

**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**
2. 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。

1. 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

1. **拟实现的具体功能**

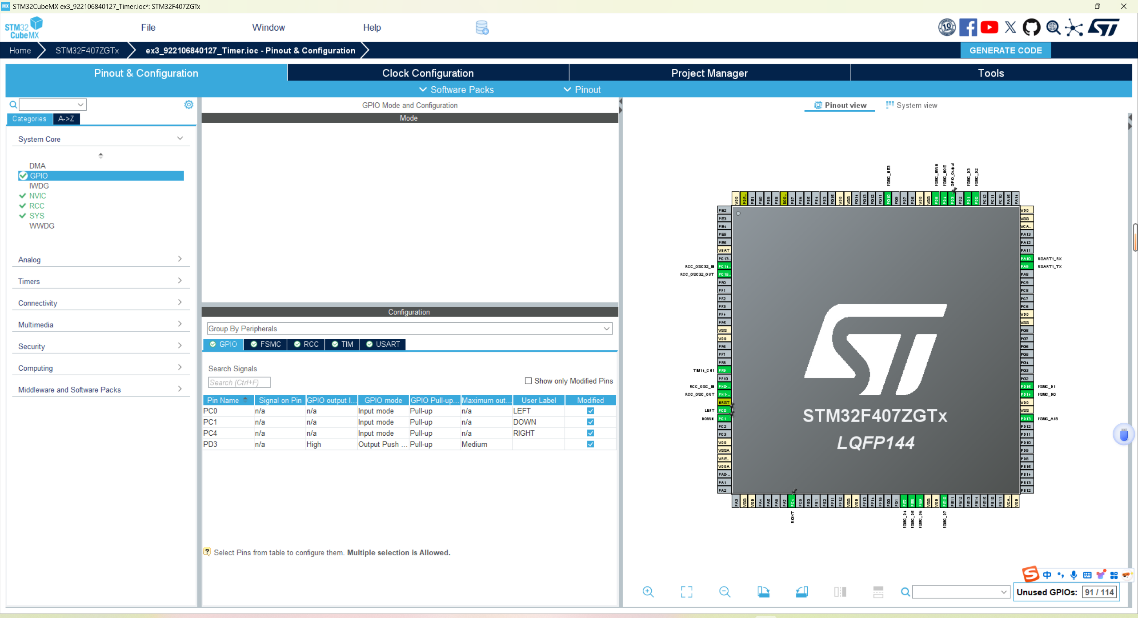
本次实验拟实现一个基于STM32的时钟显示系统，通过定时器中断自动累加并更新当前时间，然后将格式化后的时间数据显示在LCD上，同时利用三个方向键（左、下、右）实现对小时、分钟和秒数的实时调整。还加入消抖机制，避免短间隔内误触。此外，代码还通过串口输出调试信息，便于实时监控系统状态。

# 2 总体设计

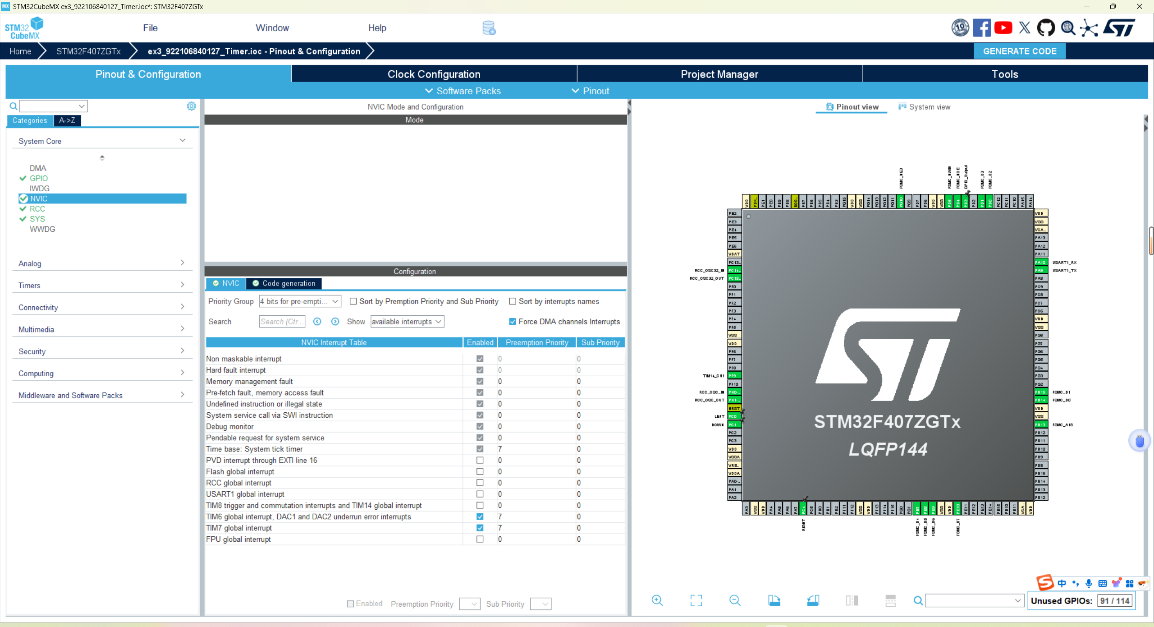
## 2.1硬件设计

**1. 硬件设计思路**

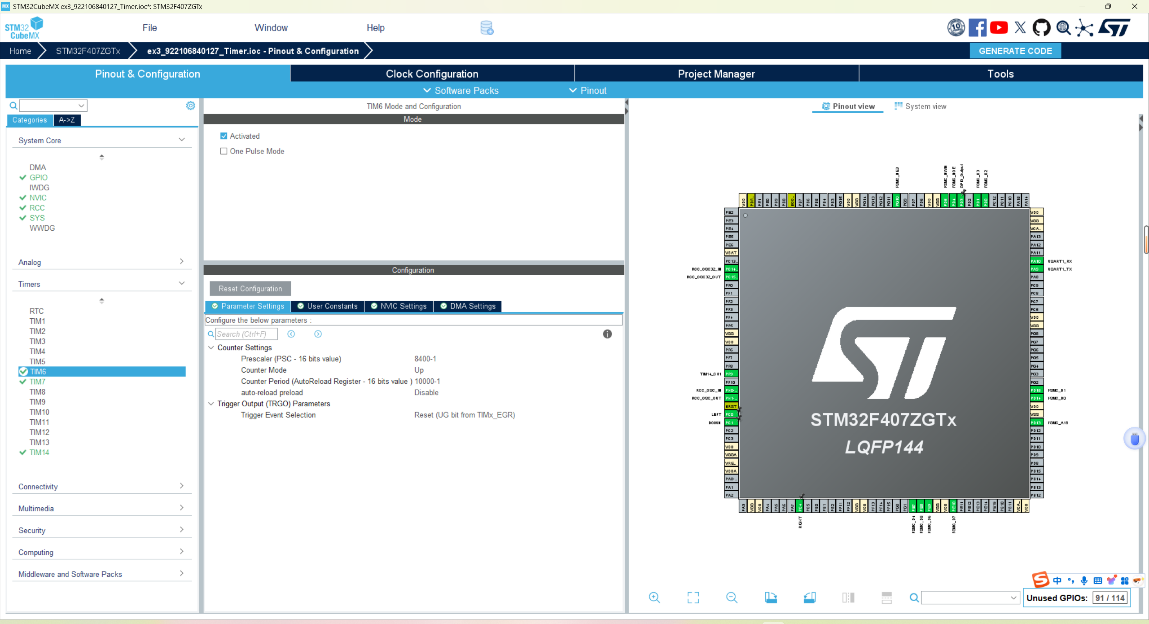
我通过查询相关开发板原理图确定了本次实验所需要的有关 GPIO 端口、 NVIC 设置、USART通信端口与TIM相关的设置，具体的代码工程配置硬件设计如下：



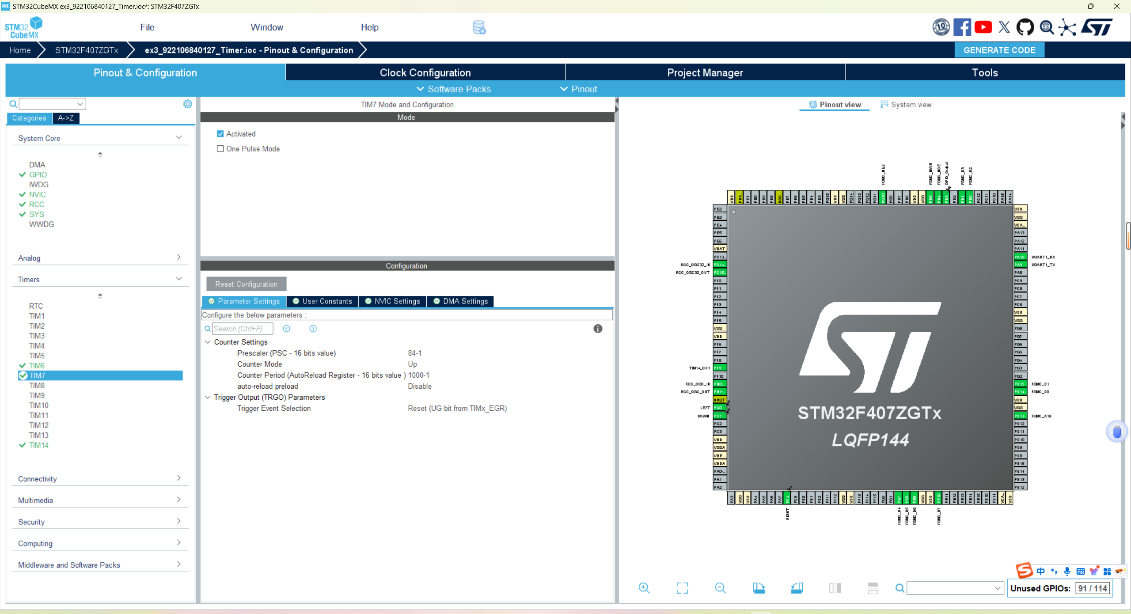
GPIO设置



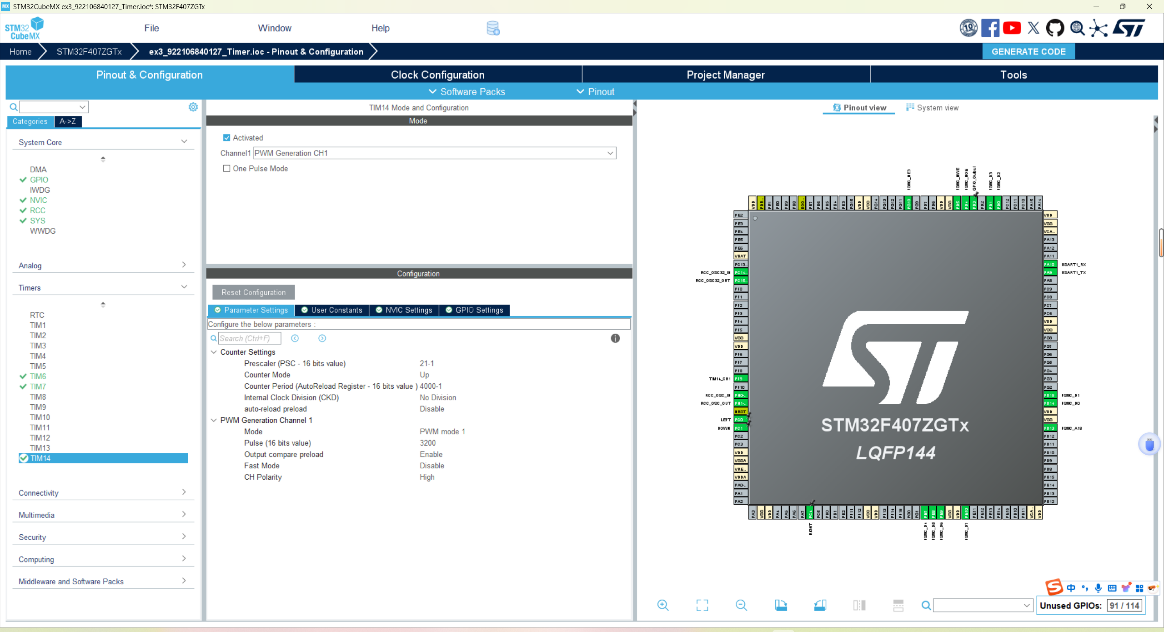
NVIC设置



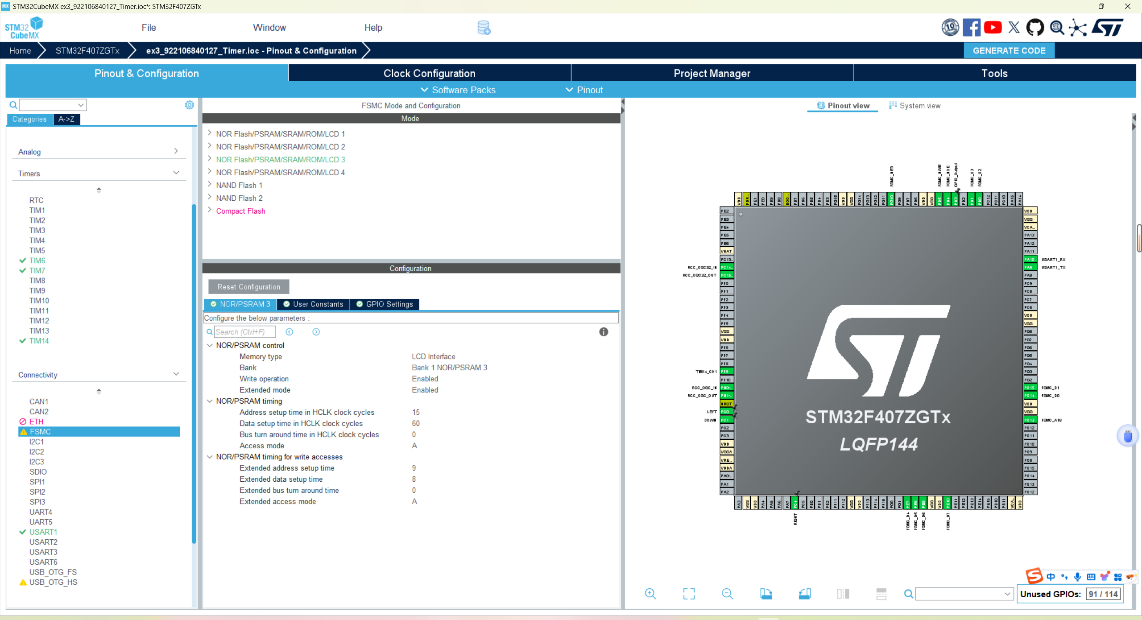
TIM6设置



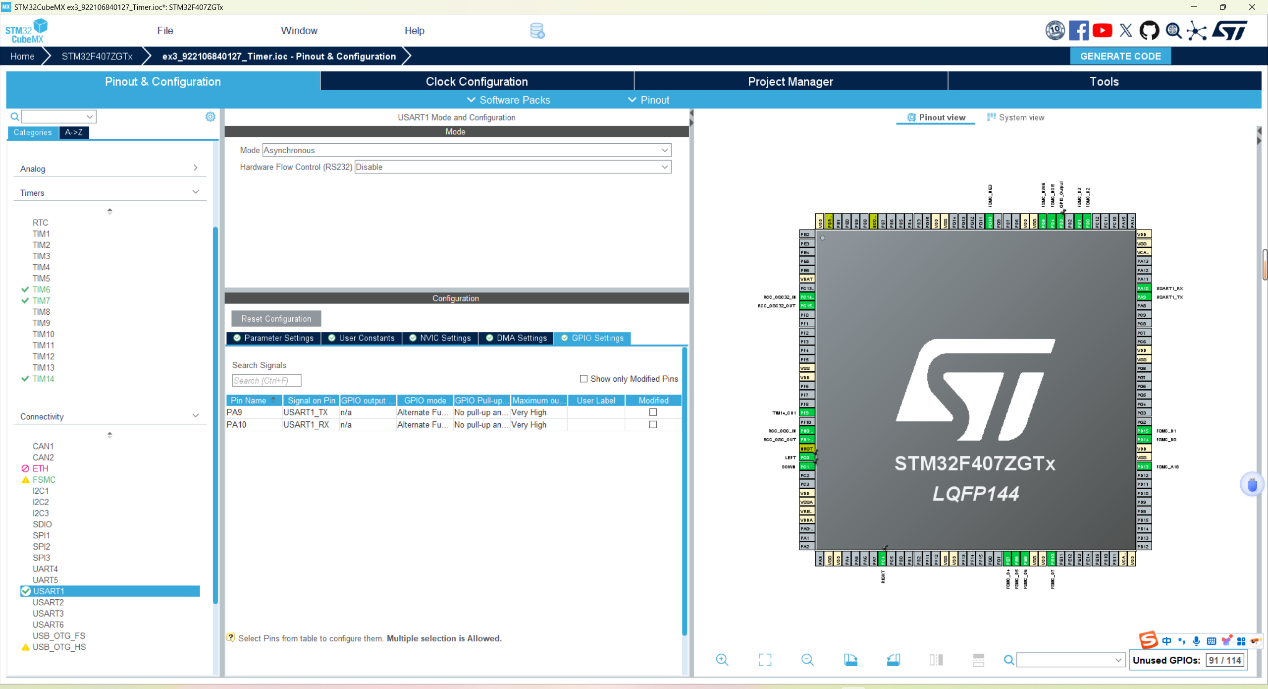
TIM7设置



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。

1. 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。

1. 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。

1. 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\_SCALE1);

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：

