第5章 定 时 器

本章将介绍 STM32F103C8T6 片内定时器的结构和用法，按照从简单到复杂的顺序依次介绍系统节拍定时器、看门狗定时器、实时时钟和通用定时器，其中，系统节拍定时器是 Cortex-M3 内核的定时器组件，主要用于为嵌入式实时操作系统提供时钟节拍（一般定时频率取为 100Hz）。STM32F103C8T6 具有 4 个定时器，其中定时器 1 为高级定时器、定时器 2～4 为通用定时器，本章将主要介绍通用定时器，且以定时器 2 为例。

本章的学习目标：

- 了解看门狗定时器与实时时钟；
- 熟悉系统节拍定时器的工作原理；
- 掌握系统节拍定时器的 HAL 类型程序设计方法；
- 熟练应用寄存器或 HAL 库进行通用定时器程序设计。

5.1 系统节拍定时器

系统节拍定时器 SysTick 是一个 24 位的减计数器，常用于产生 100Hz 的定时中断（即系统节拍定时器异常，见表 1-4），用作嵌入式实时操作系统的时钟节拍。

5.1.1 系统节拍定时器的工作原理

系统节拍定时器的结构如图 5-1 所示。

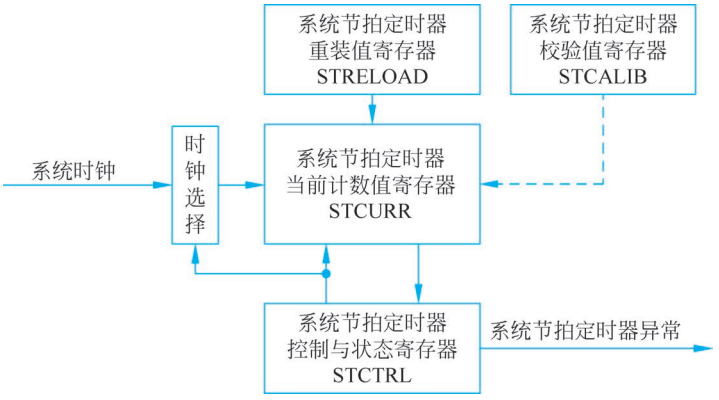

图 5-1 系统节拍定时器的结构

图 5-1 表明系统节拍定时器有 4 个相关的寄存器，即 STCTRL、STRELOAD、STCURR 和 STCALIB，了解了这 4 个寄存器的内容，即可掌握系统节拍定时器的工作原理。这 4 个寄存器的内容如表 5-1～表 5-4 所示。

表 5-1 系统节拍定时器控制与状态寄存器 STCTRL

位 号	符 号	复位值	含 义
0	ENABLE	0	写入 1，启动系统节拍定时器；写入 0，关闭系统节拍定时器
1	TICKINT	0	写入 1，开放系统节拍定时器定时中断；写入 0，关闭系统节拍定时器定时中断
2	CLKSOURCE	1	写入 1，选择系统时钟为系统节拍定时器时钟源；写入 0，选择外部时钟为系统节拍定时器时钟源，对于 STM32F103C8T6 无效
15:3	—	—	保留，仅能写入 0
16	COUNTFLAG	0	当系统节拍定时器减计数到 0 时，该位自动置位，读 STCTRL 寄存器时自动清零
31:17	—	—	保留，仅能写入 0

表 5-2 系统节拍定时器重装值寄存器 STRELOAD

位 号	符 号	复位值	含 义
23:0	RELOAD	0	系统节拍计数器计数到 0 后，下一个时钟节拍后将 RELOAD 的值装入 STCURR 寄存器中
31:24	—	—	保留，仅能写入 0

表 5-3 系统节拍定时器当前计数值寄存器 STCURR

位 号	符 号	复位值	含 义
32:0	CURRENT	0	可读出系统节拍定时器的当前定时值；写入任意值，都将清除 CURRENT 的值，并清除 STCTRL 寄存器的 COUNTFLAG 位
31:24	—	—	保留，仅能写入 0

表 5-4 系统节拍定时器校验值寄存器 STCALIB

位 号	符 号	复位值	含 义
23:0	TENMS	0x2328	当系统时钟为 9MHz 时，1ms 定时间隔的计数值，这里的 0x2328 为十进制数 9000
29:24	—	—	保留，仅能写入 0
30	SKEW	0	为 0 表示 TENMS 的值是准确的；为 1 表示 TENMS 的值是不准确的
31	NOREF	0	为 0 表示有独立的参考时钟；为 1 表示独立的参考时钟不可用

根据上述对系统节拍定时器的分析，可知设计一个定时频率为 100Hz（即定时周期为 10ms）的系统时钟节拍定时器，可采用以下语句（结合上述表 5-1～表 5-4）。

（1）配置 STCTRL 为（1uL << 1）|（1uL << 2），即关闭系统节拍定时器并开放系统节拍定时器中断，同时设置系统时钟为系统节拍定时器时钟源。此时对于 STM32F103C8T6 微控制器而言，系统时钟为 72MHz，芯片手册上明确说明系统时钟的 8 分频值用作系统节

拍定时器的输入时钟信号(见图 1-3),但实际测试发现,系统节拍定时器的输入时钟信号仍然是 72MHz,即没有所谓的 8 分频器。

(2) 向 STCURR 寄存器写入任意值,如写入 0,清除 STCURR 的值,同时清除 STCTRL 的 COUNTFLAG 位。

(3) 向 STRELOAD 寄存器写入 720000－1,即十六进制数 0x1193F。

(4) 配置 STCTRL 的第 0 位为 1(其余位保持不变),启动系统节拍定时器。

系统节拍定时器相关的寄存器定义在 CMSIS 库头文件 core_cm3.h 中,如程序段 5-1 所示。

程序段 5-1　系统节拍定时器相关的寄存器定义(摘自 core_cm3.h 文件)

```
1    typedef struct
2    {
3      __IO uint32_t CTRL;   // 偏移地址: 0x000 (可读/可写) 系统节拍定时器控制与状态寄存器
4      __IO uint32_t LOAD;   //偏移地址: 0x004 (可读/可写) 系统节拍定时器重装值寄存器
5      __IO uint32_t VAL;    //偏移地址: 0x008 (可读/可写) 系统节拍定时器当前计数值寄存器
6      __I uint32_t CALIB;   //偏移地址: 0x00C (只读) 系统节拍定时器校验值寄存器
7    } SysTick_Type;
8
9    #define SCS_BASE        (0xE000E000UL)
10   #define SysTick_BASE    (SCS_BASE + 0x0010UL)
11
12   #define SysTick         ((SysTick_Type *)SysTick_BASE)
```

系统节拍定时器的 4 个寄存器 STCTRL、STRELOAD、STCURR 和 STCALIB 的地址分别为 0xE000 E010、0xE000 E014、0xE000 E018 和 0xE000 E01C。上述程序段第 1～7 行自定义结构体类型 SysTick_Type 的各个成员与系统节拍定时器的 4 个寄存器按偏移地址一一对应(基地址为 0xE000 E010),因此,第 12 行的 SysTick 为指向系统节拍定时器的各个寄存器的结构体指针。

在 CMSIS 库头文件 core_cm3.h 中还定义了一个初始化系统节拍定时器的函数,如程序段 5-2 所示。

程序段 5-2　系统节拍定时器初始化函数(摘自 core_cm3.h 文件)

```
1    __STATIC_INLINE uint32_t SysTick_Config(uint32_t ticks)
2    {
3      if ((ticks - 1UL) > SysTick_LOAD_RELOAD_Msk)
4      {
5        return (1UL);
6      }
7      SysTick->LOAD = (uint32_t)(ticks - 1UL);
8      NVIC_SetPriority (SysTick_IRQn, (1UL << __NVIC_PRIO_BITS) - 1UL);
9      SysTick->VAL = 0UL;
10     SysTick->CTRL = SysTick_CTRL_CLKSOURCE_Msk |
11                     SysTick_CTRL_TICKINT_Msk |
12                     SysTick_CTRL_ENABLE_Msk;
13     return (0UL);
14   }
```

函数 SysTick_Config 用于初始化系统定时器 SysTick,参数 ticks 表示系统节拍定时器的计数初值。第 1 行的 uint32_t 为自定义的无符号 32 位整型类型,__STATIC_INLINE 即 static inline,用于定义静态内敛函数。第 3 行的 SysTick_LOAD_RELOAD_Msk 为宏常量 0x00FF FFFF,这是因为系统节拍定时器是 24 位的减计数器,最大值为 0x00FF FFFF,

所以，当第 3 行为真时，说明参数 ticks 的值超过了系统节拍定时器的最大计数值，故第 5 行返回 1，表示出错。第 7 行将 ticks 计数值减去 1 的值作为初值赋给 LOAD 寄存器（即系统节拍定时器重装值寄存器 STRELOAD），第 8 行调用 CMSIS 库函数 NVIC_SetPriority 设置系统节拍定时器异常的优先级号为 15（参考表 4-2 和程序段 4-2）。第 9 行向 VAL 寄存器（即系统节拍定时器当前计数值寄存器 STCURR）写入 0，使得 LOAD 内的值装入 VAL 寄存器中。第 10 行启动系统节拍定时器，并且打开系统节拍定时器中断，其中，宏常量 SysTick_CTRL_CLKSOURCE_Msk、SysTick_CTRL_TICKINT_Msk 和 SysTick_CTRL_ENABLE_Msk 依次为（1uL << 2）、（1uL << 1）和（1uL << 0）。

根据程序段 5-2 可知，设计一个定时频率为 100Hz（即定时周期为 10ms）的系统时钟节拍定时器，只需要调用语句 SysTick_Config(720000uL)即可。

5.1.2　系统节拍定时器的寄存器工程实例


视频讲解

系统节拍定时器异常一般用作嵌入式实时操作系统的时钟节拍，也可以用作普通的定时中断处理。这里使用系统节拍定时器实现 LED 灯 D3 的闪烁功能，其实现步骤如下。

（1）在工程 PRJ02 的基础上，新建寄存器类型的工程 PRJ03，保存在目录 D:\STM32F103C8T6REG\PRJ03 下，此时的工程 PRJ03 与工程 PRJ02 完全相同。

（2）新建文件 systick.c 和 systick.h，这两个文件保存在目录 D:\STM32F103C8T6REG\PRJ03\BSP 下，其代码分别如程序段 5-3 和程序段 5-4 所示。

程序段 5-3　文件 systick.c

```
1    //Filename:systick.c
2
3    #include "includes.h"
4
5    void SysTickInit(void)
6    {
7      SysTick_Config(720000uL);
8    }
9
10   void SysTick_Handler(void)
11   {
12     static Int08U i = 0;
13     i++;
14     if(i == 100)
15         LED(2,1);
16     if(i == 200)
17     {
18         i = 0;
19         LED(2,0);
20     }
21   }
```

第 5～8 行的函数 SysTickInit 调用系统函数 SysTick_Config（第 7 行），配置系统节拍定时器的工作频率为 100Hz。

第 10～21 行为系统节拍定时器异常服务函数，第 10 行的函数名 SysTick_Handler 是系统指定的，参考 1.6 节，该函数名来自启动文件 startup_stm32f10x_md.s 中同名的标号。第 12 行定义静态变量 i，如果 i 累加到 100（表示经过了 1s），则 LED 灯 D3 亮（第 15 行）；如

果 i 从 100 累加到 200(表示又经过了 1s),则 LED 灯 D3 灭(第 19 行),同时把变量 i 清零。

程序段 5-4　文件 systick.h

```
1    //Filename: systick.h
2
3    #ifndef _SYSTICK_H
4    #define _SYSTICK_H
5
6    void SysTickInit(void);
7
8    #endif
```

文件 systick.h 中声明了文件 systick.c 中定义的函数 SysTickInit(第 6 行),该函数用于系统节拍定时器的初始化。

(3) 修改文件 main.c、includes.h 和 bsp.c,分别如程序段 5-5～程序段 5-7 所示。

程序段 5-5　文件 main.c

```
1    //Filename: main.c
2
3    #include "includes.h"
4
5    int main(void)
6    {
7      BSPInit();
8      while(1)
9      {
10     }
11     return 0;
12   }
```

在文件 main.c 中,main 函数仅在第 7 行调用 BSPInit 函数实现外设的初始化,然后,进入一个空的无限循环体(第 8～10 行),因此,main 函数中不执行具体的处理工作。

程序段 5-6　文件 includes.h

```
1    //Filename: includes.h
2
3    #include "stm32f10x.h"
4
5    #include "vartypes.h"
6    #include "bsp.h"
7    #include "led.h"
8    #include "key.h"
9    #include "beep.h"
10   #include "systick.h"
```

相对于程序段 4-3 而言,这里添加了第 10 行,即包括 systick.h 头文件。

程序段 5-7　文件 bsp.c

```
1    //Filename: bsp.c
2
3    #include "includes.h"
4
5    void BSPInit()
6    {
7      LEDInit();
```

```
8       BEEPInit();
9       KEYInit();
10      EXTIKeyInit();
11      SysTickInit();
12    }
```

相对于程序段 4-4 而言，这里添加了第 11 行，即调用了系统节拍定时器初始化函数。

（4）将 systick.c 文件添加到工程管理器的 BSP 分组下，建设好的工程 PRJ03 如图 5-2 所示。

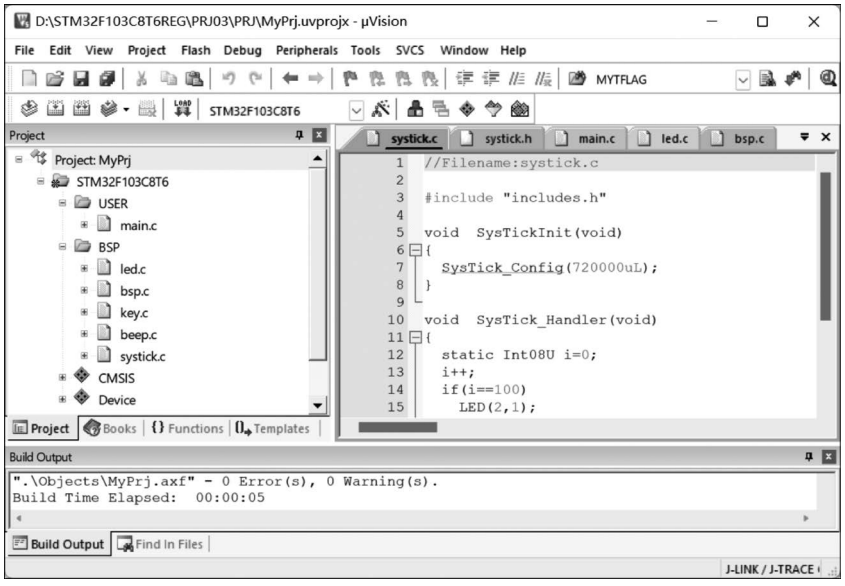

图 5-2　工程 PRJ03 工作窗口

工程 PRJ03 的工作流程如图 5-3 所示。

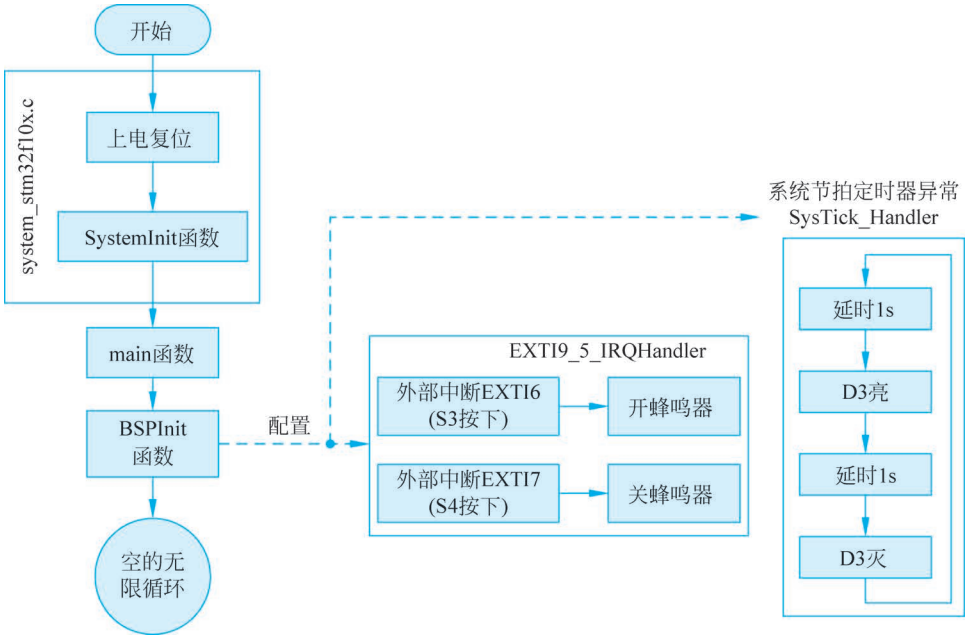

图 5-3　工程 PRJ03 的工作流程

由图 5-3 可知，在工程 PRJ03 中，主函数 main 主要完成了系统的外设初始化工作，同时，工程 PRJ03 保留了工程 PRJ02 中的按键处理功能，并添加了系统节拍定时器功能。由于配置系统节拍定时器的工作频率为 100Hz，所以，定时异常每触发 100 次相当于延时 1s。添加静态计数变量，使得系统节拍定时器异常服务函数实现了每隔 1s 使 LED 灯 D3 状态切换一次的功能。


视频讲解

5.1.3 系统节拍定时器的 HAL 工程实例

系统节拍定时器的 HAL 类型工程的建设方法如下。

(1) 在工程 HPrj02 的基础上，新建工程 HPrj03，保存在 D:\STM32F103C8T6HAL\HPrj03 目录下，此时的工程 HPrj03 与工程 HPrj02 完全相同。

(2) 基于 STM32CubeMX 生成的 HAL 类型工程在默认情况下，SysTick 定时器工作在 1000Hz 下，其中断处理函数为 HAL_IncTick，即每 1ms 执行一次该函数。

(3) 修改 mymain.c，如程序段 5-8 所示，即函数 mymain 中仅有一个空的无限循环体；新建文件 systick.c，其代码如程序段 5-9 所示。这两个文件保存在目录 D:\STM32F103C8T6HAL\HPrj03\BSP 下。

程序段 5-8 文件 mymain.c

```
1   //Filename: mymain.c
2
3   #include "includes.h"
4
5   void mymain(void)
6   {
7     while(1)
8     {
9     }
10  }
```

程序段 5-9 文件 systick.c

```
1   //Filename:systick.c
2
3   #include "includes.h"
4
5   void HAL_IncTick(void) //SysTick: 1kHz
6   {
7     static Int16U i = 0;
8     i++;
9     if(i == 1000)
10        LED(2,1);
11    if(i == 2000)
12    {
13        i = 0;
14        LED(2,0);
15    }
16  }
```

第 5 行的 HAL_IncTick 为 HAL 库函数，实质上 SysTick 定时器的中断回调函数，每 1ms 执行一次。第 7 行定义 16 位整型静态变量 i；每执行一次 HAL_IncTick 函数，i 自增 1；如果 i 累加到 1000(即经历 1s)，则执行第 9～10 行，LED 灯 D3 亮；如果 i 累加到 2000(即

又经历 1s)，则执行第 11～15 行，清零 i，同时 LED 灯 D3 灭。

(4) 将 systick.c 文件添加到工程管理器的 App/User/BSP 分组下，即完成工程 HPrj03 的建设工作，如图 5-4 所示。

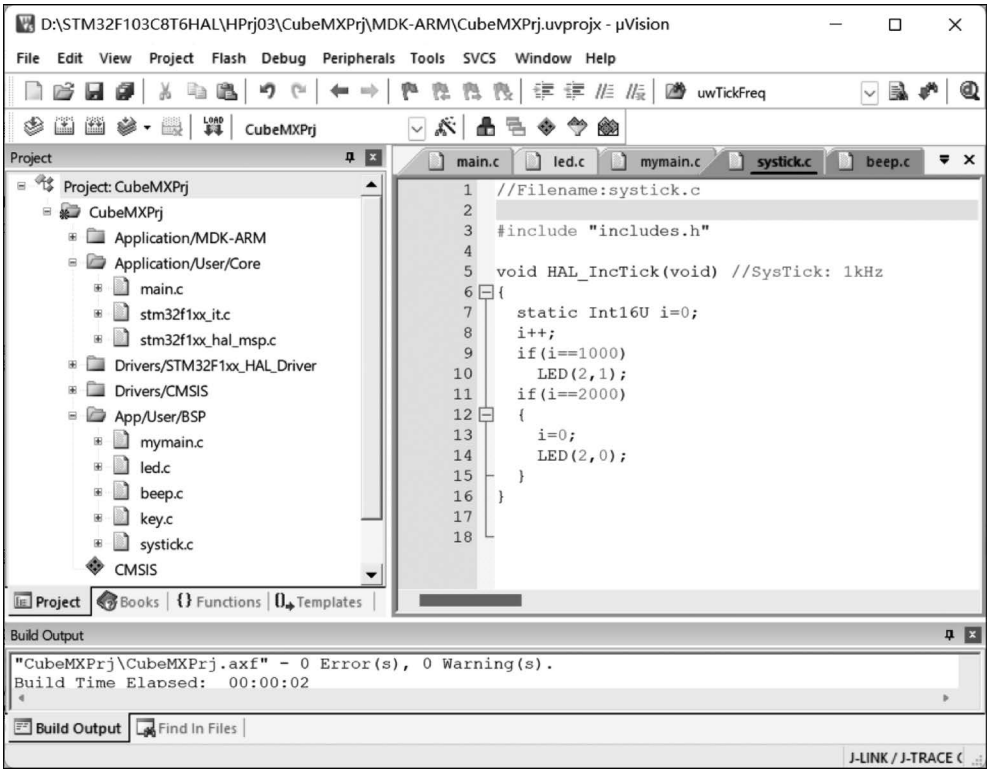

图 5-4　工程 HPrj03 工作窗口

5.2　看门狗定时器

STM32F103C8T6 微控制器中有两个看门狗，即独立看门狗和窗口看门狗。本书仅介绍相对复杂的窗口看门狗。

5.2.1　窗口看门狗定时器的工作原理

窗口看门狗定时器的结构如图 5-5 所示。

由图 5-5 可知，窗口看门狗定时器的时钟源为 PCLK1(工作频率为 36MHz)，经过 4096 分频后，再经过 WWDG_CFR 寄存器指定的分频后，送给看门狗计数器。这里的 WWDG_CFR 寄存器只有第[9:0]位有效，其中，第[8:7]位记为 WDGTB[1:0]，用于指定分频值为 $1/2^{WDGTB[1:0]}$，例如，WDGTB[1:0]设为 11b，则分频值为 1/8。WWDG_CFR 的第 9 位记为 EWI，该位置 1，则看门狗计数器 T[6:0]减计数到 0x40 时，产生看门狗中断。WWDG_CFR 的第[6:0]位为窗口，最大值为 0x7F，最小值为 0x41，当 T[6:0]的值大于 W[6:0]的值时，向 T[6:0]赋值(即喂狗)将产生复位。

WWDG_CR 寄存器只有第[7:0]位有效，其中，第[6:0]位为看门狗计数器 T[6:0]，第

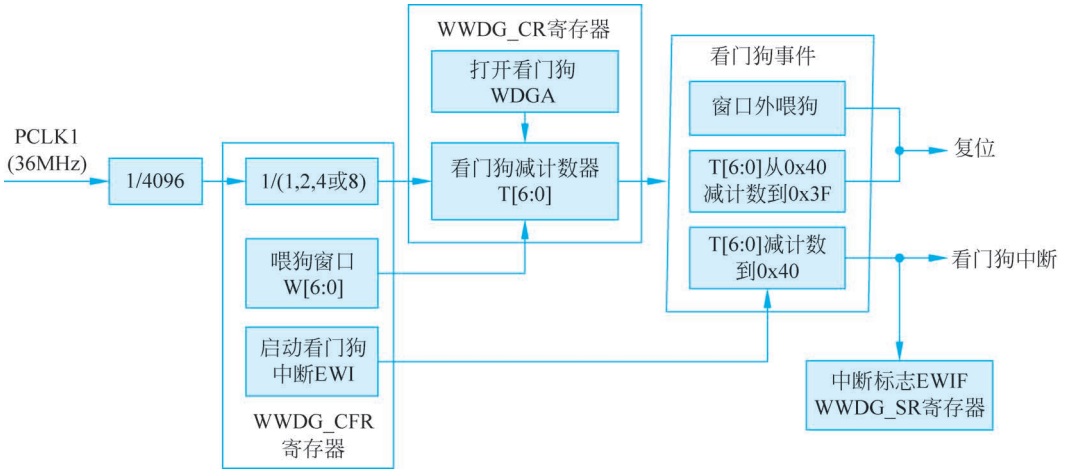

图 5-5　窗口看门狗定时器的结构

7 位记为 WDGA，设为 1 则启动看门狗，只有复位后才能自动清零。当看门狗计数器减计数到 0x40 时，将产生看门狗中断（若 EWI 位为 1）；当看门狗计数器从 0x40 减计数到 0x3F 时（即 T[6]由 1 变为 0），将产生复位。

WWDG_SR 寄存器只有第 0 位有效，记为 EWIF，当产生看门狗中断时，EWIF 位自动置 1，写入 0 可清零该位。

在图 5-5 中，如果配置 WWDG_CFR 寄存器的 WDGTB[1:0]为 11b，则看门狗每隔 910μs 减计数 1，由于看门狗中断和看门狗复位只相差一个计数时间，即相差 910μs，所以，在看门狗中断服务程序中应首先喂狗，然后执行其他处理。如果设定看门狗计数器的初始值为 0x6D，则减计数到 0x40 时，减计数值为 0x2D，即十进制数 45，所花费的时间为 40.96ms，即看门狗中断每 40.96ms 触发一次。在 5.2.2 节的工程实例中，使用了该配置方式。


视频讲解

5.2.2　窗口看门狗定时器的寄存器类型实例

本小节拟把看门狗定时器 WWDG 用作普通的定时器，实现每隔 1s LED 灯 D3 闪烁的功能。

在工程 PRJ02 的基础上，新建工程 PRJ04，保存在目录 D:\STM32F103C8T6REG\PRJ04 下，此时的工程 PRJ04 与工程 PRJ02 完全相同。然后，执行以下的步骤。

（1）修改文件 main.c，如程序段 5-5 所示，即在 main 函数的无限循环体中，不执行任何处理。

（2）新建文件 wwdog.c 和 wwdog.h，如程序段 5-10 和程序段 5-11 所示，保存在目录 D:\STM32F103C8T6REG\PRJ04\BSP 下。

程序段 5-10　文件 wwdog.c

```
1    //Filename: wwdog.c
2
3    #include "includes.h"
4
5    void WWDOGInit(void)
6    {
7      RCC->APB1ENR |= (1uL<<11);
8
9      WWDG->CR = 0x6D;                          // T[6:0] = 0x6D
10     WWDG->CFR = (1uL<<9) | (3uL<<7) | (0x40<<0);  //打开看门狗中断,8 分频
```

```
11      WWDG->SR = 0;
12      WWDG->CR |= (1uL<<7);                    //打开看门狗
13      NVIC_EnableIRQ(WWDG_IRQn);
14    }
15
16    void WWDG_IRQHandler(void)
17    {
18      static Int16U i=0;
19
20      WWDG->CR = 0x6D;
21
22      i++;
23      if(i==25)
24          LED(2,1);
25      if(i==50)
26      {
27          i=0;
28          LED(2,0);
29      }
30
31      WWDG->SR =0;
32      NVIC_ClearPendingIRQ(WWDG_IRQn);
33    }
```

第5～14行为看门狗定时器初始化函数WWDOGInit。第7行打开看门狗定时器时钟源(RCC_APB1ENR寄存器含义请参考STM32F103参考手册,其中,第11位置1表示打开看门狗定时器的时钟源);第9行向看门狗计数器赋初值0x6D;第10行设置看门狗中断有效、1/8分频值和窗口值为0x40,表示在0x40至0x7F间(这是窗口大小)间可正常“喂狗”,不触发复位异常;第11行清零中断标志位;第12行启动看门狗定时器。第13行调用CMSIS库函数打开看门狗NVIC中断。

第16～33行为看门狗中断服务函数WWDG_IRQHandler,该函数名是系统设定的(参考1.6节)。第18行定义静态变量i,第20行喂狗;第22～29行执行LED灯D3的闪烁操作,由于看门狗中断每40.96ms触发一次,触发25次约1s,当i累加到25时,LED灯D3亮,当i由25累加到50时,LED灯D3灭。第31行清零看门狗中断标志位;第32行清除看门狗中断的NVIC中断标志位。

程序段 5-11　文件 wwdog.h

```
1     //Filename: wwdog.h
2
3     #ifndef _WWDOG_H
4     #define _WWDOG_H
5
6     void WWDOGInit(void);
7
8     #endif
```

文件wwdog.h是文件wwdog.c对应的头文件,用于声明wwdog.c中定义的函数,这里第6行声明了WWDOGInit函数。

(3) 在includes.h文件的末尾添加语句#include "wwdog.h",即在总的包括头文件中包括文件wwdog.h。

(4) 在bsp.c文件的BSPInit函数中,添加对函数WWDOGInit的调用,如程序段5-12所示。

程序段 5-12　文件 bsp.c

```
1    //Filename: bsp.c
2
3    #include "includes.h"
4
5    void BSPInit()
6    {
7      LEDInit();
8      BEEPInit();
9      KEYInit();
10     EXTIKeyInit();
11     WWDOGInit();
12   }
```

文件 bsp.c 中的函数 BSPInit(第 5～12 行)用于初始化 STM32F103C8T6 微控制器的片上外设。相对于程序段 4-4 而言，这里添加的第 11 行调用了看门狗初始化函数 WWDOGInit，用于实现对看门狗定时器外设的初始化。

(5) 将 wwdog.c 文件添加到工程管理器的 BSP 分组下。完成后的工程 PRJ04 如图 5-6 所示。

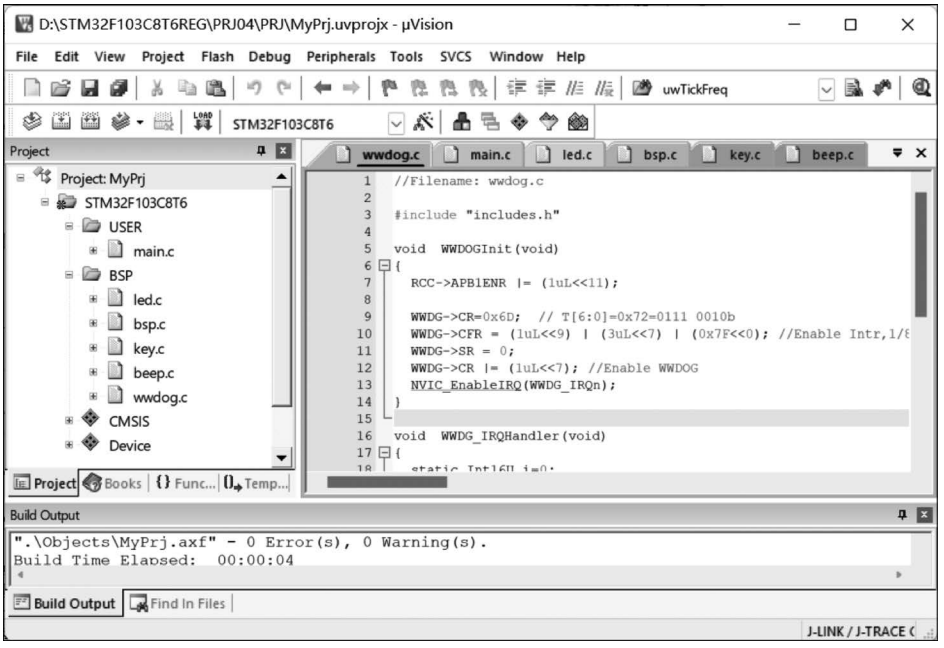

图 5-6　工程 PRJ04 工作窗口

在图 5-6 中，编译链接并运行工程 PRJ04，可以看到 STM32F103C8T6 学习板上的 LED 灯 D3 每隔约 1s 闪烁一次，从而实现了看门狗定时器的定时中断处理工作。


视频讲解

5.2.3　窗口看门狗定时器的 HAL 类型实例

在工程 HPrj02 的基础上，新建工程 HPrj04，保存在目录 D:\STM32F103C8T6HAL\HPrj04 下，此时的工程 HPrj04 与工程 HPrj02 完全相同。然后，进行如下的步骤。

(1) 在 STM32CubeMX 集成开发环境下，配置窗口看门狗，如图 5-7 所示。

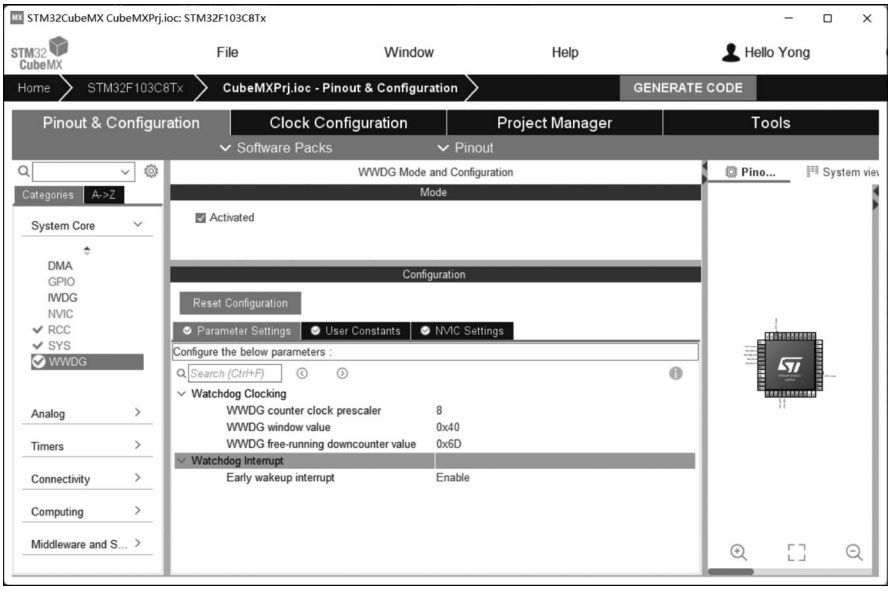

图 5-7 配置窗口看门狗

在图 5-7 左边栏中选中 WWDG，再在窗口中间部分选中 Activated，表示开启窗口看门狗；再配置 WWDG counter clock prescaler 为 8，表示分频值为 1/8；设置 WWDG window value 为 0x40，表示窗口大小为 0x40 至 0x7F；设置 WWDG free-running downcounter value 为 0x6D，即每次的“喂狗值”为 0x6D。

然后，在图 5-7 左侧选中 NVIC，如图 5-8 所示。

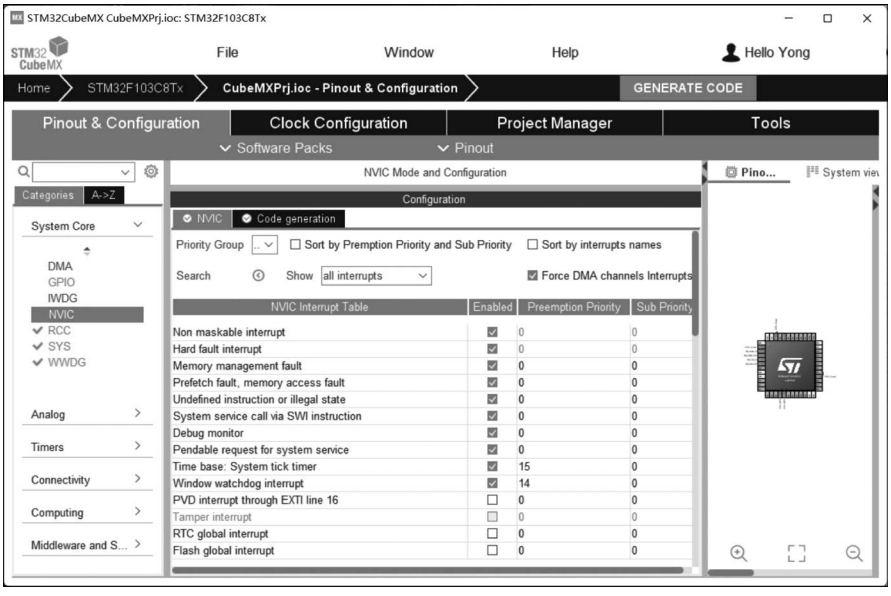

图 5-8 配置看门狗中断

在图 5-8 中，打开窗口看门狗中断 Window watchdog interrupt，设置其中断优先级号为 14。接着，在图 5-8 中，单击 GENERATE CODE 生成 CubeMXPrj 工程。

(2) 在 Keil MDK 中打开工程 CubeMXPrj，新建文件 wwdog.c，如程序段 5-13 所示，这

个文件保存在目录 D:\STM32F103C8T6HAL\HPrj04\BSP 下。

程序段 5-13　文件 wwdog.c

```
1    //Filename: wwdog.c
2
3    #include "includes.h"
4
5    void HAL_WWDG_EarlyWakeupCallback(WWDG_HandleTypeDef * hwwdg)
6    {
7      static Int16U i = 0;
8      hwwdg -> Instance -> CR = 0x6D;
9
10     i++;
11     if(i == 25)
12         LED(2,1);
13     if(i == 50)
14     {
15         i = 0;
16         LED(2,0);
17     }
18   }
```

第 5 行的 HAL_WWDG_EarlyWakeupCallback 为窗口看门狗中断的回调函数，第 8 行“hwwdg-> Instance-> CR＝0x6D;”为“喂狗”语句，即设置看门狗计数器的值为 0x6D。注意，进入看门狗中断回调函数后，第一件事情为“喂狗”。其余代码的含义请参考程序段 5-9。

(3) 修改 mymain.c 文件，如程序段 5-8 所示；添加新创建的文件 wwdog.c(保存在目录 D:\STM32F103C8T6HAL\HPrj04\BSP 下)到工程管理器的 App/User/BSP 分组下。

建设好的工程 HPrj04 如图 5-9 所示。

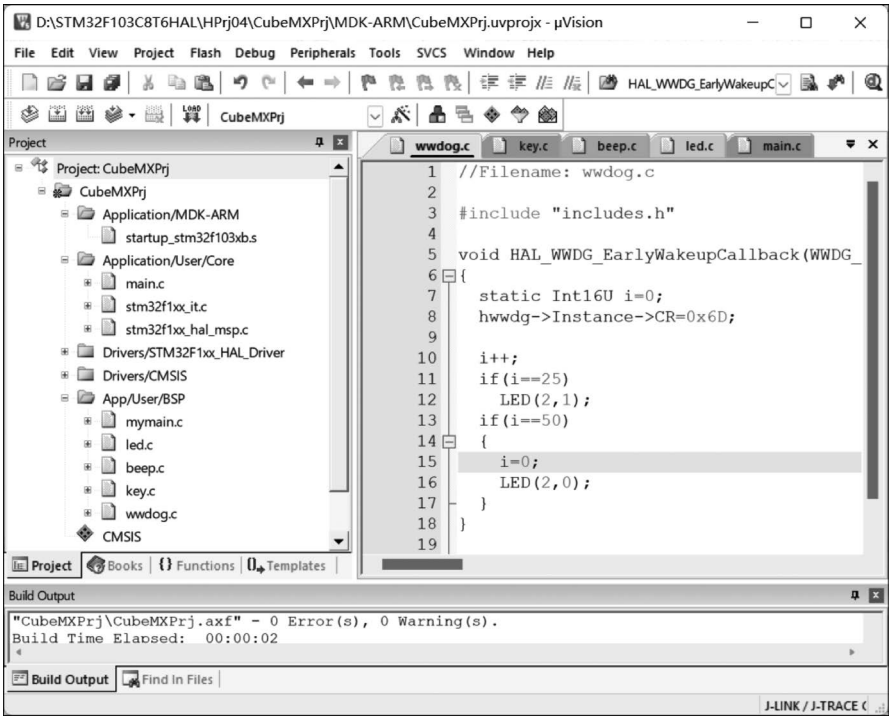

图 5-9　工程 HPrj04 工作窗口

工程 HPrj04 实现的功能与工程 PRJ04 完全相同，即将窗口看门狗定时器配置为每 40.96ms 触发一次看门狗中断的普通定时器，在看门狗中断服务函数中，通过静态的计数变量，实现每隔约 1s 时间使 LED 灯 D3 切换一次状态。

5.3　实时时钟

STM32F103C8T6 微控制器的实时时钟(RTC)模块，严格意义上讲，只是一个低功耗的定时器，如果要实现时间和日历功能，必须借助软件实现。其优点在于灵活性较强，缺点在于程序员编程时需要考虑日历变化中的闰年情况。

5.3.1　实时时钟的工作原理

STM32F103C8T6 微控制器的实时时钟结构如图 5-10 所示。

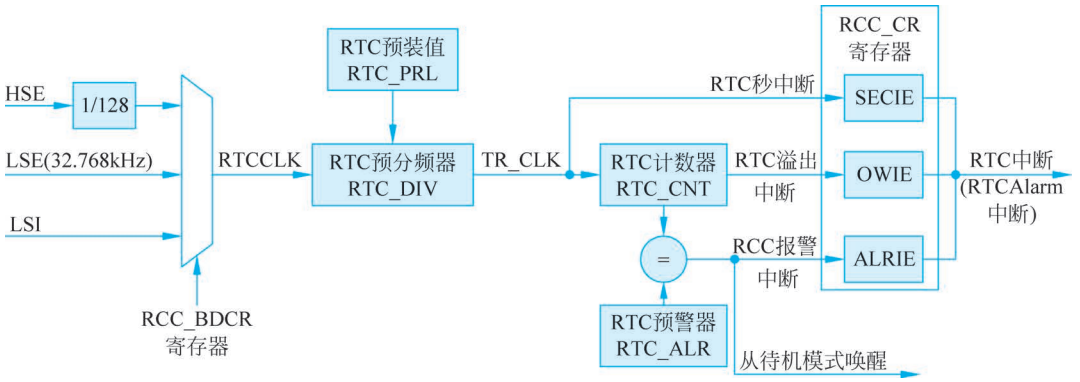

图 5-10　STM32F103C8T6 微控制器的实时时钟结构

由图 5-10 可知，RTC 模块有 3 个时钟源可供选择，一般情况下，希望选择具有较高精度的外部低速时钟 LSE(32.768kHz)。这里的 HSE 是指片外高精度高速时钟(8MHz)，LSI 指片内低速时钟(40kHz)。如果选择了 LSE 时钟，则 RTCCLK 时钟信号为 32.768kHz，如果设定 RTC 预分频器的值为 32767，则 TR_CLK = RTCCLK / (RTC_DIV + 1)，即 TR_CLK 时钟信号为 1Hz。RTC 模块可触发 3 种类型的中断，即秒中断、溢出中断和报警中断(或称闹钟中断)，通过配置 RCC_CR 寄存器实现这 3 类中断的开启。当 RTC 计数器的值与 RTC 预警器的值相等时，产生 RTC 报警中断，同时，该中断还可用于从待机模式唤醒微控制器。

图 5-10 中的 RCC_BDCR 寄存器是复位与时钟控制(RCC)模块的寄存器，其第[9:8]位设为 01b 时，RTC 模块使用 LSE 时钟源，该寄存器的详细情况请参考 STM32F103 用户手册。

下面详细介绍 RTC 模块的各个寄存器，RTC 模块的基地址为 0x4000 2800。

1) RTC 控制寄存器 RTC_CRH

RTC_CRH(偏移地址为 0x0，复位值为 0x0)是一个 16 位寄存器，只有第[2:0]位有效，第 2 位为 OWIE，为 1 表示开启溢出中断；第 1 位为 ALRIE，为 1 表示开启报警中断；第 0 位为 SECIE，为 1 表示开启秒中断。

2) RTC 控制寄存器 RTC_CRL

RTC_CRL(偏移地址为 0x04，复位值为 0x0020)是一个 16 位寄存器，只有第[5:0]位有

效。第5位为只读的RTOFF位，读出0表示写RTC寄存器正处理中，读出1表示写RTC寄存器操作已完成；第4位为CNF位，写入1表示进入配置模式，写入0表示退出配置模式；第3位为RSF位，当RTC各个寄存器同步后硬件置1，可软件方式写入0清零；第2位为溢出中断标志位OWF，为1表示发生了溢出中断，写入0清零；第1位为报警中断标志位ALRF，为1表示发生了报警中断，写入0清零；第0位为秒中断标志位SECF，为1表示发生了秒中断，写入0清零。

RTC模块的各个寄存器的访问规则为：首先，确认RTOFF位为1；然后，置CNF位为1进入配置模式；接着，配置各个RTC寄存器(包括RTC_CRH)；之后，清零CNF位退出配置模式；最后，等待RTOFF位为1。

3) RTC预装值寄存器RTC_PRLH和RTC_PRLL

RTC_PRLH和RTC_PRLL(偏移地址分别为0x08和0x0C，复位值分别为0x0和0x8000)是两个16位寄存器，RTC_PRLH的高14位保留，RTC_PRLH的第[3:0]位(作为PRL[19:16])与RTC_PRLL的第[15:0]位(作为PRL[15:0])组合成PRL[19:0]，结合图5-6，TR_CLK = RTCCLK / (PRL[19:0]+1)。

4) RTC预分频器寄存器RTC_DIVH和RTC_DIVL

RTC_DIVH和RTC_DIVL(偏移地址分别为0x10和0x14，复位值分别为0x0和0x8000)是两个只读的16位计数器，其减计数到0后，RTC_PRLH和RTC_PRLL中的预装值将自动装入RTC_DIVH和RTC_DIVL中。

5) RTC计数器寄存器RTC_CNTH和RTC_CNTL

RTC_CNTH和RTC_CNTL(偏移地址分别为0x18和0x1C，复位值均为0x0)是两个可读/可写的16位寄存器，用于保存RTC模块的时间和日历值。

6) RTC报警器寄存器RTC_ALRH和RTC_ALRL

RTC_ALRH和RTC_ALRL(偏移地址分别为0x20和0x24，复位值均为0xFFFF)是两个只写的16位寄存器，用于保存RTC模块报警时的时间和日历值。当RTC计数器寄存器RTC_CNTH和RTC_CNTL的值分别与RTC_ALRH和RTC_ALRL的值相等时，产生RTC报警中断。

5.3.2小节和5.3.3小节通过RTC模块实现LED灯D3每隔1s闪烁一次的功能，以说明RTC模块的配置方法和秒中断程序设计方法。


视频讲解

5.3.2 实时时钟的寄存器类型实例

在工程PRJ02的基础上，新建工程PRJ05，保存在目录D:\STM32F103C8T6REG\PRJ05下，此时的工程PRJ05与工程PRJ02完全相同。然后，进行如下的步骤。

(1) 修改文件main.c，如程序段5-5所示，即在主函数的无限循环体中，不做具体的处理工作。

(2) 新建文件rtc.c和rtc.h，保存在目录D:\STM32F103C8T6REG\PRJ05\BSP下，这两个文件的内容分别如程序段5-14和程序段5-15所示。

程序段5-14 文件rtc.c

```
1    //Filename: rtc.c
2
```

```
3    #include "includes.h"
4
5    void RTCInit(void)
6    {
7      Int32U i;
8
9      RCC->APB1ENR |= (1uL<<27) | (1uL<<28);  //BKP 和 PWR 使能
10     PWR->CR |= (1uL<<8);                    //使 RTC 和 BKP 可访问
11
```

第 9 行使复位与时钟控制模块的寄存器 RCC_APB1ENR 的第 27 位和第 28 位置 1，表示打开备份接口(BKP)模块和功耗管理(PWR)模块的时钟源。这两个模块与 RTC 有关。第 10 行设置 PWR_CR 寄存器的第 8 位为 1，表示可访问 RTC 和 BKP 模块的寄存器。

```
12     RCC->BDCR |= (1uL<<16);
13     RCC->BDCR &= ~(1uL<<16);                //BKP 退出复位
14     RCC->BDCR |= (1uL<<0);                  //使用 LSE 32.768kHz 时钟
15
```

第 12 行向 RCC_BDCR 寄存器的第 16 位写入 1 复位 BKP 模块；第 13 行向其写入 0，退出复位状态，进入工作状态；第 14 行向 RCC_BDCR 寄存器的第 0 位写入 1，表示开启外部的 32.768kHz 时钟源 LSE。

```
16     for(i=0;i<20000;i++);                   //等待 6 个 LSI 时钟周期
17     while((RCC->BDCR & (1uL<<1))!=(1uL<<1)); //等待 LSE 稳定
18
```

第 16 行是 RCC 模块的特殊要求，即执行了第 14 行开启 LSE 时钟源后，必须至少等待 6 个 LSE 时钟周期，使得 RCC_BDCR 寄存器的第 1 位硬件清零。如果 RCC_BDCR 寄存器的第 1 位硬件自动置 1，说明 LSE 时钟源已稳定，所以第 17 行等待 RCC_BDCR 寄存器的第 1 位置 1。

```
19     RCC->BDCR |=(1uL<<8);
20     RCC->BDCR &= ~(1uL<<9); //使用 LSE 时钟
21     RCC->BDCR |=(1uL<<15);
22
```

RCC_BDCR 寄存器的第[9:8]位为 01b，表示使用 LSE 时钟，第 19、20 行为配置其为 01b。第 21 行设置 RCC_BDCR 寄存器的第 15 位为 1，表示启用 RTC。

这里第 9～21 行使用了 BKP 和 RCC 模块的一些寄存器，书中没有详细介绍，可参考 STM32F103 参考手册的第 6 章和第 7 章。

```
23     while((RTC->CRL & (1uL<<5))!=(1uL<<5)); //RTOFF=1
24     while((RTC->CRL & (1uL<<3))!=(1uL<<3)); //RSF=1
25     RTC->CRL |= (1uL<<4);                   //CNF=1 进入配置模式
26     RTC->CRH |= (1uL<<0);                   //打开秒中断
27     RTC->PRLH = 0;
28     RTC->PRLL = 32768-1;
29     RTC->CRL &= ~(1uL<<4);                  //CNF=0 退出配置模式
30     while((RTC->CRL & (1uL<<5))!=(1uL<<5)); //RTOFF=1
31
32     NVIC_EnableIRQ(RTC_IRQn);
33   }
34
```

第 5～33 行为 RTC 模块的初始化函数 RTCInit。第 23～30 行为配置 RTC 模块的寄存器，按照其访问规则：第 23 行等待 RTOFF 位为 1；第 24 行等待 RSF 位为 1(表示 RTC 各个寄存器已同步)；第 25 行置位 CNF，进入配置模式；第 26 行打开秒中断；第 27、28 行设置分频值为 32767；第 29 行清零 CNF，退出配置模式；第 30 行等待 RTOFF 位置 1。

第 32 行调用 CMSIS 库函数打开 RTC 模块对应的 NVIC 中断。

```
35   void RTC_IRQHandler(void)
36   {
37     static Int08U i = 0;
38     i++;
39     i = i % 2;
40     if(i == 0)
41         LED(2,1);
42     else
43         LED(2,0);
44     RTC -> CRL &= ~(1uL << 0);
45     NVIC_ClearPendingIRQ(RTC_IRQn);
46   }
```

第 35～46 行为 RTC 模块的中断服务函数，函数数名必须为 RTC_IRQHandler(参考 1.6 节)，来源于 startup_stm32f10x_md.s 文件中的同名标号。第 37 行定义静态变量 i；根据 RTCInit 函数可知，RTC 中断每 1s 执行一次，每次执行第 38 行将变量 i 加 1；第 39 行将 i 的值模 2，如果为 0，则第 40 行为真，执行第 41 行 LED 灯 D3 亮；否则(第 42 行为真)，第 43 行 LED 灯 D3 灭。第 44 行向 RTC_CRL 寄存器的第 0 位写入 0，清除 RTC 秒中断标志位；第 45 行调用 CMSIS 库的 NVIC_ClearPendingIRQ 函数清除 RTC 的 NVIC 中断标志位。

程序段 5-15　文件 rtc.h

```
1    //Filename: rtc.h
2
3    #ifndef _RTC_H
4    #define _RTC_H
5
6    void RTCInit(void);
7
8    #endif
```

文件 rtc.h 中声明了文件 rtc.c 中定义的函数 RTCInit。

(3) 在 includes.h 文件的末尾添加语句“#include "rtc.h"”，即包括头文件 rtc.h。

(4) 在 bsp.c 文件的 BSPInit 函数中，添加语句“RTCInit();”，如程序段 5-16 所示。

程序段 5-16　文件 bsp.c

```
1    //Filename: bsp.c
2
3    #include "includes.h"
4
5    void BSPInit()
6    {
7      LEDInit();
8      BEEPInit();
9      KEYInit();
```

```
10      EXTIKeyInit();
11      RTCInit();
12    }
```

在文件 bsp.c 中，BSPInit 函数依次实现 LED 灯、蜂鸣器、按键、外部中断和 RTC 的初始化（如第 7～11 行所示）。

（5）将 rtc.c 文件添加到工程管理器的 BSP 分组下。建设好的工程 PRJ05 如图 5-11 所示。

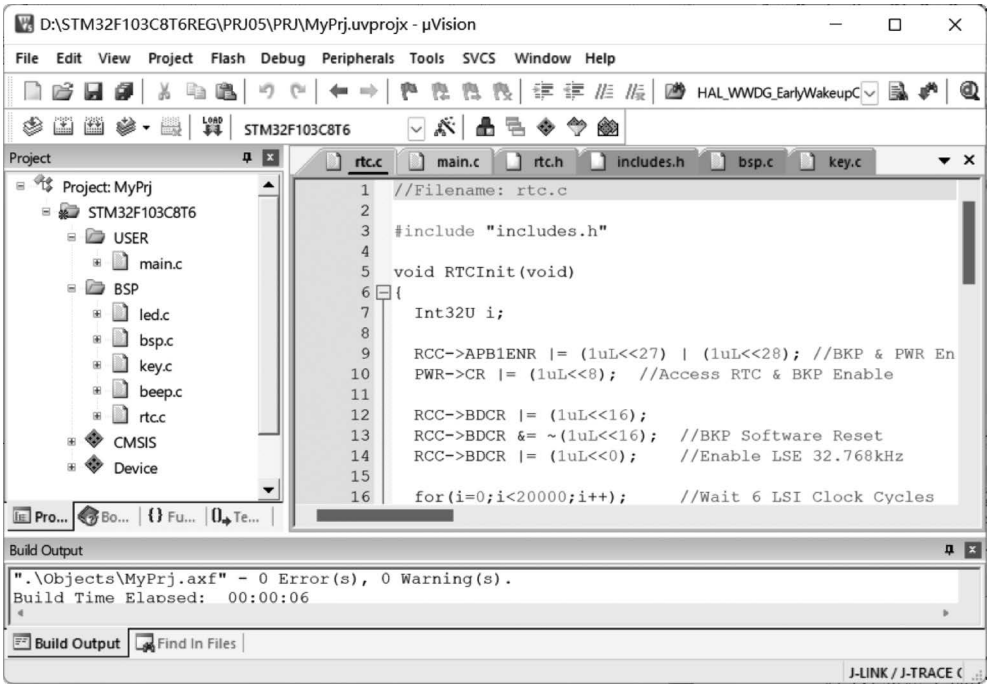

图 5-11　工程 PRJ05 工作窗口

在图 5-11 中，编译链接并运行工程 PRJ05，可观察到 STM32F103C8T6 学习板上的 LED 灯 D3 每隔 1s 闪烁一次。此外，工程 PRJ05 保留了工程 PRJ02 的按键功能。

5.3.3　实时时钟的 HAL 类型实例


视频讲解

本小节中使用 HAL 库实现工程 PRJ05 的全部功能，具体步骤如下。

（1）在工程 HPrj02 的基础上，新建工程 HPrj05，保存在目录 D:\STM32F103C8T6HAL\HPrj05 下。此时的工程 HPrj05 与工程 HPrj02 完全相同。

（2）在 STM32CubeMX 中，选中 RTC，如图 5-12 所示。

在图 5-12 中，选中 Activate Clock Source 表示为 RTC 提供时钟，然后，选中 RTC global interrupt，开放 RTC 中断，并设置其中断优先号为 13（通过选中左侧的 NVIC 进行配置）。然后，单击 GENERATE CODE 生成 CubeMXPrj 工程。

（3）在 Keil MDK 集成开发环境下，新建 rtc.c 和 rtc.h 文件，保存在目录 D:\STM32F103C8T6HAL\HPrj05\BSP 下，其中，rtc.h 文件如程序段 5-15 所示，rtc.c 文件如程序段 5-17 所示。

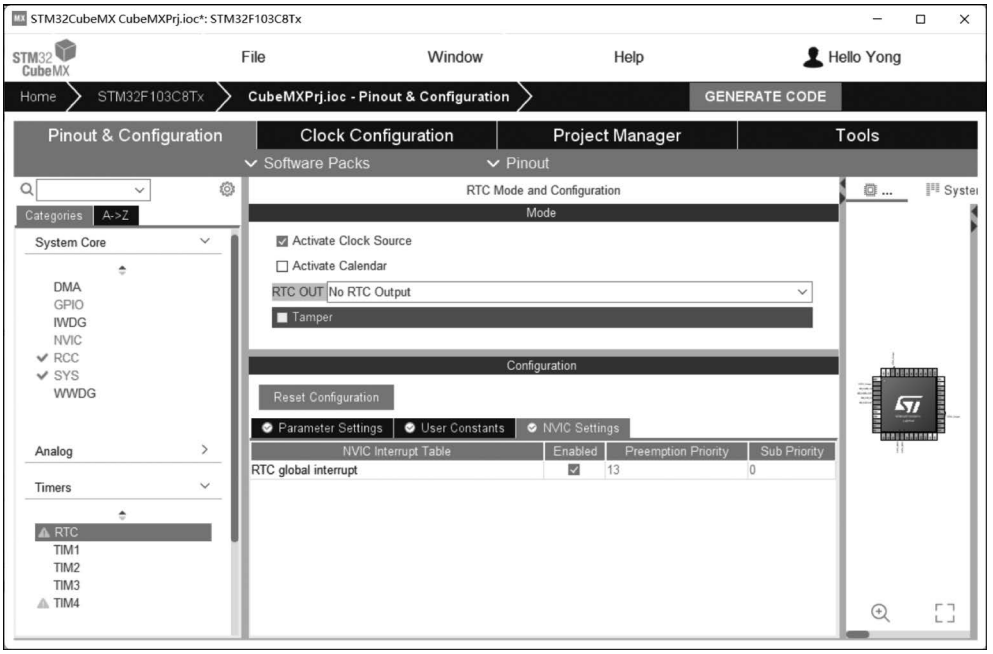

图 5-12　RTC 配置窗口

程序段 5-17　文件 rtc.c

```
1    //Filename: rtc.c
2
3    #include "includes.h"
4
5    void RTCInit(void)
6    {
7      Int32U i;
8
```

此处省略的第 9～32 行代码与程序段 5-13 的第 9～32 行代码相同。

```
33   }
34
35   void HAL_RTCEx_RTCEventCallback(RTC_HandleTypeDef * hrtc)
36   {
37     static Int08U i = 0;
38     i++;
39     i = i % 2;
40     if(i == 0)
41         LED(2,1);
42     else
43         LED(2,0);
44   }
```

第 35～44 行的函数 HAL_RTCEx_RTCEventCallback 为 RTC 的中断回调函数，RTC 中断每隔 1s 触发一次中断，每次中断调用函数 HAL_RTCEx_RTCEventCallback 一次，将切换 LED 灯 D3 的状态。

（4）修改 mymain.c 文件，如程序段 5-18 所示，即第 7 行添加对 RTC 初始化函数的调用语句。

程序段 5-18　文件 mymain.c

```
1    //Filename: mymain.c
2
3    #include "includes.h"
4
5    void mymain(void)
6    {
7      RTCInit();
8      while(1)
9      {
10     }
11   }
```

注意：第 7 行添加 RTC 初始化函数对实时时钟进行初始化，是因为 STM32CubeMX 对实时时钟的初始化工作不正常，STM32CubeMX 是一个发展中的集成开发环境，目前的版本无法实现对 RTC 时钟的完整初始化。

（5）在 includes.h 文件的末尾添加 #include "rtc.h"，即包括头文件 rtc.h。

（6）将 rtc.c 文件添加到工程管理器的 BSP 分组下，建设好的工程 HPrj05 如图 5-13 所示，工程 HPrj05 实现的功能与工程 PRJ05 完全相同。

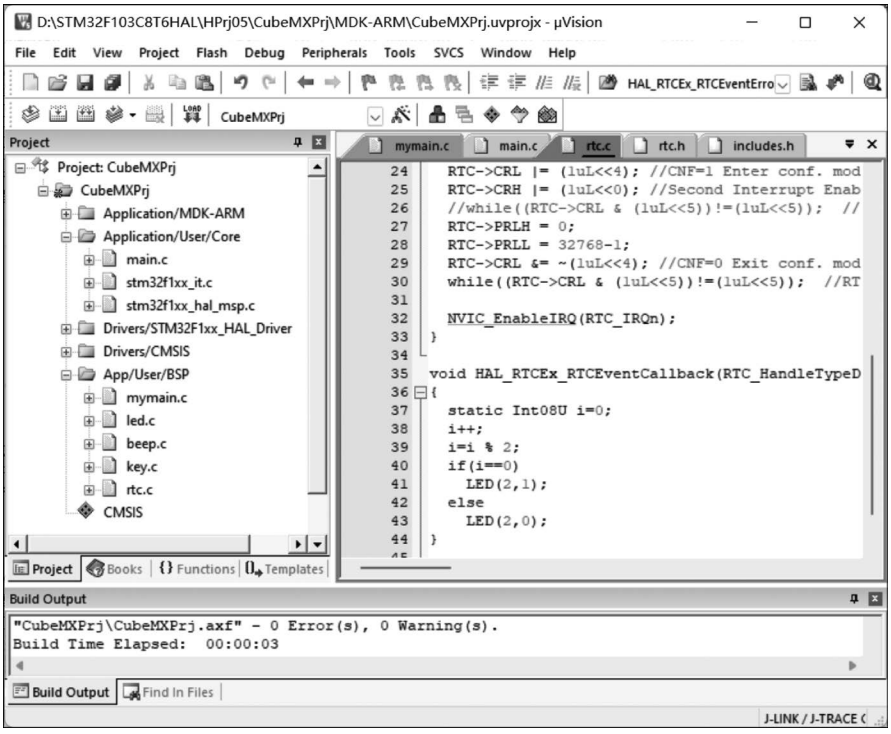

图 5-13　工程 HPrj05 工作窗口

5.4　通用定时器

STM32F103C8T6 具有 4 个定时器，其中，TIM1 为高级控制定时器，TIM2～TIM4 为通用定时器。相对于传统的 80C51 单片机的定时器而言，STM32F103C8T6 的定时器功能

更加完善和复杂。这里以 TIM2 为例介绍通用定时器的基本用法。

5.4.1 通用定时器的工作原理

STM32F103C8T6 微控制器具有 3 个通用定时器 TIM2～TIM4，它们的结构和工作原理相同。这里以通用定时器 TIM2 为例介绍通用定时器的工作原理，TIM2 的结构如图 5-14 所示。

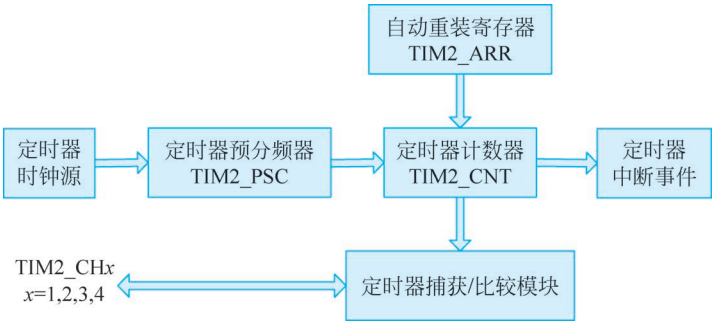

图 5-14 通用定时器 TIM2 的结构

由图 5-14 可知，定时器 TIM2 具有 4 个通道，可实现对外部输入脉冲信号的捕获（计数）和比较输出，相关的寄存器有 TIM2 捕获与比较寄存器 TIM2_CCR1～TIM2_CCR4、TIM2 捕获与比较模式寄存器 TIM2_CCMR1～TIM2_CCR2 和 TIM2 捕获与比较有效寄存器 TIM2_CCER 等。本节重点介绍通用定时器的定时计数功能，相关的寄存器如下（基地址为 0x4000 0000，见图 1-4）。

1）TIM2 控制寄存器 TIM2_CR1（偏移地址为 0x0，复位值为 0x0）

TIM2_CR1 寄存器是一个 16 位的可读/可写寄存器，如表 5-5 所示。

表 5-5 TIM2_CR1 寄存器

位　号	名　称	属　性	含　义
15:10			保留
9:8	CKD[1:0]	可读/可写	定时器捕获/比较模块中的采样时钟间的倍数值。为 0 表示相等，为 1 表示 2 分频；为 2 表示 4 分频；为 3 保留
7	ARPE	可读/可写	为 0，自动重装无缓存；为 1，自动重装带缓存
6:5	CMS	可读/可写	为 0 表示单边计数；为 1 表示双边计数模式 1，输出比较中断仅当减计数时触发；为 2 表示双边计数模式 2，输出比较中断仅当加计数时触发；为 3 表示双边计数模式 3，输出比较中断在加计数和减计数时均触发
4	DIR	可读/可写	若 CMS=00b，则 DIR 为 0 表示加计数；为 1 表示减计数
3	OPM	可读/可写	为 0 表示单拍计数方式；为 1 表示循环计数
2	URS	可读/可写	为 0 表示计数溢出和 TIM2_EGR 寄存器的第 0 位（UG 位）置位等事件均产生中断；为 1 表示仅有计数溢出时才产生中断
1	UDIS	可读/可写	为 0 表示定时器更新事件（UEV）有效；为 1 表示 UEV 无效
0	CEN	可读/可写	为 0 关闭定时器；为 1 打开定时器

如果定时器 TIM2 采用加计数方式，则可以保持其复位值，只需要配置其第 0 位为 1 打开定时器 TIM2。

2）TIM2 定时器计数器 TIM2_CNT（偏移地址为 0x24，复位值为 0x0）

TIM2_CNT 寄存器是一个 16 位的可读/可写寄存器，保存了定时器的当前计数值。

3）TIM2 定时器预分频器寄存器 TIM2_PSC（偏移地址为 0x28，复位值为 0x0）

TIM2_PSC 寄存器是一个 16 位的可读/可写寄存器，TIM2 计数器的计数频率＝定时器时钟源频率/（TIM2_PSC＋1）。如果将 72MHz 系统时钟作为 TIM2 时钟源，设置 TIM2_PSC ＝ 7200－1，则 TIM2 计数器计数频率为 10kHz。

4）TIM2 自动重装寄存器 TIM2_ARR（偏移地址为 0x2C，复位值为 0x0）

如果 TIM2 设为加计数方式，则计数值从 0 计数到 TIM2_ARR 的值时，溢出而产生中断。如果计数频率为 10kHz，设定 TIM2_ARR 为 100－1，则 TIM2 定时中断的频率为 100Hz。

5）TIM2 定时器状态寄存器 TIM2_SR（偏移地址为 0x10，复位值为 0x0）

TIM2_SR 寄存器的第 0 位为 UIF 位，当发生定时中断时，UIF 位自动置 1，向其写入 0 清零该位。

6）TIM2 定时器有效寄存器 TIM2_DIER（偏移地址为 0x0C，复位值为 0x0）

TIM2_DIER 寄存器的第 0 位为 UIE 位，写入 1 开放定时器更新中断，写入 0 关闭定时器更新中断。

关于定时器的捕获/比较功能及与 DMA 控制器相关的内容，请参考 STM32F103 用户手册。

5.4.2　通用定时器的寄存器类型实例


视频讲解

本小节使用通用定时器 TIM2 实现 LED 灯 D3 每隔 1s 闪烁一次的功能，具体实现步骤如下。

（1）在工程 PRJ02 的基础上，新建工程 PRJ06，保存在目录 D:\STM32F103C8T6REG\PRJ06 下。此时的工程 PRJ06 与工程 PRJ02 完全相同。

（2）修改 main.c 文件，如程序段 5-5 所示，即主函数的无限循环体为空。

（3）新建文件 tim2.c 和 tim2.h，保存在目录 D:\STM32F103C8T6REG\PRJ06\BSP 下，其代码分别如程序段 5-19 和程序段 5-20 所示。

程序段 5-19　文件 tim2.c

```
1    //Filename: tim2.c
2
3    #include "includes.h"
4
5    void TIM2Init(void)
6    {
7      RCC->APB1ENR |= (1uL<<0);
8      TIM2->ARR = 100-1;
9      TIM2->PSC = 7200-1;
10     TIM2->DIER |= (1uL<<0);
11     TIM2->CR1 |= (1uL<<0);
12
```

```
13      NVIC_EnableIRQ(TIM2_IRQn);
14      NVIC_SetPriority(TIM2_IRQn,8);
15    }
16
```

第5～15行为TIM2初始化函数。第7行打开TIM2定时器的时钟源；第8行设置TIM2重装计数值为99；第9行设置TIM2预分频值为7199；第10行打开定时器刷新中断；第11行启动定时器TIM2。第13行打开TIM2的NVIC中断；第14行配置该中断优先级号为8。

```
17    void TIM2_IRQHandler(void)
18    {
19      static Int08U i = 0;
20      i++;
21      if(i == 100)
22          LED(2,1);
23      if(i == 200)
24      {
25          i = 0;
26          LED(2,0);
27      }
28      TIM2 -> SR &= ～(1uL << 0);
29      NVIC_ClearPendingIRQ(TIM2_IRQn);
30    }
```

第17～30行为定时器TIM2中断服务函数。由于定时器中断触发的频率为100Hz，故100次中断的时间间隔为1s，通过静态计数变量i，实现LED灯D3每隔1s闪烁一次的功能。

程序段5-20　文件tim2.h

```
1     //Filename: tim2.h
2
3     #ifndef _TIM2_H
4     #define _TIM2_H
5
6     void TIM2Init(void);
7
8     #endif
```

文件tim2.h中声明了文件tim2.c中定义的函数TIM2Init。

(4) 在includes.h文件的末尾添加#include "tim2.h"语句，即包括头文件tim2.h。

(5) 修改bsp.c文件，如程序段5-21所示。

程序段5-21　文件bsp.c

```
1     //Filename: bsp.c
2
3     #include "includes.h"
4
5     void BSPInit()
6     {
7       LEDInit();
8       BEEPInit();
9       KEYInit();
10      EXTIKeyInit();
```

```
11      TIM2Init();
12    }
```

对比程序段 4-4 可知，这里添加了第 11 行语句，即调用 TIM2Init 函数对 TIM2 进行初始化。

（6）将文件 tim2.c 添加到工程管理器的 BSP 分组下。完成后的工程 PRJ06 如图 5-15 所示。

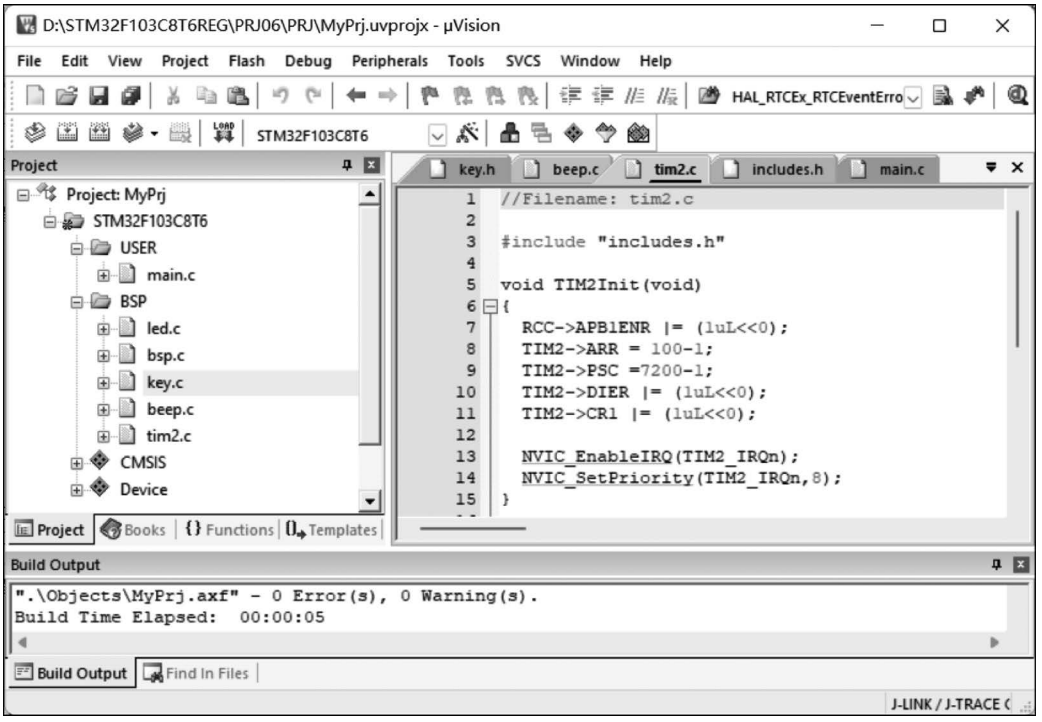

图 5-15　工程 PRJ06 工作窗口

在图 5-15 中，编译链接并运行工程 PRJ06，可以观察到 STM32F103C8T6 学习板上的 LED 灯 D3 每隔 1s 闪烁一次。此外，工程 PRJ06 保留了工程 PRJ02 的按键处理功能。

5.4.3　通用定时器的 HAL 类型实例


视频讲解

本节用 HAL 方式实现工程 PRJ06 同样的功能，具体设计步骤如下。

（1）在工程 HPrj02 的基础上，新建工程 HPrj06，保存在目录 D:\STM32F103C8T6HAL\HPrj06 下，此时的工程 HPrj06 与工程 HPrj02 完全相同。

（2）在 STM32CubeMX 开发环境中，选中 TIM2，如图 5-16 所示。

在图 5-16 中，配置定时器 TIM2 的 Clock Source 为 Internal Clock，然后，在 Parameter Settings 中按图中所示内容进行配置；接着，打开定时器 TIM2 的更新中断（在 NVIC Settings 和 NVIC 中配置）；最后，单击 GENERTATE CODE 生成 CubeMX 工程。

（3）新建 tim2.c 和 tim2.h 文件，保存在目录 D:\STM32F103C8T6HAL\HPrj06\BSP 下，其中，文件 tim2.h 如程序段 5-20 所示，文件 tim2.c 如程序段 5-22 所示。

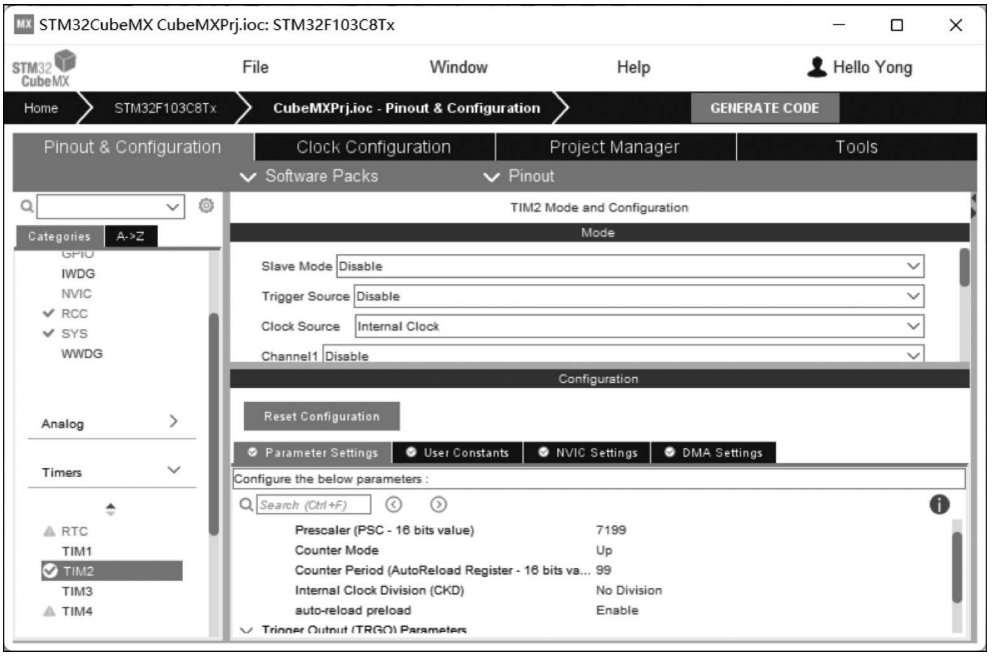

图 5-16　定时器 TIM2 配置窗口

程序段 5-22　文件 tim2.c

```
1    //Filename: tim2.c
2
3    #include "includes.h"
4
5    void TIM2Init(void)
6    {
7      RCC->APB1ENR |= (1uL<<0);
8      TIM2->ARR = 100-1;
9      TIM2->PSC = 7200-1;
10     TIM2->DIER |= (1uL<<0);
11     TIM2->CR1 |= (1uL<<0);
12   }
13
```

第 5～11 行为定时器 TIM2 初始化函数。第 7 行打开 TIM2 的时钟源；第 8、9 行分别配置定时器 TIM2 的重装值为 99、预分频值为 7199。第 10 行打开定时器 TIM2 刷新中断；第 11 行启动定时器 TIM2。

```
14   void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
15   {
16     static Int08U i=0;
17     i++;
18     if(i==100)
19         LED(2,1);
20     if(i==200)
21     {
22         i=0;
23         LED(2,0);
24     }
25   }
```

第 14～25 行为定时器 TIM2 的中断回调函数。

(4) 修改 mymain.c 文件，如程序段 5-23 所示，即在 mymain 函数中调用 TIM2Init 函数初始化定时器 TIM2。

程序段 5-23　文件 mymain.c

```
1    //Filename: mymain.c
2
3    #include "includes.h"
4
5    void mymain(void)
6    {
7      TIM2Init();
8      while(1)
9      {
10     }
11   }
```

(5) 在 includes.h 文件的末尾添加语句#include　"tim2.h"，即包括头文件 tim2.h。

(6) 将文件 tim2.c 添加到工程管理器的 App/User/BSP 分组下，完成后的工程如图 5-17 所示。

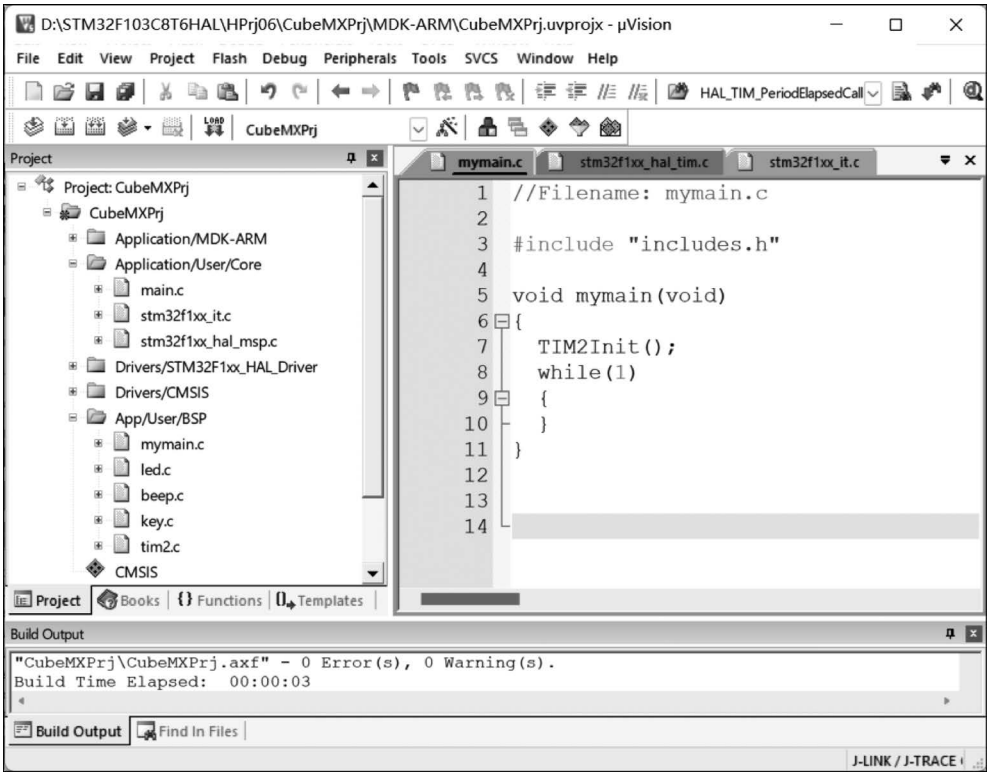

图 5-17　工程 HPrj06 工作窗口

注意，在工程 HPrj06 中，在 mymain 函数中调用了 TIM2Init 函数对定时器 TIM2 初始化。这些初始化工作可以移动到 STM32CubeMX 生成的系统初始化文件 st32f1xx_hal_msp.c 中，放在函数“HAL_TIM_Base_MspInit”中的注释/* USER CODE BEGIN TIM2_MspInit 0 */和　/* USER CODE END TIM2_MspInit 0 */之间的空行中。本书为了

保证 STM32CubeMX 生成的工程的完整性，所以，将一些外设初始化放在了自定义的 mymain.c 文件中。同样地，在工程 HPrj05 中，在 mymain 函数中调用了 RTCInit 函数对实时时钟初始化，这些初始化工作可以移动到系统初始化文件 st32f1xx_hal_msp.c 中的 HAL_RTC_MspInit 函数中，使工程更加美观。

5.5 本章小结

本章详细介绍了 STM32F103C8T6 微控制器片内系统节拍定时器、看门狗定时器、实时时钟和通用定时器的工作原理和工程程序实例。定时器是实际工程中最常用的片内外设之一，需要灵活地掌握它们的用法。建议读者朋友在学完本章后，结合 STM32F103 参考手册，编写定时器 TIM1 和 TIM3 的定时中断处理程序，并编写独立看门狗定时器的监控程序，从而加深对本章内容的理解。需要说明的是，STM32 微控制器外设有众多的控制寄存器，STM32CubeMX 配置某些 STM32 微控制器外设时还不能完全初始化，此时需要添加用户初始化代码。随着 STM32CubeMX 的进化，初始化工作将会不断完善。

习题

1. 简述系统节拍定时器的初始化方法。

2. 基于 STM32F103C8T6 学习板，编写寄存器类型工程，借助系统节拍定时器实现 LED 灯 D2 周期闪烁。

3. 基于 STM32F103C8T6 学习板，编写 HAL 类型工程，借助系统节拍定时器实现 LED 灯 D2 周期闪烁。

4. 简要说明窗口看门狗的特点和初始化方法。

5. 基于 STM32F103C8T6 学习板，编写工程借助 RTC 实现年、月、日、星期、时、分、秒的计时器，并考虑闰年的处理(提示：使用基姆拉尔森计算公式由年月日计算星期几)。

6. 简述 STM32F103C8T6 微控制器通用定时器的工作原理。

7. 基于 STM32F103C8T6 学习板，编写工程借助通用定时器 TIM4 实现 LED 灯 D2 周期闪烁。