南京理工大学

NANJING UNIVERSITY OF SCIENCE & TECHNOLOGY

计算机科学与工程学院

“嵌入式系统”实验报告书

题目：ex3_922106840127_Timer

学号：922106840127

姓名：刘宇翔

成绩

日期：　　2025 年　　4 月　　7 日

1 题目要求

1. 题目设计要求

（1） 作业内容

选择 TIM6 基本定时器实现精确的 1 秒的定时，将时、分、秒信息显示在 LCD 上，通过按键可实现时（左键）、分（下键）、秒（右键）调整。时间初值为 08:10:00。

（2） 完成要求

工程名称命名:ex3_学号_Timer,并打包成:ex3_学号_Timer.rar 压缩文件夹

实验报告 PDF 格式:ex3_学号_Timer.pdf

2. 拟实现的具体功能

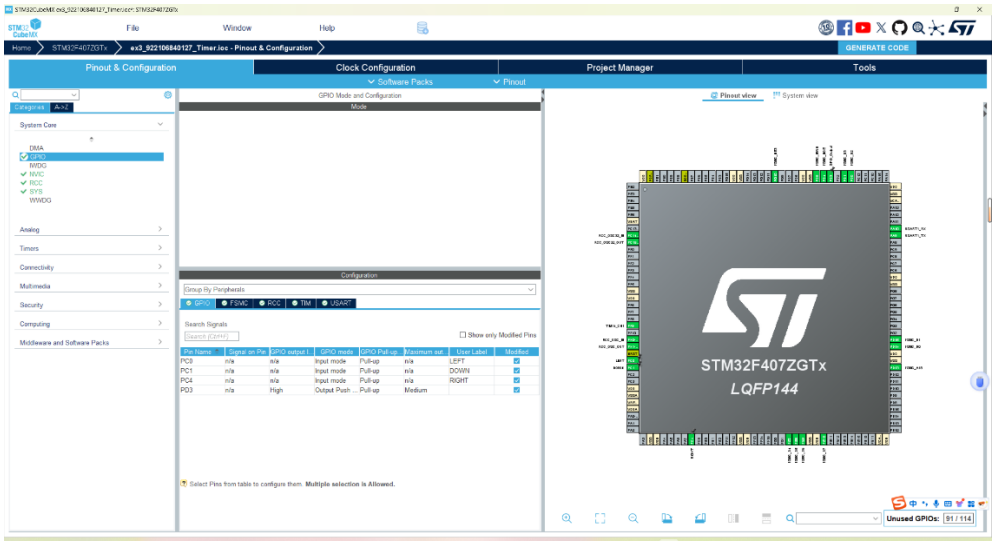
本次实验拟实现一个基于 STM32 的时钟显示系统，通过定时器中断自动累加并更新当前时间，然后将格式化后的时间数据显示在 LCD 上，同时利用三个方向键（左、下、右）实现对小时、分钟和秒数的实时调整。还加入消抖机制，避免短间隔内误触。此外，代码还通过串口输出调试信息，便于实时监控系统状态。

2 总体设计

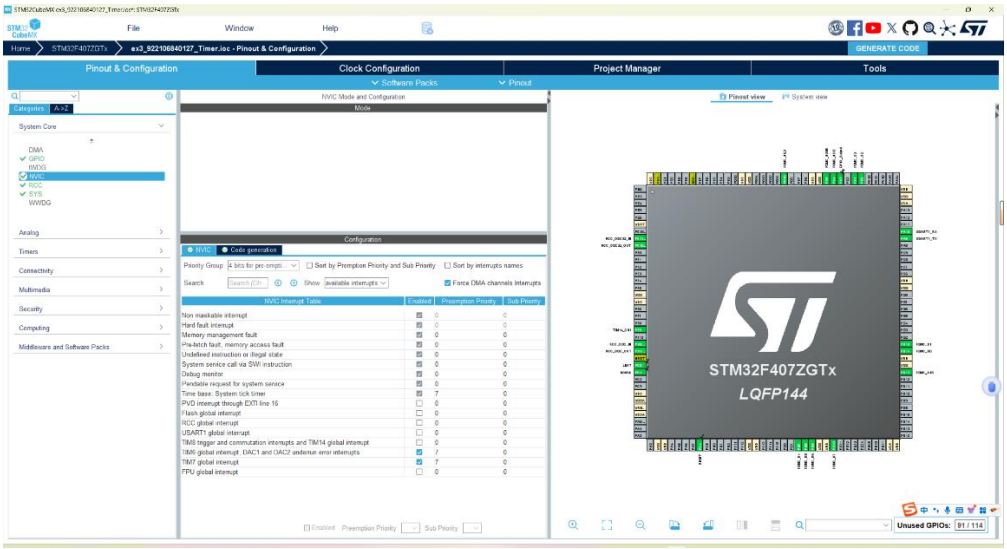
2.1 硬件设计

1. 硬件设计思路

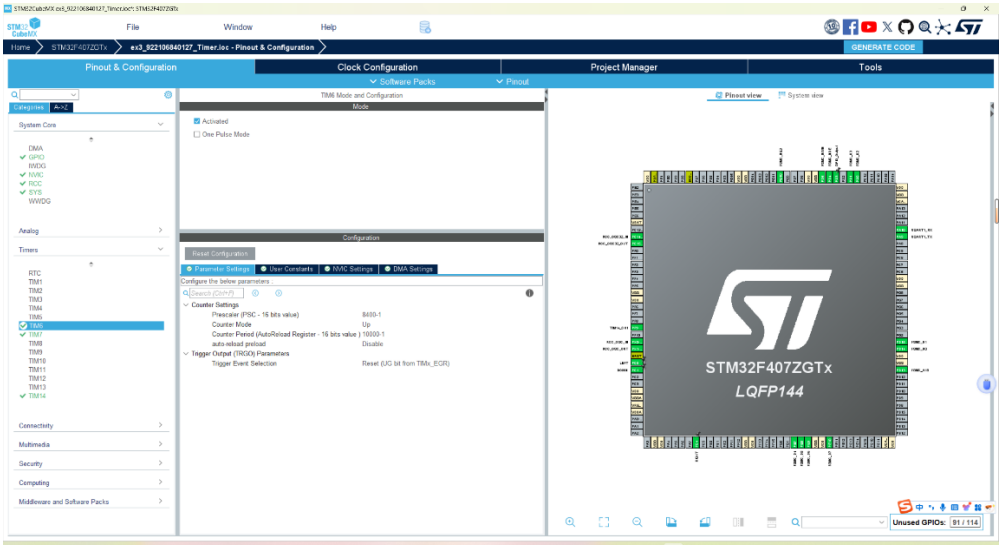
我通过查询相关开发板原理图确定了本次实验所需要的有关 GPIO 端口、NVIC 设置、USART 通信端口与 TIM 相关的设置，具体的代码工程配置硬件设计如下：



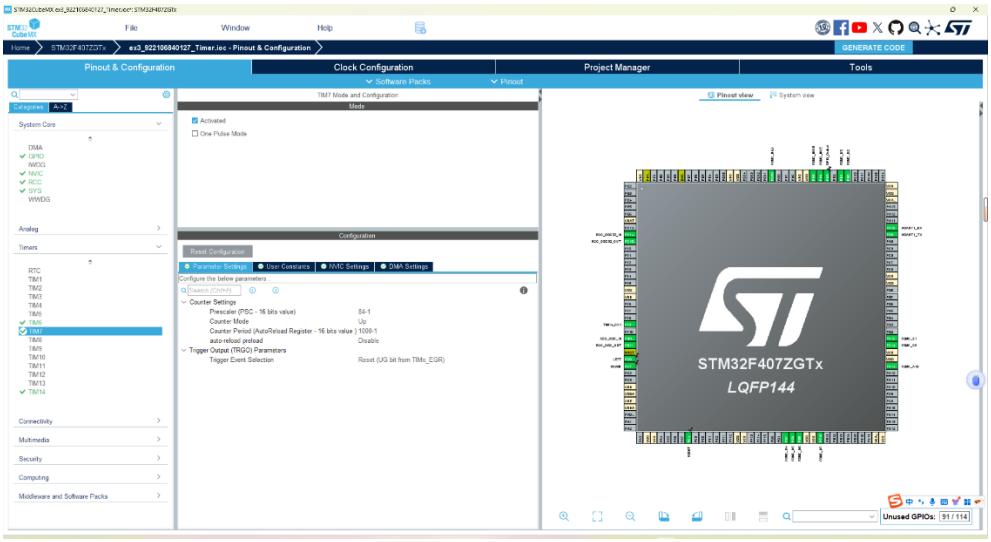
GPIO 设置



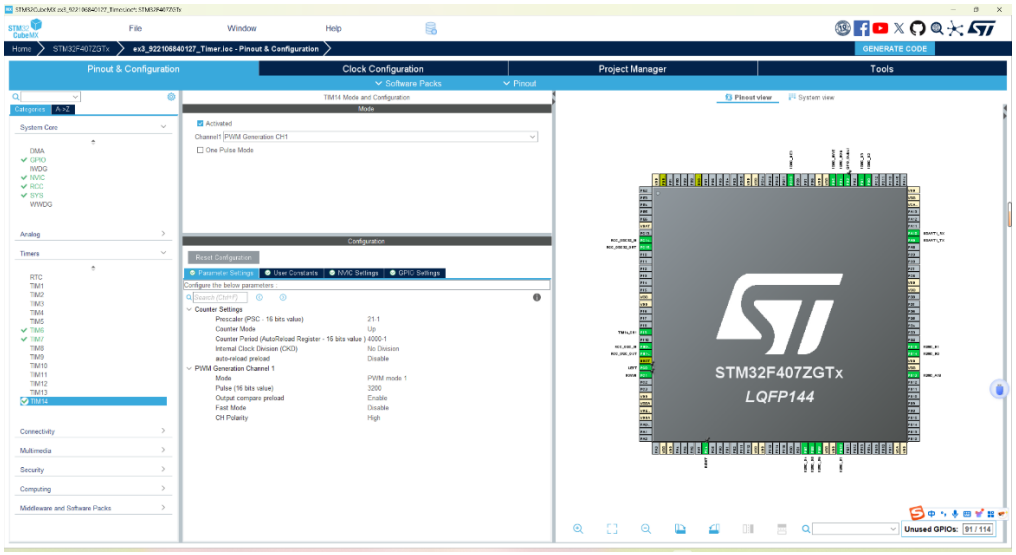
NVIC 设置



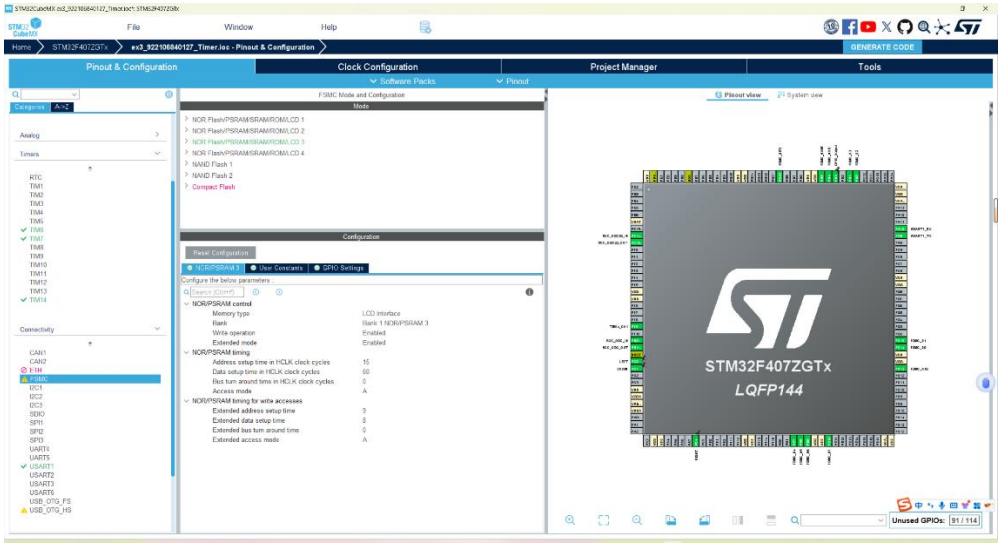
TIM6 设置



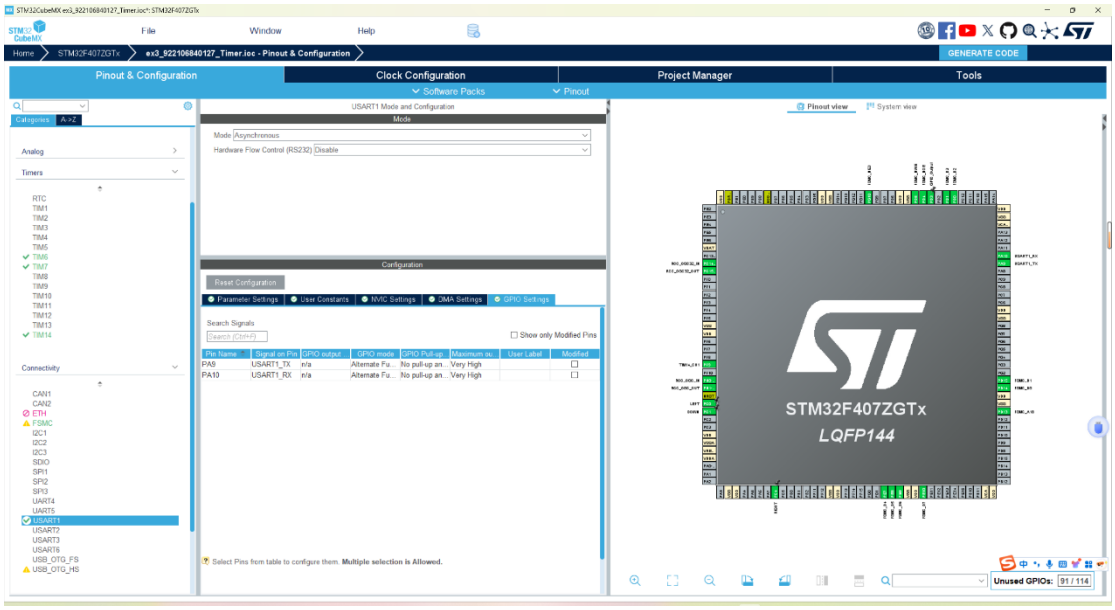
TIM7 设置



TIM14 设置



FSMC 设置



USART1 设置

如图所示，我在 NVIC 的相关设置界面对几种中断的相关属性做了具体的规定，USART 的通信模式采用的是异步通信，FSMC 是设置 LCD 相关的内容，TIM 系列计时器是本次实验的重点内容。

其他设置与前几次实验相同，如 RCC 采用 Crystal/Ceramic Resonator，使用 Serial Wire 用于调试对应的 debug 接口，对 Clock Configuration 的时钟配置进行规范，选择了 MDK-ARM V5.32 作为编译工具链，其他内容此处省略。 以上配置图是我作为 STM32CUBEMX 进行的配置设置，设置后点击 “Generate Code” 生成初始化代码。

2.2 软件设计

1. 软件设计概述

本软件主要实现基于定时器中断的电子时钟显示与按键调整功能，通过 STM32 单片机利用 TIM6 实现时间的自动累加，结合 TIM7 实现对按键输入的扫描与消抖处理，从而完成对当前时、分、秒的动态调整，并将结果实时显示在 LCD 屏幕上。系统基于 HAL 库进行外设初始化，采用模块化设计方式，整体结构简洁清晰，具备良好的可读性、稳定性与扩展性。具体设计内容如下：

（1） 系统初始化模块

在 main 函数中，系统首先通过 HAL_Init() 对底层硬件进行初始化，并调

用 SystemClock_Config() 配置系统时钟，确保各外设时序正确。随后依次初始化 GPIO、FSMC、TIM6、TIM7、TIM14 和串口 USART1，并通过 HAL_TIM_Base_Start_IT() 启动定时器中断，通过 drv_lcd_init() 初始化 LCD 显示屏，并调用 lcd_clear() 清屏设置初始显示状态。

（2） 定时器中断处理模块

本系统中 TIM6 用于实现一秒钟时间的周期性累加，在其中断回调函数中，每秒更新时间变量 g_sec，并根据进位规则同步更新 g_min 和 g_hour；为提高代码可读性，将其逻辑封装为 UpdateTime() 函数。TIM7 则用于定时扫描用户按键状态，通过读取 GPIO 引脚状态并进行简单消抖处理后，判断是否为有效按键触发，若检测到按键按下，则置位 g_key_flag 标志位，供主循环调用处理，对应功能逻辑封装为 ScanKeys() 函数。

（3） 按键事件处理模块

在主循环中通过 Handle_Key_Event() 函数对 g_key_flag 进行判断，若有按键按下，则对 g_hour、g_min、g_sec 进行加一操作，并清空标志位，实现时钟的人工校准与调整，增强了系统的可操作性与用户交互性。

（4） LCD 显示模块

通过 Update_LCD_Display() 函数，实时读取当前时间并格式化为“hh:mm:ss”格式字符串，调用 lcd_show_string() 显示在 LCD 屏幕指定区域，确保用户能够直观查看当前时钟状态。

总体而言，本软件依托 STM32 定时器中断与按键输入机制，构建了一个功能完整、响应及时的电子时钟系统，逻辑清晰，模块划分合理，为进一步扩展如闹钟、日期显示等功能打下良好的基础。

2.软件流程分解

A. 初始阶段

开始 → 系统初始化

程序启动后，首先完成各项硬件与外设的初始化配置，包括：

- 系统底层初始化（调用 HAL_Init() 对 MCU 进行复位与基础设置）
- 系统时钟配置（通过 SystemClock_Config() 配置主频与外设时钟，确保系统正常运行）

• 初始化 GPIO、FSMC、定时器 TIM6、TIM7、TIM14 以及 USART1（分别调用 MX_GPIO_Init() 、 MX_FSMC_Init() 、 MX_TIMx_Init() 与 MX_USART1_UART_Init()，为按键扫描、LCD 显示与串口通信等功能提供支持）

• 启动定时器中断（调用 HAL_TIM_Base_Start_IT() 启动 TIM6 和 TIM7 的中断功能，用于时间累加与按键检测）

• 初始化 LCD（调用 drv_lcd_init() 配置液晶模块，并通过 lcd_clear() 清屏显示初始界面）

B. 主循环结构

初始化完成 → 进入主循环

系统进入主循环后，持续监测按键标志位 g_key_flag 是否被设置，用于响应用户的按键调整请求；同时定期调用显示函数更新 LCD 屏幕中的当前时间。

C. TIM6 定时器中断触发流程

1 秒定时中断 → 进入 HAL_TIM_PeriodElapsedCallback()

当 TIM6 达到设定周期（1s）时触发中断，自动进入中断回调函数 HAL_TIM_PeriodElapsedCallback()，通过判断中断源为 TIM6 后，调用 UpdateTime() 对时间变量进行累加处理，包括秒数进位至分钟、分钟进位至小时等逻辑。

D. TIM7 定时器中断触发流程

10ms 按键扫描周期 → 进入 HAL_TIM_PeriodElapsedCallback()

TIM7 以较高频率（如 10ms）周期性触发中断，用于检测用户按键状态。在中断回调中判断中断源为 TIM7 后，调用 ScanKeys() 函数读取按键 GPIO 电平，识别是否有按键按下并设置对应标志 g_key_flag，从而支持后续时间的手动调整。

E. 按键处理逻辑

检测到按键标志 → 调用 Handle_Key_Event()

在主循环中判断 g_key_flag 是否为非零值，若检测到用户按键，则进入 Handle_Key_Event() 函数，对应修改 g_hour、g_min 或 g_sec 的值，完成当前时间的调整，同时清除标志位。

F. 显示刷新流程

周期性更新 LCD → 调用 Update_LCD_Display()

系统每轮主循环均调用 Update_LCD_Display() 函数，将当前时间格式化为标准时分秒字符串，并显示于 LCD 屏幕的指定区域，使用户可直观获取时间状态。

G. 循环机制

中断服务 → 返回主循环

每次中断处理完成后，系统退出中断服务函数，返回主循环，持续等待下一轮按键输入或定时中断触发，实现完整稳定的电子时钟运行逻辑。

3. μvision 详细代码

```
#include "main.h"
#include "tim.h"
#include "usart.h"
#include "gpio.h"
#include "fsmc.h"
#include "stdio.h"
#include "./BSP/LCD/drv_lcd.h"
#include "./BSP/LCD/rttlogo.h"

/* 配置系统时钟函数声明 */
void SystemClock_Config(void);
volatile uint8_t g_hour = 8, g_min = 10, g_sec = 0;
volatile uint8_t g_key_flag = 0;

/* 按键状态结构体，用于消抖处理 */
typedef struct {
    uint8_t debounce_cnt;
    uint8_t stable_state;
    uint8_t last_state;
} KeyState;
KeyState keys[3] = {0};    // [0] Left, [1] Down, [2] Right
```

```
/* 将 printf 重定向到 UART */
int fputc(int ch, FILE *f)
{
      HAL_UART_Transmit(&huart1, (uint8_t *)&ch, 1, 1000);
      return ch;
}

/* 更新系统时间：秒、分、时累加 */
void Update_Time(void)
{
   if (++g_sec >= 60) {
      g_sec = 0;
      if (++g_min >= 60) {
         g_min = 0;
         if (++g_hour >= 24)
            g_hour = 0;
      }
   }
}

/* 处理按键消抖，检测按键事件 */
void Process_Key_Debounce(void)
{
   const uint16_t pins[3] = {LEFT_Pin, DOWN_Pin, RIGHT_Pin};
   for (int i = 0; i < 3; i++) {
      uint8_t current = HAL_GPIO_ReadPin(GPIOC, pins[i]);
      if (current == GPIO_PIN_RESET) {
         if (keys[i].debounce_cnt < 2) {
            keys[i].debounce_cnt++;
```

```
            if (keys[i].debounce_cnt == 2) { // 消抖：连续检测 2 次有效按下
                if (keys[i].stable_state == 0) {
                    keys[i].stable_state = 1;
                    g_key_flag |= (1 << i);
                }
            }
        }
    } else {
        keys[i].debounce_cnt = 0;
        keys[i].stable_state = 0;
    }
  }
}

/* 定时器中断回调函数 */
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
  if (htim->Instance == TIM6) {
    Update_Time();
  }
  if (htim->Instance == TIM7) {
    Process_Key_Debounce();
  }
}

/* 初始化各个外设 */
void Init_Peripherals(void)
{
  MX_GPIO_Init();
  MX_FSMC_Init();
```

```
    MX_TIM14_Init();
    MX_USART1_UART_Init();
    MX_TIM6_Init();
    MX_TIM7_Init();
}

/* 启动定时器及 PWM 输出 */
void Start_Timers(void)
{
    HAL_TIM_Base_Start_IT(&htim14);
    HAL_TIM_PWM_Start(&htim14, TIM_CHANNEL_1);
    HAL_TIM_Base_Start_IT(&htim6);
    HAL_TIM_Base_Start_IT(&htim7);
}

/* 初始化 LCD 显示 */
void Init_LCD(void)
{
    drv_lcd_init();
    lcd_clear(WHITE);
    lcd_set_color(WHITE, BLACK);
}

/* 处理按键事件，根据按键调整时、分、秒 */
void Handle_Key_Event(void)
{
    if (g_key_flag) {
        if (g_key_flag & 0x01) {   // 左键：调整小时
            g_hour = (g_hour + 1) % 24;
        }
```

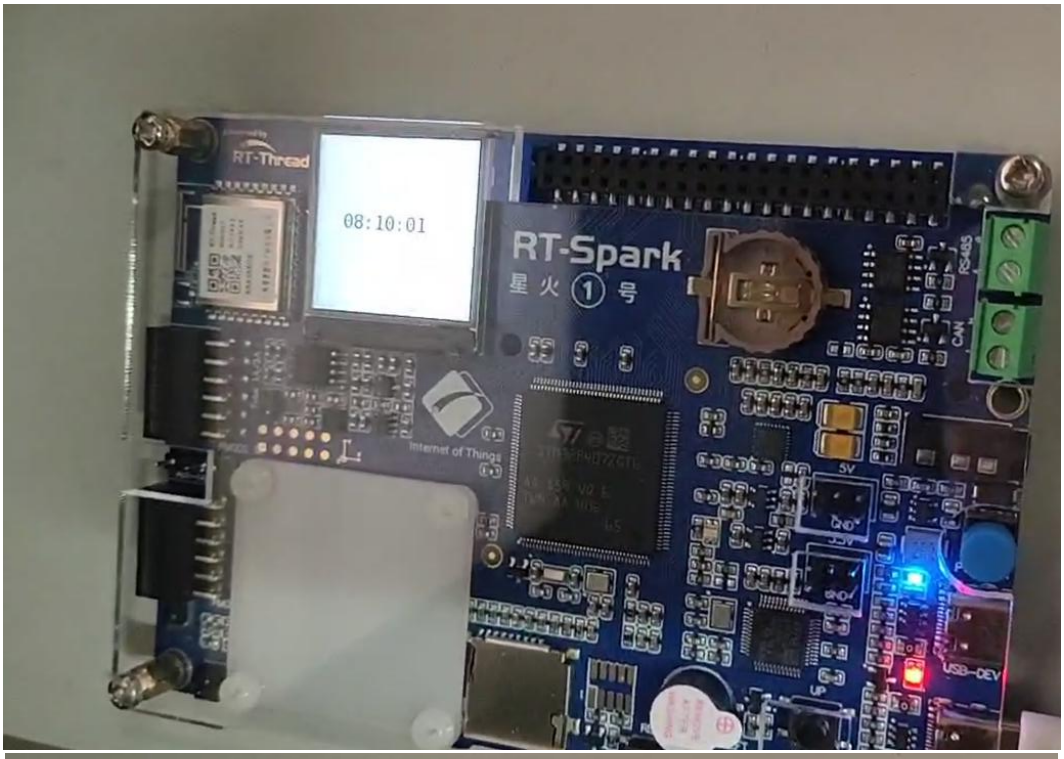
```
        if (g_key_flag & 0x02) {    // 下键：调整分钟
            g_min = (g_min + 1) % 60;
        }
        if (g_key_flag & 0x04) {    // 右键：调整秒
            g_sec = (g_sec + 1) % 60;
        }
        g_key_flag = 0;
    }
}

/* 更新 LCD 显示当前时间 */
void Update_LCD_Display(void)
{
    char time_str[20];
    sprintf(time_str, "%02d:%02d:%02d", g_hour, g_min, g_sec);
    lcd_show_string(40, 100, 32, time_str);
}

int main(void)
{
    HAL_Init();
    SystemClock_Config();
    Init_Peripherals();
    Start_Timers();
    Init_LCD();

    if (HAL_UART_Receive_IT(&huart1, (uint8_t *)&g_key_flag, 1) != HAL_OK)
    {
        Error_Handler();
    }
```

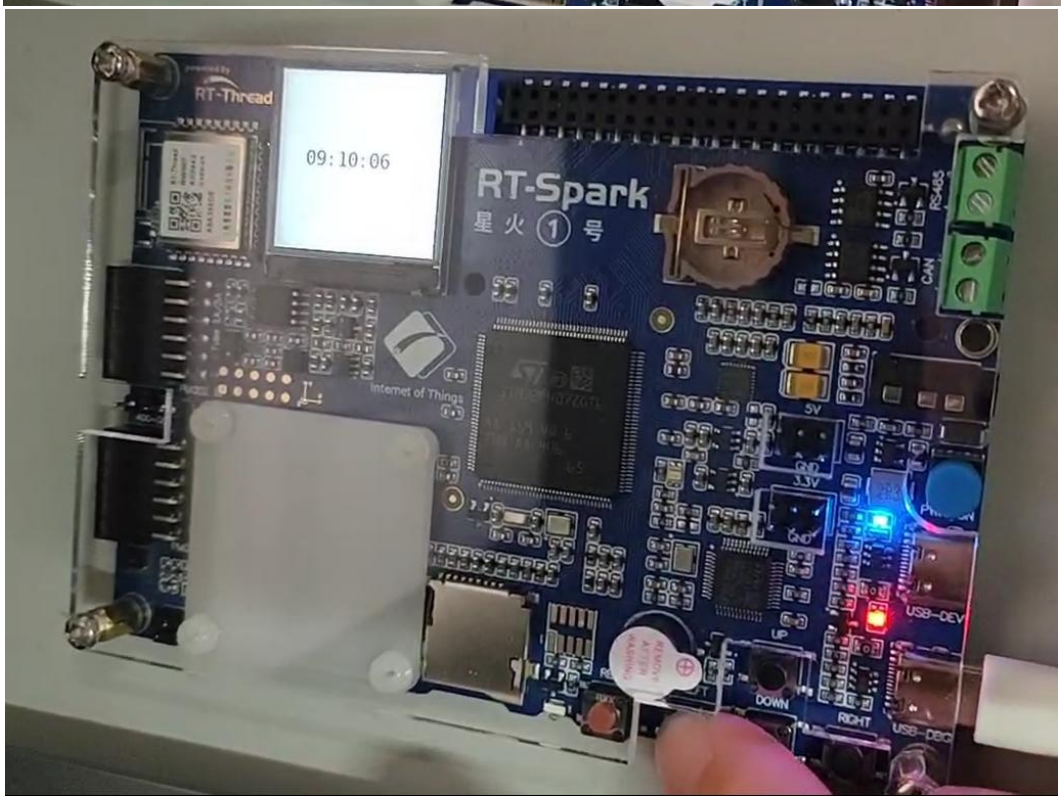
```
  while (1)
  {
    Handle_Key_Event();
    Update_LCD_Display();
    HAL_Delay(200);
  }
}

void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  __HAL_RCC_PWR_CLK_ENABLE();

__HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_S
CALE1);

  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
  RCC_OscInitStruct.HSEState = RCC_HSE_ON;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
  RCC_OscInitStruct.PLL.PLLM = 4;
  RCC_OscInitStruct.PLL.PLLN = 168;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
  RCC_OscInitStruct.PLL.PLLQ = 4;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
```

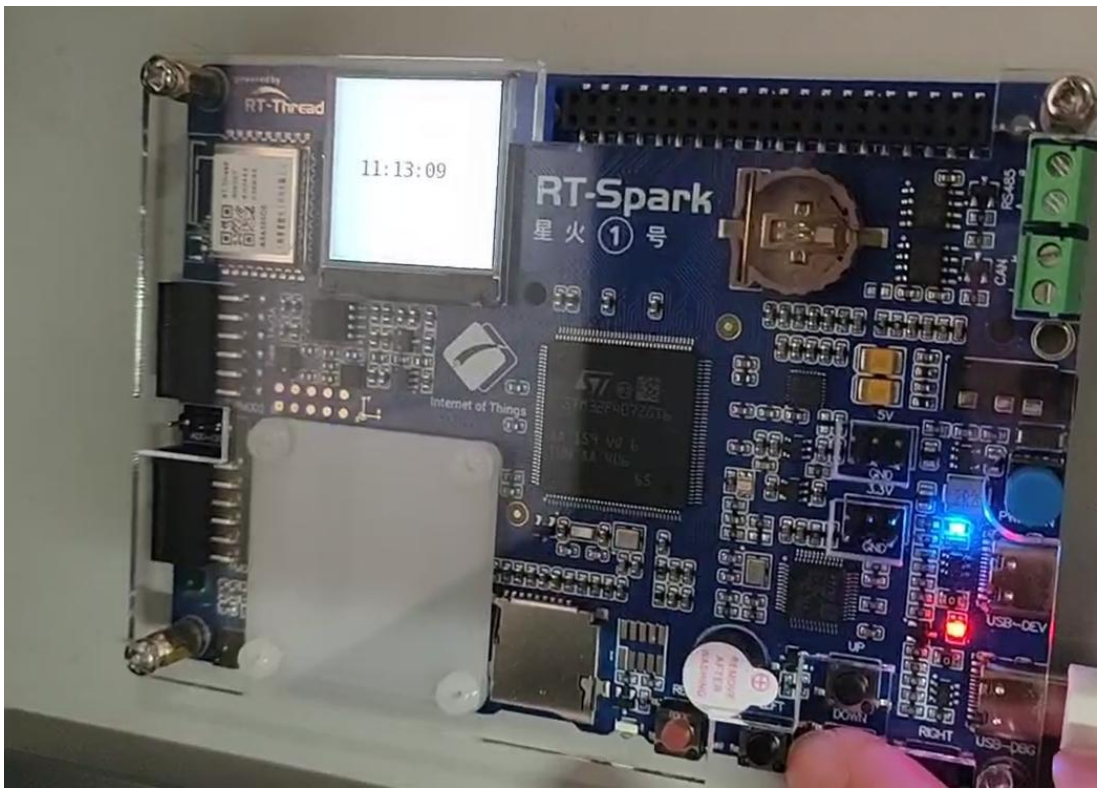
```
  }

  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK |
RCC_CLOCKTYPE_SYSCLK |
                                RCC_CLOCKTYPE_PCLK1 |
RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV4;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;
  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_5) !=
HAL_OK)
  {
    Error_Handler();
  }
}

void Error_Handler(void)
{
  __disable_irq();
  while (1)
  {
  }
}

#ifdef USE_FULL_ASSERT
void assert_failed(uint8_t *file, uint32_t line)
{
}
#endif /* USE_FULL_ASSERT */
```

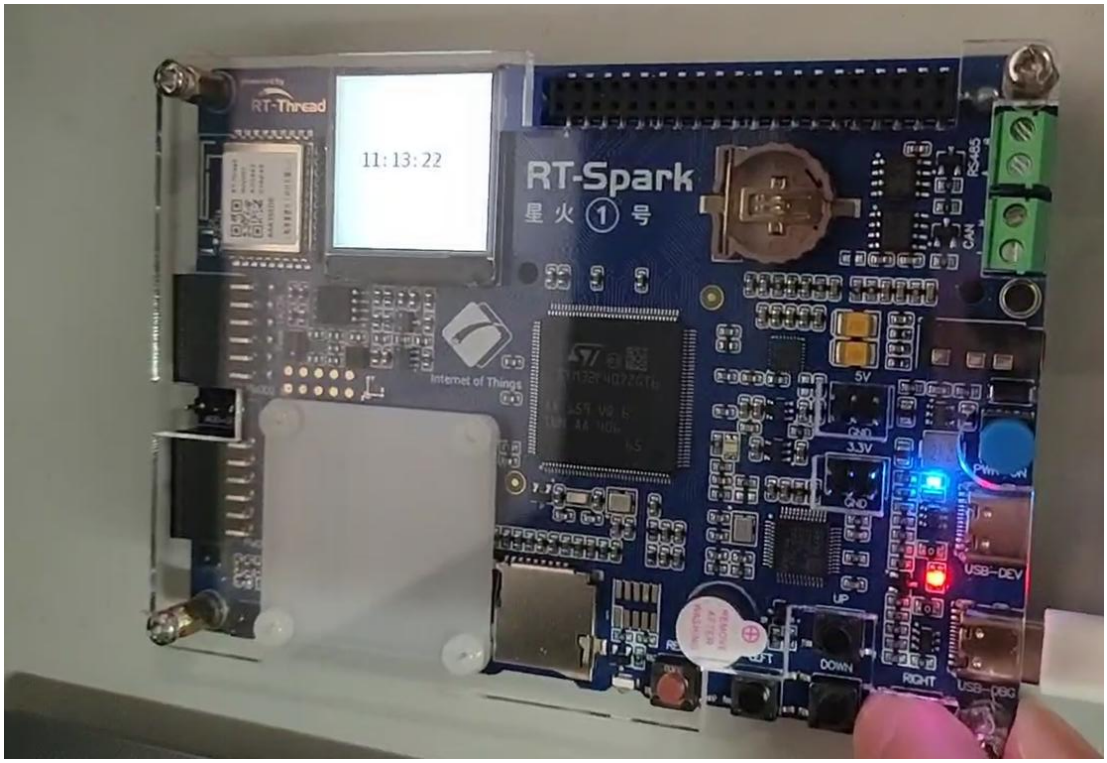
3 实验结果分析与总结

由于开发板开机和 LCD 亮起需要一点点时间，所以当 LCD 亮起时已经过了一秒，显示 08:10:01，后续我对开发板进行了相关的加时分秒的操作，由以下截图可以看到均可以成功执行，详细内容请见附带的视频：






RT-Thread
11:13:09
RT-Spark
星火①号
RS485
CAN
5V
GND
3.3V
USB-DEV
UP
DOWN
RIGHT
USB-DBG


RT-Thread
11:13:22
RT-Spark
星火①号
RS485
CAN
5V
GND
3.3V
USB-DEV
UP
DOWN
RIGHT
USB-DBG