嵌入式系统课程设计报告

课程设计题目：基于触摸屏控制的多功能电子钟

专业班级：不告诉你

姓　　名：nosteglic

学　　号：不告诉你

指导教师：不告诉你

2020 年 11 月

目录

一、 课程设计目的

1、 进一步巩固掌握嵌入式系统课程所学 STM32F4 各功能模块的工作原理；
2、 进一步熟练掌握 STM32F4 各功能模块的配置与使用方法；
3、 进一步熟练掌握开发环境 Keil MDK5 的使用与程序调试技巧；
4、 自学部分功能模块的原理、配置与使用方法，培养自学能力；
5、 培养设计复杂嵌入式应用软、硬件系统的分析与设计能力。

二、 课程设计内容

1、查阅资料，自学 STM32F4 的 RTC 模块，完成 RTC 的配置；
2、查阅资料，学习 STM32F4 与 LCD 的接口设计，完成 LCD 液晶屏驱动程序的设计，将时间、日期、星期等日历信息显示在 LCD 上；
3、能进行正常的日期、时间、星期显示；
4、有校时、校分功能，可以使用按键校时、校分，也可以通过串口调试助手由主机传送时间参数进行校时、校分；
5、行整点报时并有闹钟功能，闹钟时间可以设置多个；
6、系统关机后时间能继续运行，下次开机时间应准确；
7、查阅资料，学习 STM32F4 内部温度传感器的配置，采集、计算片内温度并显示在 LCD 上；
8、其他功能，自由发挥扩展。

三、 实验方案分析与设计

此模块描述课程设计中用到的硬件资源或模块工作原理，硬件电路的连接、模块的库函数配置方法（用到的主要库函数、配置步骤等）。

在此模块中，小组成员分别对自己所使用的硬件进行描述，并整合获得完整的硬件资源描述。在下一模块，小组成员将只对各自负责的模块进行详细说明和解释。

本次课程设计用到了以下硬件资源。

3.1 RTC

3.1.1 工作原理

STM32F4 的 RTC 是一个独立的 BCD 定时器/计数器，它提供一个日历时钟（包含年月日时分秒信息）、两个可编程闹钟（ALARM A 和 ALARM B）中断和一个具有中断功能的周期性可编程唤醒标志。

RTC 模块和时钟配置是在后备区域，在系统复位或从待机模式唤醒后 RTC 的设置和时间维持不变，只要后备区域供电正常，RTC 就会一直运行下去。

1、 时钟与分频

我们选择 LSE 作为时钟源（RTCCLK），设置 RTC 的可编程预分配器，提供 1Hz 的时钟。可编程预分配器（RTC_PRER）分为 2 个部分：

（1）7 位异步预分配器，通过 RTC_PRER 寄存器 PREDIV_A 位配置；

（2）15 位同步预分配器，通过 RTC_PRER 寄存器 PREDIV_S 位配置。

利用下面的公式计算得到 1Hz 的F_{ck_spre}：

$$F_{ck\_spre} = \frac{F_{rtcclk}}{(PREDIV\_S + 1) \times (PREDIV\_A + 1)}$$

其中，为了最大程度降低功耗，设置$PREDIV_A = 0X7F$，$PREDIV\_S = 0XFF$。

2、 日历时间（RTC_TR）和日期（RTC_DR）寄存器

RTC_TR 和 RTC_DR 分别用来存储/设置时间和日期。每隔 2 个 RTCCLK 周期，当前日历值就会复制到影子寄存器，并置位 RTC_ISR 寄存器的 RSF 位。读取 RTC_TR 和 RTC_DR 可获得当前日历和日期的 BCD 码信息，需要进行相应转换得到十进制数据。

3、 闹钟模块

RTC 单元提供两个可编程闹钟（ALARM A 和 ALARM B）中断，本次课设用到了 ALARM A，由 RTC_CR 寄存器的 ALRAE 位置 1 来使能。当日历的秒、分、小时、日期分别与闹钟寄存器 RTC_ALRMAR 中的值匹配时，则可以产生闹钟中断。

4、 周期性自动唤醒模块

周期性唤醒标志由一个 16 位可编程自动重载递减计数器生成。唤醒定时器范围可扩展至 17 位。通过将 RTC_CR 寄存器中的 WUTIE 位置 1 来使能周期性唤醒中断时，可以使 STM32F4 退出低功耗模式。系统复位以及低功耗模式（睡眠、停机和待机）对唤醒定时器没有任何影响，它仍然可以正常工作。

3.1.2 硬件电路连接

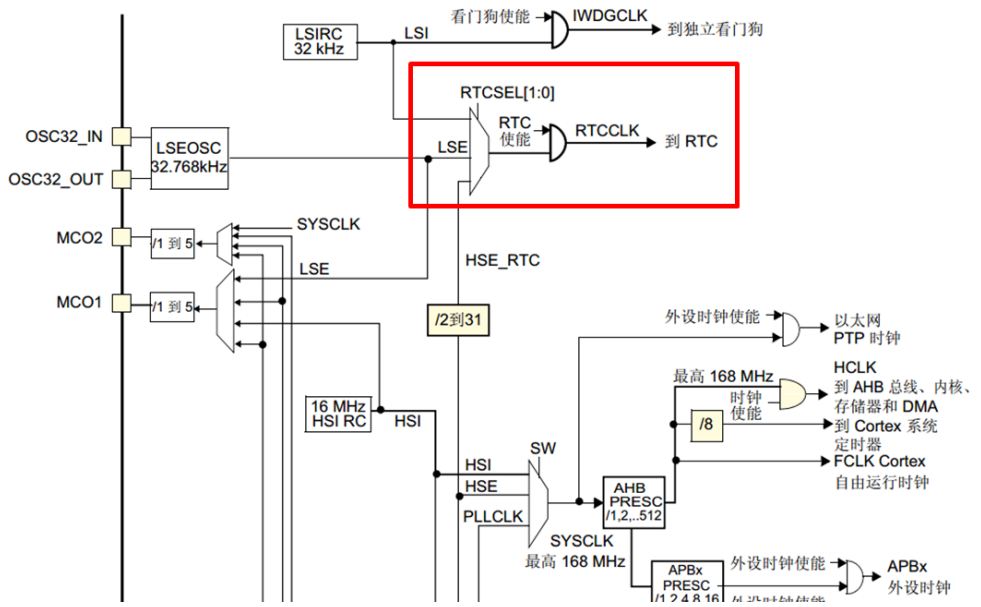


图 3.1-1 RTC 模块

3.1.3 库函数配置方法

（1）RTC 日历配置一般步骤

1° 使能 PWR 时钟

```
RCC_APB1PeriphClockCmd(RCC_APB1Periph_PWR, ENABLE);
```

2° 使能后备寄存器写访问

```
PWR_BackupAccessCmd(ENABLE);
```

3° 配置 RTC 时钟源，使能 RTC 时钟：

```
RCC_LSEConfig(RCC_LSE_ON);          //需要使用 LSE，所以必须开启 LSE
RCC_RTCCLKConfig(RCC_RTCCLKSource_LSE);
RCC_RTCCLKCmd(ENABLE);
```

4° 初始化 RTC

```
RTC_Init(&RTC_InitStructure);
//同步/异步分频系数和时钟格式
//RTC_InitStructure.RTC_AsynchPrediv = 0x7F;//RTC 异步分频系数(1~0X7F)
//RTC_InitStructure.RTC_SynchPrediv = 0xFF;//RTC 同步分频系数(0~7FFF)
//RTC_InitStructure.RTC_HourFormat = RTC_HourFormat_24;//RTC 设置为,24 小时格式
```

5° 设置时间

```
RTC_SetTime();
```

6° 设置日期

```
RTC_SetDate();
```

（2）RTC 闹钟配置步骤

1° RTC 初始化好相关参数；

2° 关闭闹钟；

```
RTC_AlarmCmd(RTC_Alarm_A,DISABLE);          //关闭 RTC 闹钟服务
```

3° 配置闹钟参数；

```
RTC_SetAlarm();                             //初始化闹钟参数
```

4° 开启闹钟；

```
RTC_AlarmCmd(RTC_Alarm_A,EABLE);            //开启 RTC 闹钟服务
```

5° 开启配置闹钟中断；

```
RTC_ITConfig();                             //开启 RTC 中断
EXTI_Init();                                //配置中断线等
NVIC_Init();                                //配置中断优先级等
```

6° 编写中断服务函数；

```
RTC_Alarm_IRQHandler();                     //RTC 闹钟中断服务函数
```

（3）RTC 周期性自动唤醒配置步骤

1° RTC 初始化好相关参数；

2° 关闭 WakeUp；

```
RTC_WakeUpCmd(DISABLE);                     //关闭 RTC 周期性自动唤醒服务
```

3° 配置 WakeUp 时钟分频系数/来源

```
RTC_WakeUpClockConfig();                    //配置 RTC 的 WakeUp 参数
```

4° 设置 WakeUp 自动装载寄存器；

```
RTC_SetWakeUpCounter();                     //配置 WakeUp 自动装载寄存器
```

5° 使能 WakeUp；

```
RTC_WakeUpCmd(  ENABLE);                    //开启 RTC 周期性自动唤醒服务
```

6° 开启配置闹钟中断：

```
RTC_ITConfig();                             //开启 RTC 中断
EXTI_Init();                                //配置中断线等
NVIC_Init();                                //配置中断优先级等
```

7° 编写中断服务函数；

```
RTC_WKUP_IRQHandler();                      //RTC 的 WakeUp 中断服务函数
```

3.2 LED

3.2.1 工作原理

LED0 和 LED1 分别对应 IO 口 F9 和 F10。将 F9、F10 位设置为 0，灯就亮；设置为 1，灯就灭。

3.2.2 硬件电路连接

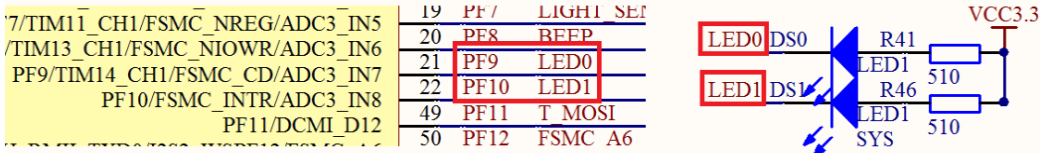


图 3.2-1 LED 与 STM32F4 连接图

3.2.3 库函数配置方法

1° 使能 GPIOF 时钟

```
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOF, ENABLE);
```

2° 初始化 GPIO

```
GPIO_Init(GPIOF, &GPIO_InitStructure);
```

3.3 LCD

3.3.1 工作原理

TFT-LCD 即薄膜晶体管液晶显示器，它在液晶显示屏的每一个像素上都设置有一个薄膜晶体管（TFT），可有效地克服非选通时的串扰，使显示液晶屏的静态特性与扫描线数无关，因此大大提高了图像质量。

所用 LCD 的颜色数据为 16 位，最低 5 位代表蓝色，中间 6 位为绿色，最高 5 位为红色。数值越大，表示该颜色越深。

TFTLCD 的通用使用流程如下：

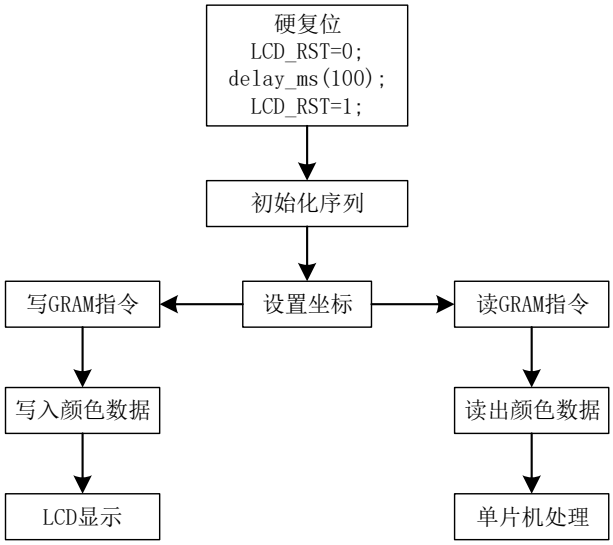


图 3.3-1 TFTLCD 使用流程

其中，硬复位和初始化序列只需要执行一次即可。

画点流程为：设置坐标→写 GRAM 指令→写入颜色数据。

读点流程为：设置坐标→读 GRAM 指令→读取颜色数据。

3.3.2 硬件电路连接

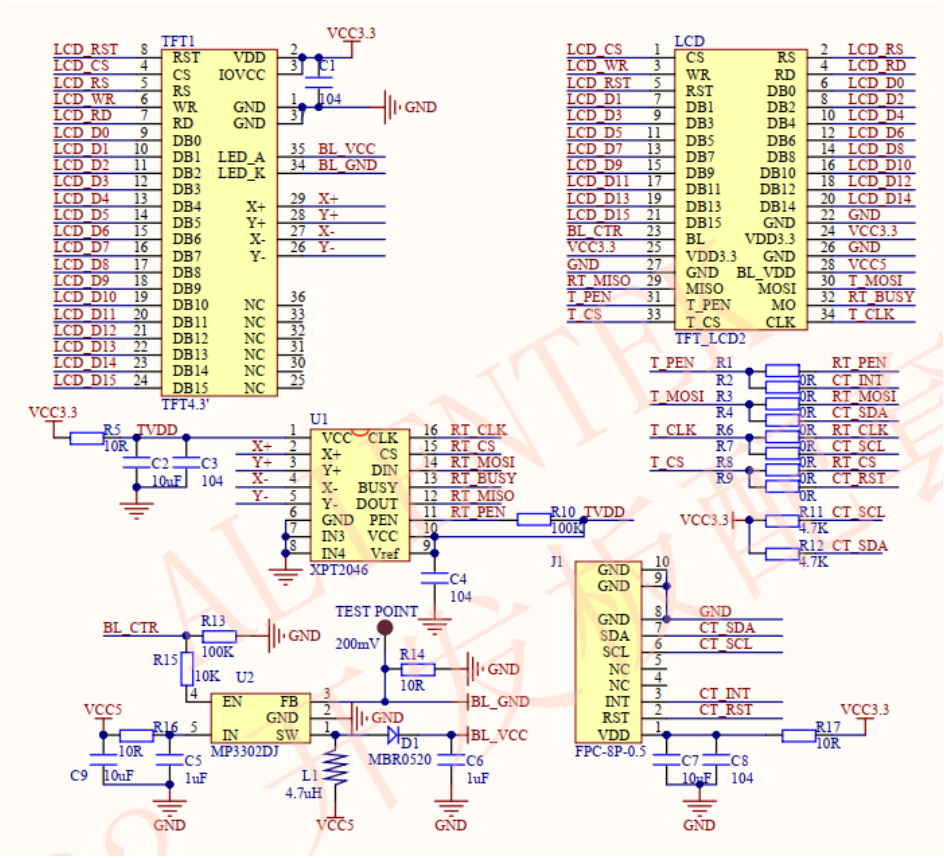


图 3.3-2 4.3' TFTLCD 原理图

3.3.3 库函数配置方法

1°GPIO，FSMC 时钟使能

```
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB|RCC_AHB1Periph_GPIOD|RCC_AHB1Periph
_GPIOE|RCC_AHB1Periph_GPIOF|RCC_AHB1Periph_GPIOG, ENABLE);
//使能 PD,PE,PF,PG 时钟
RCC_AHB3PeriphClockCmd(RCC_AHB3Periph_FSMC,ENABLE);//使能 FSMC 时钟
```

2°GPIO 初始化设置

```
GPIO_Init();
```

3°引脚复用映射设置

```
GPIO_PinAFConfig();
```

4°FSMC 初始化

```
FSMC_NORSRAMInit(&FSMC_NORSRAMInitStructure);
```

5°使能 FSMC

```
FSMC_NORSRAMCmd(FSMC_Bank1_NORSRAM4, ENABLE);
```

6°不同的 LCD 驱动器不同的初始化设置

```
根据读到的 LCD ID，对不同的驱动器执行不同的初始化代码。
```

3.4 TOUCH

3.4.1 工作原理

本次实验使用的 4.3 寸 TFTLCD 模块自带的触摸屏，采用的是投射式电容触摸屏（交互电容类型）。它是在玻璃表面的横向和纵向的 ITO 电极的交叉处形成电容。交互电容的扫描方式就是扫描每个交叉处的电容变化，来判定触摸点的位置。当触摸的时候就会影响到相邻电极的耦合，从而改变交叉处的电容量，交互电容的扫面方法可以侦测到每个交叉点的电容值和触摸后电容变化，因而它需要的扫描时间与自我电容的扫描方式相比要长一些，需要扫描检测 X×Y 根电极。

电容触摸屏对工作环境的要求是比较高的，在潮湿、多尘、高低温环境下面，都是不适合使用电容屏的。

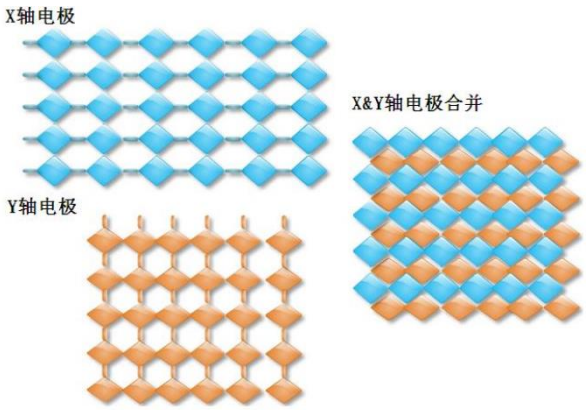


图 3.4-1 投射式电容屏电极矩阵示意图

其中，图中的电极实际是透明的，X、Y 轴的透明电极电容屏的精度、分辨率与 X、Y 轴的通道数有关。通道数越多，精度越高。

电容触摸屏的优点： 手感好、无需校准、支持多点触摸、透光性好。

电容触摸屏的缺点：成本高、精度不高、抗干扰能力差。

本实验为了简化编程复杂度，设置单点触摸。

3.4.2 硬件电路连接

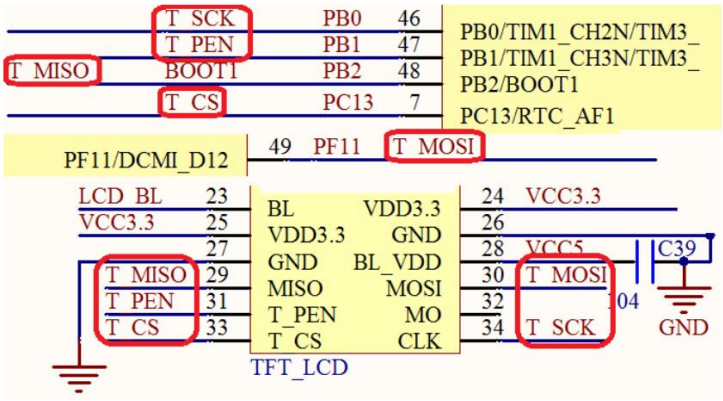


图 3.4-2 触摸屏与 STM32F4 连接图

3.4.3 库函数配置方法

1° GPIO 时钟使能

```
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB|RCC_AHB1Periph_GPIOC|RCC_AHB1Periph_GPIOF, ENABLE);//使能 GPIOB,C,F 时钟
```

2° GPIO 初始化

```
GPIO_Init();
```

3° 第一次读取初始化

```
TP_Read_XY(&tp_dev.x[0],&tp_dev.y[0]);
```

4° 初始化 24CXX

```
AT24CXX_Init();
```

3.5 TPAD

3.5.1 工作原理

通过检测电容放电时间的方法来判断是否对 TPAD 有触摸行为。

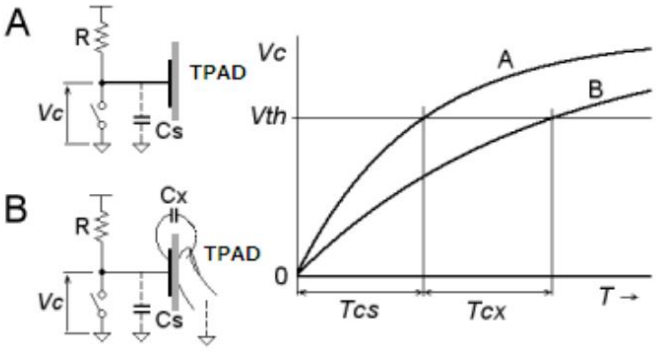


图 3.5-1 TPAD 按键原理

其中，R是外接的电容充电电阻，C_S是没有触摸按下时 TPAD 和 PCB 之间的杂散电容，C_X是有手指按下时手指与 TPAD 之间形成的电容。先用开关将C_S上的电放尽，然后断开开关，让R给C_S充电。当无手指触摸时，C_S的充电曲线如图中的 A 曲线；当有手指触摸时，C_S的充电曲线如图中的 B 曲线。在 A、B 两种情况下，V_C达到V_{th}的时间分别为T_{CS}和$T_{CS} + T_{CX}$。当充电时间在T_{CS}附近，就可以认为没有触摸，而重带你时间大于$T_{CS} + T_{CX}$时，就认为有触摸按下。

实验中，使用定时器 TIM2 的通道 1 进行输入捕获，在 MCU 每次复位重启的时候，执行一次捕获检测，记录此时的值为 tpad_default_val 作为判断依据。在后续的捕获检测，通过与 tpad_default_val 的对比来判断是不是有触摸发生。

3.5.2 硬件电路连接

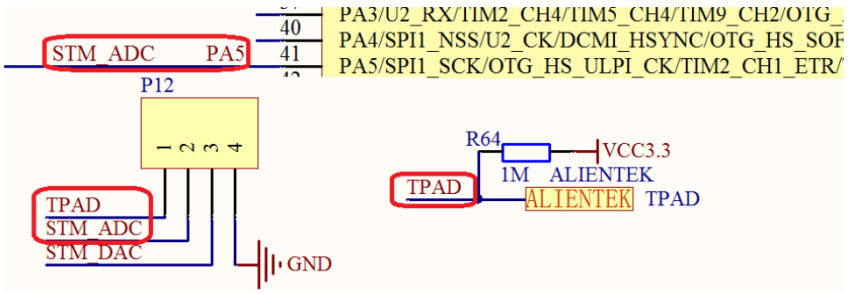


图 3.5-2 TPAD 和 STM32F4 连接图

3.5.3 库函数配置方法

1° TIM2 定时器、GPIOA 时钟初始化

```
RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2,ENABLE);        //TIM2 时钟使能
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);   //使能 PORTA 时钟
```

2° 引脚复用映射设置

```
GPIO_PinAFConfig(GPIOA,GPIO_PinSource5,GPIO_AF_TIM2);//GPIOA 复用为定时器 2
```

3° GPIO 初始化

```
GPIO_Init(GPIOA,&GPIO_InitStructure);
```

4° TIM2 初始化

```
TIM_TimeBaseInit(TIM2, &TIM_TimeBaseStructure);
TIM_ICInit(TIM2, &TIM2_ICInitStructure);
```

5° TIM2 使能

```
TIM_Cmd(TIM2,ENABLE);
```

3.6 USART

3.6.1 工作原理

串口作为 MCU 的重要外部接口，同时也是软件开发重要的调试手段。本次实验我们利用 USART 进行 printf，输出调试信息帮助编程。

调试时，打开串口调试助手，选择相应的 COM 端口，将波特率设置为 115200。

3.6.2 硬件电路连接

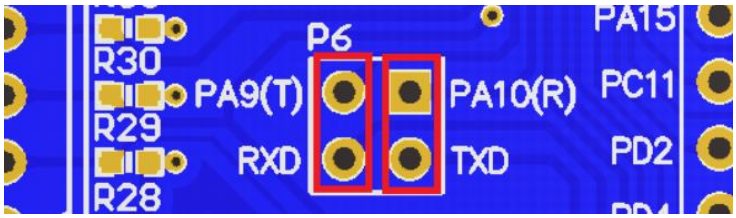


图 3.6-1 USART 连接图

把 P6 的 RXD 和 TXD 用跳线帽与 PA9 和 PA10 连接起来。

3.6.3 库函数配置方法

1° GPIO、USART 时钟使能

```
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA,ENABLE); //使能 GPIOA 时钟
RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1,ENABLE);//使能 USART1 时钟
```

2° GPIO 引脚复用映射

```
GPIO_PinAFConfig(GPIOA,GPIO_PinSource9,GPIO_AF_USART1);
//GPIOA9 复用为 USART1
GPIO_PinAFConfig(GPIOA,GPIO_PinSource10,GPIO_AF_USART1);
//GPIOA10 复用为 USART1
```

3° GPIO 端口初始化

```
GPIO_Init(GPIOA,&GPIO_InitStructure); //初始化 PA9，PA10
```

4° 串口参数初始化

```
USART_Init(USART1, &USART_InitStructure);
//设置波特率、字长、资产校验等参数
```

5° 使能串口

```
USART_Cmd(USART1, ENABLE);
```

6° 串口数据收发

```
void USART_SendData();                    //发数据，写 DR
uint16_t USART_ReceiveData();             //接受数据，读 DR
```

7° 串口传输状态获取

```
FlagStatus USART_GetFlagStatus();         //状态标志
void USART_ClearITPendingBit();           //中断标志
```

3.7 ADC

3.7.1 工作原理

STM32F4 有一个内部的温度传感器，可以用来测量 CPU 及周围的温度(TA)。该温度传感器在内部和 ADC1_IN16 输入通道相连接，此通道把传感器输出的电压转换成数字值。STM32F4 的内部温度传感器支持的温度范围为：-40~125 度。精度为±1.5℃左右。

通过以下公式计算出当前温度。

$$T(℃) = \{(V_{sense} - V_{25})/Avg\_Slope\} + 25$$

其中：

（1）$V_{25} = V_{sense}$在 25 度时的数值（典型值为：0.76）；

（2）Avg_Slope温度与V_{sense}曲线的平均斜率（单位为 mv/℃或 uv/℃）；

3.7.2 硬件电路连接

ADC 属于 STM32F4 内部资源，实际上我们只需要软件设置就可以正常工作。

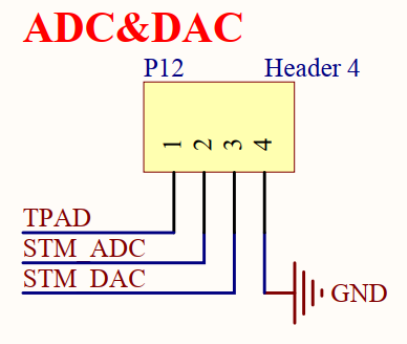


图 3.7-1 ADC 连接图

3.7.3 库函数配置方法

1° 开启 PA 口时钟和 ADC1 时钟，设置 PA1（ADC1 通道 1） 为模拟输入。

```
RCC_AHB1PeriphClockCmd (RCC_AHB1Periph_GPIOA, ENABLE);      //使能 GPIOA 时钟
RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);        //使能 ADC1 时钟
GPIO_Init();                                                //初始化 IO 口
```

2° 复位 ADC1。

```
RCC_APB2PeriphResetCmd(RCC_APB2Periph_ADC1,ENABLE);         //ADC1 复位
RCC_APB2PeriphResetCmd(RCC_APB2Periph_ADC1,DISABLE);        //ADC1 复位结束
```

3° 初始化 ADC_CCR 寄存器，同时设置 ADC1 分频因子

```
ADC_CommonInit();                                           //设置 ADC1 分频数
```

4° 初始化 ADC1 参数，设置 ADC1 的工作模式以及规则序列的相关信息。

```
void ADC_Init(ADC_TypeDef* ADCx, ADC_InitTypeDef* ADC_InitStruct)；//初始化参数
```

5° 使能 ADC

```
ADC_Cmd(ADC1, ENABLE);                                      // 开启 AD 转换器
```

6° 配置规则通道参数：

```
ADC_RegularChannelConfig();                                 //配置通道参数
```

7° 软件开启转换

```
ADC_SoftwareStartConv(ADC1);                                //开启 ADC1 转换
```

8° 等待转换完成， 读取 ADC 值

```
ADC_GetConversionValue(ADC1);                               //读取 ADC 值
```

9° 启动内部温度传感器

```
ADC_TempSensorVrefintCmd(ENABLE);                           //使能内部温度传感器
```

3.8 BEEP

3.8.1 工作原理

蜂鸣器是一种一体化结构的电子讯响器。开发板板载的蜂鸣器是电磁式的有源蜂鸣器。我们是通过三极管扩流后再驱动蜂鸣器，IO 只需要提供不到 1mA 的电流就足够了。

3.8.2 硬件电路连接

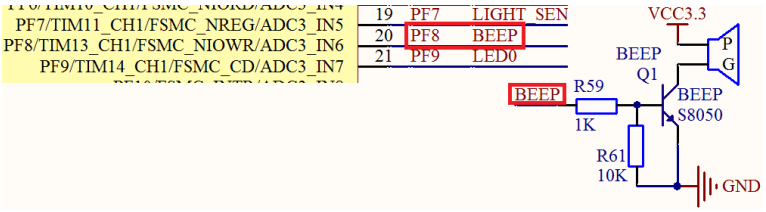


图 3.8-1 BEEP 和 STM32F4 连接图

3.8.3 库函数配置方法

1° 使能 GPIO 时钟

```
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOF, ENABLE);          //使能 GPIOF 时钟
```

2° 蜂鸣器 GPIO 引脚初始化

```
GPIO_Init(GPIOF, &GPIO_InitStructure);                          //初始化 GPIO
```

3° 初始关闭蜂鸣器

```
GPIO_ResetBits(GPIOF,GPIO_Pin_8);                               //BEEP 对应引脚拉低
```

3.9 KEY

3.9.1 工作原理

我们可以通过查询或中断方式来使用实验板上的 4 个按键，分别为 KEY0、KEY1、KEY2、KEY_UP。本课程设计我们将使用查询方式使用按键。

3.9.2 硬件电路连接

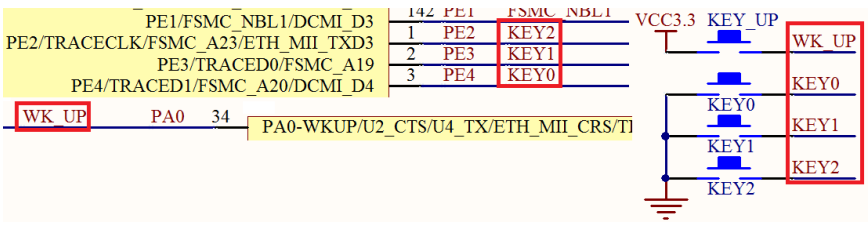


图 3.9-1 KEY 与 STM32F4 连接图

3.9.3 库函数配置方法

1°使能时钟

```
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA|RCC_AHB1Periph_GPIOE, ENABLE);
//使能 GPIOA,GPIOE 时钟
```

2°GPIO 初始化

```
GPIO_Init(GPIOE, &GPIO_InitStructure);                    //初始化 GPIOE2,3,4
```

3.10 I2S 和 WM8978

3.10.1 工作原理

（1）I2S 模块

I2S（Inter IC Sound）总线, 又称集成电路内置音频总线，是飞利浦公司为数字音频设备之间的音频数据传输而制定的一种总线标准，该总线专责于音频设备之间的数据传输，广泛应用于各种多媒体系统。它采用了沿独立的导线传输时钟与数据信号的设计，通过将数据和时钟信号分离，避免了因时差诱发的失真，为用户节省了购买抵抗音频抖动的专业设备的费用。

（2）WM8978 模块

WM8978 是欧胜（Wolfson）推出的一款全功能音频处理器。它带有一个 HI-FI 级数字信号处理内核，支持增强 3D 硬件环绕音效，以及 5 频段的硬件均衡器，可以有效改善音质；并有一个可编程的陷波滤波器，用以去除屏幕开、切换等噪音。

WM8978 的控制通过 I2S 接口（即数字音频接口）同 MCU 进行音频数据传输（支持音频接收和发送），通过两线（MODE=0，即 IIC 接口）或三线（MODE=1）接口进行配置。WM8978 的 I2S 接口，由 4 个引脚组成：

① ADCDAT：ADC 数据输出；

② DACDAT：DAC 数据输入；

③ LRC：数据左/右对齐时钟；

④ BCLK：位时钟，用于同步。

3.10.2 硬件电路连接

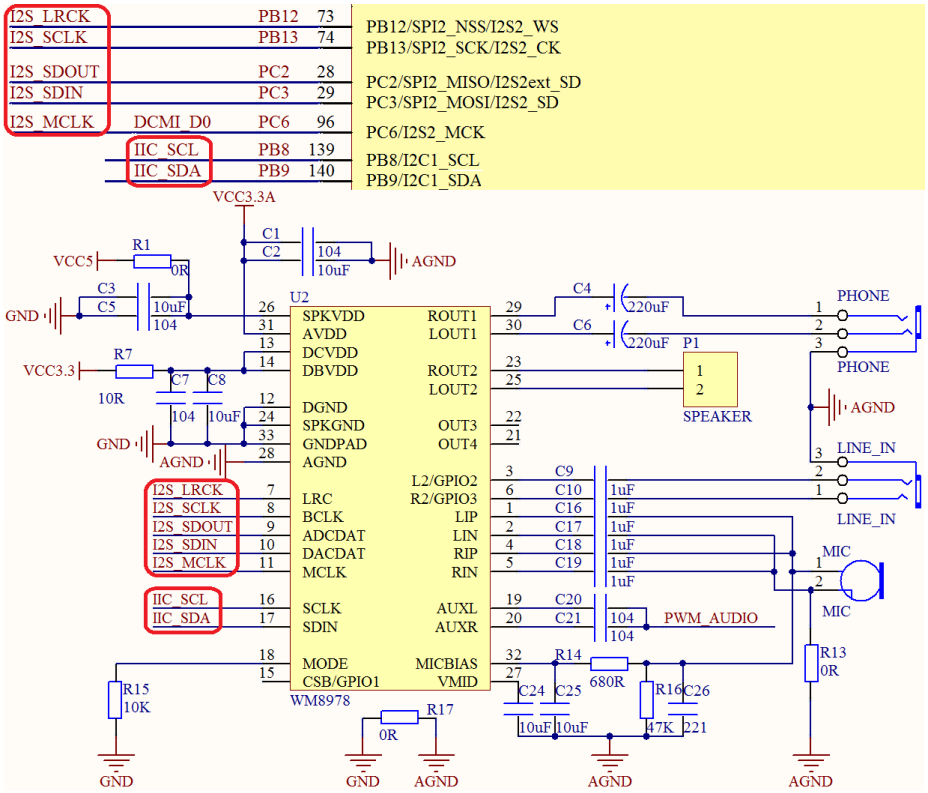


图 3.10-1 WM8978 和 STM32F4 连接图

3.10.3 库函数配置方法

1° 使能时钟

```
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB|RCC_AHB1Periph_GPIOC, ENABLE);
//使能外设 GPIOB,GPIOC 时钟
```

2° PB12/13 复用功能输出、PC2/PC3/PC6 复用功能输出

```
GPIO_Init(GPIOB, &GPIO_InitStructure);                    //初始化
GPIO_Init(GPIOC, &GPIO_InitStructure);                    //初始化
GPIO_PinAFConfig( );                                      //复用
```

3° 初始化 IIC

```
IIC_Init();                                               //初始化 IIC 接口
```

4° 软复位

```
WM8978_Write_Reg(0,0);                                    //软复位 WM8978
```

5° WM8978 通用设置

```
WM8978_Write_Reg();                                       //WM8978 配置
```

6° 音量设置

```
WM8978_HPvol_Set(40,40);                                  //耳机音量设置
WM8978_SPKvol_Set(50);                                    //喇叭音量设置
```

3.11 SPI 和 W25Q128（FLASH）

3.11.1 工作原理

（1）SPI 模块

SPI（Serial Peripheral interface）即串行外围设备接口。是 Motorola 首先在其 MC68HCXX 系列处理器上定义的。SPI 是一种高速、全双工、同步串行通信总线，并且在芯片的管脚上只占用四根线，节约了芯片的管脚，同时为 PCB 的布局上节省空间，提供方便。

SPI 接口一般使用 4 条线通信：

1、MISO 主设备数据输入，从设备数据输出。

2、MOSI 主设备数据输出，从设备数据输入。

3、SCLK 时钟信号，由主设备产生。

4、CS 从设备片选信号，由主设备控制。

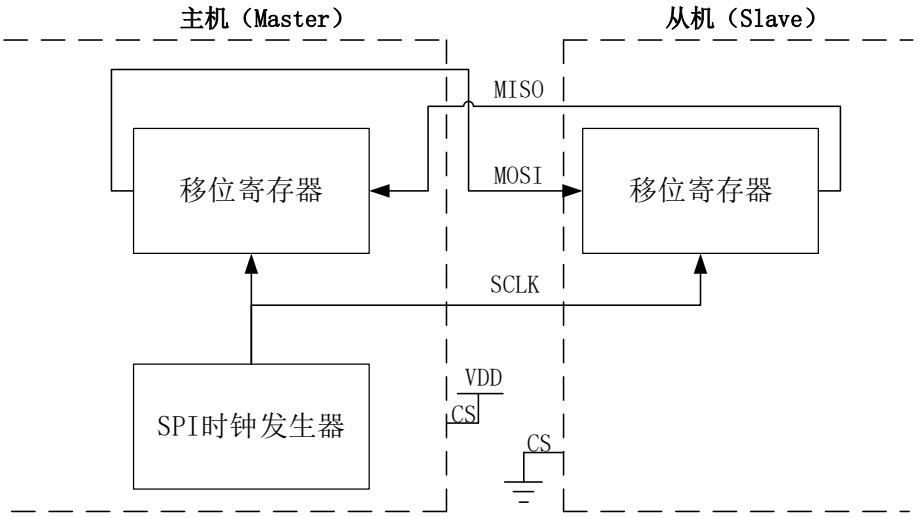


图 3.11-1 SPI 内部结构图

（2）W25Q128 模块

在本课程设计中，我们将使用 SPI 来读取外部 SPI FLASH 芯片 W25Q128。

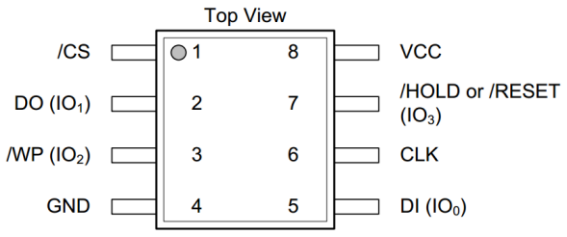


图 3.11-2 W25Q128 芯片引脚图

FLASH W25QXX 的页写是可以跨页的，与擦除指令不一样（整页擦除），页写入指令的地址并不要求按 256 字节对齐，只要确认目标存储单元是擦除状态即可（即被擦除后没有被写入过）。所以，若对“地址 x”执行页写入指令后，发送了 200 个字节数据后终止通讯，下一次再执行页写入指令，从“地址(x+200)”开始写入 200 个字节也是没有问题的（小于 256 均可）。只是在实际应用中由于基本擦除单元是 4KB，一般都以扇区为单位进行读写。

3.11.2 硬件电路连接

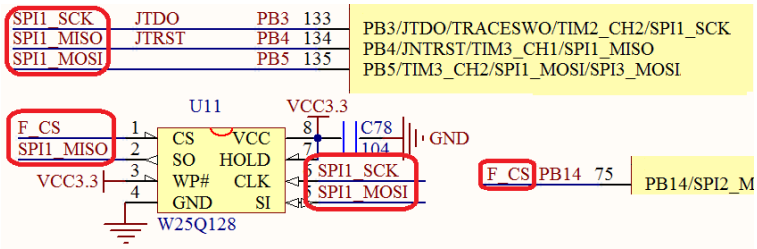


图 3.11-3 FLASH 和 STM32F4 连接图

3.11.3 库函数配置方法

（1）SPI 模块配置的步骤

1° 使能 SPIx 和 IO 口时钟

```
RCC_AHBxPeriphClockCmd();                                    //使能 GPIOA 时钟
RCC_APBxPeriphClockCmd();                                    //使能 SPI1 时钟
```

2° 初始化 IO 口为复用功能

```
void GPIO_Init();                                            //GPIOF9,F10 初始化设置
```

3° 设置引脚复用映射：

```
GPIO_PinAFConfig();                                          //引脚复用
```

4° 初始化 SPIx，设置 SPIx 工作模式

```
void SPI_Init(SPI_TypeDef* SPIx, SPI_InitTypeDef* SPI_InitStruct);//SPI 初始化
```

5° SPI 口初始化

```
RCC_APB2PeriphResetCmd(RCC_APB2Periph_SPI1,ENABLE);  //复位 SPI1
RCC_APB2PeriphResetCmd(RCC_APB2Periph_SPI1,DISABLE); //停止复位 SPI1
```

6° 使能 SPIx

```
void SPI_Cmd(SPI_TypeDef* SPIx, FunctionalState NewState);        //SPI 使能
```

7° SPI 传输数据

```
void SPI_I2S_SendData(SPI_TypeDef* SPIx, uint16_t Data);     //发送数据
uint16_t SPI_I2S_ReceiveData(SPI_TypeDef* SPIx) ;                 //接收数据
```

（2）W25Q128 芯片模块配置步骤

1°使能时钟；

```
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB, ENABLE);      //使能 GPIOB 时钟
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOG, ENABLE);      //使能 GPIOG 时钟
```

2°GPIO 初始化；

```
GPIO_Init(GPIOB, &GPIO_InitStructure);                     //初始化
```

3°防止 WIRELESS 影响我们芯片数据的读写；

```
GPIO_SetBits(GPIOG,GPIO_Pin_7);          //PG7 输出 1,防止 NRF 干扰 SPI FLASH 的通信
```

4°初始化 SPI；

```
W25QXX_CS=1;                             //SPI FLASH 不选中
SPI1_Init();                             //初始化 SPI
SPI1_SetSpeed(SPI_BaudRatePrescaler_2);  //设置为 42M 时钟,高速模式
W25QXX_TYPE=W25QXX_ReadID();             //读取 FLASH ID.
```

3.12 I2C 和 AT24C02（EEPROM）

3.12.1 工作原理

（1）IIC 模块

IIC（Inter－Integrated Circuit）总线是一种由 PHILIPS 公司开发的两线式串行总线，用于连接微控制器及其外围设备。它是由数据线 SDA 和时钟 SCL 构成的串行总线，可发送和接收数据。在 CPU 与被控 IC 之间、IC 与 IC 之间进行双向传送，高速 IIC 总线一般可达 400kbps 以上。

I2C 总线在传送数据过程中共有三种类型信号：

1、开始信号：SCL 为高电平时，SDA 由高电平向低电平跳变，开始传送数据。

2、结束信号：SCL 为高电平时，SDA 由低电平向高电平跳变，结束传送数据。

3、应答信号：接收数据的 IC 在接收到 8bit 数据后，向发送数据的 IC 发出特定的低电平脉冲，表示已收到数据。CPU 向受控单元发出一个信号后，等待受控单元发出一个应答信号，CPU 接收到应答信号后，根据实际情况作出是否继续传递信号的判断。若未收到应答信号，由判断为受控单元出现故障。

（2）AT24C02 模块

实验所用开发板板载 EEPROM 芯片型号为 24C02。该芯片的总容量是 256 个字节，该芯片通过 IIC 总线与外部连接。

I2C 的 AC24CXX 是不可以跨页写入的，地址是要对齐 8 能整除的。

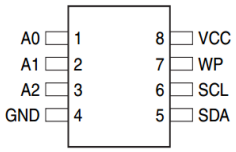


图 3.12-1 AT24C02 芯片引脚图

3.12.2 硬件电路连接

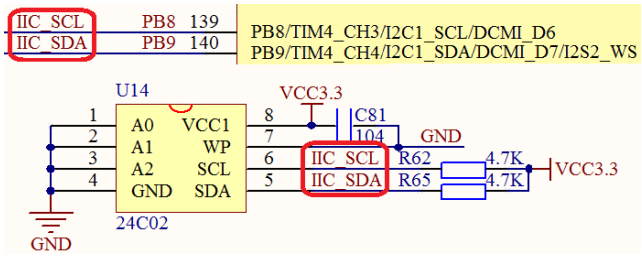


图 3.12-2 EEPROM 与 STM32F4 连接图

3.12.3 库函数配置方法

1° 使能时钟

```
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB, ENABLE);          //使能 GPIOB 时钟
```

2° GPIOB8、B9 初始化设置

```
GPIO_Init(GPIOB, &GPIO_InitStructure);                          //初始化
```

3° 空闲状态

```
IIC_SCL=1;                                                      //空闲状态
IIC_SDA=1;
```

3.13 MALLOC

3.13.1 工作原理

本次实验我们使用分块式内存管理。

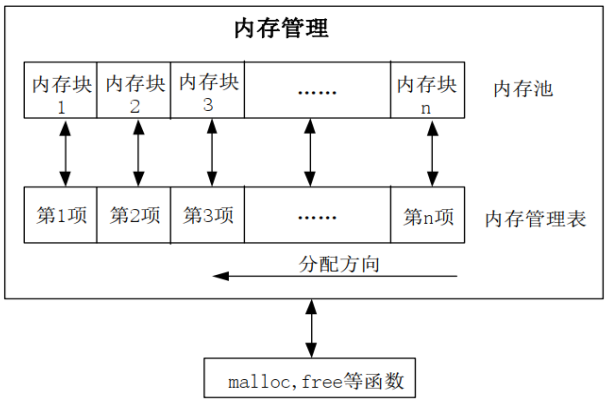


图 3.13-1 分块式内存管理原理

从上图可以看出，分块式内存管理由内存池和内存管理表两部分组成。内存池被等分为 n 块，对应的内存管理表，大小也为 n，内存管理表的每一个项对应内存池的一块内存。

内存管理表的项值代表的意义为：当该项值为 0 的时候，代表对应的内存块未被占用，当该项值非零的时候，代表该项对应的内存块已经被占用，其数值则代表被连续占用的内存块数。比如某项值为 10，那么说明包括本项对应的内存块在内，总共分配了 10 个内存块给外部的某个指针。

内寸分配方向如图所示，是从顶→底的分配方向。即首先从最末端开始找空内存。当内存管理刚初始化的时候，内存表全部清零，表示没有任何内存块被占用。

（1）分配原理

当指针 p 调用 malloc 申请内存的时候，先判断 p 要分配的内存块数（m），然后从第 n 项开始，向下查找，直到找到 m 块连续的空内存块（即对应内存管理表项为 0），然后将这 m 个内存管理表项的值都设置为 m（标记被占用），最后，把最后的这个空内存块的地址返回指针 p，完成一次分配。注意，如果当内存不够的时候（找到最后也没找到连续的 m 块空闲内存），则返回 NULL 给 p，表示分配失败。

（2）释放原理

当 p 申请的内存用完，需要释放的时候，调用 free 函数实现。free 函数先判断 p 指向的内存地址所对应的内存块，然后找到对应的内存管理表项目，得到 p 所占用的内存块数目 m（内存管理表项目的值就是所分配内存块的数目），将这 m 个内存管理表项目的值都清零，标记释放，完成一次内存释放。

3.13.2 库函数配置方法

MALLOC 内存初始化

```
my_mem_init(SRAMIN);                    //初始化内部内存池
my_mem_init(SRAMEX);                    //初始化外部内存池
my_mem_init(SRAMCCM);                   //初始化 CCM 内存池
```

3.14 FATFS 和 SRAM

3.14.1 工作原理

FATFS 是一个完全免费开源的 FAT 文件系统模块，专门为小型的嵌入式系统而设计。它完全用标准 C 语言编写，所以具有良好的硬件平台独立性。它支持 FATl2、FATl6

和 FAT32，支持多个存储媒介；有独立的缓冲区，可以对多个文件进行读/写，并特别对 8 位单片机和 16 位单片机做了优化。

FATFS 模块的层次结构如下图所示：

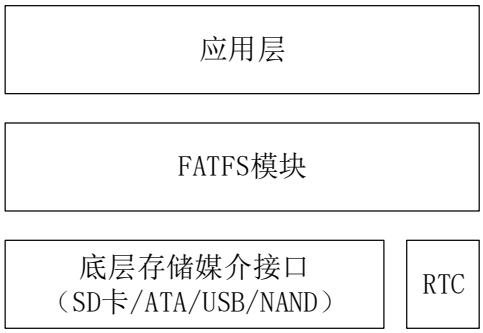


图 3.14-1 FATFS 层次结构图

3.14.2 库函数配置方法

（1）FATFS 申请内存

```
u8  exfuns_init(void);                          //为 exfuns 申请内存
```

（2）初始化外部 SRAM

1°使能时钟

```
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB|RCC_AHB1Periph_GPIOD|RCC_AHB1Periph
_GPIOE|RCC_AHB1Periph_GPIOF|RCC_AHB1Periph_GPIOG, ENABLE);
//使能 PD,PE,PF,PG 时钟
RCC_AHB3PeriphClockCmd(RCC_AHB3Periph_FSMC,ENABLE);          //使能 FSMC 时钟
```

2°GPIO 初始化

```
GPIO_Init();                                    //初始化
GPIO_PinAFConfig();                             //复用
```

3°初始化 FSMC

```
FSMC_NORSRAMInit(&FSMC_NORSRAMInitStructure);               //初始化 FSMC 配置
FSMC_NORSRAMCmd(FSMC_Bank1_NORSRAM3, ENABLE);       //使能 BANK3
```

四、 具体实现过程描述

此模块将分模块描述每个模块的具体配置流程，带注释的程序代码。

4.1 用到的 GUI 结构体函数

GUI 结构体是综合实验已封装好的，非自定义编写，在这里直接调用。

4.1.1 窗体 window

1、结构体

```
//window 结构体定义
__packed typedef struct
{
    u16 top;                        //window 顶端坐标
    u16 left;                       //window 左端坐标
    u16 width;                      //window 宽度(包含滚动条的宽度)
    u16 height;                     //window 高度

    u8 id;                          //window
    u8 type;                        //window 类型
                                    //[7]:0,没有关闭按钮.1,有关闭按钮
                                    //[6]:0,不读取背景色.1,读取背景色.
                                    //[5]:0,标题靠左.1,标题居中.
                                    //[4:2]:保留
                                    //[1:0]:0,标准的窗口(仿 XP);1,圆边窗口(仿 Andriod)


    u8 sta;                         //window 状态
                                    //[7:0]:保留

    u8 *caption;                    //window 名字
    u8 captionheight;               //caption 栏的高度
    u8 font;                        //window 文字字体
    u8 arcwinr;                     //圆角窗口的圆角的半径

    u16 captionbkcu;                //caption 的上半部分背景色
    u16 captionbkcd;                //caption 的下半部分背景色
    u16 captioncolor;               //caption 的颜色
    u16 windowbkc;                  //window 的背景色

    u16 *bkctbl;                    //背景色表(需要读取窗体背景色的时候用到)
    _btn_obj* closebtn;             //串口关闭按钮
}_window_obj;
```

2、函数

```
_window_obj * window_creat(u16 left,u16 top,u16 width,u16 height,u8 id,u8 type,u8 font);
                                                //创建一个 window 实体
void window_delete(_window_obj * window_del);  //删除一个 window 实体
void window_draw(_window_obj * windowx);        //画出已创建的 windowx
```

4.1.2 按钮 btn

1、结构体

```
//按钮结构体定义
__packed typedef struct {
    u16 top;                                //按钮顶端坐标
    u16 left;                               //按钮左端坐标
    u16 width;                              //宽度
    u16 height;                             //高度
    u8 id;                                  //按钮 ID

    u8 type;                                //按钮类型
    //[7]:      0,模式 A,按下是一种状态,松开是一种状态.
    //          1,模式 B,每按下一次,状态改变一次.按一下按下,再按一下弹起.
    //[6:4]:保留
    //[3:0]:0,标准按钮;1,图片按钮;2,边角按钮;3,文字按钮(背景透明),4,文字按钮(背景单一)

    u8 sta;                                 //按钮状态
    //[7]:坐标状态  0,松开.1,按下.(并不是实际的 TP 状态)
    //[6]:0,此次按键无效;1,此次按键有效.(根据实际的 TP 状态决定)
    //[5:2]:保留
    //[1:0]:0,激活的(松开);1,按下;2,未被激活的

    u8 *caption;                            //按钮名字
    u8 font;                                //caption 文字字体
    u8 arcbtnr;                             //圆角按钮时圆角的半径
    u16 bcfucolor;                          //button caption font up color
    u16 bcfdcolor;                          //button caption font down color

    u16 *bkctbl;                            //对于文字按钮:
                                            //背景色表(按钮为文字按钮的时候使用)
                                            //a,当为文字按钮(背景透明时),用于存储背景色
                                            //b,当为文字按钮(背景单一是),
                                                //bkctbl[0]:存放松开时的背景色;
                                                //bkctbl[1]:存放按下时的背景色.
                                            //对于边角按钮:
                                            //bkctbl[0]:圆角按钮边框的颜色
                                            //bkctbl[1]:圆角按钮第一行的颜色
                                            //bkctbl[2]:圆角按钮上半部分的颜色
                                            //bkctbl[3]:圆角按钮下半部分的颜色

    u8 *picbtnpathu;                        //图片按钮松开时的图片路径
    u8 *picbtnpathd;                        //图片按钮按下时的图片路径
}_btn_obj;
```

2、函数

```
_btn_obj * btn_creat(u16 left,u16 top,u16 width,u16 height,u8 id,u8 type);//创建一个 btn 实体
vvoid btn_delete(_btn_obj * btn_del);                                    //删除一个 btn 实体
void btn_draw(_btn_obj * btnx);                                          //画出已创建的 btnx
u8 btn_check(_btn_obj * btnx,void * in_key);                             //btnx 是否被按下检测
```

4.1.3 列表 listbox

1、结构体

```
//listbox 结构体定义
__packed typedef struct {
    u16 top;                                //listbox 顶端坐标
    u16 left;                               //listbox 左端坐标
    u16 width;                              //宽度
    u16 height;                             //高度 必须为 12/16 的倍数

    u8 type;                                //类型标记字
                                            //[bit7]:1,需要画滚动条出来(条件是 totalitems>itemsperpage);0,不需要画出来.(此位由软件自动控制)
                                            //[bit6:0]:保留

    u8 sta;                                 //listbox 状态,[bit7]:滑动标志;[bit6]:编号有效的标志;[bit5:0]:第一次按下的编号.
    u8 id;                                  //listbox 的 id
    u8 dbclick;                             //双击,
                                            //[7]:0,没有双击.1,有双击.
                                            //[6~0]:0,保留.

    u8 font;                                //文字字体 12/16
    u16 selindex;                           //选中的索引
    u16 lbkcolor;                           //内部背景颜色
    u16 lnselcolor;                         //list name 选中后的颜色
    u16 lnselbkcolor;                       //list name 选中后的背景颜色
    u16 lncolor;                            //list name 未选中的颜色
    u16 rimcolor;                           //边框颜色
//////////////////////
    u8 *fname;                              //当前选中的 index 的名字
    u16 namelen;                            //name 所占的点数.
    u16 curnamepos;                         //当前的偏移
    u32 oldtime;                            //上一次更新时间

    _scrollbar_obj * scbv;                  //垂直滚动条
    _listbox_list *list;                    //链表
}_listbox_obj;
```

2、函数

```
_listbox_obj * listbox_creat(u16 left,u16 top,u16 width,u16 height,u8 type,u8 font); //创建 listbox
void listbox_delete(_listbox_obj *listbox_del);                                      //删除 listbox
u8 listbox_addlist(_listbox_obj * listbox,u8 *name);                                 //增加一个 list
u8 listbox_check(_listbox_obj * listbox,void * in_key);                              //检查 listbox 的按下状态
void listbox_draw_listbox(_listbox_obj *listbox);                                    //重画 listbox
app_filebrower(caption,0X07);                                                        //显示标题
```

4.1.4 GUI 图示

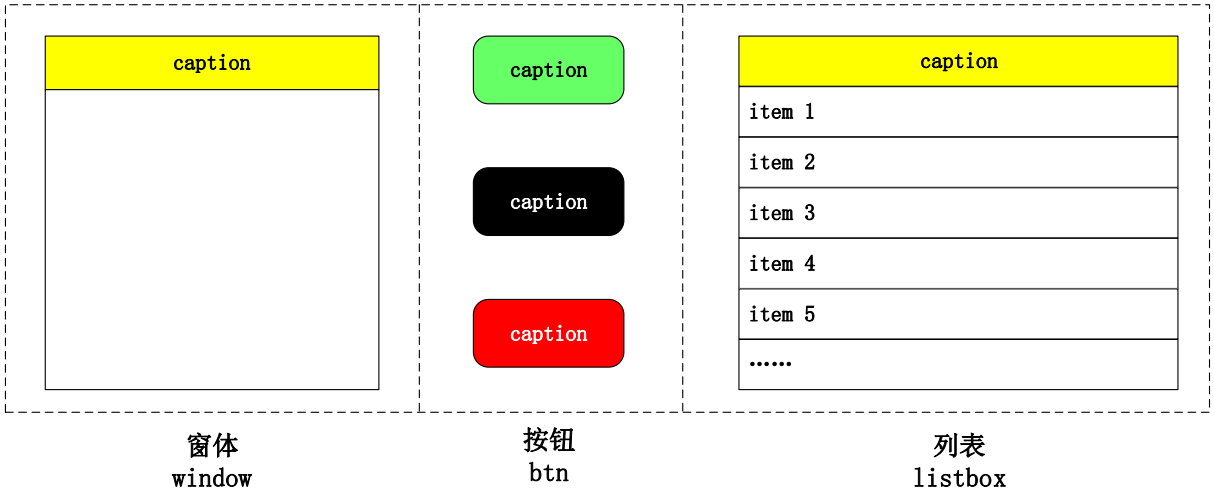


图 4.1-1 GUI 图示

4.2 提醒窗体 gui_notice

定义 gui_notice()函数用来显示各种提醒，包括出错提醒、无闹钟提醒、闹钟已满提醒、删除闹钟提醒等。另有出错处理函数 myerror_notice()调用 gui_notice 通过参数 rval 对应显示出错提示，这里对这个出错处理函数不作分析。

GUI 界面设计图示如下：

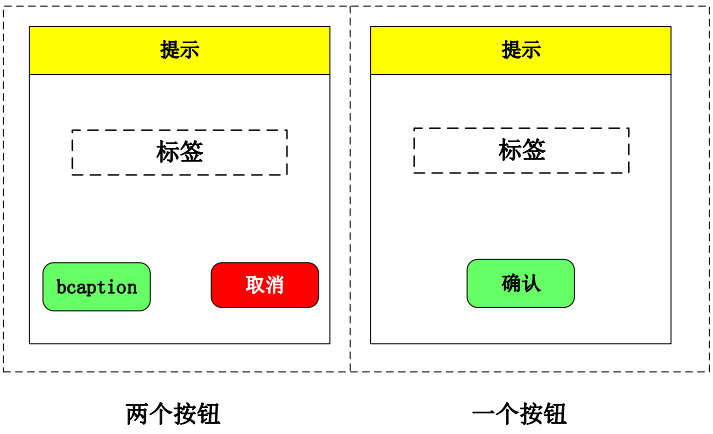


图 4.2-1 gui_notice 函数 GUI 界面设计

1、函数声明如下：

```
/*****************************************************************
 * 函数名称：gui_notice
 * 输入：      x,y:窗口坐标(窗口尺寸已经固定了的)
             mode：  0：两个按钮，添加/取消
                     1：一个按钮，确认
             caption:提示条
             bcaption：按钮名
 * 输出：u8 rval
 * 功能：提示框
 *****************************************************************/
u8 gui_notice(u16 x,u16 y,u8 mode,u8 *caption,u8 *bcaption);
```

其中，x、y 是提醒窗体的左上方点坐标。mode 为可选模式：0：两个按钮，未命名按钮/取消；1：一个按钮，确认。caption 是标签被定义需要显示的内容即提示信息，bcaption 是为未命名按钮取的按钮名。

2、函数体内部定义的变量如下：

```
    u8 rval=0,res;
    u8 flag=0;
    _window_obj *twin=0; //窗体
    _btn_obj *Abtn;          //未命令的按钮
    _btn_obj *Cbtn;          //取消按钮
    _btn_obj *Obtn;          //确认按钮
    _btn_obj *label;         //标签按钮

    twin=window_creat(x,y,300,400,0,1|1<<5,24);                                    //
创建窗口

    label=btn_creat(x+25+30,y+84+82,SYSSET_EDIT_WIDTH*2,SYSSET_EDIT_HEIGHT,0,0x04);
    //创建提示条

    /*窗体*/
    if(twin==NULL)rval=ERROR_GUI;
    else{
        twin->caption="提示";
        twin->captionheight=60;
        twin->captionbkcu=YELLOW;
        twin->captionbkcd=YELLOW;
        twin->captioncolor=BLACK;
        twin->windowbkc=WHITE;
    }

    if(mode==0){//两个按钮
```

```
        Abtn=btn_creat(x+25,y+84+234,SYSSET_BTN2_WIDTH,SYSSET_BTN2_HEIGHT,0,0x02);
        //创建按钮
        Cbtn=btn_creat(x+25+125,y+84+234,SYSSET_BTN2_WIDTH,SYSSET_BTN2_HEIGHT,0,0x
02); //创建按钮

        if(Abtn==NULL||Cbtn==NULL)rval=ERROR_GUI;

        Abtn->font=24;
        Abtn->bcfucolor=BLACK;          //松开时为黑色
        Abtn->bcfdcolor=BLACK;          //按下时为黑色
        Abtn->bkctbl[0]=WHITE;          //边框颜色
        Abtn->bkctbl[1]=GRAY;           //第一行的颜色
        Abtn->bkctbl[2]=GREEN;          //上半部分颜色
        Abtn->bkctbl[3]=GREEN;          //下半部分颜色
        Abtn->caption=bcaption;

        Cbtn->font=24;
        Cbtn->bcfucolor=WHITE;          //松开时为白色
        Cbtn->bcfdcolor=WHITE;          //按下时为白色
        Cbtn->bkctbl[0]=WHITE;          //边框颜色
        Cbtn->bkctbl[1]=GRAY;       //第一行的颜色
        Cbtn->bkctbl[2]=RED;        //上半部分颜色
        Cbtn->bkctbl[3]=RED;        //下半部分颜色
        Cbtn->caption="取消";
    }
    else{//一个按钮
        Obtn=btn_creat(x+25+SYSSET_BTN2_WIDTH/2+1,y+84+234,SYSSET_BTN2_WIDTH,SYS
SET_BTN2_HEIGHT,0,0x02);          //创建按钮

        if(Obtn==NULL)rval=ERROR_GUI;

        Obtn->font=24;
        Obtn->bcfucolor=WHITE;          //松开时为白色
        Obtn->bcfdcolor=WHITE;          //按下时为白色
        Obtn->bkctbl[0]=WHITE;          //边框颜色
        Obtn->bkctbl[1]=GRAY;           //第一行的颜色
        Obtn->bkctbl[2]=RED;        //上半部分颜色
        Obtn->bkctbl[3]=RED;        //下半部分颜色
        Obtn->caption=bcaption;
    }

    /*显示提示信息*/
    label->caption=caption;
    label->font=24;
```

```
if(rval!=ERROR_GUI){
    window_draw(twin);                    //画窗体

    /*画按钮*/
    if(mode==0){
        btn_draw(Abtn);
        btn_draw(Cbtn);
    }
    else{
        btn_draw(Obtn);
    }

    btn_draw(label);                      //画标签
```

其中，rval 是每个实现 GUI 界面函数的 u8 类型返回值，对于它每一位的定义如下：

7	6	5	4	3	2	1	0
双击按下	取消	关闭闹钟	保留	MALLOC 分配错误	ITEM 条目出错	DOUBLE 双击出错	GUI 创建失败

当 rval 作用于按钮时，按下“取消”表示将 rval 第 6 位置 1，否则当 rval 为 0X00 时表示此时作用于非“取消”按钮（例如“确认”、“添加”、“修改”、“删除”、“关闭”等。基于这样的考虑是因为，一个是 rval 的本质是为了能根据它来判断接下来应该显示哪个界面，而了解到对于一个按钮操作后可能造成的结果基本上可以归类为两种：要么停留在本页面，要么切换掉本页面，因此只需考虑 0X40 和 0X00 两个值来做“停留”和“切换”的标志；另一个是，“确认“、”添加”、“修改”在实际操作过程中意义相同，而对于“删除”和“关闭”的判断，另针对闹钟设了是否开启的标志位（state）。

rval 的后 4 位用于出错提示，位 4 保留，位 5 表示关闭闹钟，位 7 表示双击了闹钟列表的某个条目。res 用于判断相应按钮是否被按下。

flag 为跳出 while 循环的标志。

之后介绍的每个函数中包含的 rval、res 和 flag 都是同样的设置。

_window_obj 和_btn_obj 是 GUI 框架中已构建好的实体类型，前者是创建一个带标题栏的窗体，后者是创建一个按钮，这个按钮可以是图片按钮、文字按钮、圆角按钮等。

调用 window_creat()函数创建一个窗体实体，并对窗体的一些属性比如标题栏颜色、标题字体大小、标题栏大小等进行设置；调用 btn_creat()函数创建一个 label 标签，用于显示提示信息。之后根据 mode 变量定义的不同模式，决定应该是只创建一个“确认”按钮，还是创建两个按钮（其中一个固定为“取消”），并对按钮的属性进行一系列设置。假如窗体和按钮变量有返回空值的，置 rval 为 0X01，提示 GUI 出错信息。

3、函数主体定义如下：

```
        while(flag==0){
            tp_dev.scan(0);                                 //触摸屏
            in_obj.get_key(&tp_dev,IN_TYPE_TOUCH);    //得到按键键值
            if(mode==0){
                /*取消按钮检测*/
                res=btn_check(Cbtn,&in_obj);
                if(res){
                    if(((Cbtn->sta&0X80)==0)){//有有效操作
                        rval=HOLD_CLOCK_SETTING;
                        flag=1;
                        break;     //退出
                    }
                }
                /*bcaption 按钮检测*/
                res=btn_check(Abtn,&in_obj);
                if(res){
                    if((Abtn->sta&0X80)==0){//有有效操作
                        rval=SWITCH_CLOCK_SETTING;
                        flag=1;
                        break;
                    }
                }
            }//if(mode==0)
            else{
                /*确认按钮检测*/
                res=btn_check(Obtn,&in_obj);
                if(res){
                    if(((Obtn->sta&0X80)==0)){
                        rval=SWITCH_CLOCK_SETTING;
                        flag=1;
                        break;
                    }
                }
            }
        }//while(rval==0)
    }//if(rval==0)
    ////////////////////////////////////////////////////////////////
    window_delete(twin);                    //删除窗口
    /*删除按钮*/
    if(mode==0){
        btn_delete(Abtn);
        btn_delete(Cbtn);
    }
```

```
    btn_delete(label);                          //删除提示条
    printf("gui_notice:%d\n",rval);
    return  rval;
```

设置一个循环体不断查询是否有手指触摸到按钮区域。假设按的是“取消”按钮（Cbtn），将 rval 置为 0X40（HOLD_CLOCK_SETTING），并将 flag 置为 1 退出循环；若按的是其他按钮，将 rval 置为 0X00（SWITCH_CLOCK_SETTING），并将 flag 置为 1 退出循环。最后，删除 GUI 实体释放内存空间，并返回 rval。流程图如下：

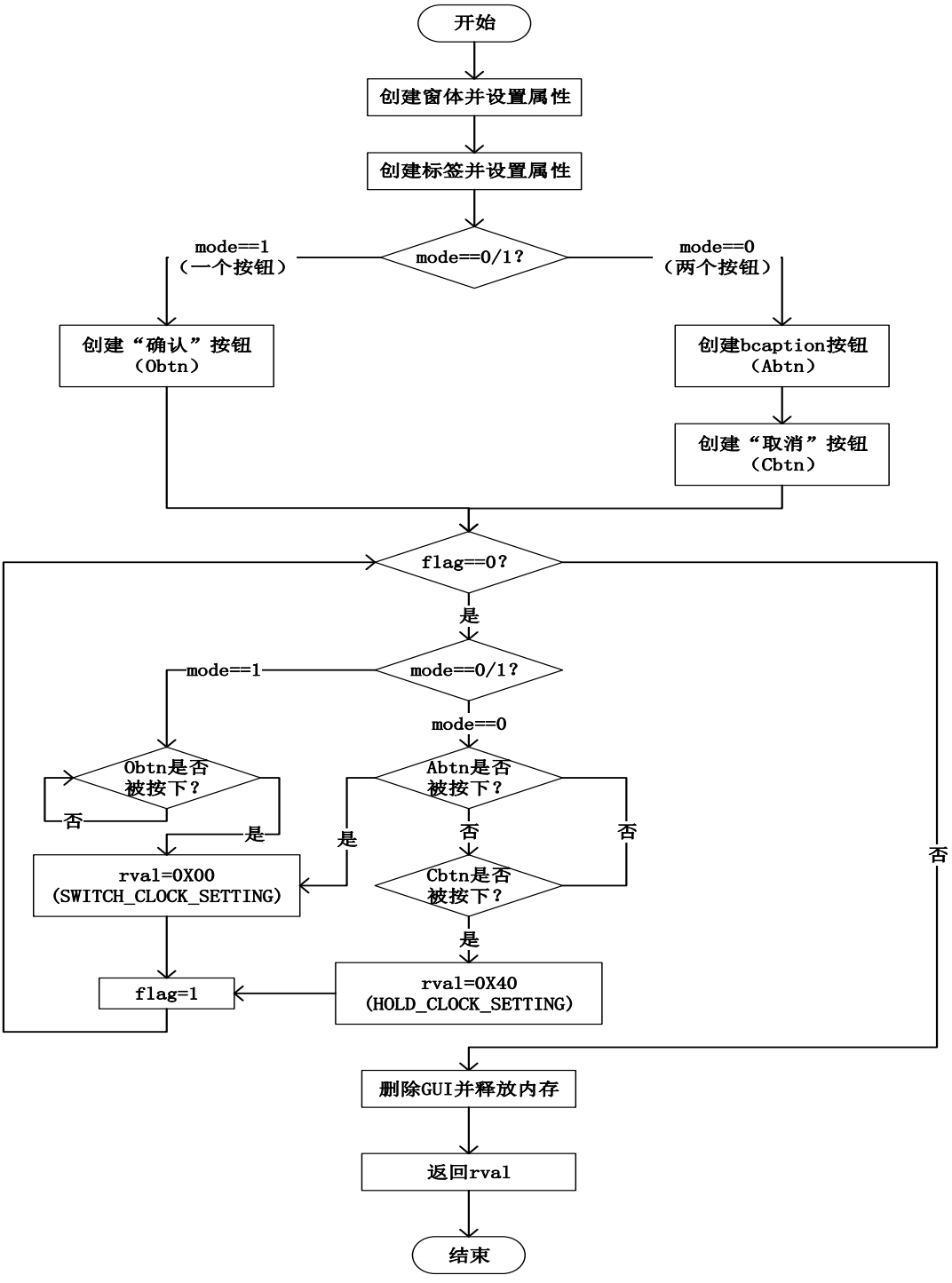


图 4.2-2 gui_notice 函数流程图

4.3 设置时间窗体 sysset_time_set

定义 sysset_time_set()函数用来显示并设置时钟的时和分。

GUI 界面设计图示如下：

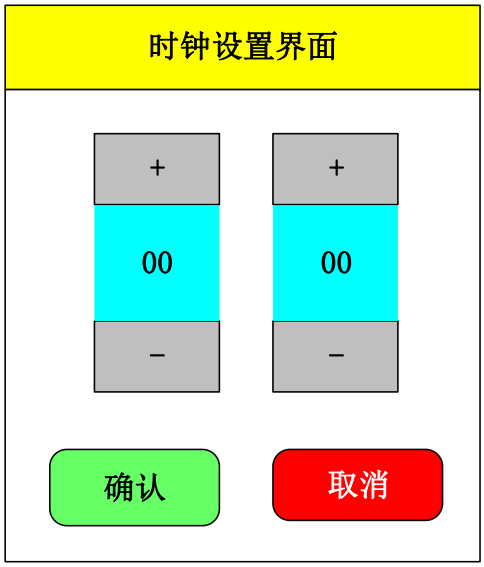


图 4.3-1 sysset_time_set()函数 GUI 界面设计

1、函数声明如下：

```
/*****************************************************************************
 * 函数名称：sysset_time_set
 * 输入：      x,y:窗口坐标(窗口尺寸已经固定了的)
               hour,min:时分
               caption:窗口名字
 * 输出：u8 rval
 * 功能：时间/闹钟设置
 *****************************************************************************/
u8 sysset_time_set(u16 x,u16 y,u8 *hour,u8 *min,u8*caption);
```

其中，x 和 y 指定整个 GUI 界面的位置（依旧是左上角），hour 和 min 分别是设置的时和分，caption 为 GUI 界面的名称。hour 和 min 设置成指针类型的变量是为了返回通过 GUI 界面改变的相应的值。

2、函数体内部变量定义如下：

```
    u8 rval=HOLD_CLOCK_SETTING;
    u8 res;
    u8 i;
    u8 flag=0;
```

其中，rval、res 和 flag 和 4.1 小节中所述相同。i 是 for 循环中使用的临时变量。

以下为对各 GUI 实体的定义与属性设置。

```
_window_obj*  twin=0; //窗体
_btn_obj  *  tbtn[6];
//总共六个按钮:0,时钟加按钮;1,时钟减按钮;2,分钟加按钮;3,分钟减按钮;4,确认按钮;5,取消按钮
twin=window_creat(x,y,300,400,0,1|1<<5,24);                                    //创建窗口
tbtn[0]=btn_creat(x+35,y+84,SYSSET_BTN1_WIDTH,SYSSET_BTN1_HEIGHT,0,0x02);
//创建按钮
tbtn[1]=btn_creat(x+35,y+84+134,SYSSET_BTN1_WIDTH,SYSSET_BTN1_HEIGHT,0,0x02);
//创建按钮
tbtn[2]=btn_creat(x+35+130,y+84,SYSSET_BTN1_WIDTH,SYSSET_BTN1_HEIGHT,0,0x02);
//创建按钮
tbtn[3]=btn_creat(x+35+130,y+84+134,SYSSET_BTN1_WIDTH,SYSSET_BTN1_HEIGHT,0,0x02);
//创建按钮
tbtn[4]=btn_creat(x+25,y+84+234,SYSSET_BTN2_WIDTH,SYSSET_BTN2_HEIGHT,0,0x02);
//创建按钮
tbtn[5]=btn_creat(x+25+125,y+84+234,SYSSET_BTN2_WIDTH,SYSSET_BTN2_HEIGHT,0,0x02);
//创建按钮
//////////////////////////////////////////////////////////////////
/*窗口设置*/
if(twin==NULL)rval=ERROR_GUI;          //出错
else{
     twin->caption=caption;                    //窗体标题
     twin->captionheight=60;                   //标题高度
     twin->captionbkcu=YELLOW;          //标题上界颜色
     twin->captionbkcd=YELLOW;          //标题下界颜色
     twin->captioncolor=BLACK;          //标题颜色
     twin->windowbkc=WHITE;             //窗体背景颜色
}
/*按钮设置*/
for(i=0;i<6;i++){
     if(tbtn[i]==NULL){                        //出错
          rval=ERROR_GUI;       break;
     }
     tbtn[i]->font=24;                    //按钮字体大小
     if(i<4){                             //加减按键
          tbtn[i]->bcfucolor=WHITE;  //松开时为白色
          tbtn[i]->bcfdcolor=WHITE;  //按下时为白色
          tbtn[i]->bkctbl[0]=WHITE;  //边框颜色
          tbtn[i]->bkctbl[1]=GRAY;    //第一行的颜色
          tbtn[i]->bkctbl[2]=GRAY;    //上半部分颜色
          tbtn[i]->bkctbl[3]=GRAY;    //下半部分颜色
     }else  if(i==4){                          //确认按键
          tbtn[i]->bcfucolor=BLACK;  //松开时为黑色
          tbtn[i]->bcfdcolor=BLACK;  //按下时为黑色
          tbtn[i]->bkctbl[0]=WHITE;  //边框颜色
          tbtn[i]->bkctbl[1]=GRAY;    //第一行的颜色
```

```
            tbtn[i]->bkctbl[2]=GREEN;  //上半部分颜色
            tbtn[i]->bkctbl[3]=GREEN;  //下半部分颜色
        }
        else{                          //取消按键
            tbtn[i]->bcfucolor=WHITE;  //松开时为白色
            tbtn[i]->bcfdcolor=WHITE;  //按下时为白色
            tbtn[i]->bkctbl[0]=WHITE;  //边框颜色
            tbtn[i]->bkctbl[1]=GRAY;   //第一行的颜色
            tbtn[i]->bkctbl[2]=RED;    //上半部分颜色
            tbtn[i]->bkctbl[3]=RED;    //下半部分颜色
        }
        /*按钮名称*/
        if(i==0||i==2)tbtn[i]->caption="+";
        if(i==1||i==3)tbtn[i]->caption="-";
        if(i==4)tbtn[i]->caption="确定";
        if(i==5)tbtn[i]->caption="取消";
    }
```

假设定义 GUI 实体没有出错，那么进入函数主体。

3、函数主体定义如下：

```
    if(rval!=ERROR_GUI){//无错误
        window_draw(twin);                      //画出窗体
        for(i=0;i<6;i++)btn_draw(tbtn[i]);      //画按钮

        /*画设置的时/分数字*/
        gui_fill_rectangle(x+35+1,y+84+82,SYSSET_EDIT_WIDTH-2,SYSSET_EDIT_HEIGHT,SYSSET_EDIT_BACK_COLOR);          //填充时钟背景
        gui_fill_rectangle(x+35+130+1,y+84+82,SYSSET_EDIT_WIDTH-2,SYSSET_EDIT_HEIGHT,SYSSET_EDIT_BACK_COLOR);          //填充分钟背景
        app_show_nummid(x+35,y+84+82,SYSSET_EDIT_WIDTH,SYSSET_EDIT_HEIGHT,*hour,2,24,BLACK,SYSSET_EDIT_BACK_COLOR);
    app_show_nummid(x+35+130,y+84+82,SYSSET_EDIT_WIDTH,SYSSET_EDIT_HEIGHT,*min,2,24,BLACK,SYSSET_EDIT_BACK_COLOR);

        while(flag==0){
            tp_dev.scan(0);                                 //触摸屏
            in_obj.get_key(&tp_dev,IN_TYPE_TOUCH);  //得到按键键值

            /*按钮扫描*/
            for(i=0;i<6;i++){
                res=btn_check(tbtn[i],&in_obj);         //确认按钮检测
                if(res){
                    if((tbtn[i]->sta&0X80)==0){       //有有效操作
                        switch(i){
                            case  0:                        //时钟增加按钮按下了
```

```
                    (*hour)++;
                    if(*hour>23)*hour=0;
                    break;
                case  1:                    //时钟减少按钮按下了
                    if(*hour)(*hour)--;
                    else  *hour=23;
                    break;
                case  2:                    //分钟增加按钮按下了
                    (*min)++;
                    if(*min>59)(*min)=0;
                    break;
                case  3:                    //分钟减少按钮按下了
                    if(*min)(*min)--;
                    else  *min=59;
                    break;
                case  4:                    //确认按钮按下
                    rval=SWITCH_CLOCK_SETTING;
                    flag=1;
                    break;
                case  5:                    //取消按钮按下
                    rval=HOLD_CLOCK_SETTING;
                    flag=1;
                    break;
            }//switch(i)
        }//if((tbtn[i]->sta&0X80)==0)
        /*显示调整的时/分*/
        app_show_nummid(x+35,y+84+82,SYSSET_EDIT_WIDTH,SYSSET_EDIT_HEIGHT,*hour,2,24,BLACK,SYSSET_EDIT_BACK_COLOR);
    app_show_nummid(x+35+130,y+84+82,SYSSET_EDIT_WIDTH,SYSSET_EDIT_HEIGHT,*min,2,24,BLACK,SYSSET_EDIT_BACK_COLOR);
            }//if(res)
        }//for(i=0;i<6;i++)
    }//while(flag==0)
}//if(rval!=ERROR_GUI)
////////////////////////////////////////////////////////////////////////////////////////////////////
window_delete(twin);                    //删除窗口
for(i=0;i<6;i++)btn_delete(tbtn[i]);    //删除按钮
printf("sysset_time_set:%d",rval);
return  rval;
```

其中，gui_fill_rectangle 和 app_show_nummid 都是为了显示所设置的时、分的数值和其背景。循环查询是否有按钮被按下：

（1）tbtn[0-1]：tbtn[0]增加“时”，tbtn[1]减少“时”，数值范围为 0~23；

（2）tbtn[2-3]：tbtn[2]增加“分”，tbtn[3]减少“分”，数值范围为 0~59；

（3）tbtn[4]：确认，返回 rval=0X00，并将标志 flag=1 退出循环；

（4）tbtn[5]：取消，返回 rval=0X40，并将标志 flag=1 退出循环。

当按下 tbtn[0-3]时，必须对设置的时分数值进行更新显示，因此再次调用函数 app_show_nummid 更新数值显示。

当按下“确认”或者“取消”按钮后，删除 GUI 实体并释放内存。流程图如下：

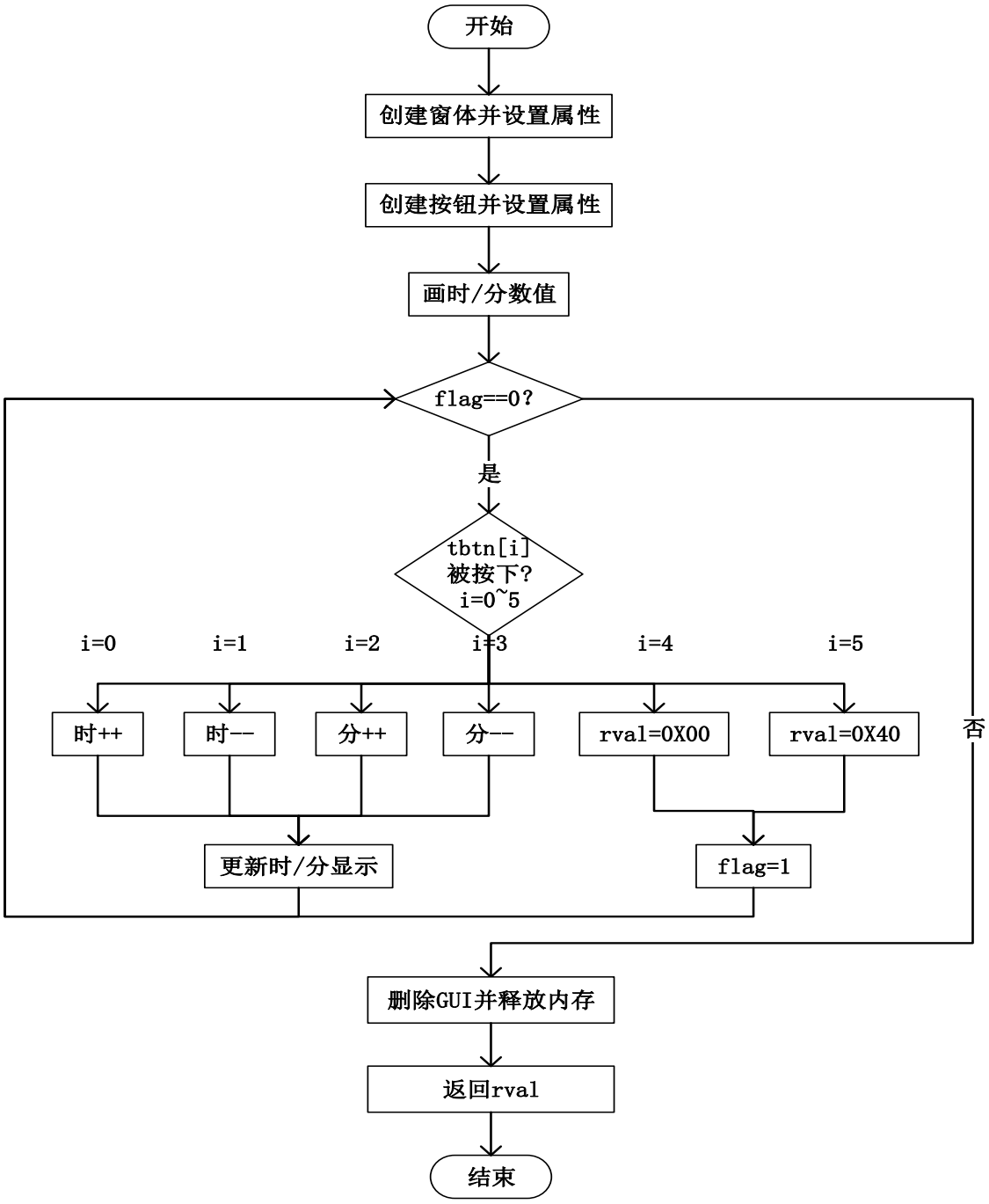


图 4.3-2 sysset_time_set 函数流程图

4.4 设置日历窗体 sysset_date_set

定义 sysset_date_set()函数用来显示并设置日历（包括年、月、日），基本上与函数 sysset_time_set 一致，只是一个设置时/分，一个设置年月日。

GUI 界面设计图示如下：

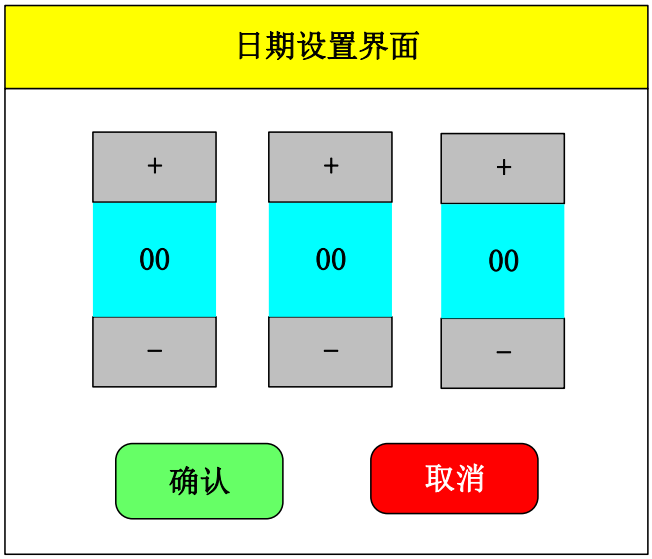


图 4.4-1 sysset_date_set()函数 GUI 界面设计

1、函数声明如下：

```
/*****************************************************************************
 * 函数名称：sysset_date_set
 * 输入：      x,y:窗口坐标(窗口尺寸已经固定了的)
               year,month,date:年月日
               caption:窗口名字
 * 输出：      u8  rval
 * 功能：日期设置
 *****************************************************************************/
u8  sysset_date_set(u16  x,u16  y,u16  *year,u8  *month,u8  *date,u8*caption);
```

其中，x 和 y 指定整个 GUI 界面的位置（依旧是左上角），year、month 和 date 分别用来设置年、月、日，caption 是整个 GUI 界面的标题。

2、函数体内部变量定义如下：

```
    u8  rval=HOLD_CLOCK_SETTING;
    u8  res;
    u8  i;
    u8  flag=0;
```

其中，rval、res 和 flag 和 4.1 小节中所述相同。i 是 for 循环中使用的临时变量。

以下为对各 GUI 实体的定义与属性设置。

```
/*窗体设置*/
if(twin==NULL)rval=ERROR_GUI;
else{
    twin->caption=caption;
    twin->captionheight=60;
    twin->captionbkcu=YELLOW;
    twin->captionbkcd=YELLOW;
    twin->captioncolor=BLACK;
    twin->windowbkc=WHITE;
}
/*按钮设置*/
for(i=0;i<8;i++)
{
    tbtn[i]->font=24;
    if(tbtn[i]==NULL)
    {
        rval=ERROR_GUI;
        break;
    }
    if(i<6)                 //加减按键
    {
        tbtn[i]->bcfucolor=WHITE;  //松开时为白色
        tbtn[i]->bcfdcolor=WHITE;  //按下时为白色
        tbtn[i]->bkctbl[0]=WHITE;  //边框颜色
        tbtn[i]->bkctbl[1]=GRAY;   //第一行的颜色
        tbtn[i]->bkctbl[2]=GRAY;   //上半部分颜色
        tbtn[i]->bkctbl[3]=GRAY;   //下半部分颜色
    }else  if(i==6) //确认按键
    {
        tbtn[i]->bcfucolor=BLACK;  //松开时为黑色
        tbtn[i]->bcfdcolor=BLACK;  //按下时为黑
        tbtn[i]->bkctbl[0]=WHITE;  //边框颜色
        tbtn[i]->bkctbl[1]=GRAY;   //第一行的颜色
        tbtn[i]->bkctbl[2]=GREEN;  //上半部分颜色
        tbtn[i]->bkctbl[3]=GREEN;  //下半部分颜色
    }
    else{                   //取消按键
        tbtn[i]->bcfucolor=WHITE;  //松开时为白
        tbtn[i]->bcfdcolor=WHITE;  //按下时为白色
        tbtn[i]->bkctbl[0]=WHITE;  //边框颜色
        tbtn[i]->bkctbl[1]=GRAY;   //第一行的颜色
        tbtn[i]->bkctbl[2]=RED;    //上半部分颜色
        tbtn[i]->bkctbl[3]=RED;    //下半部分颜色
    }
    if(i==0||i==2||i==4)tbtn[i]->caption="+";
```

```
        if(i==1||i==3||i==5)tbtn[i]->caption="-";
        if(i==6)tbtn[i]->caption="确定";
        if(i==7)tbtn[i]->caption="取消";
    }
```

假设定义 GUI 实体没有出错，那么进入函数主体。

3、函数主体定义如下：

```
    if(rval!=ERROR_GUI){//无错误
        window_draw(twin);                          //画出窗体
        for(i=0;i<8;i++)btn_draw(tbtn[i]); //画按钮

        /*日期数字*/
        gui_fill_rectangle(x+20+1,y+84+82,SYSSET_EDIT_WIDTH-2,SYSSET_EDIT_HEIGHT,SYSSET_EDIT_BACK_COLOR);          //填充年份背景
        gui_fill_rectangle(x+20+120+1,y+84+82,SYSSET_EDIT_WIDTH-2,SYSSET_EDIT_HEIGHT,SYSSET_EDIT_BACK_COLOR);    //填充月份背景
        gui_fill_rectangle(x+20+240+1,y+84+82,SYSSET_EDIT_WIDTH-2,SYSSET_EDIT_HEIGHT,SYSSET_EDIT_BACK_COLOR);    //填充日期背景

    app_show_nummid(x+20,y+84+82,SYSSET_EDIT_WIDTH,SYSSET_EDIT_HEIGHT,*year,4,24,BLACK,SYSSET_EDIT_BACK_COLOR);
    app_show_nummid(x+20+120,y+84+82,SYSSET_EDIT_WIDTH,SYSSET_EDIT_HEIGHT,*month,2,24,BLACK,SYSSET_EDIT_BACK_COLOR);
    app_show_nummid(x+20+240,y+84+82,SYSSET_EDIT_WIDTH,SYSSET_EDIT_HEIGHT,*date,2,24,BLACK,SYSSET_EDIT_BACK_COLOR);
        while(flag==0){
            tp_dev.scan(0);                                     //触摸屏检测
            in_obj.get_key(&tp_dev,IN_TYPE_TOUCH);    //得到按键键值

            /*按键*/
            for(i=0;i<8;i++){
                res=btn_check(tbtn[i],&in_obj);         //确认按钮检测
                if(res){
                    if((tbtn[i]->sta&0X80)==0){       //有有效操作
                        if(*month==2){
                            if(sysset_is_leap_year(*year))maxdate=29;//是闰年的 2 月份
                            else  maxdate=28;
                        }
                        else  if((*month==1)||(*month==3)||(*month==5)||(*month==7)||(*month==8)||(*month==10)||(*month==12))maxdate=31;
                        else  maxdate=30;
                        switch(i){
                            case  0://年份增加按钮按下了
                                (*year)++;
                                if(*year>2100)*year=2000;
```

```
                                    break;
                                case  1://年份减少按钮按下了
                                    if(*year>2000)(*year)--;
                                    else  *year=2100;
                                    break;
                                case  2://月份增加按钮按下了
                                    (*month)++;
                                    if(*month>12)(*month)=1;
                                    break;
                                case  3://月份减少按钮按下了
                                    if(*month>1)(*month)--;
                                    else  *month=12;
                                    break;
                                case  4://日期增加按钮按下了
                                    (*date)++;
                                    if(*date>maxdate)(*date)=1;
                                    break;
                                case  5://日期减少按钮按下了
                                    if(*date>1)(*date)--;
                                    else  *date=maxdate;
                                    break;
                                case  6://确认按钮按下
                                    rval=SWITCH_CLOCK_SETTING;
                                    flag=1;
                                    break;
                                case  7://取消按钮按下
                                    rval=HOLD_CLOCK_SETTING;
                                    flag=1;
                                    break;
                            }
                        }
    app_show_nummid(x+20,y+84+82,SYSSET_EDIT_WIDTH,SYSSET_EDIT_HEIGHT,*year,4,24,BLACK,SYSSET_EDIT_BACK_COLOR);
    app_show_nummid(x+20+120,y+84+82,SYSSET_EDIT_WIDTH,SYSSET_EDIT_HEIGHT,*month,2,24,BLACK,SYSSET_EDIT_BACK_COLOR);
    app_show_nummid(x+20+240,y+84+82,SYSSET_EDIT_WIDTH,SYSSET_EDIT_HEIGHT,*date,2,24,BLACK,SYSSET_EDIT_BACK_COLOR);
                    }//if(res)
                }//for(i=0;i<8;i++)
            }//while(rval==0)
    }
    ////////////////////////////////////////////////////
    window_delete(twin);                    //删除窗口
    for(i=0;i<8;i++)btn_delete(tbtn[i]);//删除按钮
    return  rval;
```

函数主体结构基本上与 sysset_time_set 一致，需要注意的是：

（1）“年”的数值范围为 2000~2100；

（2）“月”的数值范围为 1~12；

（3）“日”的数值范围需要根据“年”和“月”来判断：

① 根据是否是“闰年”决定“2 月”有几天（闰年 29 天，非闰年 28 天）；

② 根据是“大月”还是“小月”决定一个月有几天（大月 31 天，小月 30 天）。

判断是否是闰年的函数如下：

```
/**************************************************************************
 * 函数名称：sysset_is_leap_year
 * 输入：     year
 * 输出：     该年份是不是闰年.1,是.0,不是
 * 功能：判断是否是闰年函数
//月份    1  2  3  4  5  6  7  8  9  10 11 12
//闰年    31 29 31 30 31 30 31 31 30 31 30 31
//非闰年 31 28 31 30 31 30 31 31 30 31 30 31
 **************************************************************/
u8 sysset_is_leap_year(u16 year){
    if(year%4==0){                          //必须能被 4 整除
        if(year%100==0){
            if(year%400==0)return 1;        //如果以 00 结尾,还要能被 400 整除
            else return 0;
        }else return 1;
    }else return 0;
}
```

当按下“确认”或者“取消”按钮后，删除 GUI 实体并释放内存。流程图如下：

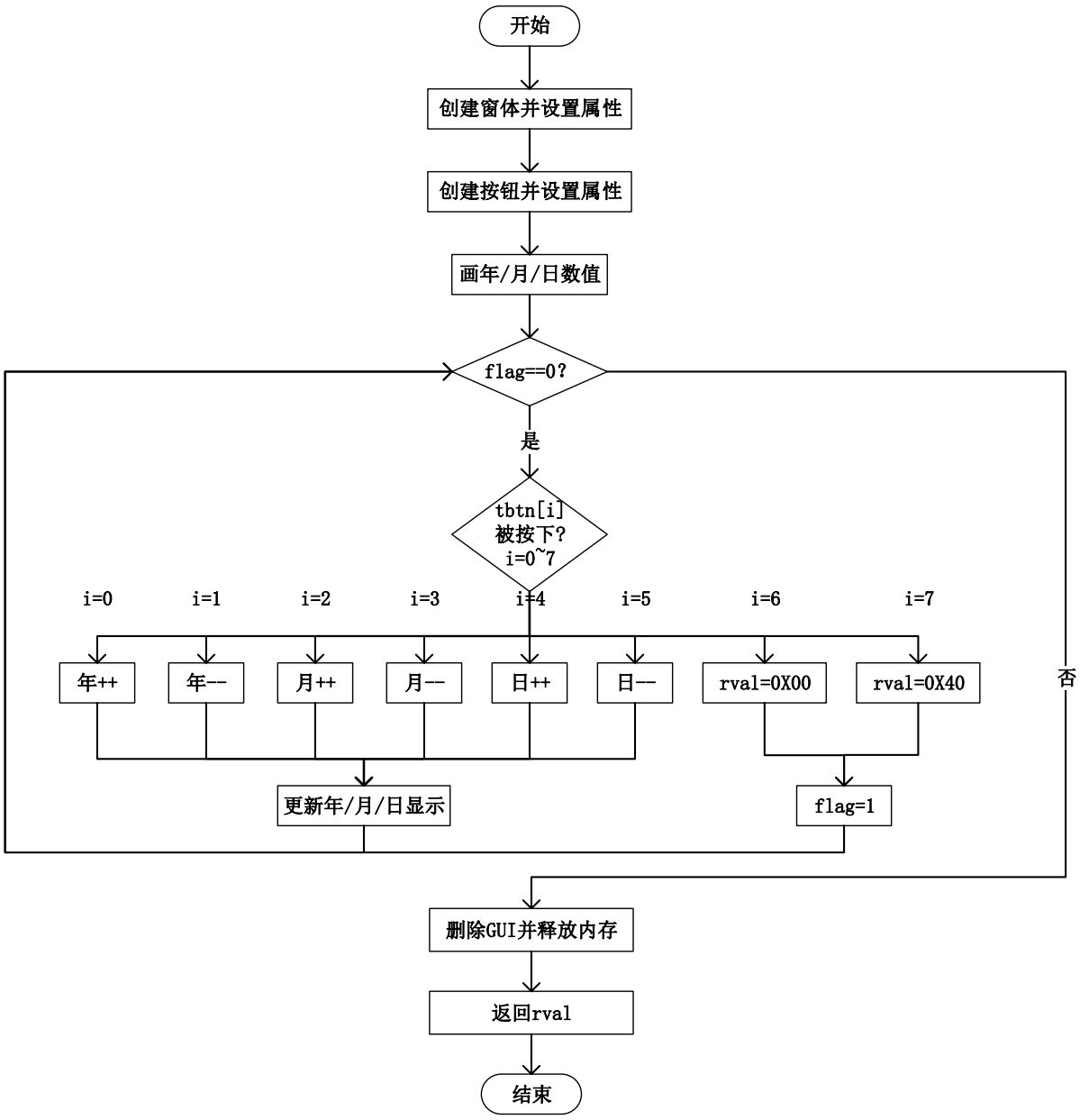


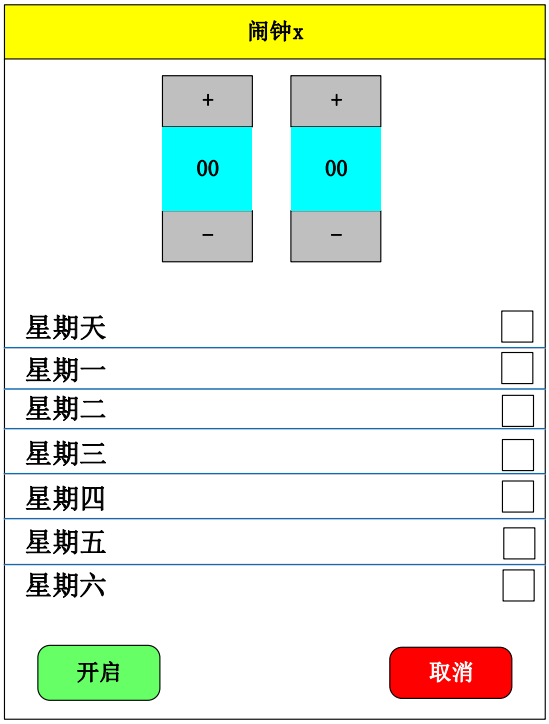
图 4.4-2 sysset_date_set 函数流程图

4.5 设置闹钟模块窗体

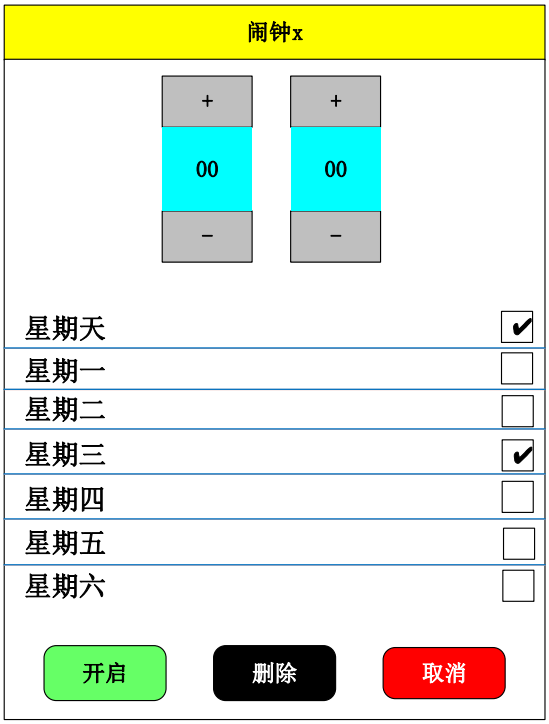
定义函数 alarm_play()用来显示已设置的闹钟列表界面，定义函数 app_items_sel()用来显示设置某个闹钟的界面。

4.5.1 app_items_sel()

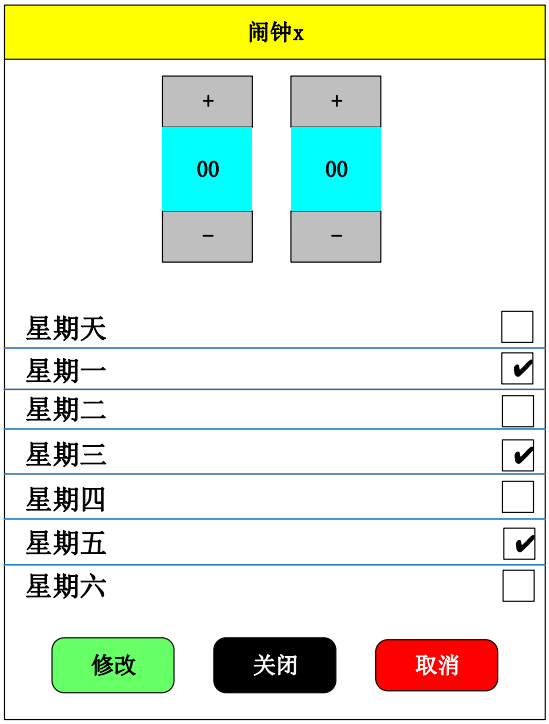
GUI 界面设计图示如下：



第一次添加闹钟x



关闭中的闹钟x



开启中的闹钟x

图 4.5-1 app_items_sel()函数 GUI 界面设计

1、函数声明如下：

```
/************************************
 * 函数名称：app_items_sel
 * 输入：      //x,y,width,height:坐标尺寸(width 最小为 150,height 最小为 72)
               //items[]:条目名字集
               //itemsize:总条目数(最大不超过 8 个)
               //selx:结果.多选模式时,对应各项的选择情况.单选模式时,对应选择的条目.
               //mode:
                    //[7]:0,无 OK 按钮;1,有 OK 按钮
                    //[6]:0,不读取背景色;1,读取背景色
                    //[5]:0,单选模式;1,多选模式
                    //[4]:0,不加载图标;1,加载图标
                    //[3:0]:保留
                    //caption:窗口名字
 * 输出：u8 rval
 * 功能：闹钟界面
 ************************************/
u8 app_items_sel(u16 x,u16 y,u16 width,u16 height,u8 *items[],u8 itemsize,u8 *hour,u8 *min,u8 *selx,u8 mode,u8*caption,u8 *is_open80);
```

2、函数体内部变量定义如下：

```
    u8 rval=0,res;
    u8 selsta=0;  //选中状态为 0,
                  //[7]:标记是否已经记录第一次按下的条目;
                  //[6:4]:保留
                  //[3:0]:第一次按下的条目

    u16 i;
    u8 temp;              //暂存 i

    u16 itemheight=0;     //每个条目的高度
    u16 itemwidth=0;      //每个条目的宽度
    u8* unselpath=0;      //未选中的图标的路径
    u8* selpath=0;            //选中图标的路径
    u8* icopath=0;
    u8 flag=0;
////////////////////////////////////////////////////////////////////////////////
    _window_obj* twin=0; //窗体
    _btn_obj * tbtn[7];       //时间设置的 6 个按钮
                              //tbtn[0-3]：加减
                              //tbtn[4]:开启/修改
                              //tbtn[5]:取消
                              //tbtn[6]:关闭
```

```
    if(itemsize>8||itemsize<1)return  ERROR_ITEM_SIZE;     //条目数错误
    if(width<150||height<72)return  ERROR_ITEM_SIZE;       //尺寸错误

    itemheight=(height-450)/itemsize-1;                    //得到每个条目的高度
    itemwidth=width;                                       //每个条目的宽度

    twin=window_creat(0,0,lcddev.width,lcddev.height,0,1|(1<<5)|((1<<6)&mode),24);        //创建窗口

    tbtn[0]=btn_creat(x+120,y+114,SYSSET_BTN1_WIDTH,SYSSET_BTN1_HEIGHT,0,0x02);
        //创建按钮
    tbtn[1]=btn_creat(x+120,y+114+134,SYSSET_BTN1_WIDTH,SYSSET_BTN1_HEIGHT,0,0x02);
        //创建按钮
    tbtn[2]=btn_creat(x+120+130,y+114,SYSSET_BTN1_WIDTH,SYSSET_BTN1_HEIGHT,0,0x02);
        //创建按钮
    tbtn[3]=btn_creat(x+120+130,y+114+134,SYSSET_BTN1_WIDTH,SYSSET_BTN1_HEIGHT,0,0x0
2);      //创建按钮

    /*窗体*/
    if(twin==NULL){
        spb_delete();                          //释放 SPB 占用的内存
        twin=window_creat(0,0,lcddev.width,lcddev.height,0,1|(1<<5)|((1<<6)&mode),24);//重新创建窗
口
        rval=ERROR_GUI;
    }
    /*按钮*/
    for(i=0;i<4;i++){
        if(tbtn[i]==NULL){
            rval=ERROR_GUI;
            break;
        }
        tbtn[i]->font=24;
        tbtn[i]->bcfucolor=WHITE;              //松开时为白色
        tbtn[i]->bcfdcolor=WHITE;              //按下时为白色
        tbtn[i]->bkctbl[0]=WHITE;              //边框颜色
        tbtn[i]->bkctbl[1]=GRAY;               //第一行的颜色
        tbtn[i]->bkctbl[2]=GRAY;               //上半部分颜色
        tbtn[i]->bkctbl[3]=GRAY;               //下半部分颜色

        if(i==0||i==2)tbtn[i]->caption="+";
        if(i==1||i==3)tbtn[i]->caption="-";
    }
    if(mode&(1<<7)){                           //有 OK 按钮
        tbtn[4]=btn_creat(0,lcddev.height-y-SYSSET_BTN2_HEIGHT,SYSSET_BTN2_WIDTH,SYSS
ET_BTN2_HEIGHT,0,0x02);                                    //创建开启/修改按钮
        tbtn[5]=btn_creat(lcddev.width-SYSSET_BTN2_WIDTH,lcddev.height-y-SYSSET_BTN2_HEI
```

```
GHT,SYSSET_BTN2_WIDTH,SYSSET_BTN2_HEIGHT,0,0x02);
                                    //创建取消按钮
        tbtn[6]=btn_creat((lcddev.width-SYSSET_BTN2_WIDTH)/2,lcddev.height-y-SYSSET_BTN2_
HEIGHT,SYSSET_BTN2_WIDTH,SYSSET_BTN2_HEIGHT,0,0x02);
                                    //创建关闭按钮
        if(tbtn[6]==NULL)rval=ERROR_GUI;
        else{
            if((*is_open80)==0X80)tbtn[6]->caption="关闭";
            else  tbtn[6]->caption="删除";
            tbtn[6]->font=24;
            tbtn[6]->bcfucolor=WHITE;  //松开时为白色
            tbtn[6]->bcfdcolor=WHITE;  //按下时为白色
            tbtn[6]->bkctbl[0]=WHITE;  //边框颜色
            tbtn[6]->bkctbl[1]=GRAY;    //第一行的颜色
            tbtn[6]->bkctbl[2]=BLACK;  //上半部分颜色
            tbtn[6]->bkctbl[3]=BLACK;  //下半部分颜色
        }
        if(twin==NULL||tbtn[4]==NULL)rval=ERROR_GUI;
        else{
            tbtn[4]->font=24;
            tbtn[4]->bcfucolor=BLACK;       //松开时为黑色
            tbtn[4]->bcfdcolor=BLACK;       //按下时为黑色
            tbtn[4]->bkctbl[0]=WHITE;       //边框颜色
            tbtn[4]->bkctbl[1]=GRAY;        //第一行的颜色
            tbtn[4]->bkctbl[2]=GREEN;       //上半部分颜色
            tbtn[4]->bkctbl[3]=GREEN;       //下半部分颜色
            if((*is_open80)==0X80)  tbtn[4]->caption="修改";
            else  tbtn[4]->caption="开启";

        }

    }else{                                          //没有 ok 按钮
        tbtn[5]=btn_creat(lcddev.width-SYSSET_BTN2_WIDTH,lcddev.height-y-SYSSET_BTN2_HEI
GHT,SYSSET_BTN2_WIDTH,SYSSET_BTN2_HEIGHT,0,0x02);  //创建取消按钮
        if(twin==NULL||tbtn[5]==NULL)rval=ERROR_GUI;
    }
    if(rval!=ERROR_GUI||rval!=ERROR_ITEM_SIZE){//之前的操作正常
        /*窗体*/
        twin->font=24;
        twin->caption=caption;
        twin->captionheight=60;
        twin->captionbkcu=YELLOW;
        twin->captionbkcd=YELLOW;
        twin->captioncolor=BLACK;
        twin->windowbkc=WHITE;
```

```
        /*按钮*/
        tbtn[5]->font=24;
        tbtn[5]->caption="取消";                //取消
        tbtn[5]->bcfucolor=WHITE;               //松开时为白色
        tbtn[5]->bcfdcolor=WHITE;               //按下时为白色
        tbtn[5]->bkctbl[0]=WHITE;               //边框颜色
        tbtn[5]->bkctbl[1]=GRAY;                //第一行的颜色
        tbtn[5]->bkctbl[2]=RED;                 //上半部分颜色
        tbtn[5]->bkctbl[3]=RED;                 //下半部分颜色

        if(mode&(1<<4)){                        //需要加载图标
            if(mode&(1<<5)){                    //多选模式
                unselpath=(u8*)APP_CANCEL_PIC;      //未选中的图标的路径
                selpath=(u8*)APP_OK_PIC;            //选中图标的路径
            }else{                              //单选模式
                unselpath=(u8*)APP_UNSELECT_PIC;    //未选中的图标的路径
                selpath=(u8*)APP_SELECT_PIC;        //选中图标的路径
            }
        }

        window_draw(twin);                      //画出窗体
        btn_draw(tbtn[5]);                      //画按钮
        if(mode&(1<<7)){
            if((*is_open80)!=0X02)
                btn_draw(tbtn[6]);
            btn_draw(tbtn[4]);
        }
        for(i=0;i<4;i++)btn_draw(tbtn[i]); //画按钮

        gui_fill_rectangle(x+120+1,y+114+82,SYSSET_EDIT_WIDTH-2,SYSSET_EDIT_HEIGHT,SYSSET_EDIT_BACK_COLOR);        //填充时钟背景
        gui_fill_rectangle(x+120+130+1,y+114+82,SYSSET_EDIT_WIDTH-2,SYSSET_EDIT_HEIGHT,SYSSET_EDIT_BACK_COLOR);        //填充分钟背景
    app_show_nummid(x+120,y+114+82,SYSSET_EDIT_WIDTH,SYSSET_EDIT_HEIGHT,*hour,2,24,BLACK,SYSSET_EDIT_BACK_COLOR);
    app_show_nummid(x+120+130,y+114+82,SYSSET_EDIT_WIDTH,SYSSET_EDIT_HEIGHT,*min,2,24,BLACK,SYSSET_EDIT_BACK_COLOR);
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
        /*星期*/
        for(i=0;i<itemsize;i++){
            icopath=app_get_icopath(mode&(1<<5),selpath,unselpath,*selx,i);
                    //得到图标路径
    app_show_items(x+5,y+375+i*(itemheight+1),itemwidth,itemheight,items[i],icopath,BLACK,twin->windowbkc);  //显示所有的条目
```

```
            if((i+1)!=itemsize)app_draw_smooth_line(x+5,y+375+(i+1)*(itemheight+1)-1,itemwidth,1,
0Xb1ffc4,0X1600b1);//画彩线
        }
```

此函数主要涉及到两大模块的设计，分别是设置闹钟的时/分，设置闹钟的星期掩码。

设置闹钟的时/分参考 sysset_time_set 函数，保留除窗体外的其他 GUI 实体，属性也保持一致；设置闹钟的星期掩码参考综合实验中闹钟星期掩码的设置，包括各星期条目的彩色分割线和勾选框的图片样式，修改了条目间隔距离。

其中，涉及到了 mode 和 is_open 变量，现作如下解释：

针对 mode 每一位的含义在函数声明前的注释中已给出，mode 涉及到两方面的影响：一个是是否显示“确认”按钮，实验中恒保留“确认”意义的按钮，因此 mode 最高位恒为 1；另一个是针对星期掩码模式的设置，实验中为“多选”、“加载图标”，因此 mode 的第 4、5 位为 1。

is_open 变量反映闹钟状态，其每一位的定义如下：

7	6	5	4	3	2	1	0
闹钟开启中	保留					闹钟是第一次添加	点击闹钟删除标志

当 is_open=0X00 时，代表闹钟关闭中；is_open=0X01 时，代表点击了闹钟的删除标志，但还没有确认删除；is_open=0X02 时，代表闹钟是第一次添加，不应该有“删除”按钮；is_open=0X80 时，代表闹钟被开启了。

我们规定：当闹钟处于关闭状态时，才可以对闹钟进行删除，即“关闭”和“删除”是相斥显示的。对于闹钟设置界面仅有的三个按钮，做如下按钮名称分配：

（1）闹钟将第一次添加，此时对于这个闹钟还没有真正保存它的设置，所以不存在“删除”按钮（无法删除一个还不存在的闹钟），此时显示的按钮有“开启”和“取消”；

（2）闹钟处于关闭状态，此时显示的按钮有“开启”、“删除”、“取消”；

（3）闹钟处于开启状态，此时显示的按钮有“修改”、“关闭”、“取消”。

3、函数主体定义如下：

```
while(flag==0){
        tp_dev.scan(0);
        in_obj.get_key(&tp_dev,IN_TYPE_TOUCH);    //得到按键键值

        for(i=0;i<7;i++){
            res=btn_check(tbtn[i],&in_obj);          //确认按钮检测
            if(res){
                if((tbtn[i]->sta&0X80)==0){        //有有效操作
```

```
                    switch(i){
                        case  0://时钟增加按钮按下了
                            (*hour)++;
                            if(*hour>23)*hour=0;
                            break;
                        case  1://时钟减少按钮按下了
                            if(*hour)(*hour)--;
                            else  *hour=23;
                            break;
                        case  2://分钟增加按钮按下了
                            (*min)++;
                            if(*min>59)(*min)=0;
                            break;
                        case  3://分钟减少按钮按下了
                            if(*min)(*min)--;
                            else  *min=59;
                            break;
                        case  4://确认按钮按下
                            if(mode&(1<<7)){
                                (*is_open80)=0X80;
                                rval=SWITCH_CLOCK_SETTING;
                                flag=1;
                            }
                            break;
                        case  5://取消按钮按下
                            rval=HOLD_CLOCK_SETTING;
                            flag=1;
                            break;
                        case  6:
                            if(mode&(1<<7)){
                                if((*is_open80)==0X80){
                                    (*is_open80)=0X00;//关闭
                                    rval=CLOSE_CLOCK;
                                }
                                else{
                                    (*is_open80)=0X01;//删除
                                    rval=SWITCH_CLOCK_SETTING;
                                }
                                flag=1;
                            }
                            break;
                    }
                }
    app_show_nummid(x+120,y+114+82,SYSSET_EDIT_WIDTH,SYSSET_EDIT_HEIGHT,*hour,2,24,
BLACK,SYSSET_EDIT_BACK_COLOR);
```

```
    app_show_nummid(x+120+130,y+114+82,SYSSET_EDIT_WIDTH,SYSSET_EDIT_HEIGHT,*min,
2,24,BLACK,SYSSET_EDIT_BACK_COLOR);
                }//if(res)
            }//for(i=0;i<6;i++)
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
            temp=0XFF;//标记量,如果为 0XFF,在松开的时候,说明是不在有效区域内的.如果非 0XF
F,则表示 TP 松开的时候,是在有效区域内.
            for(i=0;i<itemsize;i++){
                if(tp_dev.sta&TP_PRES_DOWN){//触摸屏被按下
                    if(app_tp_is_in_area(&tp_dev,x+5,y+375+i*(itemheight+1),itemwidth,itemheigh
t)){//判断某个时刻,触摸屏的值是不是在某个区域内
                        if((selsta&0X80)==0){//还没有按下过
                            icopath=app_get_icopath(mode&(1<<5),selpath,unselpath,*selx,i);  //得
到图标路径                            app_show_items(x+5,y+375+i*(itemheight+1),itemwidth,itemheight,ite
ms[i],icopath,BLACK,APP_ITEM_SEL_BKCOLOR);//反选条目
                            selsta=i;        //记录第一次按下的条目
                            selsta|=0X80;  //标记已经按下过了
                        }
                        break;
                    }
                }else{  //触摸屏被松开了
                    if(app_tp_is_in_area(&tp_dev,x+5,y+375+i*(itemheight+1),itemwidth,itemheight)){//
判断某个时刻,触摸屏的值是不是在某个区域内
                        temp=i;
                        break;
                    }
                }
            }
            if((selsta&0X80)&&(tp_dev.sta&TP_PRES_DOWN)==0){//有按下过,且按键松开了
                if((selsta&0X0F)==temp){//松开之前的坐标也是在按下时的区域内.
                    if(mode&(1<<5)){//多选模式,执行取反操作
                        if((*selx)&(1<<temp))*selx&=~(1<<temp);
                        else  *selx|=1<<temp;
                    }else{//单选模式
    app_show_items(x+5,y+375+(*selx)*(itemheight+1),itemwidth,itemheight,items[*selx],unselpath,BL
ACK,twin->windowbkc);//取消之前选择的条目
                        *selx=temp;
                    }
                }else  temp=selsta&0X0F;//得到当时按下的条目号
                icopath=app_get_icopath(mode&(1<<5),selpath,unselpath,*selx,temp);  //得到图标路
径
    app_show_items(x+5,y+375+temp*(itemheight+1),itemwidth,itemheight,items[temp],icopath,BLAC
K,twin->windowbkc);//反选条目
                selsta=0;//取消
            }
```

```
        }
```

“//////”分割线后的部分为选择星期掩码部分的实现，为直接参考这里不做详细解释，着重解释分割线前的部分。

依然是循环扫描每个按钮的状态：

（1）tbtn[0-1]：按下 tbtn[0]即对闹钟的时进行加，按下 tbtn[1]即对闹钟的时进行减；

（2）tbtn[2-3]：按下 tbtn[2]即对闹钟的分进行加，按下 tbtn[3]即对闹钟的分进行减；

（3）tbtn[4]：无论是“开启”还是“修改”，都将把更新闹钟的时/分值，置 is_open 为 0X80 表示闹钟启动了，置 rval 为 0X00 表示“确认”，置 flag=1 退出循环；

（4）tbtn[5]：置 rval 为 0X40 表示“取消”，置 flag=1 退出循环；

（5）tbtn[6]：

① 若闹钟开启着（is_open==0X80），显示“关闭”，当按下该按钮时，置 is_open 为 0X00 表示闹钟关闭，置 rval 为 0X20（CLOSE_CLOCK）表示闹钟关闭，置 flag=1 退出循环；

② 若闹钟关闭着（is_open==0X00），显示“删除”，当按下该按钮时，置 is_open 为 0X01 表示闹钟待删除，置 rval 为 0X00 表示“确认”，置 flag=1 退出循环。

退出循环后，删除 GUI 实体并释放内存。流程图如下：

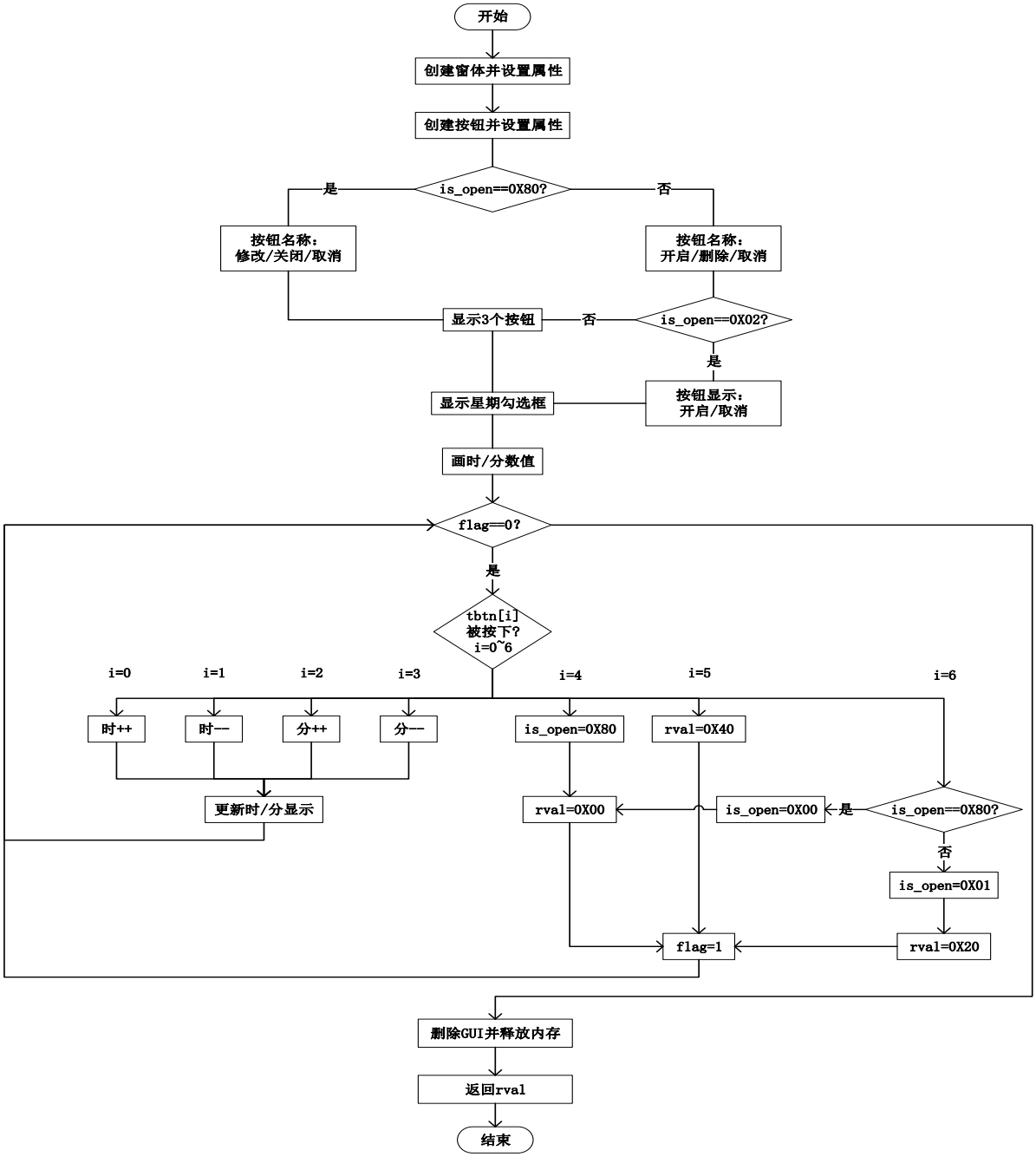


图 4.5-2 app_items_sel()函数流程图

4.5.2 alarm_play()

GUI 界面设计图示如下：

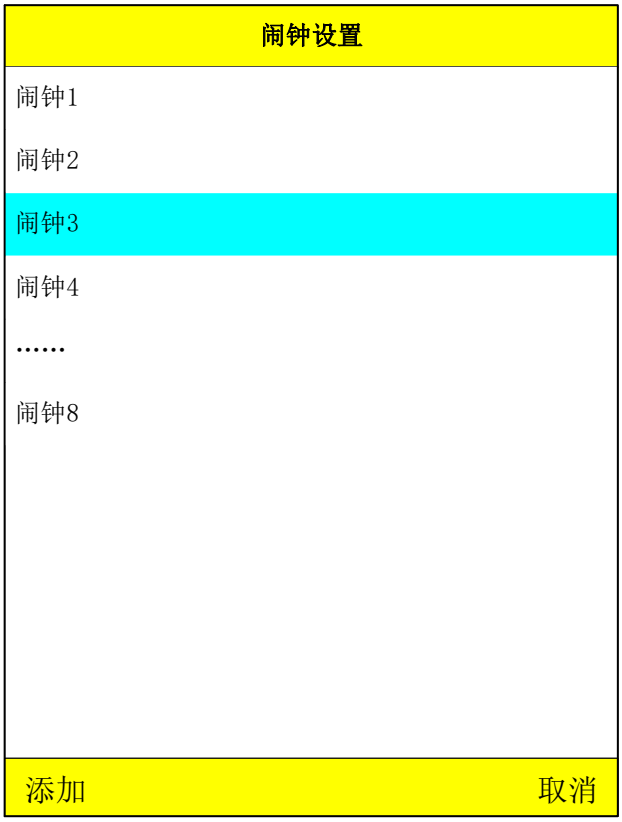


图 4.5-3 alarm_play()函数 GUI 界面设计

1、函数声明如下：

```
/*********************************************************************
 * 函数名称：alarm_play
 * 输入：count：当前闹钟数
 * 输出：u8 rval
 * 功能：显示闹钟列表界面
 *********************************************************************/
u8 alarm_play(u8 count);
```

其中，count 为当前已添加的闹钟总数，只有当 count>0 即至少有一个闹钟已添加的前提下，才会调用 alarm_play()函数。

2、函数体内部变量定义如下：

```
    u8 temp1,temp2,temp3,is_open;
    //temp1:闹钟的时
    //temp2:闹钟的分
    //temp3:闹钟的星期掩码
    //is_open:闹钟状态标志位
    u8 rval=0,res;
    u8 rval_temp;
```

```
u8  flag=0;
u8  flag_temp=0;
////////////////////////////////////////////////////////////
u8  selitem=0;              //selitem:当前选中的条目
u8  topitem=0;              //topitem:当前最顶部的条目
u8  **items;                //items[]:条目名字集
////////////////////////////////////////////////////////////
items=(u8**)alarm_tbl[gui_phy.language];//获得闹钟名
```

其中，rval、res 和 flag 含义依旧如 4.1 小节中所言。添加 rval_temp 和 flag_temp 用于 while 循环内部的 while 循环，rval_temp 的作用本质上和 rval 相同，flag_temp 的作用本质上和 flag 相同。

selitem、topitem 为列表（listbox）GUI 实体所用，items 为列表中每一条目的名称。

3、函数主体定义如下：

```
while(flag==0){
    res=app_listbox_select(&selitem,&topitem,"闹钟设置",items,count);
    if(res&SWITCH_CLOCK_SETTING){           //假如点了”添加“
        rval=SWITCH_CLOCK_SETTING;
        flag=1;
    }
    else  if(res&HOLD_CLOCK_SETTING){  //”假如点了“取消”
        rval=HOLD_CLOCK_SETTING;
        flag=1;
    }
    else  if(res&0X80){                     //双击第 selitem 个闹钟
        flag_temp=0;
        temp1=Alarm[selitem].hour;
        temp2=Alarm[selitem].min;
        temp3=Alarm[selitem].weekmask;
        is_open=Alarm[selitem].state;      //判断闹钟开不开
        if(is_open==0X50)temp3=0;       //当是单次闹钟时，掩码显示为空
        while(flag_temp==0){
rval=app_items_sel(0,0,lcddev.width,lcddev.height,(u8**)calendar_week_table[gui_phy.language],7,
(u8*)&temp1,(u8*)&temp2,(u8*)&temp3,0XB0,items[selitem],(u8  *)&is_open);//多选
            if(rval==SWITCH_CLOCK_SETTING){   //假如点了“开启/修改/删除”
                if(is_open==0X01){      //删除
                    rval_temp=gui_notice((lcddev.width-300)/2,(lcddev.height-400)/2,0,"确认删
除该闹钟?","确认");
                    if(rval_temp==SWITCH_CLOCK_SETTING)  {//确定删除
                        alarm_delete(selitem);          //删除当前的闹钟
                        selitem=0;
                        count=SYSTEM_Alarm_NUM;
                        //rval=SWITCH_CLOCK_SETTING;        //回到闹钟栏界面
```

```
                        flag_temp=1;
                    }
                    else{is_open==0X00;}        //关闭闹钟
                }
                else{                       //开启/修改
                    if(alarm_equal(selitem,temp1,temp2)){//有相同设置的闹钟
                        rval=gui_notice((lcddev.width-300)/2,(lcddev.height-400)/2,1,"相同闹
钟已存在!","确认");
                    }
                    else{
                        temp3=temp3<<1;
                        alarm_Set(selitem,temp1,temp2,is_open,temp3,0);//保存闹钟配置
                        flag_temp=1;
                    }
                }
            }//if(rval==SWITCH_CLOCK_SETTING)
            else  if(rval==HOLD_CLOCK_SETTING){
                flag_temp=1;
            }
        }//while(flag_temp==0)
    }//else  if(res&0X80)
    else  flag=1;                           //出错
}//while
printf("alarm_play:%d\n",rval);
return  rval;
```

其中，定义 while(flag==0)是为了保持闹钟列表 GUI 界面。在闹钟列表界面，可以执行三种操作：

（1）点击“添加”按钮，这时返回值 rval 置为 0X00，置标志位 flag 为 1 退出循环；

（2）点击“取消”按钮，这时返回值 rval 置为 0X40，置标志位 flag 为 1 退出循环；

（3）双击“闹钟 x”条目（x 为已添加的闹钟中某个闹钟的编号），这时返回值 rval 的最高位置为 1（即 rval&0X80==1），进入“闹钟 x”的设置界面。现对双击后设置的 while()循环做出如下解释。

设置 while(flag_temp)循环是为了解决这样一个问题：在开启中的“闹钟 x”设置界面点击“关闭”按钮后，应跳转到关闭的“闹钟 x“设置界面，并且对于跳转到的这个界面中的按钮可以进行对应的操作并返回正确的值。根据 while(flag_temp)中函数 app_items_sel()的返回值做出如下动作：

（1）返回值为 0X00。

① is_open==0X01：说明点了“删除”按钮，此时需要调用 gui_notice()弹出“删

除闹钟“提示框，并将提示框函数的返回值送入 rval_temp 中。假如 rval_temp==0X00，说明确认删除闹钟，需要调用 alarm_delete()函数对双击选中的闹钟进行删除，并修改 count 值为当前剩余闹钟数，将 selitem 置 0 初始化，置 flag_temp 为 1 跳出 while(flag_temp)循环，返回闹钟列表显示界面。假如 rval_temp≠0X00，说明取消删除，依然保持在闹钟 x 的设置界面。

② is_open≠0X01：说明点了“开启”或“修改”，此时需要考虑设置的时间是否与其他已设置闹钟的时间重复，如果重复了需要给予提示，提示后仍然保持在闹钟 x 的设置界面。假如没有时间没有重复，调用 alarm_Set()对闹钟进行设置。将星期掩码左移一位（temp3=temp3<<1）是因为，后端的掩码设置为高 7 位从低到高分别对应星期一~星期日，而在设置界面中，temp3 是低 7 位设置掩码。保存闹钟配置后置 flag_temp 为 1 跳出 while(flag_temp)循环。

（2）返回值为 0X40。说明在闹钟 x 的设置界面点了“取消”按钮，此时需要返回到闹钟列表，即置 flag_temp=1 跳出 while(flag_temp)循环。

函数流程图如下：

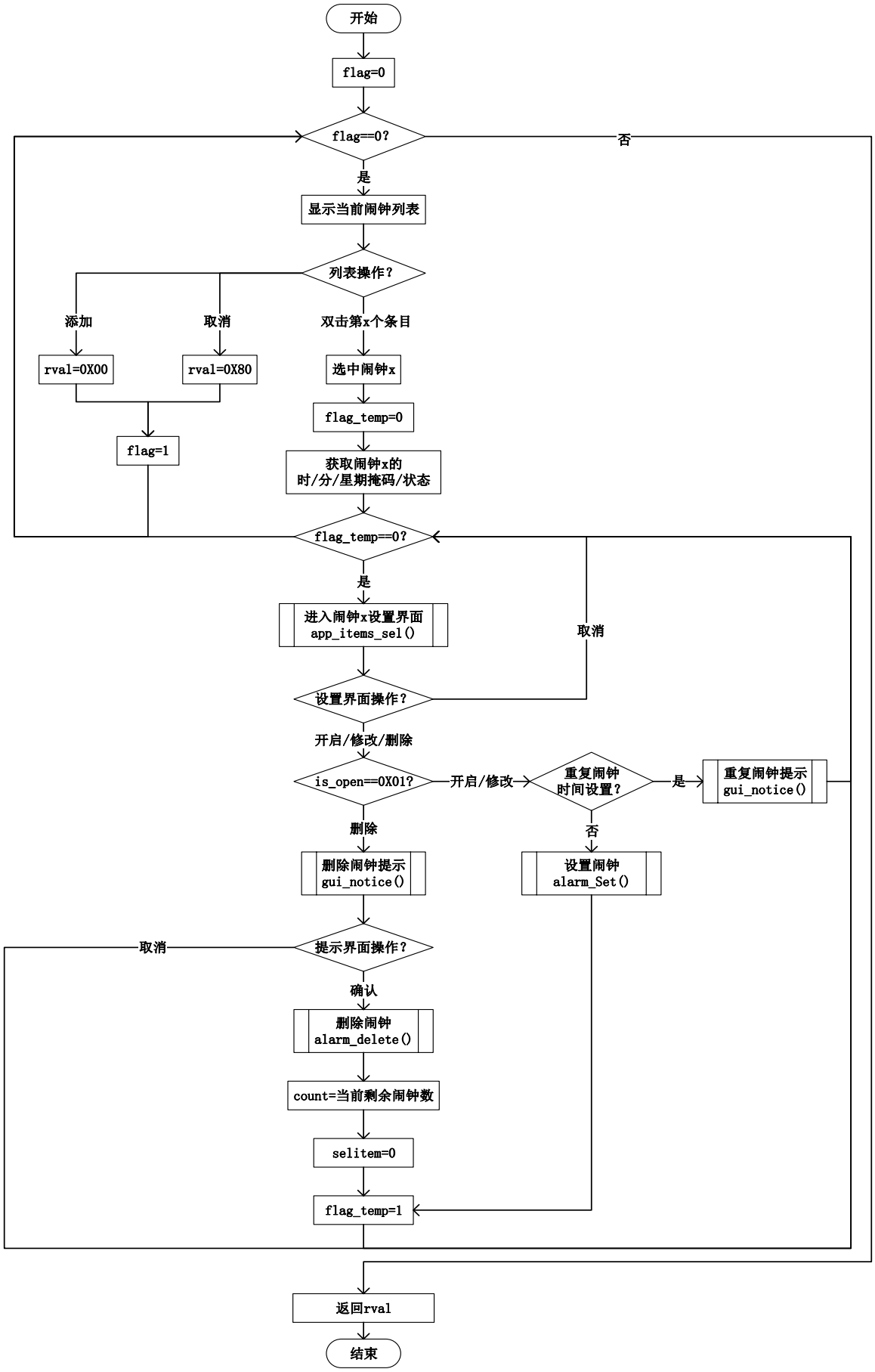


图 4.5-4 alarm_play 函数流程图

4.6 设置菜单栏窗体 setting_play

定义函数 setting_play()显示设置总菜单栏。

GUI 界面设计图示如下：

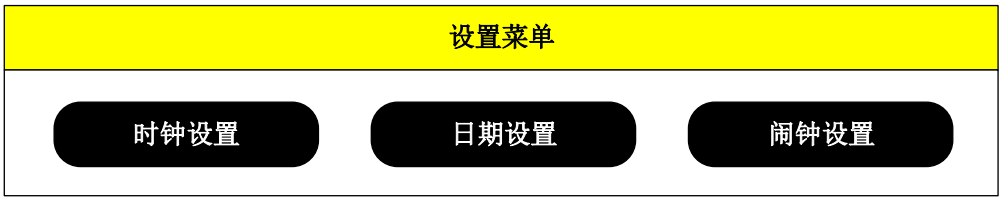


图 4.6-1 setting_play()函数 GUI 界面设计

1、函数声明如下：

```
/*****************************************************************************
 * 函数名称：setting_play
 * 输入：void
 * 输出：u8 rval
 * 功能：显示功能菜单栏，并引发功能操作
 *****************************************************************************/
u8 setting_play(void);
```

2、函数体内部变量定义如下：

```
    int i;
    u8 rval=HOLD_CLOCK_SETTING,res;
    u8 **items;
    /*****时间/日期暂存************/
    u16 temp1;
    u8 temp2,temp3;
    u8 is_open;
    _window_obj *menu;              //功能菜单栏
    _btn_obj *menu_btn[3];      /********功能键：边角按钮 0X02********
                                        menu_btn[0]：时钟设置
                                        menu_btn[1]：日期设置
                                        menu_btn[2]：闹钟设置
                                    ************************************/
    items=(u8**)alarm_tbl[gui_phy.language];
```

其中，temp1、temp2、temp3 用来存放设置的中间变量值，比如时/分、年/月/日。

3、函数主体定义如下：

```
    while(1){
        ////////////////////////////////////////////////////////////////////////////////////////////
        /***********创建功能菜单栏********/
        menu=window_creat(menu_left,menu_top,menu_width,menu_height,menu_id,1<<5,24);
        //菜单栏创建
        /***********创建功能键************/
```

```
    menu_btn[0]=btn_creat(3,menu_button_top,menu_button_width,menu_button_height,menu_button_id,0x02);
    menu_btn[1]=btn_creat(6+menu_button_width,menu_button_top,menu_button_width,menu_button_height,menu_button_id,0x02);
    menu_btn[2]=btn_creat(6+2*menu_button_width,menu_button_top,menu_button_width,menu_button_height,menu_button_id,0x02);
        ////////////////////////////////////////////////////////////////////////////////////////////
        /*********************************配置*****************************/

        /*****************界面配置******************/
        //菜单配置
        if(menu==NULL){
            rval=ERROR_GUI;
            myerror_notice(rval);
        }
        else{
            menu->caption="设置菜单";
            menu->captionheight=40;
            menu->captionbkcu=YELLOW;
            menu->captionbkcd=YELLOW;
            menu->captioncolor=BLACK;
            menu->windowbkc=WHITE;
        }
        //功能键配置
        for(i=0;i<3;i++){
            if(menu_btn[i]==NULL){
                rval=ERROR_GUI;
                myerror_notice(rval);
                break;
            }
            menu_btn[i]->font=24;
            menu_btn[i]->bcfucolor=WHITE;
            menu_btn[i]->bcfucolor=WHITE;
            menu_btn[i]->bkctbl[0]=WHITE;
            menu_btn[i]->bkctbl[1]=GRAY;
            menu_btn[i]->bkctbl[2]=BLACK;
            menu_btn[i]->bkctbl[3]=BLACK;
            if(i==0)menu_btn[i]->caption="时钟设置";
            if(i==1)menu_btn[i]->caption="日期设置";
            if(i==2)menu_btn[i]->caption="闹钟设置";

        }
        ////////////////////////////////////////
        if(rval!=ERROR_GUI){                                    //GUI 创建成功的话
            /*画组件*/
```

```
        window_draw(menu);                                  //画功能菜单栏
        for(i=0;i<3;i++)btn_draw(menu_btn[i]);              //画功能键
        //////////
        rval=1;
        while(rval!=SWITCH_CLOCK_SETTING&&rval!=HOLD_CLOCK_SETTING){
            tp_dev.scan(0);                                 //判断触摸屏有无触摸
            in_obj.get_key(&tp_dev,IN_TYPE_TOUCH);          //得到按键键值
            if(TPAD_Scan(0)){                               //按了 TPAD，返回时钟显示界面
                rval=SWITCH_CLOCK_SETTING;
                break;
            }
            for(i=0;i<3;i++){                               //判断三个功能键
                res=btn_check(menu_btn[i],&in_obj);         //menu_btn[i]是否被按下检测
                if(res){                                    //如果有功能键被按下
                    if((menu_btn[i]->sta&0X80)==0){         //有有效操作
                        /*根据功能键进入相应功能使能*/
                        switch(i){
                            case 0:/*时钟设置*/
                                【0】;
                                break;
                            case 1:/*日期设置*/
                                【1】;
                                break;
                            case 2:/*闹钟设置*/
                                【2】;
                                break;
                        }//switch(i)：判断设置什么
                    }//if((menu_btn[i]->sta&0X80)==0)：第 i 个菜单按钮有有效操作
                }//if(flag_ok)：有菜单按钮按下
            }//for(i=0;i<3;i++)

        }//rval!=SWITCH_CLOCK_SETTING||rval!=HOLD_CLOCK_SETTING
    }//if(!flag_break)：一开始有无出错
    ////////////////////////////////////////////////////////////////////////////////////////////////
    window_delete(menu);                                    //删除功能菜单栏
    for(i=0;i<3;i++)btn_delete(menu_btn[i]);                //删除功能键
    printf("setting_play:%d\n",rval);
    return rval;
}
```

在菜单栏界面时，不断测试是否有手指按下 TPAD，如果有就返回到时钟界面。注意：在进入到具体设置界面时，不对 TPAD 进行扫描。

这个函数靠三个按钮触发关键事件，由于页面美观问题，上述代码中用【0】【1】【2】代替核心代码，下面依次对这三部分代码进行解释。

4.6.1 【0】——设置时间

```
temp1=RTC_TimeStruct.RTC_Hours;
temp2=RTC_TimeStruct.RTC_Minutes;
rval=sysset_time_set((lcddev.width-300)/2,(lcddev.height-400)/2,(u8*)&temp1,(u8*)&temp2,"时钟设置界面");
if(rval==SWITCH_CLOCK_SETTING)  RTC_Set_Time(temp1,temp2,0,0);    //将设置的时间保存
```

这部分代码和【1】中代码都比较简单，先获取当前时间的时/分，然后调用函数 sysset_time_set()进入设置时间界面。如果设置时间界面点了“确认”，就调用函数 RTC_Set_Time 改变当前时间，并返回到时钟界面。

RTC_Set_Time 函数如下：

```
//RTC 时间设置
//hour,min,sec:小时,分钟,秒钟
//ampm:@RTC_AM_PM_Definitions  :RTC_H12_AM/RTC_H12_PM
//返回值:SUCEE(1),成功
//          ERROR(0),进入初始化模式失败
ErrorStatus RTC_Set_Time(u8 hour,u8 min,u8 sec,u8 ampm){
    RTC_TimeTypeDef RTC_TimeTypeInitStructure;
    RTC_TimeTypeInitStructure.RTC_Hours=hour;
    RTC_TimeTypeInitStructure.RTC_Minutes=min;
    RTC_TimeTypeInitStructure.RTC_Seconds=sec;
    RTC_TimeTypeInitStructure.RTC_H12=ampm;
    return RTC_SetTime(RTC_Format_BIN,&RTC_TimeTypeInitStructure);
}
```

4.6.2 【1】——设置日期

```
temp1=RTC_DateStruct.RTC_Year+2000;
//由于 RTC_Set_Date 的 year 只有后两个数字，所以要加上 2000
temp2=RTC_DateStruct.RTC_Month;
temp3=RTC_DateStruct.RTC_Date;
rval=sysset_date_set((lcddev.width-380)/2,(lcddev.height-400)/2,&temp1,(u8*)&temp2,(u8*)&temp3,"日期设置界面");
if(rval==SWITCH_CLOCK_SETTING)
    RTC_Set_Date(temp1-2000,temp2,temp3,RTC_Get_Week(temp1,temp2,temp3));//设置日期
```

同样先获取当前时间的年/月/日，然后调用函数 sysset_date_set()进入设置日期界面。如果设置日期界面点了“确认”，就调用函数 RTC_Set_Date 改变当前日期，并返回到时钟界面。

其中，函数 RTC_Get_Week 是为了通过设置的日期计算出星期。函数如下：

```
u8 const table_week[12]={0,3,3,6,1,4,6,2,5,0,3,5}; //月修正数据表
//获得现在是星期几
```

```
//功能描述:输入公历日期得到星期(只允许 1901-2099 年)
//year,month,day：公历年月日
//返回值：星期号(1~7,代表周 1~周日)
u8 RTC_Get_Week(u16 year,u8 month,u8 day)
{
    u16 temp2;
    u8 yearH,yearL;
    yearH=year/100;    yearL=year%100;
    // 如果为 21 世纪,年份数加 100
    if (yearH>19)yearL+=100;
    // 所过闰年数只算 1900 年之后的
    temp2=yearL+yearL/4;
    temp2=temp2%7;
    temp2=temp2+day+table_week[month-1];
    if (yearL%4==0&&month<3)temp2--;
    temp2%=7;
    if(temp2==0)temp2=7;
    return temp2;
}
```

RTC_Set_Date 函数如下：

```
//RTC 日期设置
//year,month,date:年(0~99),月(1~12),日(0~31)
//week:星期(1~7,0,非法!)
//返回值:SUCEE(1),成功
//          ERROR(0),进入初始化模式失败
ErrorStatus RTC_Set_Date(u8 year,u8 month,u8 date,u8 week){
    RTC_DateTypeDef RTC_DateTypeInitStructure;
    RTC_DateTypeInitStructure.RTC_Date=date;
    RTC_DateTypeInitStructure.RTC_Month=month;
    RTC_DateTypeInitStructure.RTC_WeekDay=week;
    RTC_DateTypeInitStructure.RTC_Year=year;
    return RTC_SetDate(RTC_Format_BIN,&RTC_DateTypeInitStructure);
}
```

4.6.3 【2】——设置闹钟

由于这一块代码实在太多了，所以分成两块进行解释。

1、第一部分：

```
if(SYSTEM_Alarm_NUM==0){                    //此时没有闹钟
    temp1=RTC_TimeStruct.RTC_Hours;
    temp2=RTC_TimeStruct.RTC_Minutes;
    temp3=0;
    is_open=Alarm[0].saveflag;                    //判断是否是第一次添加该闹钟
    rval=gui_notice((lcddev.width-300)/2,(lcddev.height-400)/2,0,"未添加闹钟!","添加");
```

```
    if(rval==SWITCH_CLOCK_SETTING){          //如果点击“添加”
        if(is_open==0)  is_open=0X02;                //如果是第一次添加
        rval=app_items_sel(0,0,lcddev.width,lcddev.height,(u8**)calendar_week_table[gui_phy.languag
e],7,(u8  *)&temp1,(u8  *)&temp2,(u8  *)&temp3,0XB0,"闹钟 1",(u8  *)&is_open);//添加闹钟界面
        if(rval==SWITCH_CLOCK_SETTING){    //如果开启
            if(alarm_equal(SYSTEM_Alarm_NUM,(u8)temp1,temp2)){//有相同设置的闹钟
                rval=gui_notice((lcddev.width-300)/2,(lcddev.height-400)/2,1,"相同闹钟已存在!","确
认");
        }
        else{
                temp3=temp3<<1;
                alarm_Set(SYSTEM_Alarm_NUM,(u8)temp1,temp2,is_open,temp3,0);
                //保存闹钟配置
            }
        }
    }
}
else  rval=SWITCH_CLOCK_SETTING;                          //否则，进入闹钟列表
```

第一部分的代码主要是为了区分在按下设置闹钟的时候，此时系统中有没有保存的已设置的闹钟。

假如此时系统中没有闹钟，需要先调用 gui_notice()函数做“未添加闹钟”提示，当选择了取消，则返回到菜单栏界面；当选择了确认，需要进入“闹钟 1”的设置界面。在设置界面时，假如按了“取消”，同样返回到菜单栏界面；按了“确认”后还需进行重复闹钟判断，假如没有重复，则保存“闹钟 1”的设置，此时系统中的闹钟数由 0 变 1。

假如此时系统中已经有了闹钟，那么需要直接进入闹钟列表界面，返回值应和闹钟设置界面设置非重复闹钟按了“确认”后的返回值相同，为 0X00。

2、第二部分：

```
while(rval==SWITCH_CLOCK_SETTING){
    printf("闹钟菜单栏：  %d\n",rval);
    rval=alarm_play(SYSTEM_Alarm_NUM);                  //显示闹钟列表
    if(rval==SWITCH_CLOCK_SETTING){                     //点了闹钟列表界面的添加
        if(SYSTEM_Alarm_NUM>=RTC_Alarm_MAX_NUM){//超过规定可以添加的闹钟数
            rval=gui_notice((lcddev.width-300)/2,(lcddev.height-400)/2,1,"闹钟已满!","确认");
            SYSTEM_Alarm_NUM=RTC_Alarm_MAX_NUM;
        }
        else{                                           //未超过闹钟数
            temp1=RTC_TimeStruct.RTC_Hours;
            temp2=RTC_TimeStruct.RTC_Minutes;
            temp3=0;
            is_open=Alarm[SYSTEM_Alarm_NUM].saveflag;
```

```
            if(is_open==0)  is_open=0X02;                    //第一次添加
            rval=app_items_sel(0,0,lcddev.width,lcddev.height,(u8**)calendar_week_table[gui_phy.lan
guage],7,(u8 *)&temp1,(u8 *)&temp2,(u8 *)&temp3,0XB0,items[SYSTEM_Alarm_NUM],(u8 *)&is_o
pen);
            if(rval==SWITCH_CLOCK_SETTING){          //点了“开启”
                if(alarm_equal(SYSTEM_Alarm_NUM,(u8)temp1,temp2)){//有相同设置的闹钟
                    rval=gui_notice((lcddev.width-300)/2,(lcddev.height-400)/2,1,"相同闹钟已存在!
","确认");
                }
                else{
                    temp3=temp3<<1;
                    alarm_Set(SYSTEM_Alarm_NUM,(u8)temp1,temp2,is_open,temp3,0);
                    //保存闹钟配置
                }
            }
        }
    }//if(rval==SWITCH_CLOCK_SETTING)
    printf("alarm_num_open:%d\n\n\r\n\n",SYSTEM_Alarm_NUM);
}
```

这一部分是为了保持闹钟菜单栏界面，并判断闹钟数是否已满，已满将调用 gui_notice()函数进行“闹钟已满”提示，不予添加新闹钟。当 rval=0X00 时，进入 while 循环，调用 alarm_play()函数显示闹钟菜单栏界面。对于闹钟菜单栏界面的双击操作以在 alarm_play()函数内部进行了解释，“添加”和“取消”将放在 setting_play()中解释

假如点了闹钟列表界面的“添加”按钮（返回值 rval=0X00），需要判断当前闹钟数是否已满，如果未满，需要对本次添加闹钟 x 的状态进行标记，表示这是第一次添加闹钟 x，接着进入没有“删除”按钮的闹钟 x 设置界面。

在闹钟 x 设置界面假如点了“确认”（返回值 rval=0X00），需要进行重复闹钟判断，重复了给予提示，否则调用 alarm_Set()函数对保存闹钟 x 的设置。

假如点了闹钟列表界面的：取消：按钮（返回值 rval=0X40），则跳出 while()循环，返回到菜单栏界面。

4.6.4 设置菜单栏函数流程图

函数流程图如下：

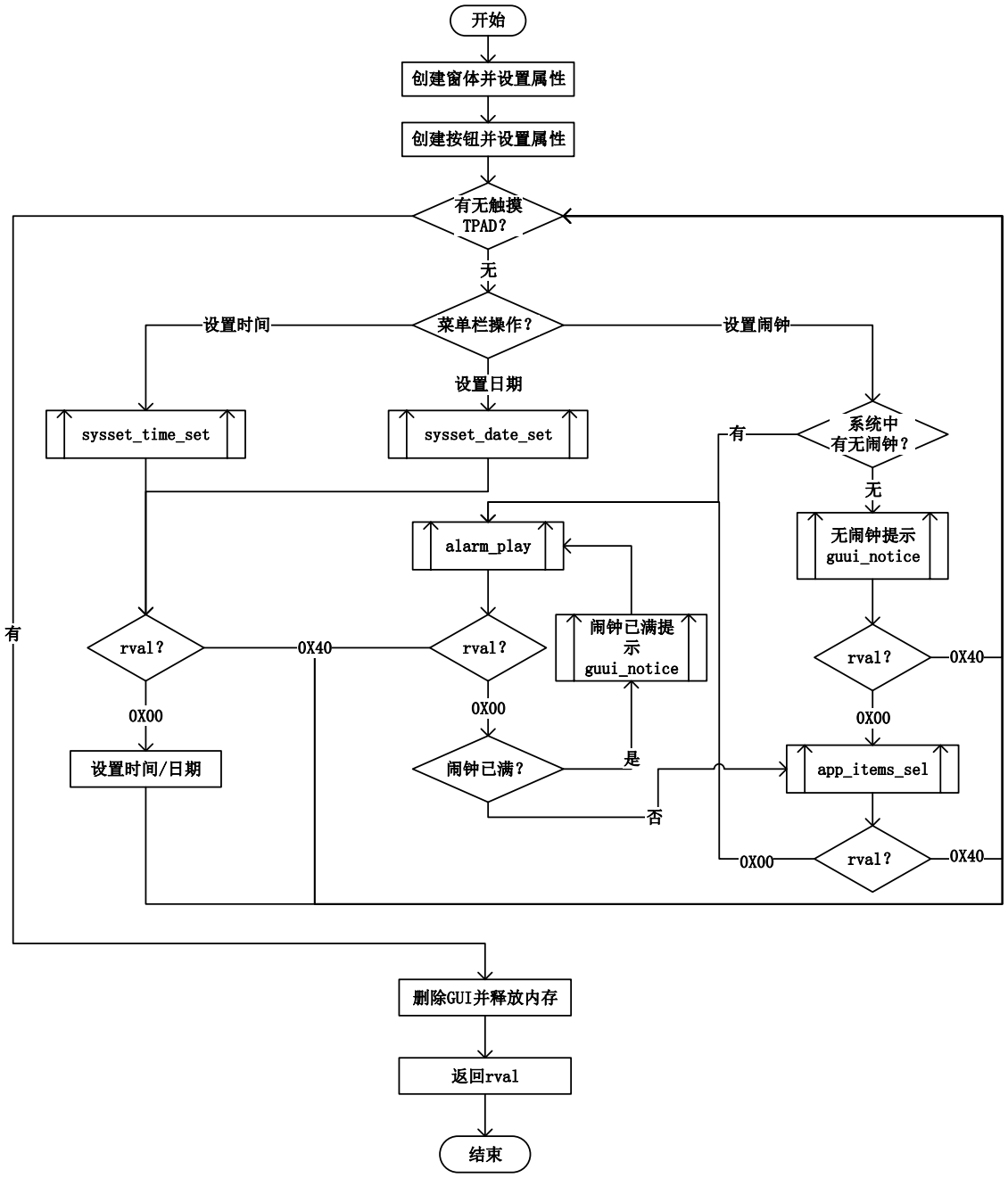


图 4.6-2 setting_play()函数流程图

4.7 GUI 界面操作整体流程图

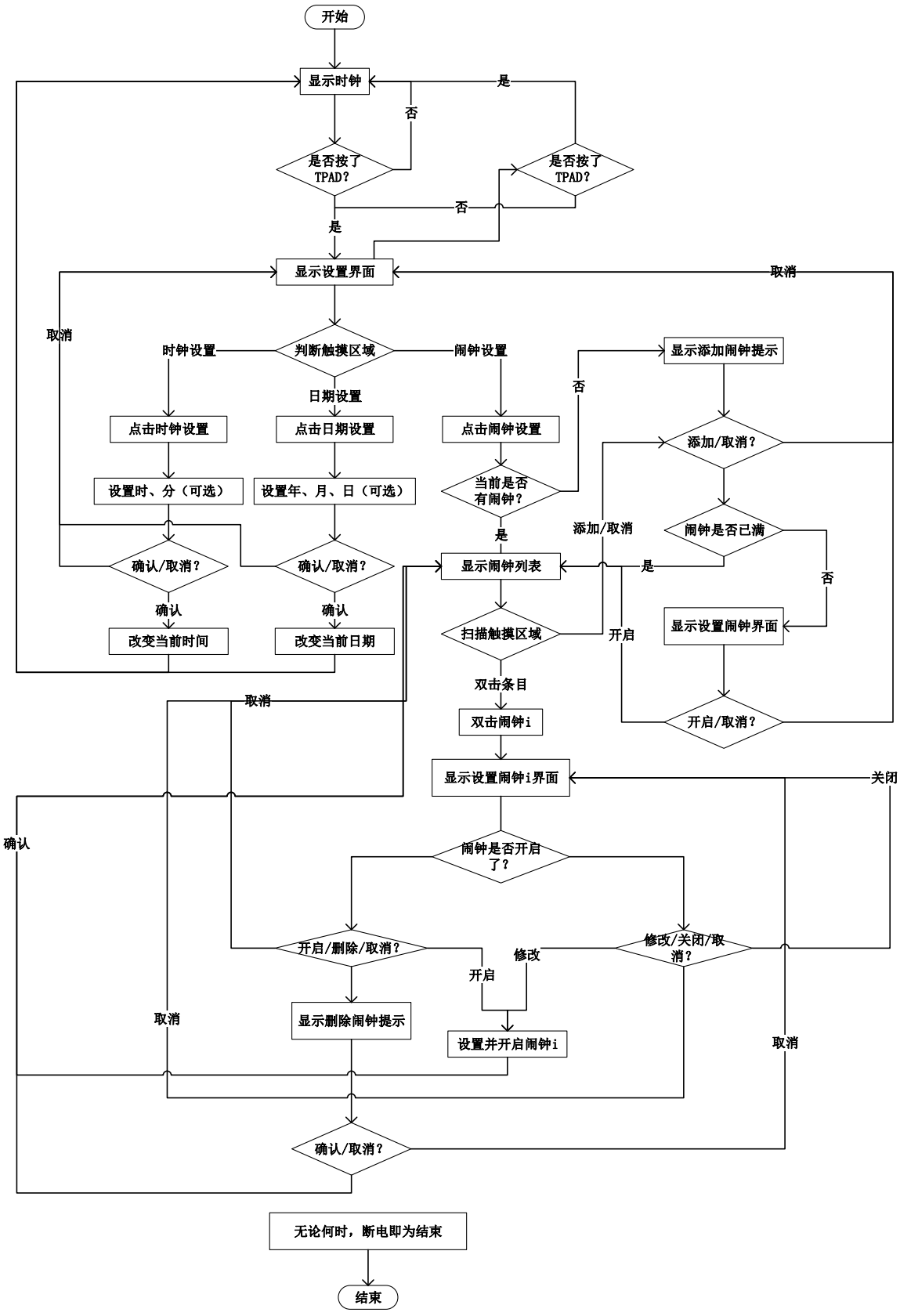


图 4.7-1 GUI 界面操作流程图

4.8 主函数 main

```
int  main(void){
    /**********变量***********/
    u8  rval=HOLD_CLOCK_SETTING;
    u8  temp=0;
    /**********函数***********/
    system_init();                          //系统各种初始化
    while(1){
        rval=calendar_play(0);              //显示时钟界面
        printf("calendar_play:%d\n",rval); //一切 printf 语句都是为了调试而设
        if(rval==SWITCH_CLOCK_SETTING){     //进入设置界面
            do{
                rval=setting_play();    //菜单栏函数
                printf("setting_play:%d\n",rval);
                BACK_COLOR=BLACK;           //背景色为黑色
                LCD_Clear(BLACK);           //清屏函数
                temp=rval;
                rval=calendar_play(2);//运行时钟
                rval=temp;
                alarm_ring();                   //判断闹钟是否要响
            }while(rval==HOLD_CLOCK_SETTING);
        }
        alarm_ring();                       //判断闹钟是否要响
    }
}
```

先调用 calendar_play()函数显示时钟界面，假如按了 TPAD，进入设置界面；直到在设置菜单栏界面再次按了 TPAD，再返回到时钟显示界面。alarm_ring()函数用于判断闹钟是否需要响铃。为了时钟的正常显示，需要进行清屏。

流程图如下：

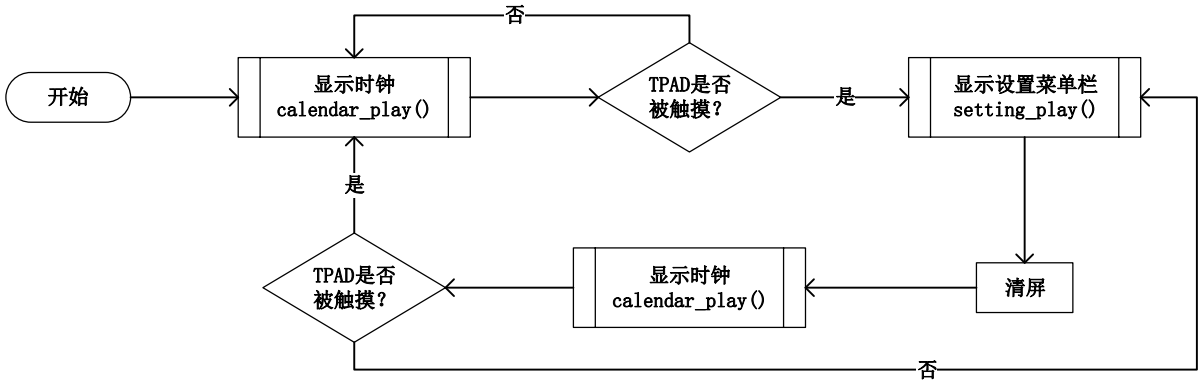


图 4.8-1 main()函数流程图

4.9 模块初始化函数 system_init

这一函数主要就是对各硬件模块进行初始化。

```
void  system_init(){
    /******MALLOC*****/
    FSMC_SRAM_Init();           //初始化外部 SRAM
    my_mem_init(SRAMIN);        //初始化内部内存池
    my_mem_init(SRAMEX);        //初始化外部内存池
    my_mem_init(SRAMCCM);       //初始化 CCM 内存池
    /*******GUI******/
    piclib_init();              //piclib 初始化
    exfuns_init();              //FATFS  申请内存
    /*********************************初始化**********************************/
    delay_init(168);            //初始化延时函数
    uart_init(115200);          //初始化串口波特率为 115200
    LED_Init();                 //初始化 LED
    TPAD_Init(8);               //初始化触摸按键,以 84/4=21Mhz 频率计数
    LCD_Init();                 //LCD 初始化
    LCD_Clear(BLACK);           //清屏函数
    gui_init();                 //GUI 插件初始化
    tp_dev.init();              //触摸屏初始化
    My_RTC_Init();              //初始化 RTC
    usmart_dev.init(84);        //初始化 USMART
    Adc_Init();                 //ADC 初始化
    BEEP_Init();                //蜂鸣器初始化
    AT24CXX_Init();             //EEPROM 初始化
    WM8978_Init();              //WM8978 初始化
    WM8978_HPvol_Set(40,40);    //耳机音量设置
    WM8978_SPKvol_Set(50);      //喇叭音量设置;
    /*****************************************************************************/
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2);//设置系统中断优先级分组 2
    RTC_Set_WakeUp(RTC_WakeUpClock_CK_SPRE_16bits,0);//配置 WAKE  UP 中断,1 秒中断一次
    /*********************************汉字************************************/
    font_init();                //汉字字体初始化
    W25QXX_Init();              //W25QXX 初始化
    f_mount(fs[1],"1:",1);      //挂载 SPI  FLASH
    /*********************************数码管字体*******************************/
    font_set();                 //设置特殊字体
    /*********************************AT24CXX 检测*****************************/
    AT24CXX_Check();            //检测 IIC 器件是否正常
    /*****************************闹钟*****************************************/
    alarm_init();               //闹钟初始化
}
```

4.10 zox 完成的工作

在此不多做介绍，以 zox 所画流程图显示，具体由 zox 的报告予以说明：

4.10.1 时钟显示 calendar_play

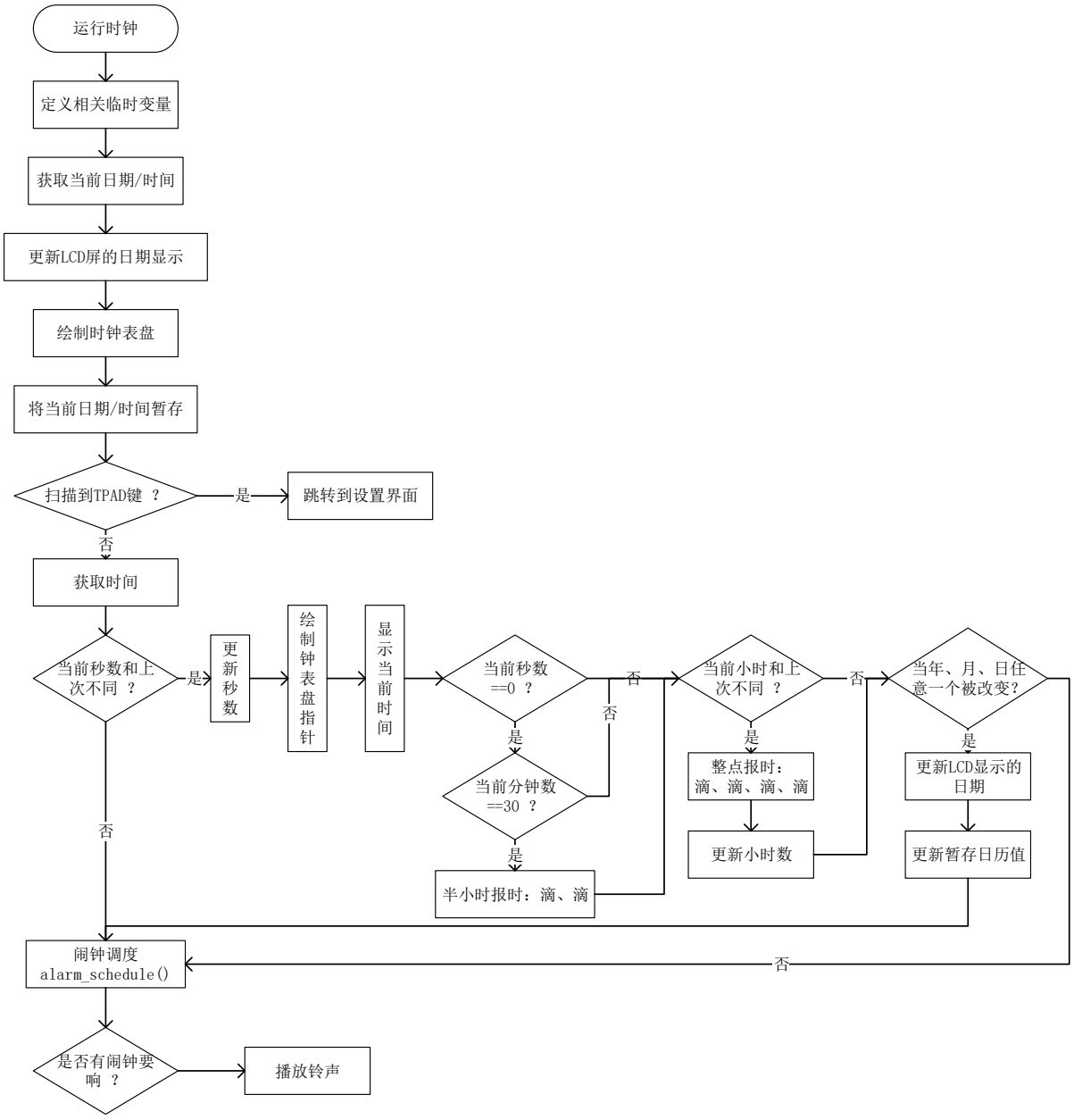


图 4.10-1 时钟显示流程图

4.10.2 闹钟 alarm_init

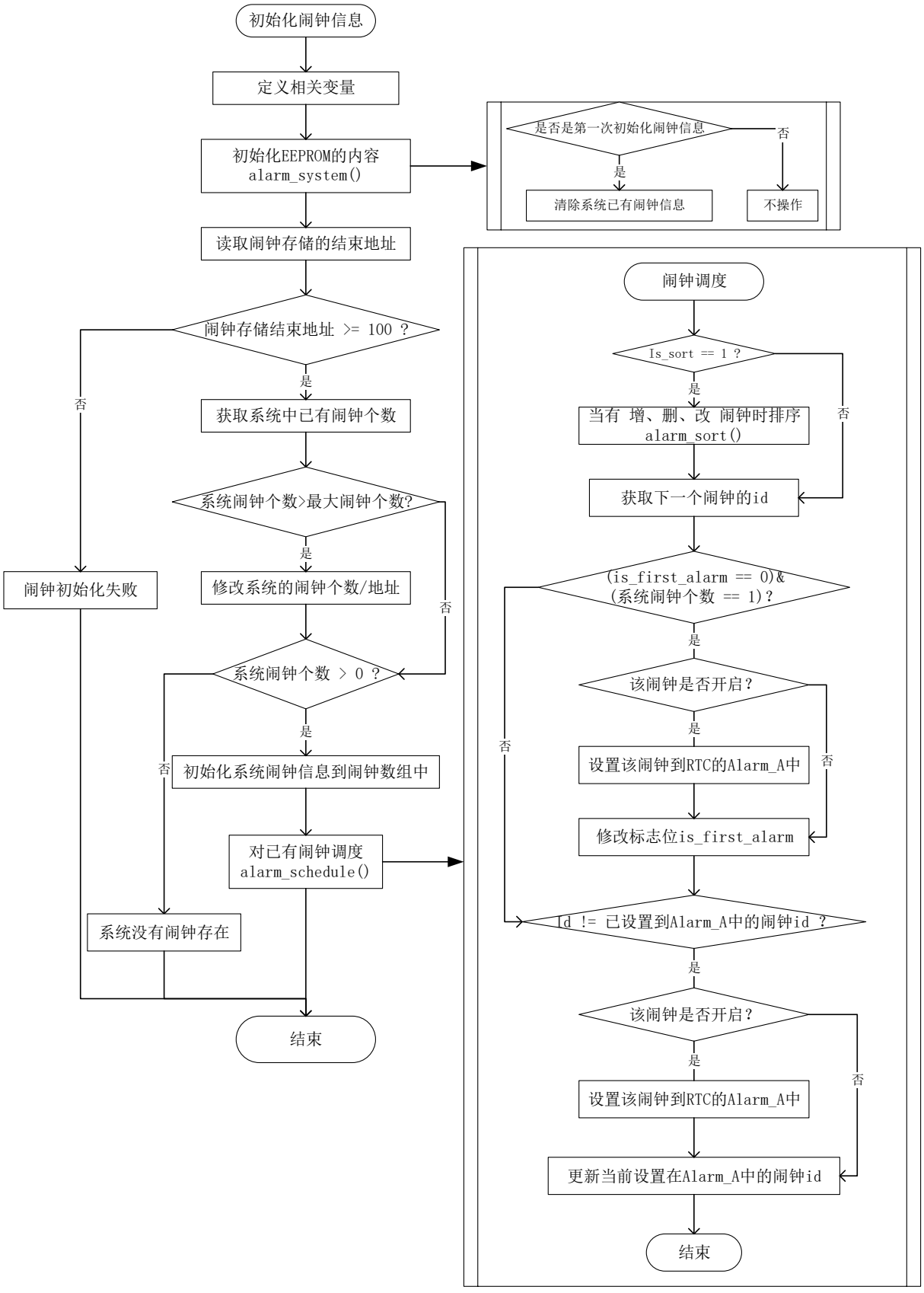


图 4.10-2 闹钟流程图

五、 实现效果

此模块给出实现结果的效果图，PC 机上的屏幕截图、开发板的运行效果照片。

5.1 时钟显示和设置界面显示

5.2 设置时间和日期


2020-10-11
星期日
时钟设置界面
确定
取消
温度: 23.92℃
设置菜单
时钟设置
日期设置
闹钟设置
点击
ALIENTEK 4.3' TFTLCD

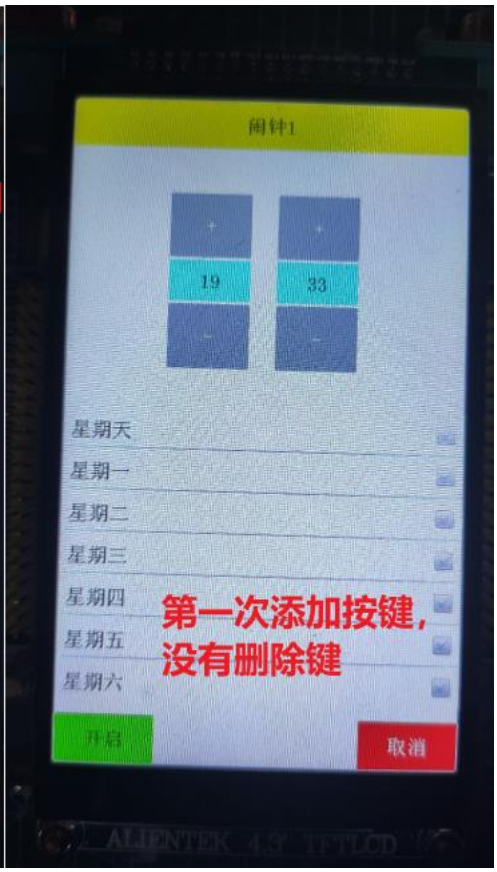

2020-10-11
星期日
19:33:04
温度: 24.25℃
时间已修改
ALIENTEK 4.3' TFTLCD

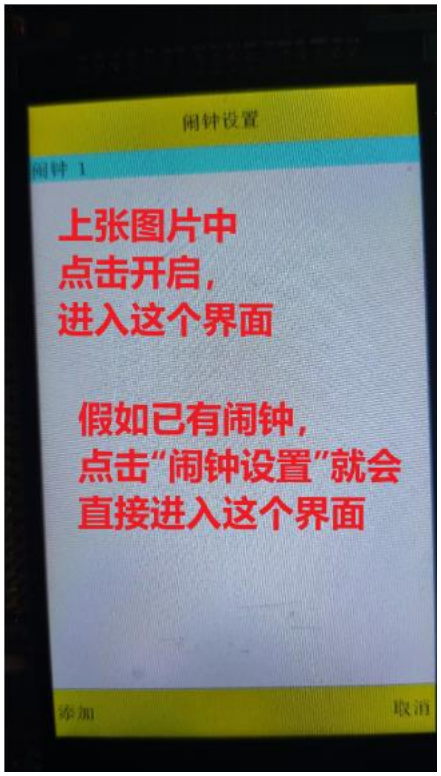

2020-10-11
星期日
日期设置界面
2020
11
13
确定
取消
温度: 24.89℃
设置菜单
时钟设置
日期设置
闹钟设置
点击
ALIENTEK 4.3' TFTLCD


2020-11-13
星期五
日期已修改
19:33:22
温度: 24.57℃
ALIENTEK 4.3' TFTLCD

5.3 闹钟设置

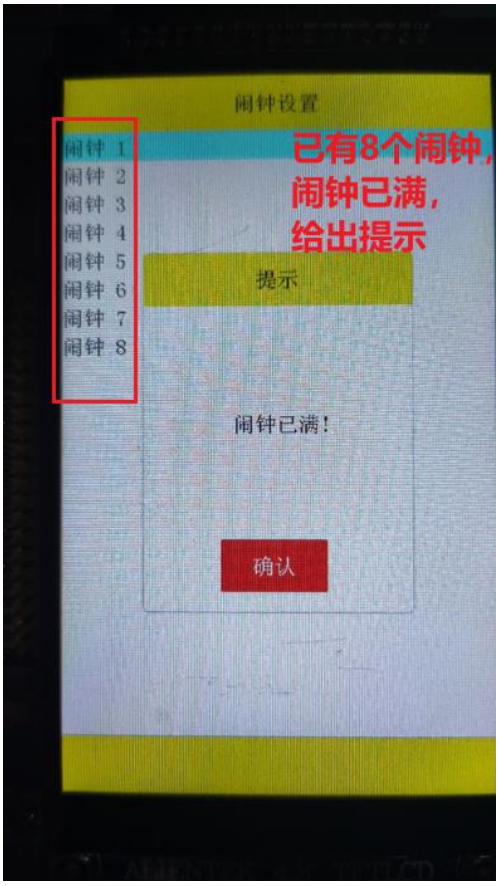

2020-11-13
星期五
此时没有闹钟，显示提示窗口
提示
未添加闹钟!
添加
取消
温度：
设置菜单
时钟设置
日期设置
闹钟设置
ALIENTEK 4.3' TFTLCD

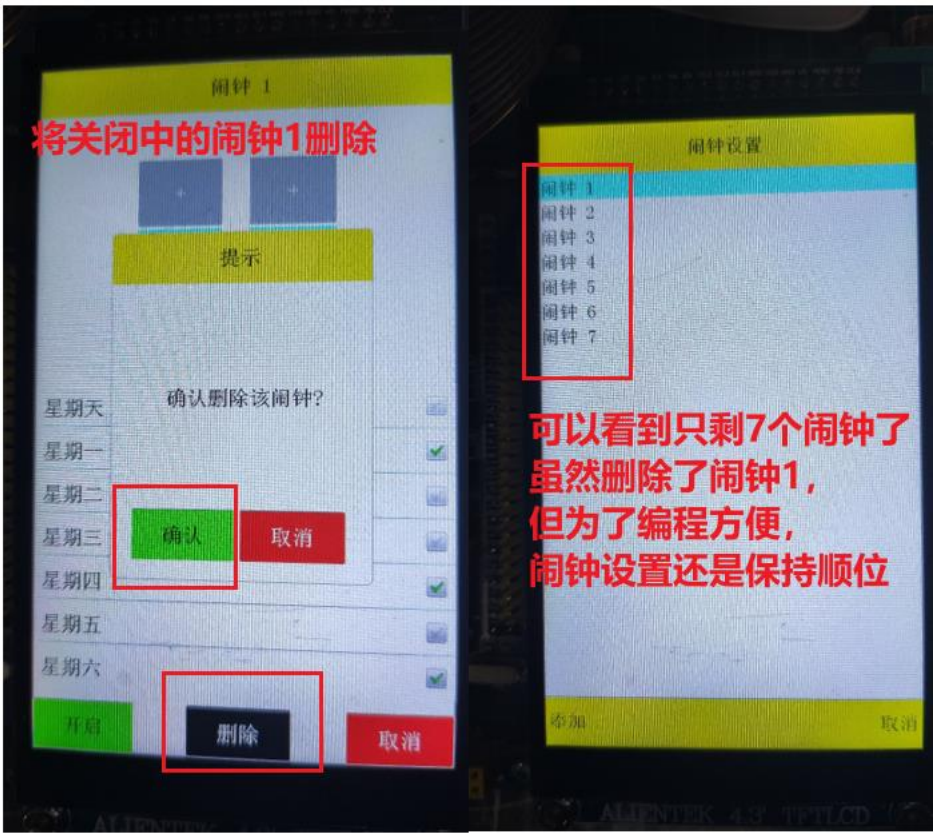

闹钟1
19
33
星期天
星期一
星期二
星期三
星期四
星期五
星期六
第一次添加按键，
没有删除键
开启
取消
ALIENTEK 4.3' TFTLCD


闹钟设置
闹钟 1
上张图片中
点击开启，
进入这个界面
假如已有闹钟，
点击“闹钟设置”就会
直接进入这个界面
添加
取消

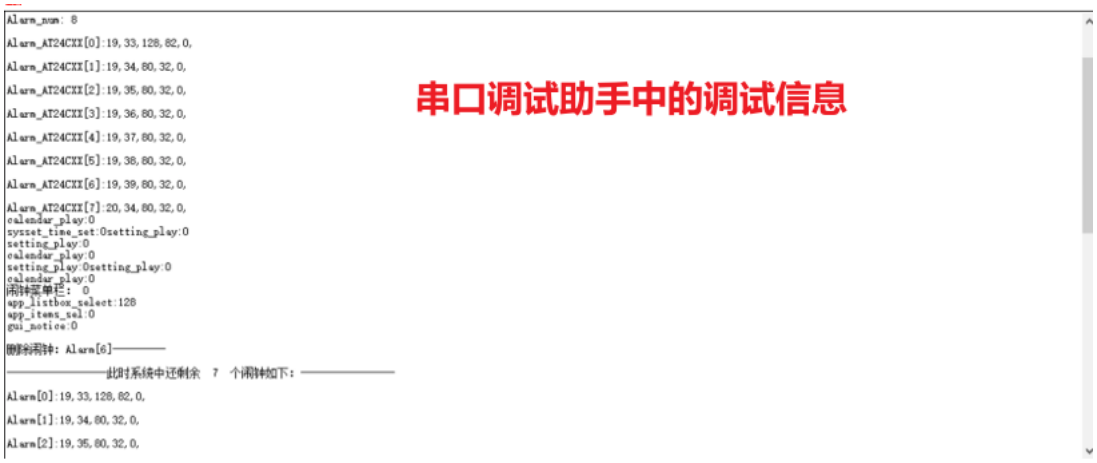

点击“闹钟1”条目后，进入“闹钟1”设置界面
闹钟 1
19
33
星期天
星期一
星期二
星期三
星期四
星期五
星期六
开启中的闹钟
修改
关闭
取消
闹钟 1
19
33
星期天
星期一
星期二
星期三
星期四
星期五
星期六
关闭中的闹钟
开启
删除
取消


闹钟 3
19
34
添加闹钟3时，时间和闹钟2一样，会给出重复闹钟提示
星期天
星期一
星期二
星期三
星期四
星期五
星期六
开启
取消
闹钟 3
提示
相同闹钟已存在!
确认
星期天
星期一
星期二
星期三
星期四
星期五
星期六
开启
取消


闹钟设置
闹钟 1
闹钟 2
闹钟 3
闹钟 4
闹钟 5
闹钟 6
闹钟 7
闹钟 8
已有8个闹钟，
闹钟已满，
给出提示
提示
闹钟已满！
确认


闹钟 1
将关闭中的闹钟1删除
提示
确认删除该闹钟?
星期天
星期一
星期二
星期三
星期四
星期五
星期六
确认
取消
开启
删除
取消
闹钟设置
闹钟 1
闹钟 2
闹钟 3
闹钟 4
闹钟 5
闹钟 6
闹钟 7
可以看到只剩7个闹钟了
虽然删除了闹钟1，
但为了编程方便，
闹钟设置还是保持顺位
添加
取消

5.4 调试



六、 总结

6.1 已实现的功能

1、表盘、日历、时间、内部温度显示；

2、整点报时；

3、触摸设置时钟时间：时、分；

4、触摸设置日历：年、月、日；

5、闹钟设置，包括：

（1）添加和删除闹钟；

（2）开启和关闭闹钟；

（3）设置闹钟的时间；

（4）显示已设置的闹钟；

（5）未添加闹钟提醒；

（6）闹钟已满提醒；

（7）闹钟删除提醒；

6、修改某个变量，可以清空实验板已有闹钟数。

6.2 小组分工

本次嵌入式课程设计由我和 zox 同学协作完成。我负责前端 GUI 界面开发，zox 同

学提供后端函数接口。

在设计过程中，我们先讨论好各自定义的函数模块所需要传入的参数，增量修改课设的程序模型。先实现了时钟的显示以及设置日期、时间等基础功能，再进行闹钟部分的设计。此次设计的难点着重在闹钟部分，直到设计结束的那一天，仍有很大的空间进行改进，我会在 6.2 小节中给予说明。

6.3 所遇问题

6.3.1 小组问题

1、传参问题，主要是指针问题。由于函数局部变量需要对结构体的某些变量进行修改，设计指针形式达到传递参数地址的目的。

2、各函数接口衔接问题。由于是小组合作，在相互磨合的过程中，不免遇到了两人定义的函数衔接出现问题，具体表现在：（1）缺少需要的参数；（2）A 定义的函数无法作用于 B 定义的函数等。但总体上函数还是封装地较好的，基本上都是逻辑问题，不用大幅度去修改代码。

3、调试方案：printf＋while(1)。为了更加清晰地显示每一步造成的变化，使用串口调试助手进行调试，对函数返回值以及修改的变量给予显示。合理运用 while(1)观察程序是否运行到了我们需要的那一步。

4、模块初始化。起初忘记了硬件初始化，造成硬件模块无法启动的后果。

5、使用全局变量的问题。对某一文件中的全局变量需要在外部引用的话，必须 extern。

6.3.2 个人问题

1、GUI 框架搭建。GUI 框架是从 STM32F4 的综合测试实验中扒下来的，由于实验室实验板无法进行综合实验的测试，前两天一直无法更直接地去了解 GUI 的运作方式，着实费了一番功夫。

2、页面嵌套显示问题，应该返回什么值，返回到哪个页面。GUI 界面操作看似很简单，实际上针对每个函数的返回值，应该去执行什么操作，执行完操作后又应该停留在哪个页面，这些都是逻辑性挺强的东西。本质上是调用函数对页面进行刷新。

3、GUI 位置、大小。为了页面的美观和使用舒适度，调整 GUI 的位置和大小就调试了非常非常多次。

5、设置界面时钟显示问题。由于对各界面的调度本质上基于刷新，因此在设置界面

操作完成按下“取消”后，无法进入时钟时分的显示函数体中。想到的解决方案是在calendart_play()函数中设置 tag=2，解决设置界面时钟的时分也可显示的问题。

6、时钟和设置界面切换问题。主要是理清逻辑，这一问题主要体现在主函数中，设置 do-while 循环是为了按下“取消”按钮后，还保持在设置界面。

6.3.3 可改进的地方

1、调试时 printf 的输出问题。有时候 printf 会输出乱码，并且设置位置不恰当导致循环显示重复的调试信息。这里可改进，不过对功能影响不大。

2、运用 ucosII 解决 while(1)问题。我们的设计过程中，涉及到很多的 while(1)循环，当进入一个循环后，除非是中断，否则将无法执行其他的函数。运用 ucosII 将有效解决 while(1)循环造成的问题，比如：

（1）闹钟响铃触摸控制停止问题。拟设计当闹钟响铃时，弹出“关闭闹钟”提示，按下“确认”按钮停止闹铃。而实验中，采用的是按键停止闹钟响铃。

（2）闹钟响铃时时钟不走秒问题。由于时钟走秒和闹钟响铃在不同的 while 循环中，闹钟响铃过程中无法进行时钟走秒。

3、单次闹钟和按星期设置闹钟切换。考虑到在实际生活中，闹钟可以只想一次，也可以周期性响铃。实验中只设置了按星期周期性响铃，考虑到技术难度和成员实现不同两个方面，给出两个解决方案：

（1）由写后端的同学实现这样一个函数，当我不选择星期的时候默认为一次性闹钟。那么就需要判断假设在设置界面的闹钟过期问题。

（2）由写前端的同学实现这样一个过程，当我不选择星期的时候提示必须至少选择一个星期，否则不允许设置闹钟。那么就需要考虑按下“请选择星期”提示界面的“确认”按钮后，应该返回到那个界面。

6.4 心得体会

本次嵌入式课程设计，满打满算整整花了两个星期，包括在实验室做实验的时间以及在宿舍自己用开发板调试的时间。最后的成品还是很漂亮的，唯一的遗憾是对于闹钟的模式设置还是存在一些问题，但整体功能还是比较完善的。

我们采用小组合作的方式进行课程设计，相比于之前的汇编课程设计，此次合作对于每个人的工作有了更高的区分度，进一步体会了分工合作的奥义。在此感谢小组成员

zox 同学的努力和付出。

感谢老师领进 STM32F4 的门，基于仔细的课堂讲解和认真的课堂学习，才能很好地完成这一次的课设任务。

本科期间，硬件实验起于电子钟，终于电子钟，有始有终，感慨万千。