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：ex4_922106840127_ PWM

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

设置 TIM2_CH4(PA3)产生 PWM 信号，然后通过 TIM9_CH1(PA2)来测量该信号频率和占空比，同时将频率和占空比显示在 LCD 上，保留一位小数。

（2） 完成要求

工程名称命名:ex4_学号_PWM,并打包成:ex4_学号_ PWM.rar 压缩文件夹

实验报告 PDF 格式:ex4_学号_PWM.pdf

2. 拟实现的具体功能

本次实验拟实现一个基于 STM32 的 PWM 测量系统，通过配置 TIM2 通道 4（PA3）输出 PWM 波形，并利用 TIM9 通道 1（PA2）对该 PWM 信号进行实时捕获，以测量其频率与占空比。实验中设置了上下限保护及按键调节功能，用户可通过左右按键对 PWM 的占空比进行步进式调节。

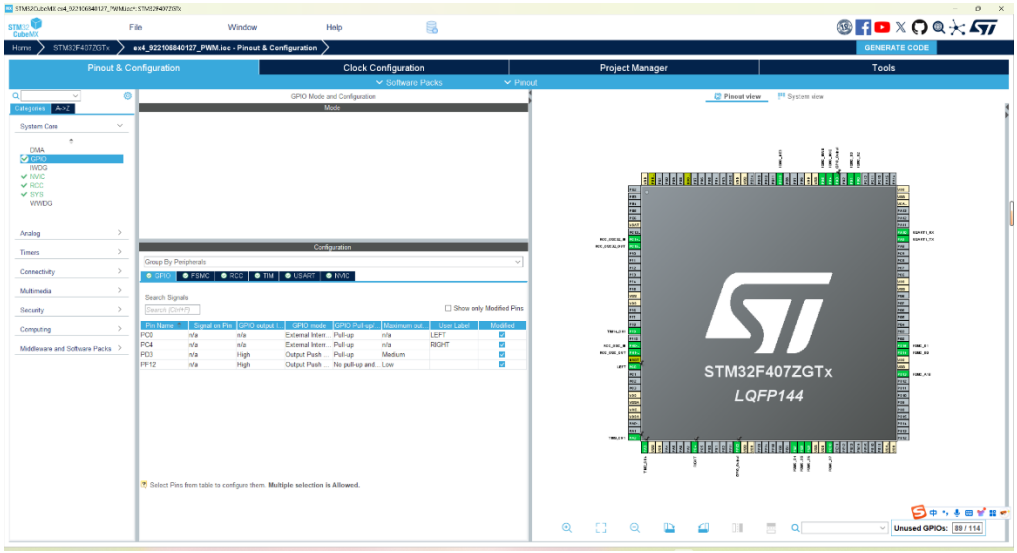
系统采用 TIM9 的输入捕获功能获取 PWM 周期与高电平时间，并据此计算出实际频率与占空比。所有测量结果均实时显示在 LCD 屏幕上，频率与占空比的数值均保留一位小数以提升数据显示精度与可读性。同时系统实现了按键消抖处理，并通过串口输出调试信息，便于监控 PWM 调节与测量的全过程。

2 总体设计

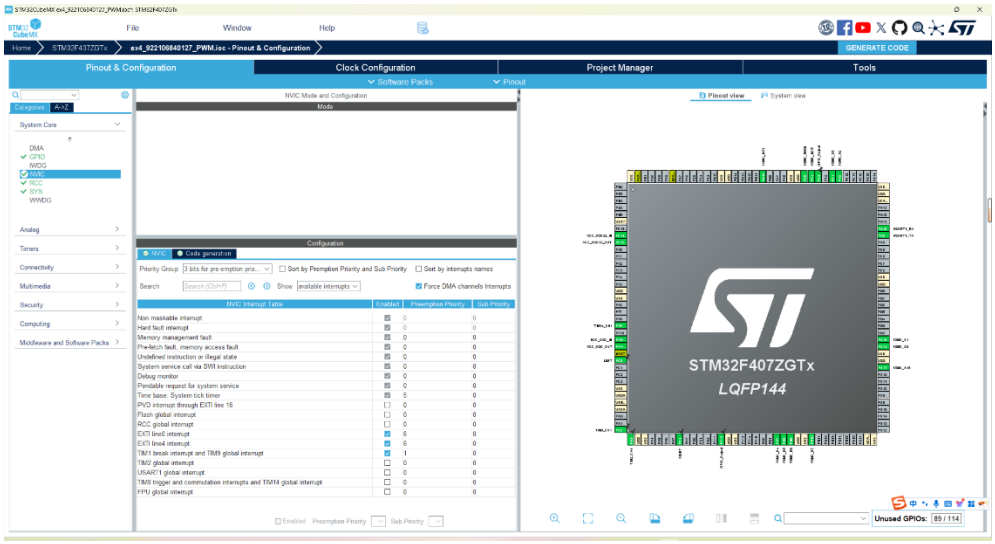
2.1 硬件设计

1. 硬件设计思路

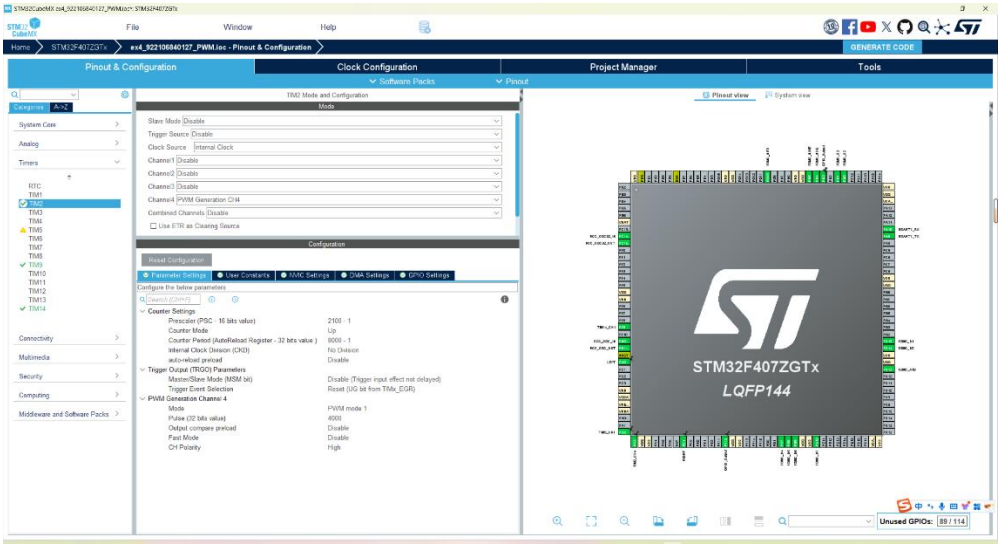
我通过查询相关开发板原理图确定了本次实验所需要的有关 GPIO 端口、NVIC 设置、USART 通信端口与 TIM 相关的设置，具体的代码工程配置硬件设计如下：



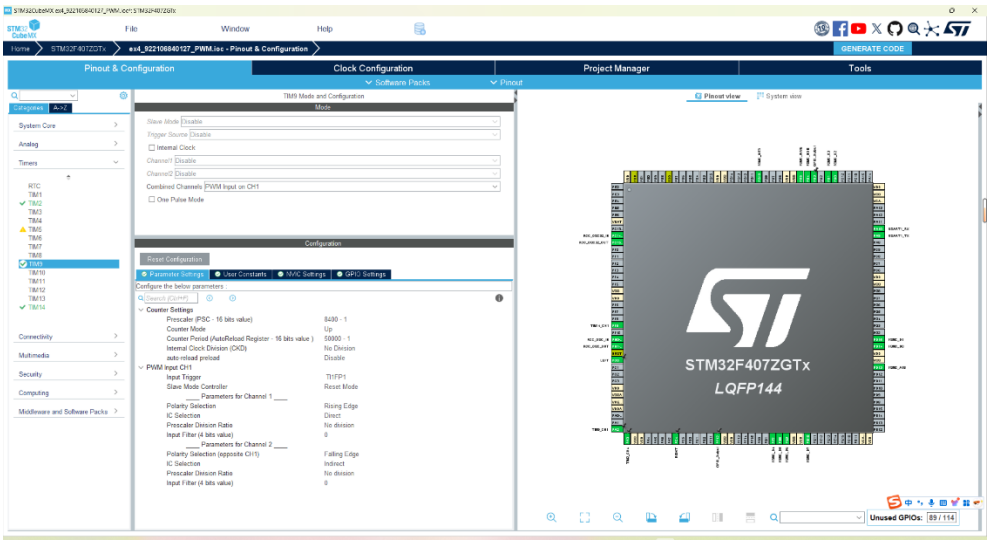
GPIO 设置



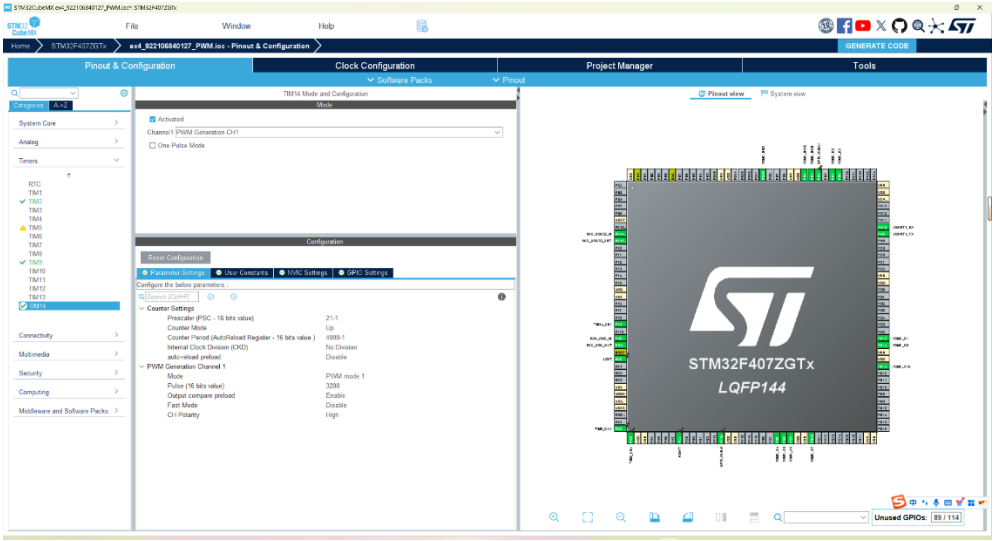
NVIC 设置



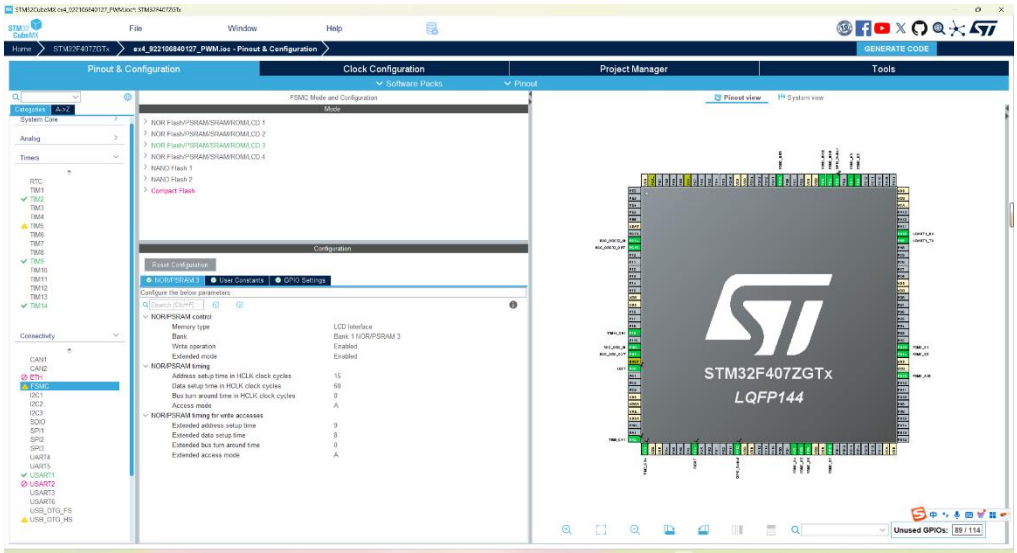
TIM2 设置



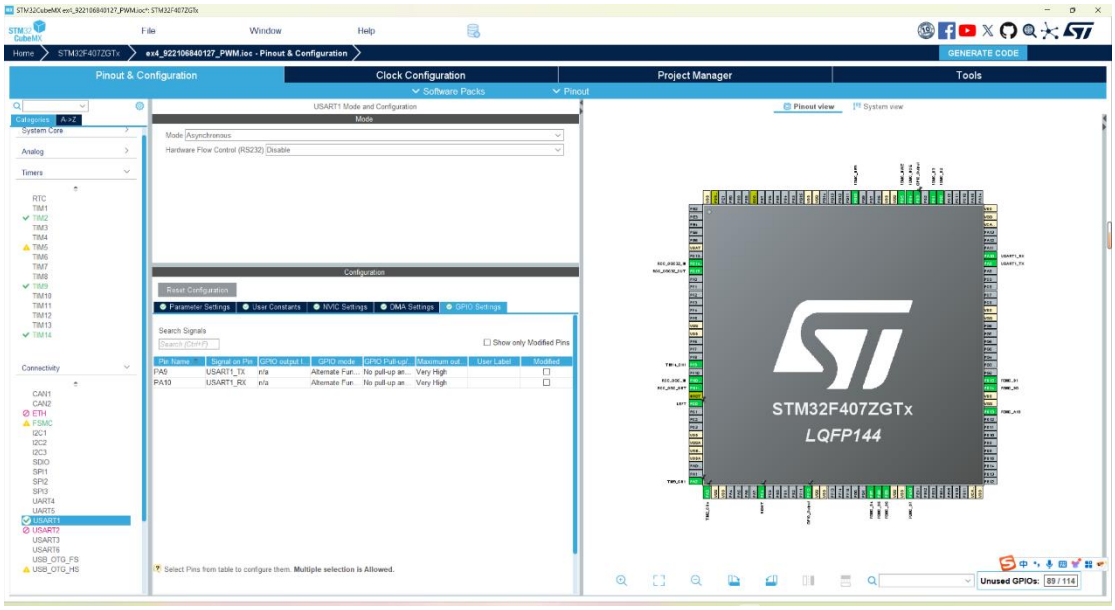
TIM9 设置



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输入捕获的 PWM 信号频率与占空比测量与显示功能，借助 STM32 单片机的 TIM2 产生 PWM 波形，并通过 TIM9 输入捕获通道对该波形的频率与占空比进行实时测量。同时，系统集成了左右方向按键功能，用于调节 PWM 的占空比，并将最新测量结果在 LCD 屏幕上实时显示，保留一位小数精度。整个软件基于 HAL 库开发，采用模块化设计思想，具有良好的结构清晰度、可读性与可维护性。具体设计内容如下：

（1） 系统初始化模块

在 main 函数中，系统首先调用 HAL_Init() 完成 HAL 库初始化，随后通过 SystemClock_Config() 配置系统时钟，确保各外设运行频率合理。随后依次初始化 GPIO、FSMC、TIM2、TIM9、TIM14 以及串口 USART1，并通过 HAL_TIM_Base_Start_IT() 启动定时器中断，同时调用 HAL_TIM_PWM_Start() 与 HAL_TIM_IC_Start_IT() 分别启动 PWM 输出与输入捕获功能。LCD 显示屏则通过 drv_lcd_init() 进行初始化，并使用 lcd_clear() 设置初始显示背景与颜色。

（2） PWM 信号测量模块

该模块核心为 UpdatePWMMeasurement() 函数，通过读取 TIM9 通道 1 与通道 2 的捕获值计算出输入 PWM 波的频率与占空比。具体而言，通道 1 用于测量完整周期，通道 2 用于测量高电平宽度，从而推导出实际频率与 DutyCycle。该函数被封装在 TIM 输入捕获中断回调函数 HAL_TIM_IC_CaptureCallback() 中，确保捕获事件一发生即自动完成数据更新。

（3） 按键调节模块

系统配置了左右按键输入，通过外部中断方式触发 HAL_GPIO_EXTI_Callback() 回调函数，并结合 AdjustPWMDutyCycle() 函数判断按键类型与当前 PWM 占空比，进行增减调节。该过程包括简单的按键消抖处理，通过延时避免误触，提高系统稳定性。PWM 占空比调节通过修改 TIM2_CH4 的比较值 __HAL_TIM_SetCompare() 实现，占空比范围限定在 400～7600 之间。

（4） LCD 显示模块

该模块通过 DisplayPWMInfo() 函数将当前测得的 PWM 频率与占空比值格式化为字符串，并调用 lcd_show_string() 显示在 LCD 指定位置。特别地，为避免格式误差与小数位数不一致问题，还对浮点数显示精度进行统一处理，确保结果精确至一位小数，提升用户可读性与观感体验。

总体而言，本软件基于 STM32 的定时器输入捕获与 PWM 输出机制，实现了一个结构清晰、交互友好且具有实用性的 PWM 测量与显示系统，为后续扩展如多通道测量、曲线绘制等功能奠定了良好基础。

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 STM32 单片机进行基础设置与复位配置；
- **系统时钟配置**：通过 SystemClock_Config() 设置系统主频及外设时钟，确保各模块正常运行；
- **初始化外设资源**：包括 GPIO、FSMC、TIM2、TIM9、TIM14 以及串口 USART1，分别通过 MX_XXX_Init() 函数完成初始化，保障 PWM 输出、信号捕获与串口调试功能正常运行；
- **启动定时器功能**：调用 HAL_TIM_Base_Start_IT() 启动 TIM14 和 TIM2 定时器中断，并通过 HAL_TIM_PWM_Start() 与 HAL_TIM_IC_Start_IT() 分别启用 TIM2 的 PWM 输出（通道 4）与 TIM9 的输入捕获（通道 1 和 2）；
- **LCD 显示屏初始化**：调用 drv_lcd_init() 初始化液晶显示器，通过 lcd_clear() 清除屏幕并设置背景色与前景色，准备显示初始信息。

B. 主循环结构

初始化完成 → 进入主循环

系统初始化完成后，程序进入主循环，不断执行以下操作：

- **读取并显示 PWM 测量信息**：调用 DisplayPWMInfo() 获取当前测得的频率与占空比，并将格式化后的结果显示在 LCD 指定区域，显示精度保留一位小数；
- **周期性延时**：调用 HAL_Delay(200) 每 200ms 更新一次显示内容，提高刷新频率与用户体验。

C. 输入捕获中断处理流程（TIM9）

PWM 信号变化 → 进入 HAL_TIM_IC_CaptureCallback()

当 TIM9 的输入捕获通道捕获到新边沿信号后，自动触发 HAL_TIM_IC_CaptureCallback() 中断回调函数：

- **判断中断来源**：若当前捕获事件来自 TIM9 的通道 1，说明有新的 PWM 周期信号产生；
- **更新频率与占空比**：调用 UpdatePWMMeasurement() 读取 TIM9 的通道 1 和通道 2 的捕获值，分别代表周期和高电平宽度，结合系统时钟计算出频率与占空比。

D. 外部中断触发流程（按键）

用户按下按键 → 进入 HAL_GPIO_EXTI_Callback()

当用户按下左键或右键时，触发外部中断回调函数 HAL_GPIO_EXTI_Callback()：

- **消抖处理**：调用 HAL_Delay(10) 等待 10ms 进行简单按键消抖；
- **占空比调节逻辑**：调用 AdjustPWMDutyCycle() 根据按下的是左键或右键，对变量 duty 进行增减，并通过 __HAL_TIM_SetCompare() 修改 TIM2_CH4 的 PWM 占空比，控制输出波形形态。

E. 显示刷新流程

周期性更新 LCD → 调用 DisplayPWMInfo()

主循环中调用 DisplayPWMInfo() 函数，将当前 PWM 的频率与占空比值格式化为字符串（如 "Frequency: 123.4"），调用 lcd_show_string() 显示在屏幕顶部区域，保证用户实时获取信号测量信息。

F. 循环机制

中断服务 → 返回主循环

每当中断（输入捕获或按键）处理完成后，系统退出对应的回调函数，重新回到主循环，继续执行显示与检测逻辑，实现对 PWM 信号的动态测量、显示与交互调整，构建出一个完整、稳定的 PWM 测量与控制系统。

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"

__IO float IC2ValuePWM = 0;
__IO float DutyCyclePWM = 0;
__IO float FrequencyPWM = 0;
void SystemClock_Config(void);
float FormatTo1Decimal(float value);
float duty = 4000;

// 重定向 printf 到串口
int fputc(int ch, FILE *f) {
    HAL_UART_Transmit(&huart1, (uint8_t *)&ch, 1, 1000);
    return ch;
}

// 更新 PWM 测量值：计算占空比和频率
void UpdatePWMMeasurement(TIM_HandleTypeDef *htim) {
    IC2ValuePWM = HAL_TIM_ReadCapturedValue(htim, TIM_CHANNEL_1); //
获取周期
    if (IC2ValuePWM != 0) {
        DutyCyclePWM = ((HAL_TIM_ReadCapturedValue(htim,
TIM_CHANNEL_2)) * 100) / IC2ValuePWM; // 占空比 = 高电平时间 / 周期
        FrequencyPWM = (HAL_RCC_GetPCLK2Freq() * 2 / (TIM9->PSC + 1)) /
IC2ValuePWM; // 频率 = 定时器时钟 / 周期
    } else {
        DutyCyclePWM = 0;
        FrequencyPWM = 0;
    }
}
```

```
// 输入捕获中断回调函数，触发更新 PWM 测量值
void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim) {
    if (htim->Instance == TIM9 && htim->Channel ==
HAL_TIM_ACTIVE_CHANNEL_1) {
        UpdatePWMMeasurement(htim);
    }
}

// 调整 PWM 占空比（通过左右按键控制）
void AdjustPWMDutyCycle(uint16_t GPIO_Pin) {
    if (HAL_GPIO_ReadPin(GPIOC, GPIO_Pin) == GPIO_PIN_RESET) {
        if (GPIO_Pin == LEFT_Pin && duty > 400) {
            duty -= 400;
            __HAL_TIM_SetCompare(&htim2, TIM_CHANNEL_4, duty);
        }
        if (GPIO_Pin == RIGHT_Pin && duty < 7600) {
            duty += 400;
            __HAL_TIM_SetCompare(&htim2, TIM_CHANNEL_4, duty);
        }
    }
}

// 外部中断回调函数：检测按键并调整 PWM 占空比
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin) {
    HAL_Delay(20); // 简单消抖
    AdjustPWMDutyCycle(GPIO_Pin);
}

// 初始化所有外设
void InitPeripherals(void) {
```

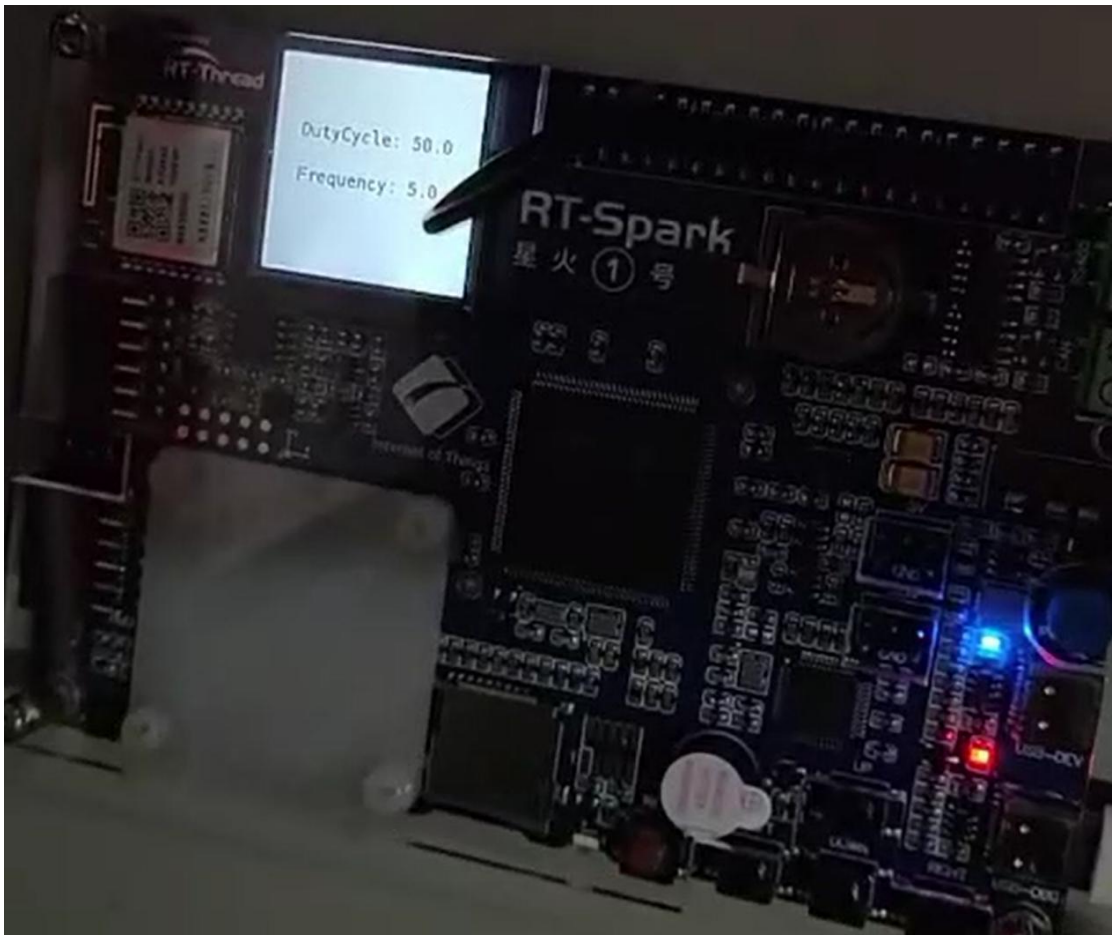
```
    MX_GPIO_Init();
    MX_FSMC_Init();
    MX_TIM14_Init();
    MX_USART1_UART_Init();
    MX_TIM2_Init();
    MX_TIM9_Init();

    // 启动定时器和 PWM
    HAL_TIM_Base_Start_IT(&htim14);
    HAL_TIM_PWM_Start(&htim14, TIM_CHANNEL_1);
    HAL_TIM_Base_Start_IT(&htim2);
    HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_4);

    // 启动输入捕获
    HAL_TIM_IC_Start_IT(&htim9, TIM_CHANNEL_1);
    HAL_TIM_IC_Start_IT(&htim9, TIM_CHANNEL_2);

    drv_lcd_init(); // 初始化 LCD
}

//时刻检查是否为一位小数
float FormatTo1Decimal(float value) {
     return (float)((int)(value * 10 + 0.5)) / 10.0;
}


// 显示 PWM 占空比和频率信息
void DisplayPWMInfo(void) {
    char DutyCycle[20];
    char Frequency[20];
```

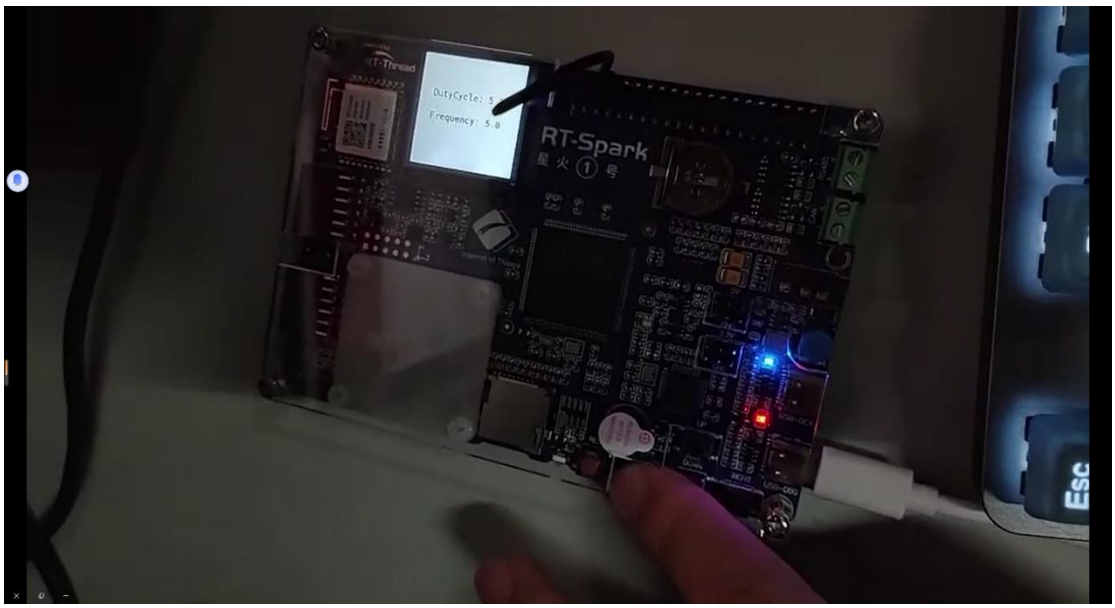
```
    sprintf(DutyCycle, "DutyCycle: %.1f", DutyCyclePWM);     // 保留 1 位小数
    sprintf(Frequency, "Frequency: %.1f", FrequencyPWM);
    lcd_show_string(30, 70, 24, DutyCycle);
    lcd_show_string(30, 120, 24, Frequency);
}

int main(void) {
    HAL_Init();                     // 初始化 HAL 库
    SystemClock_Config();       // 配置系统时钟
    InitPeripherals();            // 初始化所有外设

    lcd_clear(WHITE);
    lcd_set_color(WHITE, BLACK); // 设置前景背景色

    while (1) {
        DisplayPWMInfo();       // 显示 PWM 信息
        HAL_Delay(200);          // 延时 200ms
    }
}

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_VOLTAG
E_SCALE1);

    RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
    RCC_OscInitStruct.HSEState = RCC_HSE_ON;
```

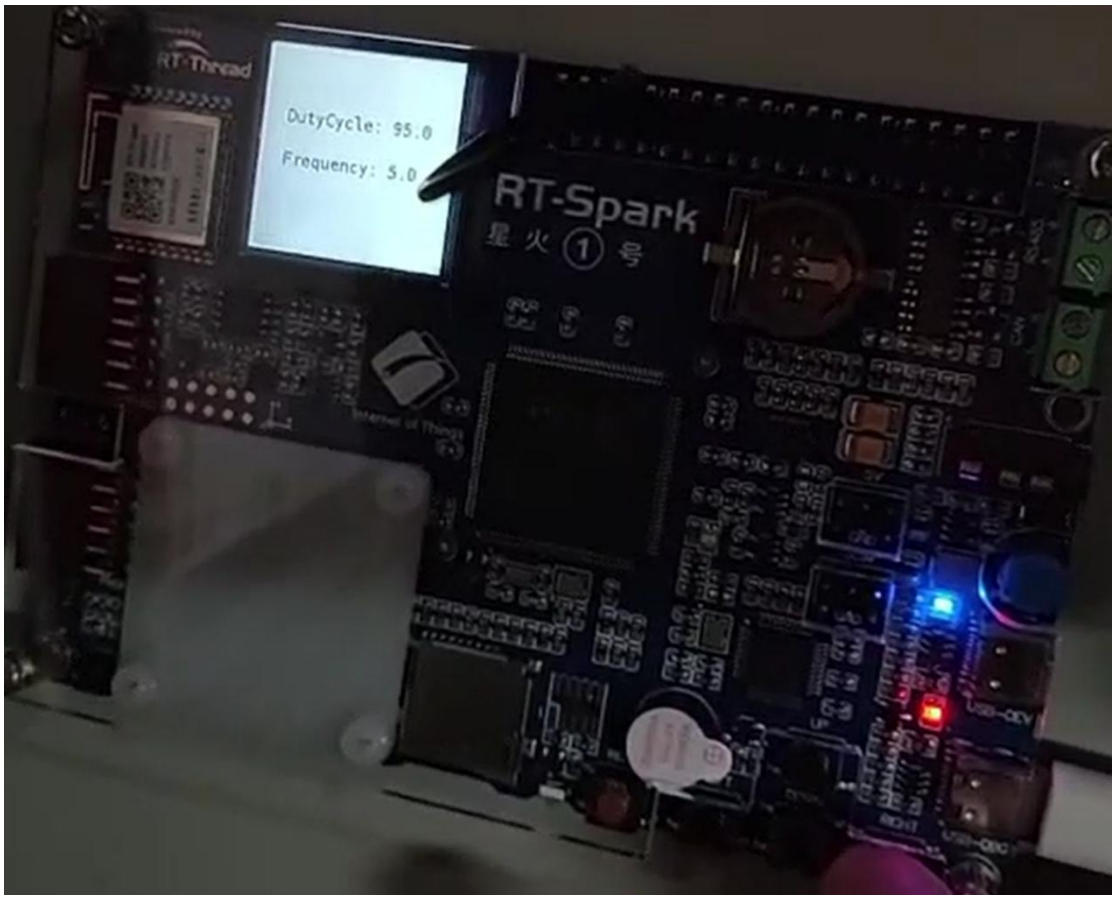
```
    RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
    RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
    RCC_OscInitStruct.PLL.PLLM = 4;
    RCC_OscInitStruct.PLL.PLLN = 168;
    RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
    RCC_OscInitStruct.PLL.PLLQ = 4;
    if (HAL_RCC_OscConfig(&RCC_OscInitStruct)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

    RCC_ClkInitStruct.ClockType =
RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK|RCC_CLOCKTYPE_P
CLK1|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) {}
}
```

3 实验结果分析与总结

将本项目代码工程通过编译，上板验证，可以看到占空比和频率初始化为 50.0 与 5.0，完成实验要求，接着我按下左键进行占空比的减少，可以看到最低减到了 5.0，再按下右键增加占空比，可以看到最高可以加到 95.0，完成实验要求。




RT-Spark
星火①号


RT-Thread
DutyCycle: 95.0
Frequency: 5.0
RT-Spark
星火①号