

浙江大学



嵌入式系统·实验报告

(本科)

课程名称: 嵌入式系统

学院: 机械工程学院

专业: 机械工程

学号: 3230103743

姓名: 徐屹寒

指导教师: _____

年 月 日

目录

实验一 STM32 项目创建实验

实验二 基于寄存器的跑马灯实验

实验三 基于库函数的跑马灯实验

实验四 按键输入实验

实验五 外部中断实验

实验六 串口通讯实验

实验七 定时器中断实验

实验八 定时器计时实验

实验一 STM32 项目创建实验

一、实验目的

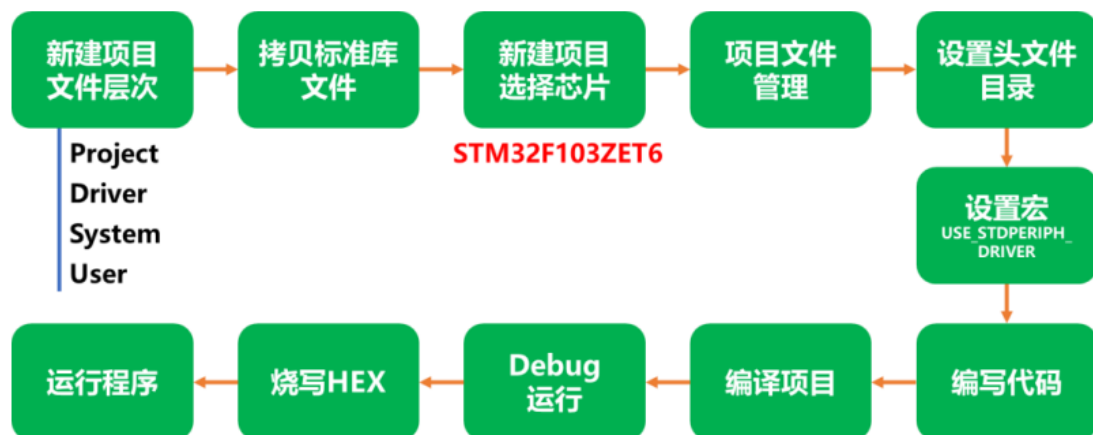
1. 掌握基于 MDK5 的两种 STM32 项目创建方法。
2. 掌握 MDK5 下程序配置、编译、调试、烧写等方法。

二、实验内容

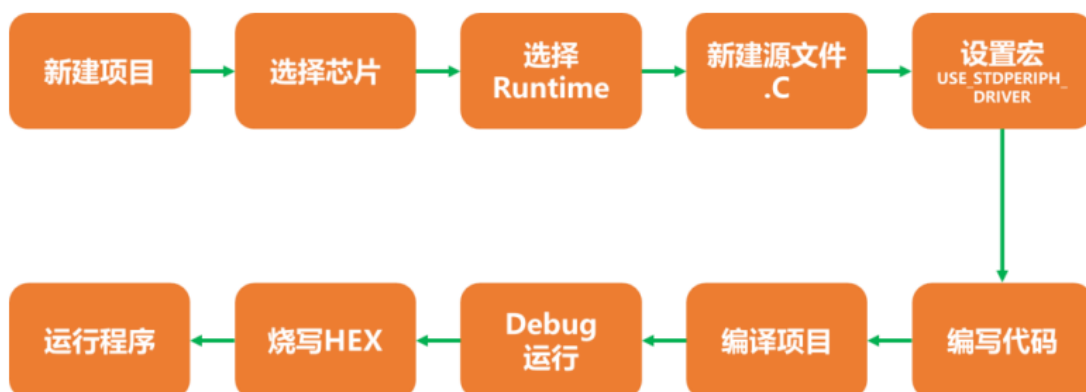
1. 基于标准库的 STM32 项目创建
2. 基于 MDK5 的 STM32 项目创建

三、实验步骤

1. 基于标准库的 STM32 项目创建



2. 基于 MDK5 的 STM32 项目创建



四、实验结果

```
#include "stm32f10x.h"
```

```

void delay_ms(int ms)
{
    int i;
    while(ms-->0)
    {
        i=7500;
        while(i-->0);
    }
}

int main()
{
    GPIO_InitTypeDef initStructure;
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOE,ENABLE);
    initStructure.GPIO_Pin=GPIO_Pin_5;
    initStructure.GPIO_Speed=GPIO_Speed_50MHz;
    initStructure.GPIO_Mode=GPIO_Mode_Out_PP;
    GPIO_Init(GPIOE,&initStructure);
    GPIO_ResetBits(GPIOE,GPIO_Pin_5);

    while(1)
    {
        GPIO_SetBits(GPIOE,GPIO_Pin_5);
        delay_ms(1000);
        GPIO_ResetBits(GPIOE,GPIO_Pin_5);
        delay_ms(1000);
    }
    return 1;
}

```

五、思考题

1. 在基于 MDK5 的 STM32 项目中，如何添加或删除 Runtime？

添加 Runtime

- 1) 点击菜单栏中的 “Project”，然后选择 “Manage” – “Run-Time Environment”。
- 2) 在弹出的对话框中可以看到一系列可用的软件包和组件。根据需要勾选相应的 Runtime 组件，例如 FreeRTOS 等。
- 3) 勾选组件后，可能会弹出配置界面，可以在这里设置组件的相关参数，如 FreeRTOS

的堆栈大小、任务优先级等。

- 4) 配置完成后, 点击 “OK” 按钮, 所选的 Runtime 组件将被添加到项目中, 并且相关的头文件和源文件会自动导入。

删除 Runtime

- 1) 点击菜单栏中的 “Project”, 然后选择 “Manage Run-Time Environment”。
 - 2) 在弹出的对话框中, 找到之前添加的 Runtime 组件, 取消勾选它。点击 “OK” 按钮, 该 Runtime 组件将从项目中移除, 相关的文件也会被清理。
2. 在图 1.8 中, 如果不去配置 DLL, 结果会怎样? 为什么?

Dialog DLL 是 Keil 与硬件调试器或软件仿真器通信的接口。它确保 Keil 能够正确地与目标硬件或模拟器交互, 支持调试功能(如下载代码、设置断点、读取寄存器等)。如果不去配置, 调试功能会不可用。

3. 在图 1.17 中, 如何修改 Project 下的 Source Group 1 名称?

在项目窗口中找到 Source Group 1, 单击选中, 再次单击 Source Group 1 的名称, 稍等片刻后, 名称会进入可编辑状态, 此时可以直接输入新的名称

实验二 基于寄存器的跑马灯实验

一、实验目的

1. 掌握 STM32 编程 IO 配置流程。
2. 掌握 STM32 相关寄存器文档资料检索方法。
3. 了解 GPIO 的几种输入/输出工作模式和寄存器配置方法。

二、实验内容

使用基于寄存器编程的方式，结合延时程序，使开发板上的 LED0 和 LED1 灯间隔 0.5 秒钟交替闪烁，即：LED0 亮，LED1 灭，0.5 秒钟后 LED0 灭，LED1 亮，再过 0.5 秒钟 LED0 亮，LED1 灭……如此往复。

三、实验步骤



四、实验结果

```
#include "stm32f10x.h"

void delay_ms(int ms)
{
    int i;
    while(ms-->0)
    {
        i=7500;
        while(i-->0);
    }
}

int main()
{
    RCC->APB2ENR |= (1<<3);
```

```

RCC->APB2ENR |= (1<<6);
GPIOB->CRL &= 0xFF0FFFFF;
GPIOE->CRL &= 0xFF0FFFFF;
GPIOB->CRL |= 0x00300000;
GPIOE->CRL |= 0x00300000;
while(1)
{
    GPIOB->ODR |= 1<<5;
    GPIOE->ODR &= ~(1<<5);
    delay_ms (1000);
    GPIOE->ODR |= 1<<5;
    GPIOB->ODR &= ~(1<<5);
    delay_ms (1000);
}

return 1;
}

```

五、思考题

1. 修改程序，使得开发板启动后，两盏 LED 灯同时闪烁 5 次，随后交替闪烁。

```

#include "stm32f10x.h"

void delay_ms(int ms)
{
    int i;
    while(ms--)
    {
        i=7500;
        while(i--);
    }
}

int main()
{
    int count = 0;

    RCC->APB2ENR |= (1<<3);
    RCC->APB2ENR |= (1<<6);
    GPIOB->CRL &= 0xFF0FFFFF;
    GPIOE->CRL &= 0xFF0FFFFF;
    GPIOB->CRL |= 0x00300000;
    GPIOE->CRL |= 0x00300000;
}

```

```

while(1)
{
    if(count < 5)
    {
        GPIOB->ODR |=1<<5;
        GPIOE->ODR |=1<<5;
        delay_ms(1000);
        GPIOB->ODR &=~(1<<5);
        GPIOE->ODR &=~(1<<5);
        delay_ms(1000);
        count++;
    }
    else
    {
        GPIOB->ODR |=1<<5;
        GPIOE->ODR &=~(1<<5);
        delay_ms(1000);
        GPIOE->ODR |=1<<5;
        GPIOB->ODR &=~(1<<5);
        delay_ms(1000);
    }
}
return 1;
}

```

2. 查阅资料，获取相关寄存器地址，尝试直接通过地址（指针）操控寄存器，实现跑马灯功能。

```
#include <stdint.h>
```

```

#define PERIPH_BASE          ((uint32_t)0x40000000) /* Peripheral base
address */
#define APB1PERIPH_BASE     PERIPH_BASE
#define APB2PERIPH_BASE     (PERIPH_BASE + 0x00010000)
#define AHBPERIPH_BASE      (PERIPH_BASE + 0x00020000)

#define RCC_BASE             (AHBPERIPH_BASE + 0x1000) // 0x40021000

#define GPIOA_BASE           (APB2PERIPH_BASE + 0x0800) // 0x40010800
#define GPIOB_BASE           (APB2PERIPH_BASE + 0x0C00) // 0x40010C00
#define GPIOC_BASE           (APB2PERIPH_BASE + 0x1000) // 0x40011000
#define GPIOD_BASE           (APB2PERIPH_BASE + 0x1400) // 0x40011400
#define GPIOE_BASE           (APB2PERIPH_BASE + 0x1800) // 0x40011800

```



```

// 自定义 RCC 结构体
typedef struct
{
    volatile uint32_t CR;           // 偏移量 0x00
    volatile uint32_t CFGR;        // 偏移量 0x04
    volatile uint32_t CIR;         // 偏移量 0x08
    volatile uint32_t APB2RSTR;    // 偏移量 0x0C
    volatile uint32_t APB1RSTR;    // 偏移量 0x10
    volatile uint32_t AHBENR;      // 偏移量 0x14
    volatile uint32_t APB2ENR;     // 偏移量 0x18
} RCC_TypeDef_Custom;

// 自定义 GPIO 结构体 (仅包含我们需要的成员)
typedef struct
{
    volatile uint32_t CRL;         // 偏移量 0x00
    volatile uint32_t CRH;         // 偏移量 0x04
    volatile uint32_t IDR;         // 偏移量 0x08
    volatile uint32_t ODR;         // 偏移量 0x0C
    volatile uint32_t BSRR;        // 偏移量 0x10
    volatile uint32_t BRR;         // 偏移量 0x14
    volatile uint32_t LCKR;        // 偏移量 0x18
} GPIO_TypeDef_Custom;

// 定义指向这些结构体的指针
#define RCC ((RCC_TypeDef_Custom *) RCC_BASE)
#define GPIOB ((GPIO_TypeDef_Custom *) GPIOB_BASE)
#define GPIOE ((GPIO_TypeDef_Custom *) GPIOE_BASE)

// LED 引脚定义
#define LED0_PIN 5 // PB5
#define LED1_PIN 5 // PE5

// 延时函数
void delay_ms(uint32_t ms) {
    uint32_t i, j;
    for (i = 0; i < ms; i++) {
        for (j = 0; j < 7500; j++) {
        }
    }
}

int main(void) {

```

```

// 1. 使能 IO 时钟
// 使能 GPIOB (RCC_APB2ENR 的第3位 IOPBEN)
RCC->APB2ENR |= (1 << 3);
// 使能 GPIOE (RCC_APB2ENR 的第6位 IOPEEN)
RCC->APB2ENR |= (1 << 6);

// 2. 设置 IO 模式
// 配置 PB5 (LED0) 为推挽输出模式 (50MHz)
// CNF5[1:0]MODE5[1:0] (位 23:22, 21:20)
// 清空 PB5 原来的模式设置 (CRL 寄存器中的位 20-23)
GPIOB->CRL &= ~(0xF << (LED0_PIN * 4)); // 0xF 表示清除 4 个位,
LED0_PIN*4 是起始位
// 设置 PB5 为推挽输出, 50MHz (CNF=00, MODE=11 => 0011b = 0x3)
GPIOB->CRL |= (0x3 << (LED0_PIN * 4));

// 配置 PE5 (LED1) 为推挽输出模式 (50MHz)
// 清空 PE5 原来的模式设置 (CRL 寄存器中的位 20-23)
GPIOE->CRL &= ~(0xF << (LED1_PIN * 4));
// 设置 PE5 为推挽输出, 50MHz (CNF=00, MODE=11 => 0011b = 0x3)
GPIOE->CRL |= (0x3 << (LED1_PIN * 4));

// 3. 循环程序 (IO 高低电平控制 LED 闪烁)
while (1) {
    // LED0 (PB5) 亮, LED1 (PE5) 灭
    GPIOB->ODR |= (1 << LED0_PIN);    // PB5 输出高电平
    GPIOE->ODR &= ~(1 << LED1_PIN);    // PE5 输出低电平
    delay_ms(500); // 延时约 0.5 秒 (实际时间依赖于 delay_ms 的精确度)

    // LED0 (PB5) 灭, LED1 (PE5) 亮
    GPIOB->ODR &= ~(1 << LED0_PIN);    // PB5 输出低电平
    GPIOE->ODR |= (1 << LED1_PIN);    // PE5 输出高电平
    delay_ms(500); // 延时约 0.5 秒
}
}

```

实验三 基于库函数的跑马灯实验

一、实验目的

1. 熟悉 GPIO 相关的标准库函数和结构体，掌握其功能和使用方法。
2. 熟练掌握基于库函数的 STM32 程序编写流程。

二、实验内容

使用基于库函数编程的方式，结合延时程序，使开发板上的 LED0 和 LED1 灯间隔 0.5 秒钟交替闪烁，即：LED0 亮，LED1 灭，0.5 秒钟后 LED0 灭，LED1 亮，再过 0.5 秒钟 LED0 亮，LED1 灭……如此往复。

三、实验步骤



四、实验结果

```
#include "stm32f10x.h"
```

```
void delay_ms(int ms)
{
    int i;
    while(ms-->0)
    {
        i=7500;
        while(i-->0);
    }
}
```

```
int main()
```

```

{
    GPIO_InitTypeDef initStructure;
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOE, ENABLE);
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);
    initStructure.GPIO_Pin=GPIO_Pin_5;
    initStructure.GPIO_Speed=GPIO_Speed_50MHz;
    initStructure.GPIO_Mode=GPIO_Mode_Out_PP;
    GPIO_Init(GPIOE, &initStructure);
    GPIO_Init(GPIOB, &initStructure);
    GPIO_ResetBits(GPIOE, GPIO_Pin_5);
    GPIO_ResetBits(GPIOB, GPIO_Pin_5);

    while(1)
    {
        GPIO_SetBits(GPIOE, GPIO_Pin_5);
        GPIO_ResetBits(GPIOB, GPIO_Pin_5);
        delay_ms(500);
        GPIO_ResetBits(GPIOE, GPIO_Pin_5);
        GPIO_SetBits(GPIOB, GPIO_Pin_5);
        delay_ms(500);
    }

    return 1;
}

```

五、思考题

1. 基于库函数，编写程序，控制蜂鸣器循环发出不同频率的声音，并探索频率和声调的关系。

```

#include "stm32f10x.h"

void Delay_ms(int ms)
{
    int i;
    while (ms--)
    {
        i = 7500;
        while (i);
    }
}

void delay_tone_us_approx(int us_approx)

```

```

{
    int i_outer;
    int j_inner;

    for (i_outer = 0; i_outer < us_approx; i_outer++)
    {
        j_inner = 8;
        while (j_inner--);
    }
}

void Buzzer_Init(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;

    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);

    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_8;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOB, &GPIO_InitStructure);

    GPIO_ResetBits(GPIOB, GPIO_Pin_8);
}

void Play_Sound(int freq, int duration)
{
    int half_us;
    int cycles;
    int i;

    half_us = freq ? 500000 / freq : 0;
    cycles = freq ? (duration * 500) / half_us : 0;

    if (half_us == 0) {
        Delay_ms(duration);
        return;
    }

    for (i = 0; i < cycles; i++) {
        GPIO_SetBits(GPIOB, GPIO_Pin_8);
        delay_tone_us_approx(half_us);
        GPIO_ResetBits(GPIOB, GPIO_Pin_8);
        delay_tone_us_approx(half_us);
    }
}

```

```

    }
}

int main(void)
{
    int notes[8] = {262, 294, 330, 349, 392, 440, 494, 523};
    int note_cnt = 8;
    int i;

    Buzzer_Init();

    while(1)
    {
        for(i = 0; i < note_cnt; i++)
        {
            Play_Sound(notes[i], 300);
            Delay_ms(100);
        }
        Delay_ms(500);
    }
}

```

频率越高，声音的音调越高；频率越低，声音的音调越低。

2. 尝试重写 Delay 函数，使其运行时尽量少占用 CPU 资源。

```

#include "stm32f10x.h" // 根据具体芯片型号包含头文件

uint32_t ms_delay;

// SysTick 中断服务函数
void SysTick_Handler(void)
{
    if (ms_delay > 0)
    {
        ms_delay--;
    }
}

// 延时函数
void Delay_ms(uint32_t ms)
{
    ms_delay = ms;
    while (ms_delay > 0)
    {

```

```

        __WFI(); // 进入休眠模式，等待中断唤醒（不会占用CPU资源）
    }
}

// 系统时钟配置时初始化 SysTick（通常在 SystemInit() 中）
void SysTick_Init(void)
{
    SysTick->LOAD = (SystemCoreClock / 1000) - 1; // 设置 1ms 重载值
    SysTick->VAL = 0; // 清除当前值
    SysTick->CTRL = SysTick_CTRL_TICKINT_Msk | // 启用中断
                   SysTick_CTRL_ENABLE_Msk; // 启动定时器
}

```

使用方法：

1. 在系统时钟初始化时调用 SysTick_Init()
2. 需要延时时直接调用 Delay_ms(需要延时的毫秒数)

实验四 按键输入实验

一、实验目的

1. 了解 STM32F1 的 IO 口作为输入口的使用方法。

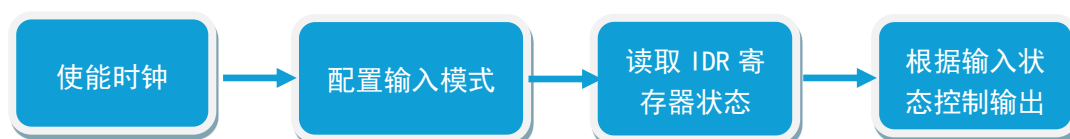
二、实验内容

利用键盘扫描的方式实现以下功能

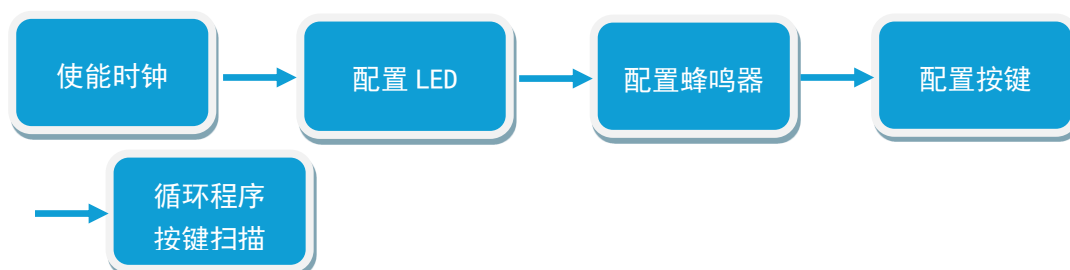
1. KEY0 按下：DS0、DS1 长亮；
2. KEY1 按下：DS0、DS1 均不亮；
3. KEY_UP 按下：蜂鸣器的状态翻转一次（由响到不响或者由不响到响）；
4. 先软件仿真，结果正确后下载到开发板上运行。

三、实验步骤

1. IO 口作为输入口的使用流程



2. 软件流程图



四、实验结果

```
#include "stm32f10x.h"

void delay_ms(int ms)
{
    int i;
    while(ms--)
    {
```



```

        i=7500;
        while(i--);
    }
}

void GPIO_InitPorts(void) {
    GPIO_InitTypeDef GPIO_InitStructure;

    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA | RCC_APB2Periph_GPIOB |
RCC_APB2Periph_GPIOE, ENABLE);

    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_5;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOB, &GPIO_InitStructure);
    GPIO_Init(GPIOE, &GPIO_InitStructure);

    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_8;
    GPIO_Init(GPIOB, &GPIO_InitStructure);
    GPIO_SetBits(GPIOB, GPIO_Pin_8);

    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_3 | GPIO_Pin_4;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPU;
    GPIO_Init(GPIOE, &GPIO_InitStructure);

    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPD;
    GPIO_Init(GPIOA, &GPIO_InitStructure);
}

void Key_Scan(void) {
    static uint8_t buzzer_state = 0;

    if(GPIO_ReadInputDataBit(GPIOE, GPIO_Pin_4) == 0) {
        delay_ms(10);
        if(GPIO_ReadInputDataBit(GPIOE, GPIO_Pin_4) == 0) {
            GPIO_SetBits(GPIOB, GPIO_Pin_5);
            GPIO_SetBits(GPIOE, GPIO_Pin_5);
            while(!GPIO_ReadInputDataBit(GPIOE, GPIO_Pin_4));
        }
    }
    else if(GPIO_ReadInputDataBit(GPIOE, GPIO_Pin_3) == 0) {
        delay_ms(10);
        if(GPIO_ReadInputDataBit(GPIOE, GPIO_Pin_3) == 0) {

```

```

        GPIO_ResetBits(GPIOB, GPIO_Pin_5);
        GPIO_ResetBits(GPIOE, GPIO_Pin_5);
        while(!GPIO_ReadInputDataBit(GPIOE, GPIO_Pin_3));
    }
}
else if(GPIO_ReadInputDataBit(GPIOA, GPIO_Pin_0) == 1) {
    delay_ms(10);
    if(GPIO_ReadInputDataBit(GPIOA, GPIO_Pin_0) == 1) {
        buzzer_state = !buzzer_state;
        if(buzzer_state) {
            GPIO_ResetBits(GPIOB, GPIO_Pin_8);
        } else {
            GPIO_SetBits(GPIOB, GPIO_Pin_8);
        }
        while(GPIO_ReadInputDataBit(GPIOA, GPIO_Pin_0));
    }
}
}

int main(void) {
    GPIO_InitPorts();

    while(1) {
        Key_Scan();
        delay_ms(10);
    }
}

```

五、思考题

1. 写出 IO 作为输入口的重要寄存器。
 - GPIOx_CRL/CRH: 配置引脚模式
 - GPIOx_IDR: 输入数据寄存器
 - GPIOx_ODR: 输出数据寄存器
 - GPIOx_BSRR: 位设置/清除寄存器
2. 本实验中，PA0、PE3、PE4、PB8 设成哪种输入模式？并分析原因。

PA0: 下拉输入 (KEY_UP 高电平有效)

PE3/PE4: 上拉输入 (KEY0/KEY1 低电平有效)

PB8: 推挽输出（控制蜂鸣器）

实验五 外部中断实验

一、实验目的

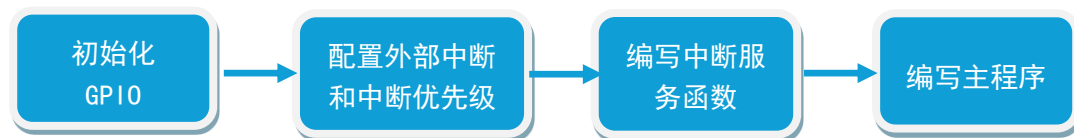
- 1 学习如何将 STM32F1 的 IO 口作为外部中断输入用
2. 学习 STM32 中断的设置及中断服务函数的编写。

二、实验内容

利用按键“中断”实现以下功能：

1. 主程序：DS0 闪亮；
2. KEY1 按下产生中断：DS1 亮 5s；
3. KEY_UP 按下产生中断：蜂鸣器叫 5s；
4. 要求：KEY1、KEY_UP 中断设在 group2，KEY_UP 优先级高于 KEY1。

三、实验步骤



四、实验结果

```
#include "stm32f10x.h"

void delay_ms(int ms) {
    int i, j;
    for(i = 0; i < ms; i++) {
        for(j = 0; j < 7500; j++);
    }
}

void LED_Config(void) {
    GPIO_InitTypeDef GPIO_InitStructure;
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB | RCC_APB2Periph_GPIOE,
    ENABLE);

    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_5;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
```

```

    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOB, &GPIO_InitStructure);
    GPIO_ResetBits(GPIOB, GPIO_Pin_5);

    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_5;
    GPIO_Init(GPIOE, &GPIO_InitStructure);
    GPIO_SetBits(GPIOE, GPIO_Pin_5);
}

void Buzzer_Config(void) {
    GPIO_InitTypeDef GPIO_InitStructure;
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);

    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_8;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOB, &GPIO_InitStructure);
    GPIO_ResetBits(GPIOB, GPIO_Pin_8);
}

void Key_Config(void) {
    GPIO_InitTypeDef GPIO_InitStructure;
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOE | RCC_APB2Periph_GPIOA,
    ENABLE);

    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_4;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPU;
    GPIO_Init(GPIOE, &GPIO_InitStructure);

    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_3;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPU;
    GPIO_Init(GPIOE, &GPIO_InitStructure);

    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPD;
    GPIO_Init(GPIOA, &GPIO_InitStructure);
}

void EXTI_NVIC_Config(void) {
    EXTI_InitTypeDef EXTI_InitStructure;
    NVIC_InitTypeDef NVIC_InitStructure;

    RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO, ENABLE);
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2);
}

```

```

GPIO_EXTILineConfig(GPIO_PortSourceGPIOA, GPIO_PinSource0);
EXTI_InitStructure.EXTI_Line = EXTI_Line0;
EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Rising;
EXTI_InitStructure.EXTI_LineCmd = ENABLE;
EXTI_Init(&EXTI_InitStructure);

NVIC_InitStructure.NVIC_IRQChannel = EXTI0_IRQn;
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
NVIC_Init(&NVIC_InitStructure);

GPIO_EXTILineConfig(GPIO_PortSourceGPIOE, GPIO_PinSource3);
EXTI_InitStructure.EXTI_Line = EXTI_Line3;
EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Falling;
EXTI_InitStructure.EXTI_LineCmd = ENABLE;
EXTI_Init(&EXTI_InitStructure);

NVIC_InitStructure.NVIC_IRQChannel = EXTI3_IRQn;
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 1;
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
NVIC_Init(&NVIC_InitStructure);
}

void EXTI0_IRQHandler(void) {
    if (EXTI_GetITStatus(EXTI_Line0) != RESET) {
        GPIO_SetBits(GPIOB, GPIO_Pin_8);
        delay_ms(5000);
        GPIO_ResetBits(GPIOB, GPIO_Pin_8);
        EXTI_ClearITPendingBit(EXTI_Line0);
    }
}

void EXTI3_IRQHandler(void) {
    if (EXTI_GetITStatus(EXTI_Line3) != RESET) {
        GPIO_ResetBits(GPIOE, GPIO_Pin_5);
        delay_ms(5000);
        GPIO_SetBits(GPIOE, GPIO_Pin_5);
        EXTI_ClearITPendingBit(EXTI_Line3);
    }
}

```

```

}

int main(void) {
    LED_Config();
    Buzzer_Config();
    Key_Config();
    EXTI_NVIC_Config();
    while (1) {
        GPIO_SetBits(GPIOB, GPIO_Pin_5);
        delay_ms(500);
        GPIO_ResetBits(GPIOB, GPIO_Pin_5);
        delay_ms(500);
    }
}

```

五、思考题

1、简述使用外部 I/O 口引脚中断的基本步骤

- 1) 开启复用时钟、I/O 口时钟。调用函数：RCC_APB2PeriphClockCmd();
- 2) 配置 I/O 口，包括引脚名称、传输速率、引脚 工作模式调用函数：GPIO_Init();
- 3) 建立 I/O 口与中断线的映射 GPIO_EXTI_LineConfig();
- 4) 中断初始化：配置中断线、中断模式、中断触发方式、中断使能 函数：EXTI_Init();
- 5) 中断优先级分组：NVIC_PriorityGroupConfig();
- 6) 中断优先级初始化 NVIC_Init();
- 7) 写中断服务函数：EXTIx_IRQHandler();

2、如何配置中断优先级

- 1) 选择中断优先级分组。

```
NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2); // 选择优先级分组 2
```

- 2) 为每个中断配置抢占优先级和子优先级。

```
NVIC_InitTypeDef NVIC_InitStructure;
```

```
// 配置中断0
```

```
NVIC_InitStructure.NVIC_IRQChannel = EXTI0_IRQn; // 指定中断通道
```

```

NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0; // 抢占
优先级为0
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0; // 子优先级为0
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE; // 使能中断通道
NVIC_Init(&NVIC_InitStructure);

```

// 配置中断3

```

NVIC_InitStructure.NVIC_IRQChannel = EXTI3_IRQn; // 指定中断通道
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 1; // 抢占
优先级为1
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0; // 子优先级为0
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE; // 使能中断通道
NVIC_Init(&NVIC_InitStructure);

```

3) 使能中断。

```

NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE; // 使能中断通道

NVIC_Init(&NVIC_InitStructure);

```

4) 编写中断服务函数。

```

void EXTI0_IRQHandler(void) {

    if (EXTI_GetITStatus(EXTI_Line0) != RESET) { // 检查中断标志

        // 中断处理代码

        EXTI_ClearITPendingBit(EXTI_Line0); // 清除中断标志

    }

}

```


实验六 串口通讯实验

一、实验目的

1. 掌握串口异步通讯的设置方法；
2. 掌握串口异步通讯（中断）的设置方法，学会串口中断服务函数的编写方法。

二、实验内容

实验一：

1. 初始化串口通讯；
2. 串口接收上位机发送的字符，并将接收到的字符发送回上位机。

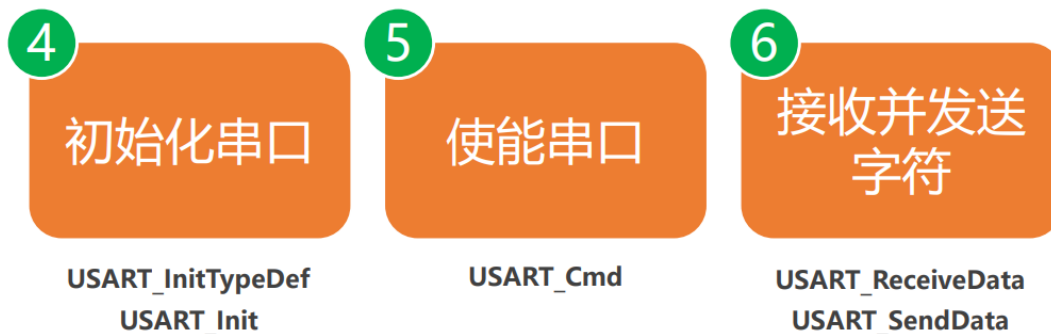
实验二：

1. 初始化串口中断；
2. 串口接收上位机发送的字符，并将接收到的字符发送回上位机；
3. LED0 间隔约 0.2 秒闪烁一次。

三、实验步骤

实验一：





实验二：



四、实验结果

实验一：

```
#include "stm32f10x.h"
```

```
void delay_ms(int ms)
```

```

{
int i;
    while(ms--)
    {
        i=7500;
        while(i--);
    }
}

void USART1_Init(unsigned int bound) {
    GPIO_InitTypeDef GPIO_InitStructure;
    USART_InitTypeDef USART_InitStructure;

    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1, ENABLE);

    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
    GPIO_Init(GPIOA, &GPIO_InitStructure);

    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
    GPIO_Init(GPIOA, &GPIO_InitStructure);

    USART_DeInit(USART1);

    USART_InitStructure.USART_BaudRate = bound;
    USART_InitStructure.USART_WordLength = USART_WordLength_8b;
    USART_InitStructure.USART_StopBits = USART_StopBits_1;
    USART_InitStructure.USART_Parity = USART_Parity_No;
    USART_InitStructure.USART_HardwareFlowControl =
USART_HardwareFlowControl_None;
    USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;
    USART_Init(USART1, &USART_InitStructure);

    USART_Cmd(USART1, ENABLE);
}

int main(void) {
    unsigned short received_data;

    USART1_Init(9600);

```

```

while (1) {
    if (USART_GetFlagStatus(USART1, USART_FLAG_RXNE) != RESET) {
        received_data = USART_ReceiveData(USART1);
        USART_SendData(USART1, received_data);
        while (USART_GetFlagStatus(USART1, USART_FLAG_TC) == RESET);
    }
}
}

```

实验二:

```
#include "stm32f10x.h"
```

```

void delay_ms(int ms)
{
    int i;
    while(ms-->0)
    {
        i=7500;
        while(i-->0);
    }
}

```

```

void LED0_Init(void) {
    GPIO_InitTypeDef GPIO_InitStructure;
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_5;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOB, &GPIO_InitStructure);
    GPIO_SetBits(GPIOB, GPIO_Pin_5);
}

```

```

void USART1_Interrupt_Init(unsigned int bound) {
    GPIO_InitTypeDef GPIO_InitStructure;
    USART_InitTypeDef USART_InitStructure;
    NVIC_InitTypeDef NVIC_InitStructure;

    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1, ENABLE);

    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
}

```

```

GPIO_Init(GPIOA, &GPIO_InitStructure);

GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
GPIO_Init(GPIOA, &GPIO_InitStructure);

USART_DeInit(USART1);

USART_InitStructure.USART_BaudRate = bound;
USART_InitStructure.USART_WordLength = USART_WordLength_8b;
USART_InitStructure.USART_StopBits = USART_StopBits_1;
USART_InitStructure.USART_Parity = USART_Parity_No;
USART_InitStructure.USART_HardwareFlowControl =
USART_HardwareFlowControl_None;
USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;
USART_Init(USART1, &USART_InitStructure);

USART_ITConfig(USART1, USART_IT_RXNE, ENABLE);

NVIC_InitStructure.NVIC_IRQChannel = USART1_IRQn;
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
NVIC_Init(&NVIC_InitStructure);

USART_Cmd(USART1, ENABLE);
}

void USART1_IRQHandler(void) {
    unsigned short received_char;
    if (USART_GetITStatus(USART1, USART_IT_RXNE) != RESET) {
        received_char = USART_ReceiveData(USART1);
        USART_SendData(USART1, received_char);
        while (USART_GetFlagStatus(USART1, USART_FLAG_TC) == RESET);
    }
}

int main(void) {
    LED0_Init();
    USART1_Interrupt_Init(9600);
    while (1) {
        GPIO_ResetBits(GPIOB, GPIO_Pin_5);
        delay_ms(200);
        GPIO_SetBits(GPIOB, GPIO_Pin_5);
    }
}

```

```
    delay_ms(200);  
}
```

五、思考题

1. 阐述 STM32 开发板上非中断串口通讯的实现流程，完成实验一，给出程序流程图及代码，描述实验结果；

实现流程

1. 初始化串口：

- ✓ 配置 GPIO 引脚（如 PA9 为 USART1 的 TX，PA10 为 USART1 的 RX）。
- ✓ 配置 USART1 的波特率、字长、停止位、校验位等参数。
- ✓ 使能 USART1。

2. 发送和接收数据：

- ✓ 在主循环中，通过轮询 USART1 的接收标志位（USART_FLAG_RXNE）来判断是否有数据接收。
- ✓ 如果有数据接收，读取接收到的数据（USART_ReceiveData）。
- ✓ 将接收到的数据发送回上位机（USART_SendData）。
- ✓ 等待发送完成（通过轮询发送完成标志位 USART_FLAG_TC）。

实验结果

- ✓ 上位机发送的字符能够被 STM32 开发板接收。
- ✓ STM32 开发板将接收到的字符原样发送回上位机。
- ✓ 串口通讯正常，数据传输无误。

2. 阐述 STM32 开发板上中断触发的串口通讯的实现流程，完成实验二，给出程序流程图及代码，描述实验结果

实现流程

1. 初始化串口：

- ✓ 配置 GPIO 引脚（如 PA9 为 USART1 的 TX，PA10 为 USART1 的 RX）。
- ✓ 配置 USART1 的波特率、字长、停止位、校验位等参数。
- ✓ 使能 USART1。

2. 配置中断：

- ✓ 使能 USART1 的接收中断（USART_IT_RXNE）。
- ✓ 配置 NVIC 中断优先级。
- ✓ 使能 USART1 中断。

3. 中断服务函数：

- ✓ 在中断服务函数 USART1_IRQHandler 中，判断是否接收到数据（USART_IT_RXNE）。
- ✓ 如果接收到数据，读取接收到的数据（USART_ReceiveData）。
- ✓ 将接收到的数据发送回上位机（USART_SendData）。
- ✓ 等待发送完成（通过轮询发送完成标志位 USART_FLAG_TC）。

4. 主程序：

- ✓ 初始化 LED0。
- ✓ 初始化串口中断。
- ✓ 在主循环中，控制 LED0 以 0.2 秒的间隔闪烁。

实验结果

- ✓ 上位机发送的字符能够被 STM32 开发板接收。
- ✓ STM32 开发板将接收到的字符原样发送回上位机。
- ✓ 串口通讯正常，数据传输无误。
- ✓ LED0 能够按照要求以 0.2 秒的间隔闪烁。

- ✓ 串口通讯和 LED 闪烁互不干扰，系统运行稳定。

实验七 定时器中断实验

一、实验目的

1. 掌握定时器中断的设置方法；
2. 掌握定时器中断服务函数的编写方法。

二、实验内容

1. 设置定时器中断，周期为 1 秒；
2. 触发定时器中断后，反转 LED1 和 BEEP 的状态：当 LED1 熄灭时，BEEP 响；当 LED1 点亮 时，BEEP 不响；
3. LED0 间隔 0.2 秒闪烁一次。

三、实验步骤



四、实验结果

```
#include "stm32f10x.h"

void delay_ms(int ms)
```

```

{
    int i;
    while(ms--)
    {
        i=7500;
        while(i--);
    }
}

void All_GPIO_Init(void) {
    GPIO_InitTypeDef GPIO_InitStructure;
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB | RCC_APB2Periph_GPIOE,
ENABLE);

    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_5;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOB, &GPIO_InitStructure);
    GPIO_ResetBits(GPIOB, GPIO_Pin_5);

    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_5;
    GPIO_Init(GPIOE, &GPIO_InitStructure);
    GPIO_SetBits(GPIOE, GPIO_Pin_5);

    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_8;
    GPIO_Init(GPIOB, &GPIO_InitStructure);
    GPIO_SetBits(GPIOB, GPIO_Pin_8);
}

void TIM3_Interruption_Init(void) {
    TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure;
    NVIC_InitTypeDef NVIC_InitStructure;
    unsigned short arr_val;
    unsigned short psc_val;

    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3, ENABLE);

    // Tout = ((arr+1)*(psc+1))/Tclk
    // Tclk = 72MHz
    // Tout = = 1,000,000
    // (arr+1)*(psc+1) = 72,000,000
    // psc_val = 7200 - 1 = 7199
    // arr_val+1 = 72,000,000 / 7200 = 10,000
    // arr_val = 10000 - 1 = 9999

```

```

arr_val = 9999;
psc_val = 7199;

TIM_TimeBaseStructure.TIM_Period = arr_val;
TIM_TimeBaseStructure.TIM_Prescaler = psc_val;
TIM_TimeBaseStructure.TIM_ClockDivision = TIM_CKD_DIV1;
TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
TIM_TimeBaseInit(TIM3, &TIM_TimeBaseStructure);

TIM_ITConfig(TIM3, TIM_IT_Update, ENABLE);

NVIC_InitStructure.NVIC_IRQChannel = TIM3_IRQn;
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
NVIC_Init(&NVIC_InitStructure);

TIM_Cmd(TIM3, ENABLE);
}

void TIM3_IRQHandler(void) {
    if (TIM_GetITStatus(TIM3, TIM_IT_Update) != RESET) {
        if (GPIO_ReadOutputDataBit(GPIOE, GPIO_Pin_5) == Bit_RESET) {
            GPIO_SetBits(GPIOE, GPIO_Pin_5);
        } else {
            GPIO_ResetBits(GPIOE, GPIO_Pin_5);
        }

        if (GPIO_ReadOutputDataBit(GPIOE, GPIO_Pin_5) == Bit_RESET) {
            GPIO_ResetBits(GPIOB, GPIO_Pin_8);
        } else {
            GPIO_SetBits(GPIOB, GPIO_Pin_8);
        }

        TIM_ClearITPendingBit(TIM3, TIM_IT_Update);
    }
}

int main(void) {
    All_GPIO_Init();
    TIM3_Interrupt_Init();
    while (1) {
        GPIO_SetBits(GPIOB, GPIO_Pin_5);
        delay_ms(200);
    }
}

```

```
        GPIO_ResetBits(GPIOB, GPIO_Pin_5);  
        delay_ms(200);  
    }  
}
```

五、思考题

1. 完成本节实验，给出程序流程图及代码，描述实验结果；

实验结果描述

1. LED1 和 BEEP 的状态反转：

- 定时器中断周期设置为 1 秒。
- 每次定时器中断触发时，LED1（连接到 GPIOE 的 5 号引脚）的状态会反转。
- 当 LED1 熄灭时，BEEP（连接到 GPIOB 的 8 号引脚）会响；当 LED1 点亮时，BEEP 不会响。
- 这种状态反转每秒发生一次，符合实验要求。

2. LED0 的闪烁：

- LED0（连接到 GPIOB 的 5 号引脚）以 0.2 秒的间隔闪烁。
- 由于 LED0 的闪烁是通过主循环中的延时函数实现的，与定时器中断无关，因此它会独立于定时器中断运行。

2. 对于本节的实验，每次中断执行完后，若不清除定时器中断标志，会发生什么现象？

1. 定时器中断标志位（TIM_IT_Update）在中断触发时被硬件自动设置为 1。如果不清除该标志位，中断服务函数会不断被触发，因为中断标志位始终为 1。这会导致中断服务函数不断重复执行，无法退出中断服务函数。
2. 由于中断服务函数不断重复执行，主程序（main 函数中的 LED0 闪烁逻辑）将无法正常运行。系统可能会卡死，或者出现异常行为，如 LED0 闪烁停止，或者 LED1 和 BEEP 的状态无法正常反转。
3. 如果系统中有多中断，且定时器中断的优先级较高，不清除中断标志可能会导致其他中断无法正常响应。这会进一步影响系统的稳定性和响应能力。