STM32Cube高效开发教程（基础篇）

第11章　实时时钟

王维波
中国石油大学（华东）控制科学与工程学院

STM32Cube高效开发教程（基础篇）

作者：王维波，鄢志丹，王钊

人民邮电出版社

2021年9月出版

如果有读者需要本书课件的PPT版本用于备课，可以给作者发邮件免费获取，并可加入专门的教学和技术交流QQ群

邮箱：wangwb@upc. edu. cn

第11章　实时时钟

11.1 RTC功能概述

11.2 闹钟和周期唤醒功能的使用

11.1 RTC功能概述

11.1.1 RTC的功能

11.1.2 工作原理

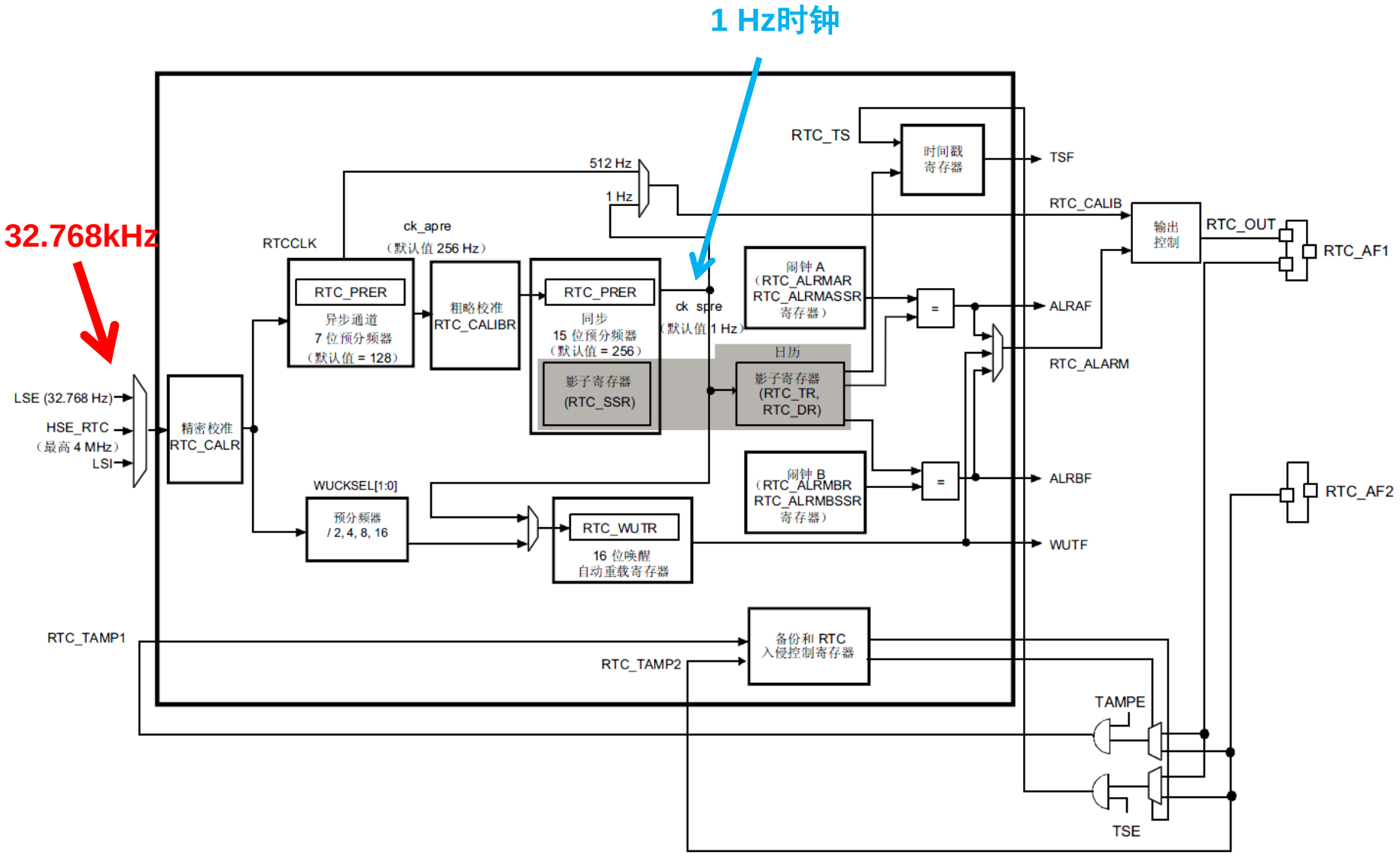
11.1.3 RTC中断和复用引脚功能小结

11.1.1 RTC的功能

RTC（Real-time Clock，实时时钟）是由时钟信号驱动的日历时钟，提供日期和时间数据。

- RTC能提供BCD或二进制的秒、分钟、小时（12或24小时制）、星期几、日期、月份、年份数据
- RTC模块和时钟配置都使用备用存储区域，使用VBAT备用电源，即使主电源断电或系统复位也不影响RTC的工作
- RTC有两个可编程闹钟中断，可以设定任意组合和重复性的闹钟
- 有一个可周期性唤醒的中断，唤醒中断可以当做一个普通定时器使用
- 具有时间戳和入侵检测功能

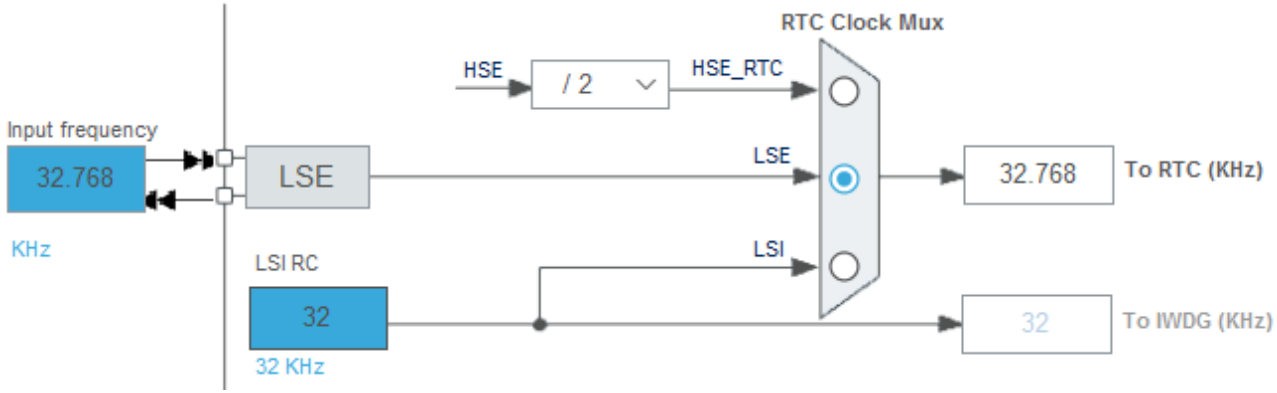
11.1.2 工作原理


1 Hz时钟
32.768kHz
LSE (32.768 Hz)
HSE_RTC
（最高 4 MHz）
LSI
精密校准
RTC_CALR
RTCCLK
512 Hz
1 Hz
ck_apre
（默认值 256 Hz）
RTC_PRER
异步通道
7 位预分频器
（默认值 = 128）
粗略校准
RTC_CALIBR
RTC_PRER
同步
15 位预分频器
（默认值 = 256）
影子寄存器
(RTC_SSR)
ck_spre
（默认值 1 Hz）
日历
影子寄存器
(RTC_TR,
RTC_DR)
RTC_TS
时间戳
寄存器
TSF
RTC_CALIB
输出
控制
RTC_OUT
RTC_AF1
闹钟 A
（RTC_ALRMAR
RTC_ALRMASSR
寄存器）
=
ALRAF
RTC_ALARM
WUCKSEL[1:0]
预分频器
/ 2, 4, 8, 16
RTC_WUTR
16 位唤醒
自动重载寄存器
闹钟 B
（RTC_ALRMBR
RTC_ALRMBSSR
寄存器）
=
ALRBF
WUTF
RTC_AF2
RTC_TAMP1
RTC_TAMP2
备份和 RTC
入侵控制寄存器
TAMPE
TSE

1. RTC的时钟信号源

可以从3个时钟信号源中选择一个作为RTC的时钟：

- LSI，处理器内部的32KHz时钟信号
- LSE，处理器外接的32.768KHz时钟信号
- HSE经过2~31分频后的时钟信号HSE_RTC



一般选择LSE作为RTC的时钟源，因为32.768KHz经过多次2分频后可以得到精确的1Hz时钟信号。

2. 预分频器

如果选用LSE作为RTCCLK，经过异步预分频器128分频后的信号ck_apre是512Hz。512Hz的时钟信号再经过同步预分频器256分频后得到1Hz时钟信号ck_spre。这个1Hz信号用于更新日历，也可以作为周期唤醒定时器的时钟源。

ck_apre和ck_spre经过一个选择器后，可以选择一个时钟信号作为RTC_CALIB时钟信号，这个时钟信号再经过输出控制选择，可以输出到复用管脚RTC_AF1，也就是向外部提供一个512Hz或1Hz的时钟信号。

3. 实时时钟和日历数据

图11-1中有3个影子寄存器，影子寄存器就是计数器的数值暂存寄存器，系统每隔两个RTCCLK周期就将当前的日历值复制到影子寄存器。

当程序读取日期时间数据时，读取的是影子寄存器的内容，不会影响日历计数器的工作。

4. 周期性自动唤醒

RTC内有一个16位自动重载递减计数器可以产生周期性的唤醒中断。唤醒定时器的时钟输入有2个来源：

（1）同步预分频器输出的ck_spre时钟信号，通常是1Hz

（2）RTCCLK经过2、4、8或16分频后的时钟信号

唤醒中断产生信号WUTF，这个信号可以配置输出到复用引脚RTC_AF1

5. 可编程闹钟

RTC有2个可编程闹钟，即闹钟A和闹钟B。

可以设置闹钟的时间和重复方式，闹钟触发时可以产生中断信号ALRAF和ALRBF。

这2个信号和中期唤醒中断信号WUTF一起经过一个选择器，可以选择其中一个信号作为RTC_ALARM闹钟输出信号，再通过输出控制可以输出到复用引脚RTC_AF1。

6. 时间戳（Timestamp）

时间戳就是当某个外部事件（上跳沿或下跳沿变化）发生时的当时的日历时间，例如行车记录仪在发生碰撞时保存发生碰撞时的RTC日期时间数据就是时间戳。

7. 入侵检测（Tamper detection）

tamper的意思是“篡改”

有2个入侵输入检测信号源，RTC_TAMP1和RTC_TAMP2，这两个信号可以映射到复用引脚RTC_AF1和RTC_AF2。可以配置为边沿检测，或带滤波的电平检测。

MCU上有20个32位备份寄存器，在系统主电源关闭或复位时，备份寄存器的数据不会丢失。当检测到入侵事件发生时，MCU就会复位这20个备份寄存器的内容。

检测到入侵事件时会产生中断信号，同时还可以记录时间戳数据。

11.1.3 RTC中断和复用引脚功能小结

中断事件	事件标志	可输出到引脚	输入引脚
闹钟A	ALRAF	RTC_AF1	
闹钟B	ALRBF	RTC_AF1	
周期唤醒	WUTF	RTC_AF1	
时间戳	TSF		RTC_AF1或RTC_AF2
入侵检测1	TAMP1F		RTC_AF1或RTC_AF2
入侵检测2	TAMP2F		RTC_AF1或RTC_AF2

复用引脚RTC_AF1是引脚PC13，RTC_AF2是引脚PI8。只有176引脚的MCU上才有PI8

11.2 闹钟和周期唤醒功能的使用

11.2.1 示例功能

11.2.2 STM32CubeMX配置

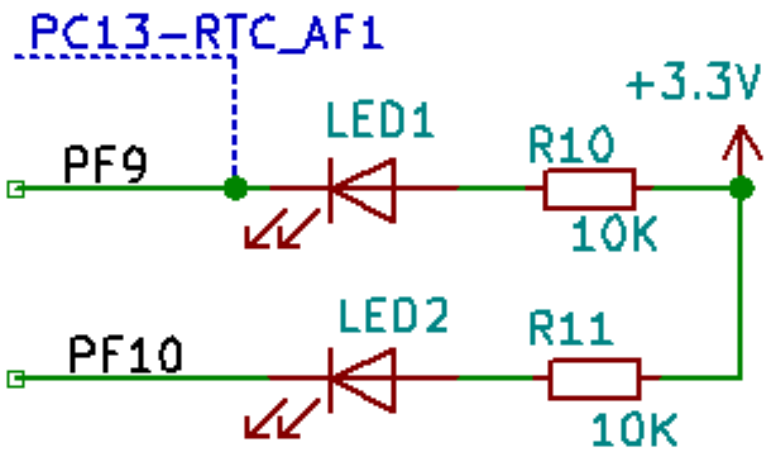
11.2.3 初始化项目代码分析

11.2.4 编写用户功能代码

11.2.1 示例功能

示例Demo11_1RTC使用闹钟A、闹钟B和周期唤醒功能

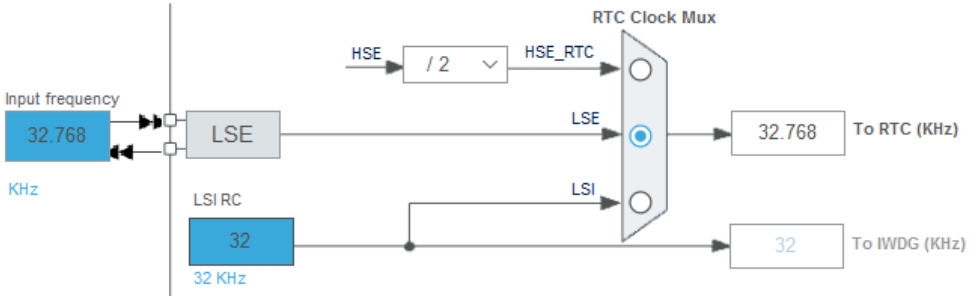
- 使用LSE的32.768kHz时钟作为RTC的时钟源
- 系统复位时初始化RTC日期时间为2019-3-16 7:15:10
- 每1秒钟周期性唤醒，在唤醒中断里读取当前日期和时间，并在LCD上显示
- 将周期唤醒中断信号WUTF输出到复用引脚RTC_AF1（PC13），用杜邦线连接PC13和LED1的引脚PF9。这样，LED1的亮灭由PC13的输出状态控制



- 闹钟A设置为在时间xx:16:05触发，即在每个小时的16分5秒时刻触发闹钟A。对闹钟A中断次数计数，并在LCD上显示
- 闹钟B设置为在时间xx:xx:30触发，即在每分钟的30秒时刻触发闹钟B。对闹钟B中断次数计数，并在LCD上显示

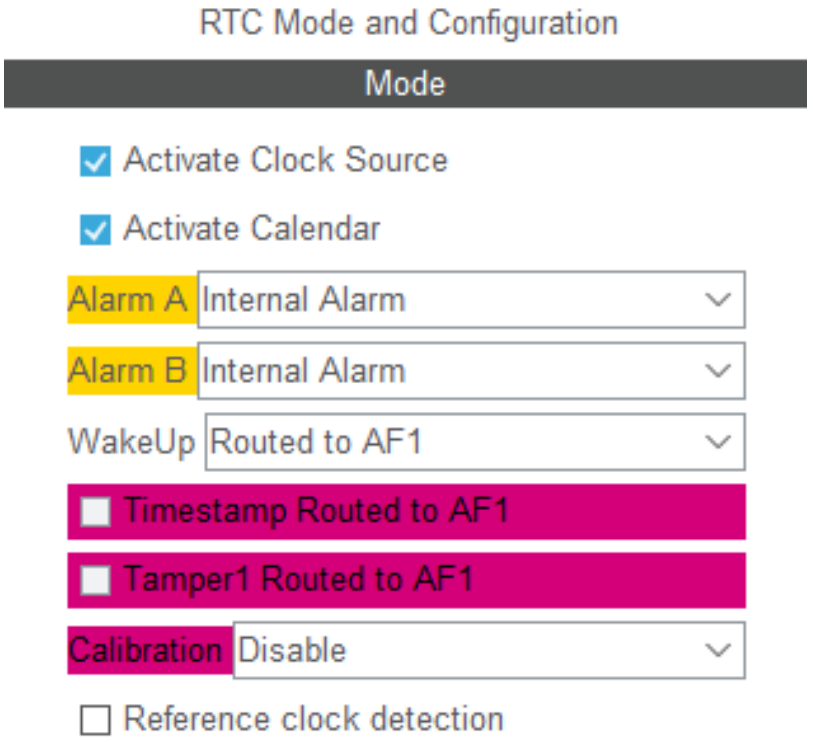
11.2.2 STM32CubeMX配置

在时钟树中配置使用LSE作为RTC的时钟源。



1. RTC模式设置

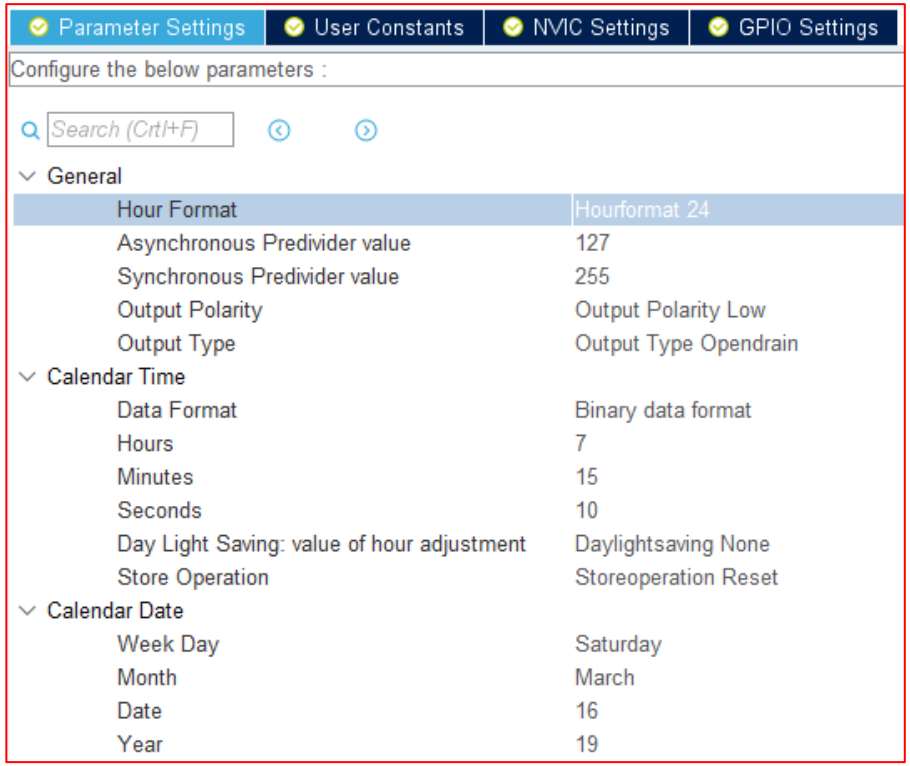
- 启用时钟源和日历。
- 将闹钟A和闹钟B都设置为Internal Alarm
- 将周期唤醒（WakeUp）设置为Routed to AF1，即输出到复用引脚RTC_AF1



2. RTC基本参数设置

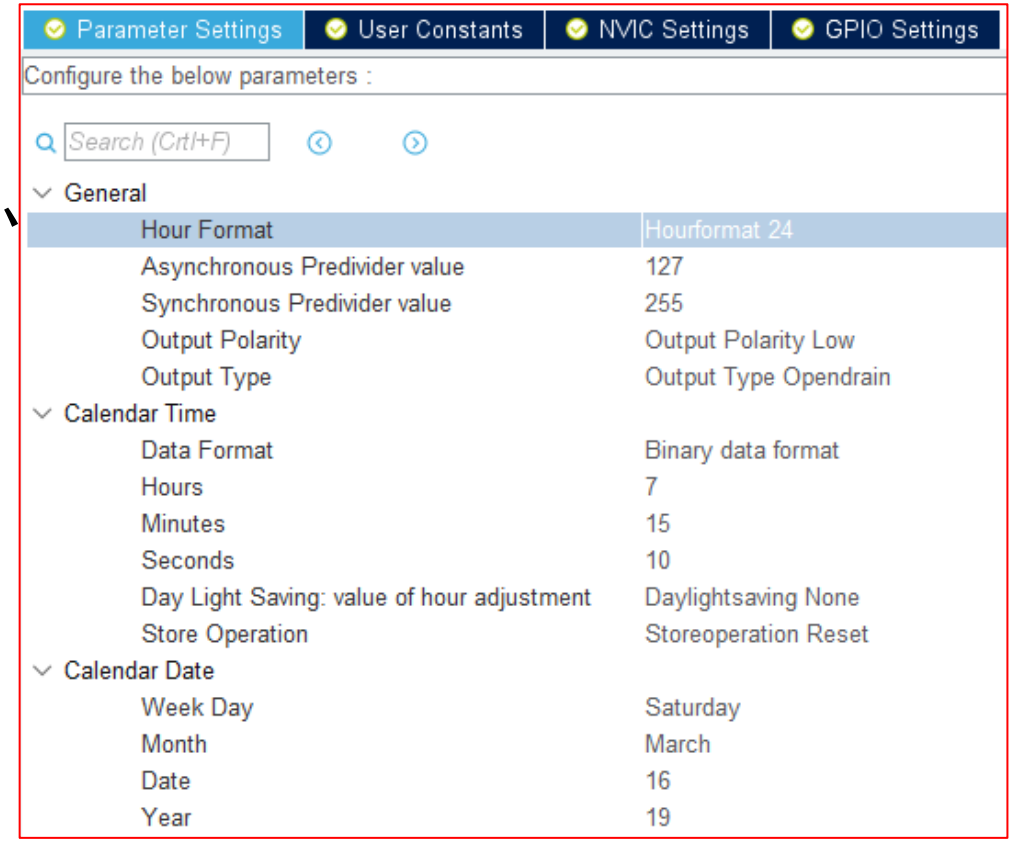
(1)General分组

- Hour Format，时间格式，12或24小时制
- Asynchronous Predivider value，异步预分频器值。127对应128分频
- Synchronous Predivider value，同步预分频器值。255对应256分频
- Output Polority，输出极性。闹钟A、闹钟B、周期唤醒中断信号RTC_ALARM有效时的输出极性
- Output Type，输出类型。复用引脚RTC_AF1的输出类型



（2）Calendar Time分组

- Data Format，数据格式，二进制格式或BCD格式
- 初始化时间数据，包括小时、分钟、秒的数据
- Day Light Saving，夏令时设置，None就是不使用夏令时
- Store Operation，表示是否已经对夏令时设置做修改。设Reset表示未修改夏令时



日历初始化日期数据中，年的设置范围是0-99，表示2000年至2099年。

3. 闹钟定时设置

闹钟的触发时间设置：

设置RTC的初始时间为7:15:10，闹钟A设置为在时间xx:16:05触发，即每个小时的16分5秒时刻触发。

在图11-6中，闹钟A的设置主要是闹钟日期时间设置和屏蔽设置

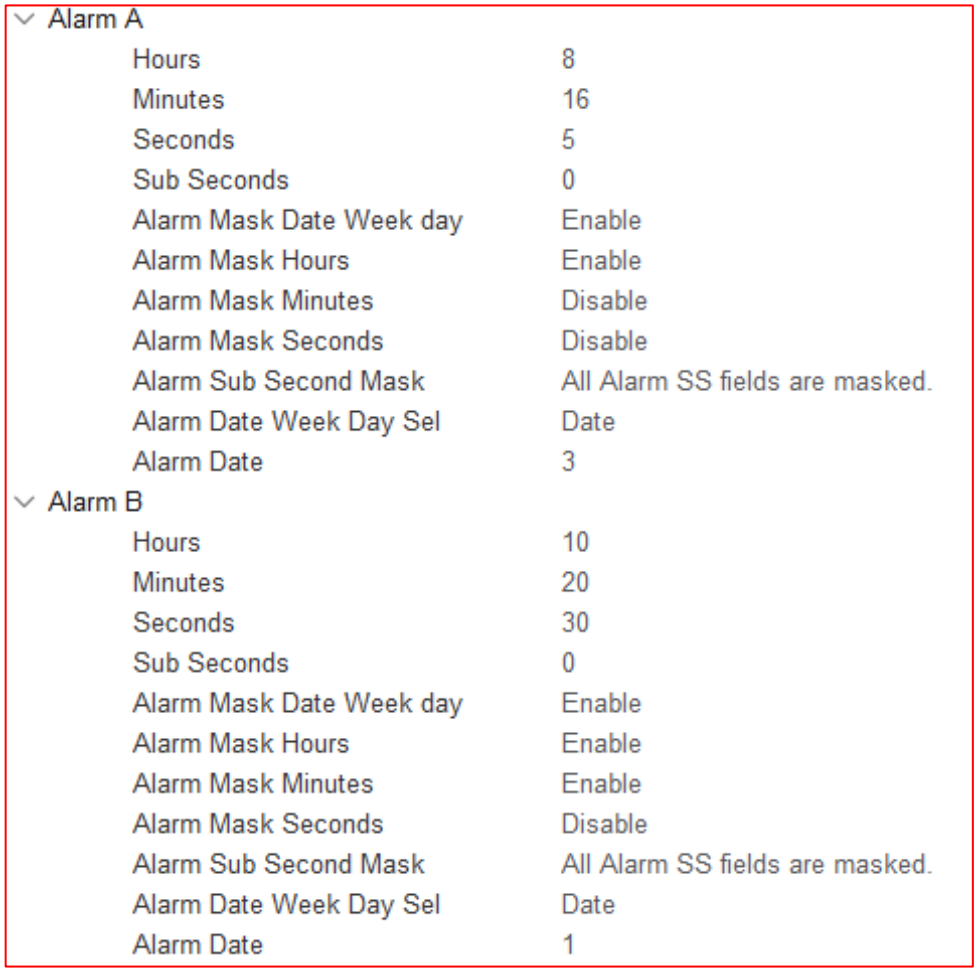


图11-6

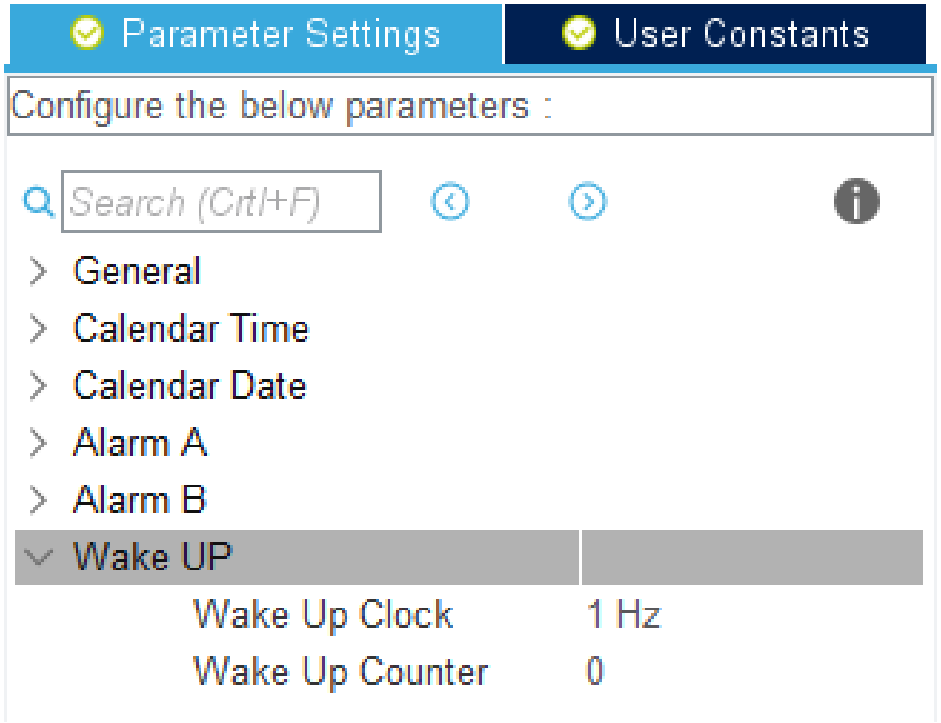
表11-2 闹钟A的参数设置

参数	意义	取值示例	数据范围
Hours	小时	8	0-23
Minutes	分钟	16	0-59
Seconds	秒	5	0-59
Sub Seconds	亚秒	0	0-59
Alarm Mask Date Week day	屏蔽日期	Enable	Enable =屏蔽，闹钟与日期数据无关 Disable =日期数据参与比对
Alarm Mask Hours	屏蔽小时	Enable	Enable =屏蔽，闹钟与小时数据无关 Disable =小时数据参与比对
Alarm Mask Minutes	屏蔽分钟	Disable	Enable =屏蔽，闹钟与分钟数据无关 Disable =分钟数据参与比对
Alarm Mask Seconds	屏蔽秒	Disable	Enable=屏蔽，闹钟与秒数据无关 Disable =秒数据参与比对
Alarm Sub Second Mask	屏蔽亚秒	All Alarm SS fields are masked	All Alarm SS fields are masked =屏蔽，闹钟与亚秒数据无关，其他选项用于亚秒数据比对
Alarm Date Week Day Sel	日期形式	Date	Date=按1-31日表示日期 Weekday=用Monday到Sunday表示星期几
Alarm Data	日期	3	1-31或Monday到Sunday

4. 唤醒周期设置

（1）Wake Up Clock，周期唤醒的时钟源。

- 1Hz，来自于ck_spre的1Hz信号（常用）
- 其他【打开软件解释】

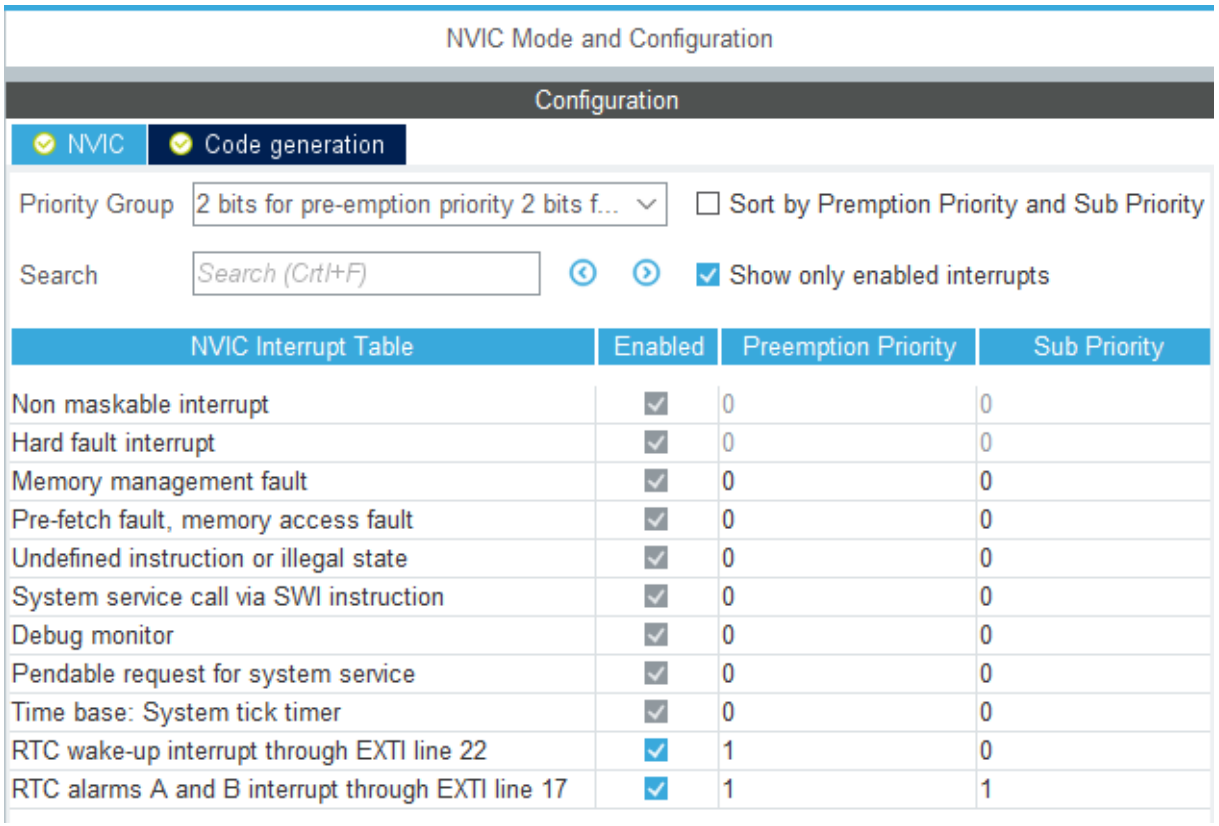


（2）Wake Up Counter，唤醒计数器。表示时钟计数达到这个值时才触发一次WakeUp中断。设置为0，则每个时钟周期中断1次。

5. 中断设置

开启RTC的Wakeup中断，它使用EXTI中断线22，设置抢占优先级为1，次优先级为0

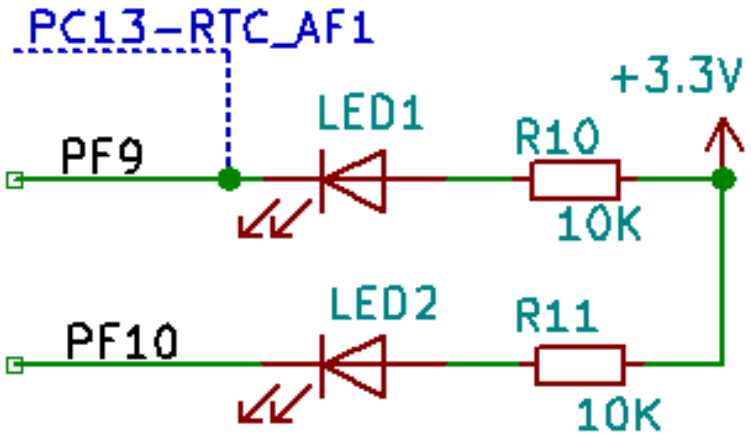
闹钟A和闹钟B共用EXTI中断线17，设置抢占优先级为1，次优先级为1



NVIC Interrupt Table	Enabled	Preemption Priority	Sub Priority
Non maskable interrupt	☑	0	0
Hard fault interrupt	☑	0	0
Memory management fault	☑	0	0
Pre-fetch fault, memory access fault	☑	0	0
Undefined instruction or illegal state	☑	0	0
System service call via SWI instruction	☑	0	0
Debug monitor	☑	0	0
Pendable request for system service	☑	0	0
Time base: System tick timer	☑	0	0
RTC wake-up interrupt through EXTI line 22	☑	1	0
RTC alarms A and B interrupt through EXTI line 17	☑	1	1

6. LED的GPIO引脚设置

用杜邦线将开发板上PC13和PF9引脚的排针相连接，用PC13的输出控制LED1的亮灭。与LED1和LED2连接的引脚PF9和PF10不要做任何设置，使其处于复位后状态。



11.2.3 初始化项目代码分析

1.主程序

主程序框架，外设初始化

```
int main(void)
{
        HAL_Init();  //复位所有外设，初始化Flash接口和Systick
        SystemClock_Config();          //配置系统时钟

  /*  初始化所有已配置外设  */
        MX_GPIO_Init();                    //GPIO初始化
        MX_FSMC_Init();      //FSMC 初始化，用于TFT LCD
        MX_RTC_Init();                     //RTC初始化
        while (1)
        {
        }
}
```

2. RTC初始化函数MX_RTC_Init()

（1） RTC基本参数设置

打开源程序rtc.c源分析

```
#include "rtc.h"
RTC_HandleTypeDef hrtc;                    //表示RTC对象的变量

/*  RTC 初始化函数   */
void MX_RTC_Init(void)
{
  RTC_TimeTypeDef sTime = {0};
  RTC_DateTypeDef sDate = {0};
  RTC_AlarmTypeDef sAlarm = {0};

  /**  RTC 参数设置  */
  hrtc.Instance = RTC;              //指向寄存器RTC
  hrtc.Init.HourFormat = RTC_HOURFORMAT_24;              //24小时制
  hrtc.Init.AsynchPrediv = 127;              //异步预分频器系数
  hrtc.Init.SynchPrediv = 255;              //同步预分频器系数
  hrtc.Init.OutPut = RTC_OUTPUT_WAKEUP;        //输出WakeUp信号到AF1
  hrtc.Init.OutPutPolarity = RTC_OUTPUT_POLARITY_LOW;              //输出低有效
  hrtc.Init.OutPutType = RTC_OUTPUT_TYPE_OPENDRAIN;              //开漏输出
  if (HAL_RTC_Init(&hrtc) != HAL_OK)
  {
    Error_Handler();
  }
```

（2）RTC时间和日期设置

打开源程序rtc.c源分析

```
/**  设置 RTC 的日期和时间  */
 sTime.Hours = 7;
 sTime.Minutes = 15;
 sTime.Seconds = 10;
 sTime.DayLightSaving = RTC_DAYLIGHTSAVING_NONE;                //夏令时
 sTime.StoreOperation = RTC_STOREOPERATION_RESET;
 if (HAL_RTC_SetTime(&hrtc, &sTime, RTC_FORMAT_BIN) != HAL_OK)
 {
   Error_Handler();
 }
 sDate.WeekDay = RTC_WEEKDAY_SATURDAY;
 sDate.Month = RTC_MONTH_MARCH;                                 //月份
 sDate.Date = 16;                  //取值范围1-31
 sDate.Year = 19;                  //取值范围0-99，表示2000-2099年
 if (HAL_RTC_SetDate(&hrtc, &sDate, RTC_FORMAT_BIN) != HAL_OK)
 {           //设置RTC日期和时间
   Error_Handler();
 }
```

自动生成的代码与STM32CubeMX中的设置是对应的

（3）闹钟A和闹钟B的设置

```
/** 闹钟A设置  */
 sAlarm.AlarmTime.Hours = 8;
 sAlarm.AlarmTime.Minutes = 16;
 sAlarm.AlarmTime.Seconds = 5;
 sAlarm.AlarmTime.SubSeconds = 0;
 sAlarm.AlarmTime.DayLightSaving = RTC_DAYLIGHTSAVING_NONE;
 sAlarm.AlarmTime.StoreOperation = RTC_STOREOPERATION_RESET;
 sAlarm.AlarmMask = RTC_ALARMMASK_DATEWEEKDAY|RTC_ALARMMASK_HOURS;
 sAlarm.AlarmSubSecondMask = RTC_ALARMSUBSECONDMASK_ALL;
 sAlarm.AlarmDateWeekDaySel = RTC_ALARMDATEWEEKDAYSEL_DATE;
 sAlarm.AlarmDateWeekDay = 3;
 sAlarm.Alarm = RTC_ALARM_A;
 if (HAL_RTC_SetAlarm_IT(&hrtc, &sAlarm, RTC_FORMAT_BIN) != HAL_OK)
 {
   Error_Handler();
 }
```

```
/**闹钟B设置  */
 sAlarm.AlarmTime.Hours = 10;
 sAlarm.AlarmTime.Minutes = 20;
 sAlarm.AlarmTime.Seconds = 30;
 sAlarm.AlarmMask =
     RTC_ALARMMASK_DATEWEEKDAY|RTC_ALARMMASK_HOURS
                    |RTC_ALARMMASK_MINUTES;
 sAlarm.AlarmDateWeekDay = 1;
 sAlarm.Alarm = RTC_ALARM_B;
 if (HAL_RTC_SetAlarm_IT(&hrtc, &sAlarm, RTC_FORMAT_BIN) != HAL_OK)
 {
   Error_Handler();
 }
```

闹钟A和闹钟B需要分别设置

（4）周期唤醒设置

```
/** 周期性唤醒设置  */
 if (HAL_RTCEx_SetWakeUpTimer_IT(&hrtc, 0,
                                 RTC_WAKEUPCLOCK_CK_SPRE_16BITS) != HAL_OK)
 {
   Error_Handler();
 }
```

RTC初始化的MSP函数HAL_RTC_MspInit()

```
void HAL_RTC_MspInit(RTC_HandleTypeDef* rtcHandle)
{
  if(rtcHandle->Instance==RTC)
  {
    __HAL_RCC_RTC_ENABLE();    // 使能RTC时钟
    /* RTC 中断初始化 */
    HAL_NVIC_SetPriority(RTC_WKUP_IRQn, 1, 0);
    HAL_NVIC_EnableIRQ(RTC_WKUP_IRQn);
    HAL_NVIC_SetPriority(RTC_Alarm_IRQn, 1, 1);
    HAL_NVIC_EnableIRQ(RTC_Alarm_IRQn);
  }
}
```

函数HAL_RTC_Init()内部会调用HAL_RTC_MspInit()，rtc.c里重新实现了这个函数，功能就是设置RTC的两个中断。

3. 闹钟和周期唤醒中断响应函数分析

闹钟A和闹钟B共用EXTI22中断，周期唤醒使用EXTI17中断，在文件stm32f4xx_it.c自动生成了相应的ISR函数框架

```
/* RTC 周期唤醒中断响应函数  */
void RTC_WKUP_IRQHandler(void)
{
      HAL_RTCEx_WakeUpTimerIRQHandler(&hrtc);
}

/* RTC 闹钟A和闹钟B中断响应函数   */
void RTC_Alarm_IRQHandler(void)
{
      HAL_RTC_AlarmIRQHandler(&hrtc);
}
```

周期唤醒中断回调函数：HAL_RTCEx_WakeUpTimerIRQHandler()

闹钟A的回调函数：HAL_RTC_AlarmAEventCallback()

闹钟B的回调函数：HAL_RTCEx_AlarmBEventCallback()

11.2.4 编写用户功能代码

1. 主程序

RTC在初始化之后就开始运行

```
int main(void)
{
  HAL_Init();             //复位所有外设，初始化Flash接口和Systick
  SystemClock_Config();         //配置系统时钟

  /*  初始化所有已配置外设  */
  MX_GPIO_Init();                 //GPIO初始化
  MX_FSMC_Init();                 //FSMC 初始化，用于TFT LCD
  MX_RTC_Init();                  //RTC初始化

  /* USER CODE BEGIN 2 */
  TFTLCD_Init();      //LCD软件初始化
  LCD_ShowString(10,10,(uint8_t *)"Demo11_1:RTC");
  /* USER CODE END 2 */
   while (1)
  {
  }
}
```

2. 中断回调函数的实现

```
//周期唤醒中断回调函数
void HAL_RTCEx_WakeUpTimerEventCallback(RTC_HandleTypeDef *hrtc)
{
	RTC_TimeTypeDef sTime;
	RTC_DateTypeDef sDate;
//读取时间和日期，必须都读取出来，否则无法解锁，就不能连续读取了
	if (HAL_RTC_GetTime(hrtc, &sTime,  RTC_FORMAT_BIN) == HAL_OK)
	{
		HAL_RTC_GetDate(hrtc, &sDate,  RTC_FORMAT_BIN);
		uint16_t  xPos=20, yPos=50;
		//显示日期  年-月-日
		char str[40];
		sprintf(str,"RTC Date= %4d-%2d-%2d", 2000+sDate.Year, sDate.Month,
	sDate.Date);
		LCD_ShowStr(xPos,yPos, (uint8_t*)str);
		//显示 时间  hh:mm:ss
		yPos=yPos+LCD_SP15;
		sprintf(str,"RTC Time= %2d:%2d:%2d", sTime.Hours, sTime.Minutes,
	sTime.Seconds);
		LCD_ShowStr(xPos,yPos, (uint8_t*)str);
	}
}
```

使用函数HAL_RTC_GetTime()读取RTC的当前时间，再使用HAL_RTC_GetDate()读取当前日期。

注意，调用HAL_RTC_GetTime()之后必须调用HAL_RTC_GetDate()以解锁数据，才能连续更新日期和时间。因为调用HAL_RTC_GetTime()时会将日历影子寄存器的当前值锁定，直到日期数据被读出。所以，即使不使用日期数据，也需要调用HAL_RTC_GetDate()读取出日期数据。

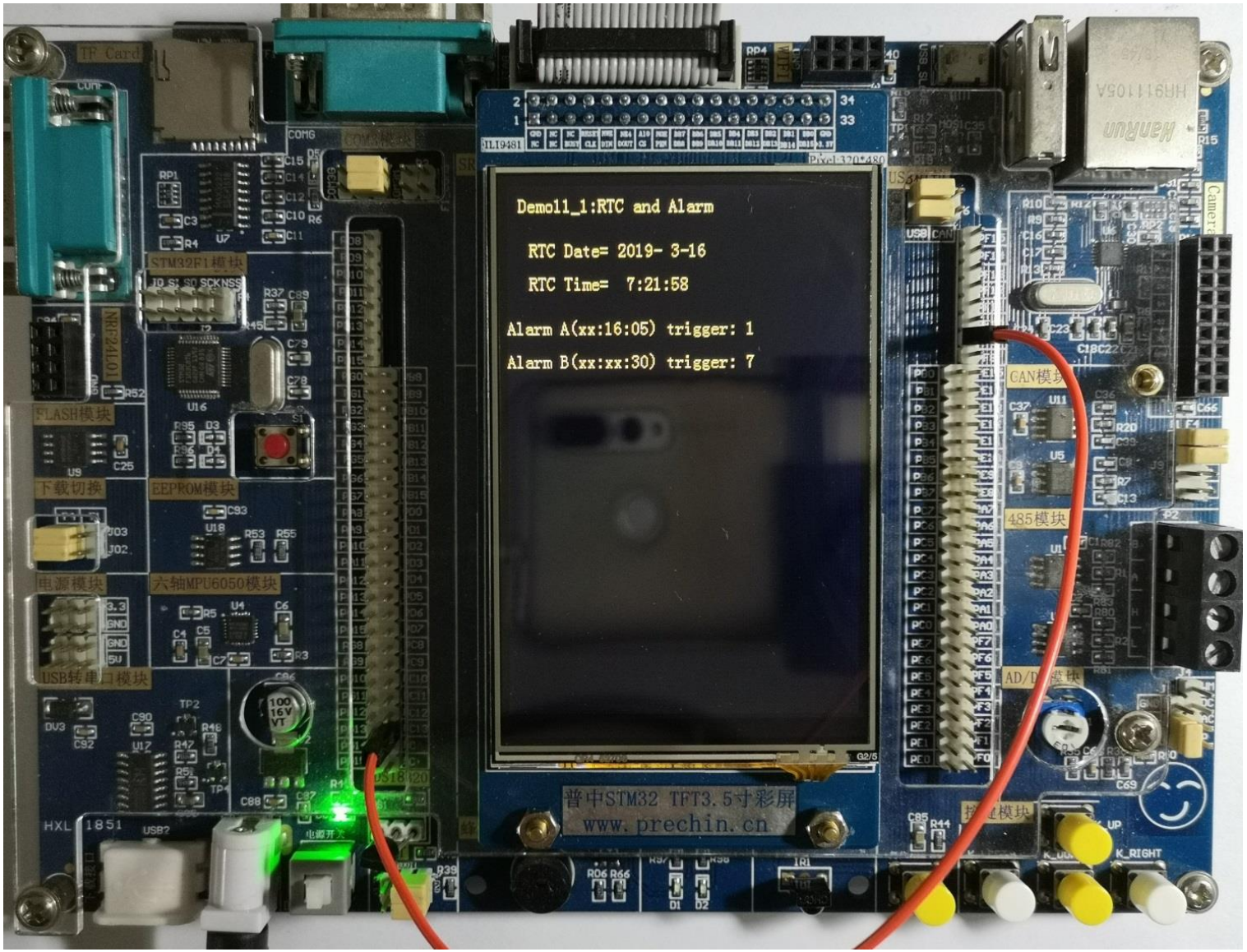
```
//闹钟A中断的回调函数
void HAL_RTC_AlarmAEventCallback(RTC_HandleTypeDef *hrtc)
{
      uint16_t        yPos=120;
      char *infoA="Alarm A(xx:16:05) trigger: ";
      LCD_ShowString(0, yPos, (uint8_t *)infoA);
      triggerCntA++;  //闹钟A触发次数加1
      LCD_ShowUint(LCD_CurPosX, yPos, triggerCntA);  //显示中断次数
}

//闹钟B中断的回调函数
void HAL_RTCEx_AlarmBEventCallback(RTC_HandleTypeDef *hrtc)
{
      uint16_t        yPos=150;
      char infoB[]="Alarm B(xx:xx:30) trigger: ";
      LCD_ShowString(0, yPos, (uint8_t *)infoB);
      triggerCntB++;  //闹钟B触发次数加1
      LCD_ShowUint(LCD_CurPosX, yPos, triggerCntB);
}
```

闹钟A在xx:16:05时刻触发，闹钟B在xx:xx:30时刻触发

示例运行效果

- 在LCD上以“yyyy-m-d”的形式显示日期，如“2019-3-16”
- 在LCD的下一行上以“hh:mm:ss”的形式显示时间，如“7:23:16”
- 如果将PC13与LED1的引脚PF9用杜邦线连接。每秒钟LED1会闪亮一下，这是因为RTC_AF1输出了一个低电平信号，只是这个信号持续时间比较段，LED1只是一闪即灭
- 闹钟A在xx:16:05时刻触发，LCD上显示的计数值变化
- 闹钟B在xx:xx:30时刻触发，LCD上显示的计数值变化

示例运行效果


Demo11_1:RTC and Alarm
RTC Date= 2019- 3-16
RTC Time= 7:21:58
Alarm A(xx:16:05) trigger: 1
Alarm B(xx:xx:30) trigger: 7
普中STM32 TFT3.5寸彩屏
www.prechin.cn

练习任务

阅读书上11.3节备份寄存器部分的内容，编程实现示例Demo11_2RTC_BKUP的功能