方式1：普通延时（纯软件循环）

delay_sw.h

c 复制

```c
#ifndef __DELAY_SW_H
#define __DELAY_SW_H

#include "stdint.h"

void DelaySW_Init(void);
void DelaySW_Us(uint32_t us);
void DelaySW_Ms(uint32_t ms);

#endif
```

delay_sw.c

c 复制

```c
#include "delay_sw.h"
#include "stm32f10x.h"

// 微秒延时系数（72MHz 下，需根据编译优化等级实测调整）
#define DELAYSW_US_FACTOR  18
#define DELAYSW_MS_FACTOR  18000

/**
 * @brief  初始化软件延时（空函数，保持接口统一）
 * @retval None
 */
void DelaySW_Init(void) {
    // 无需初始化，仅保持接口一致性
}

/**
 * @brief  微秒级软件延时（阻塞式）
 * @param  us: 延时的微秒数
 * @retval None
 * @note   精度低，受编译优化影响大，仅适合粗略延时
 */
void DelaySW_Us(uint32_t us) {
    for(uint32_t i = 0; i < us * DELAYSW_US_FACTOR; i++) {
        __NOP();
    }
}

/**
 * @brief  毫秒级软件延时（阻塞式）
 * @param  ms: 延时的毫秒数
 * @retval None
 * @note   通过循环调用微秒延时实现
 */
void DelaySW_Ms(uint32_t ms) {
    for(uint32_t i = 0; i < ms; i++) {
        DelaySW_Us(1000);  // 延时1毫秒
    }
}
```

方式2：SysTick中断延时（推荐）

delay_systick_irq.h

c 复制

```c
#ifndef __DELAY_SYSTICK_IRQ_H
#define __DELAY_SYSTICK_IRQ_H

#include "stdint.h"

void DelayIRQ_Init(void);
uint32_t DelayIRQ_GetTime(void);
void DelayIRQ_Ms(uint32_t ms);
void DelayIRQ_Us(uint32_t us);

#endif
```

delay_systick_irq.c

c 复制

```c
#include "delay_systick_irq.h"
#include "stm32f10x.h"

// 全局系统tick计数器（在SysTick中断中递增）
volatile uint32_t g_system_tick = 0;

/**
 * @brief  SysTick中断服务函数，每1ms触发一次
 * @note   需在stm32f10x_it.c中注释掉同名的定义函数
 */
void SysTick_Handler(void) {
    g_system_tick++;
}

/**
 * @brief  初始化SysTick中断延时
 * @retval None
 * @note   配置为1ms中断一次
 */
void DelayIRQ_Init(void) {
    if(SysTick_Config(SystemCoreClock / 1000)) {
        while(1); // 配置失败，死循环报警
    }
}

/**
 * @brief  获取系统运行时间（毫秒）
 * @retval 系统自动初始化后的毫秒数
 */
uint32_t DelayIRQ_GetTime(void) {
    return g_system_tick;
}

/**
 * @brief  毫秒级精确延时（阻塞式）
 * @param  ms: 延时的毫秒数
 * @retval None
 * @note   基于全局计数器，精确度高
 */
void DelayIRQ_Ms(uint32_t ms) {
    uint32_t start = g_system_tick;
    while((g_system_tick - start) < ms);
}

/**
 * @brief  微秒级延时（阻塞式，轮询方式）
 * @param  us: 延时的微秒数
 * @retval None
 * @note   72MHz下系数18需根据编译优化实测校准
 */
void DelayIRQ_Us(uint32_t us) {
    for(uint32_t i = 0; i < us * 18; i++) {
        __NOP();
    }
}
```

方式3：SysTick轮询延时（无中断）

delay_systick_poll.h

c 复制

```c
#ifndef __DELAY_SYSTICK_POLL_H
#define __DELAY_SYSTICK_POLL_H

#include "stdint.h"

void DelayPoll_Init(void);
void DelayPoll_Ms(uint32_t ms);
void DelayPoll_Us(uint32_t us);

#endif
```

delay_systick_poll.c

c 复制

```c
#include "delay_systick_poll.h"
#include "stm32f10x.h"

/**
 * @brief  初始化SysTick轮询延时
 * @retval None
 * @note   开启SysTick但不使能中断，通过查询COUNTFLAG标志
 */
void DelayPoll_Init(void) {
    // 配置SysTick重装载值（最大24位）
    SysTick->LOAD = 0xFFFFFF;  // 设为最大值

    // 选择时钟源为HCLK(72MHz)，使能SysTick，不使能中断
    SysTick->CTRL = SysTick_CTRL_ENABLE_Msk | SysTick_CTRL_CLKSOURCE_Msk;

    // 当前值清零
    SysTick->VAL = 0;
}

/**
 * @brief  微秒级轮询延时（阻塞式）
 * @param  us: 延时的微秒数
 * @retval None
 * @note   通过查询SysTick COUNTFLAG标志实现，精确度高
 */
void DelayPoll_Us(uint32_t us) {
    uint32_t start = SysTick->VAL;
    uint32_t ticks = us * 72; // 72MHz 下每个tick约13.89ns

    while(ticks > 0) {
        if(SysTick->CTRL & SysTick_CTRL_COUNTFLAG_Msk) {
            ticks -= 0xFFFFFF; // 24位计数器溢出
        } else {
            uint32_t current = SysTick->VAL;
            if(current <= start) {
                ticks -= (start - current);
            } else {
                ticks -= (start + (0xFFFFFF - current));
            }
            start = current;
        }
    }
}

/**
 * @brief  毫秒级轮询延时（阻塞式）
 * @param  ms: 延时的毫秒数
 * @retval None
 */
void DelayPoll_Ms(uint32_t ms) {
    for(uint32_t i = 0; i < ms; i++) {
        DelayPoll_Us(1000);  // 延时1毫秒
    }
}
```

使用方法与注意事项

1. **普通延时**：无需配置，直接调用，但精度最低
2. **中断延时**：
   - 调用 `DelayIRQ_Init()` 初始化
   - **关键**：在 `stm32f10x_it.c` 中注释掉 `SysTick_Handler` 的弱定义
   - 可获取系统时间，适合多任务场景
3. **轮询延时**：
   - 调用 `DelayPoll_Init()` 初始化
   - 不占用中断资源，纯硬件轮询
   - 阻塞CPU但比软件延时精确

工程配置：在 `stm32f10x_it.c` 和 `stm32f10x_it.h` 中注释掉 `SysTick_Handler` 相关代码即可兼容三种方式。

#include "delay_systick_irq.h"