 = 36MHZ。

软件延时

```
void Delay(__IO uint32_t nCount)
{
    for (; nCount != 0; nCount--);
}
```

软件延时函数，使用不同的系统时钟，延时时间不一样，可以通过 LED 闪烁的频率来判断。

MCO 输出

在 STM32F103 系列中，PA8 可以复用为 MCO 引脚，对外提供时钟输出，我们也可以用示波器监控该引脚的输出来判断我们的系统时钟是否设置正确。

代码 3 MCO GPIO 初始化

```
/*
 * 初始化 MCO 引脚 PA8
 * 在 F103 系列中 MCO 引脚只有一个，即 PA8，在 F4 系列中，MCO 引脚有两个
 */
void MCO_GPIO_Config(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;
    // 开启 GPIOA 的时钟
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);

    // 选择 GPIO8 引脚
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_8;

    //设置为复用功能推挽输出
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;

    //设置 IO 的翻转速率为 50M
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;

    // 初始化 GPIOA8
    GPIO_Init(GPIOA, &GPIO_InitStructure);
}
```

代码 4 MCO 输出时钟选择

```
// 设置 MCO 引脚输出时钟，用示波器即可在 PA8 测量到输出的时钟信号，
// 我们可以把 PLLCLK/2 作为 MCO 引脚的时钟来检测系统时钟是否配置准确
// MCO 引脚输出可以是 HSE,HSI,PLLCLK/2,SYSCLK
//RCC_MCOConfig(RCC_MCO_HSE);
//RCC_MCOConfig(RCC_MCO_HSI);
//RCC_MCOConfig(RCC_MCO_PLLCLK_Div2);
RCC_MCOConfig(RCC_MCO_SYSCLK);
```

我们初始化 MCO 引脚之后，可以直接调用库函数 RCC_MCOConfig()来选择 MCO 时钟来源。

主函数

```
int main(void)
{
    // 程序来到 main 函数之前，启动文件：statup_stm32f10x_hd.s 已经调用
    // SystemInit()函数把系统时钟初始化成 72MHZ
    // SystemInit()在 system_stm32f10x.c 中定义
    // 如果用户想修改系统时钟，可自行编写程序修改

    // 重新设置系统时钟，这时候可以选择使用 HSE 还是 HSI

    // 使用 HSE 时，SYSCLK = 8M * RCC_PLLMul_x, x:[2,3,...16],最高是 128M
    HSE_SetSysClock(RCC_PLLMul_9);

    // 使用 HSI 时，SYSCLK = 4M * RCC_PLLMul_x, x:[2,3,...16],最高是 64MH
    //HSI_SetSysClock(RCC_PLLMul_16);

    // MCO 引脚初始化
```

```
17      MCO_GPIO_Config();
18
19      // 设置 MCO 引脚输出时钟，用示波器即可在 PA8 测量到输出的时钟信号，
20      // 我们可以把 PLLCLK/2 作为 MCO 引脚的时钟来检测系统时钟是否配置准确
21      // MCO 引脚输出可以是 HSE,HSI,PLLCLK/2,SYSCLK
22      //RCC_MCOConfig(RCC_MCO_HSE);
23      //RCC_MCOConfig(RCC_MCO_HSI);
24      //RCC_MCOConfig(RCC_MCO_PLLCLK_Div2);
25      RCC_MCOConfig(RCC_MCO_SYSCLK);
26
27      // LED 端口初始化
28      LED_GPIO_Config();
29      while (1) {
30          LED1( ON );         // 亮
31          Delay(0x0FFFFF);
32          LED1( OFF );        // 灭
33          Delay(0x0FFFFF);
34      }
35  }
```

在主函数中，可以调用 HSE_SetSysClock()或者 HSI_SetSysClock()这两个函数把系统时钟设置成各种常用的时钟，然后通过 MCO 引脚监控，或者通过 LED 闪烁的快慢体验不同的系统时钟对同一个软件延时函数的影响。

1.3.5 下载验证

把编译好的程序下载到开发板，可以看到设置不同的系统时钟时，LED 闪烁的快慢不一样。更精确的数据我们可以用示波器监控 MCO 引脚看到。

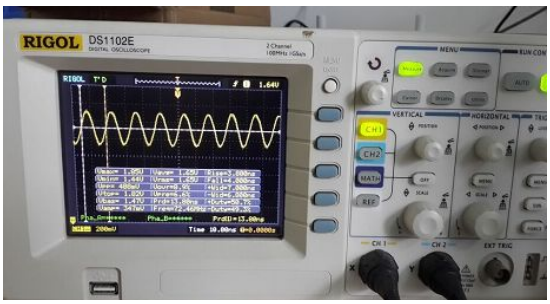


图 2 MCO=SYSCLK=72M

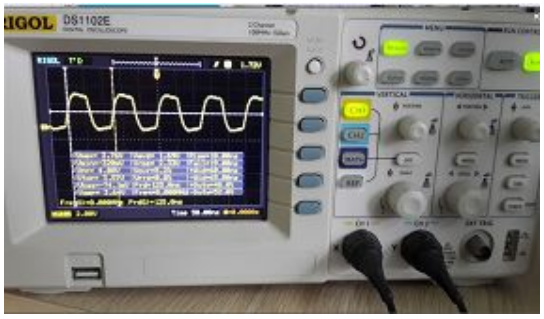


图 3 MCO=HSI=8M