



**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。

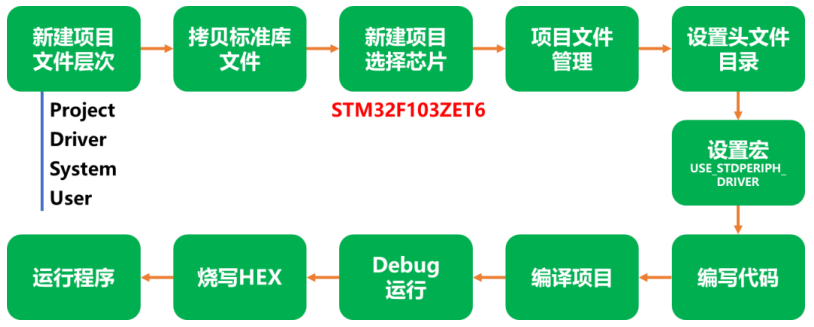
二、实验内容

1. 基于标准库的 STM32 项目创建

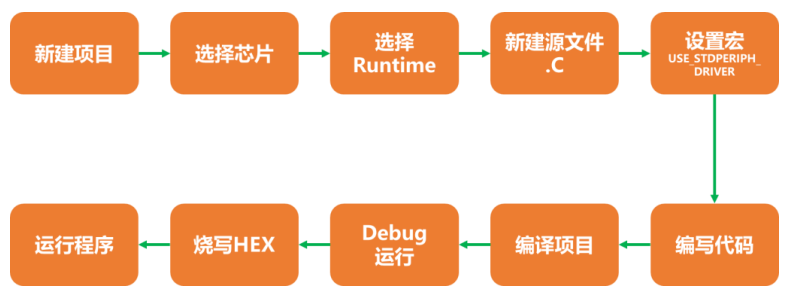
2. 基于 MDK5 的 STM32 项目创建

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--)

    {

    i=7500;

    while(i--);

    }

}

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。
3. 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。

1. 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--)

    {

    i=7500;

*while*(i--);

    }

}

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--)

    {

    i=7500;

*while*(i--);

    }

}

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\_InitStruct;

    RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_GPIOB, ENABLE);

    GPIO\_InitStruct.GPIO\_Pin = GPIO\_Pin\_8;

    GPIO\_InitStruct.GPIO\_Mode = GPIO\_Mode\_Out\_PP;

    GPIO\_InitStruct.GPIO\_Speed = GPIO\_Speed\_50MHz;

    GPIO\_Init(GPIOB, &GPIO\_InitStruct);

    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

使能时钟

配置LED

配置蜂鸣器

配置按键

循环程序

按键扫描

2. 软件流程图

使能时钟

配置输入模式

读取IDR寄存器状态

根据输入状态控制输出

**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**

初始化GPIO

配置外部中断和中断优先级

编写中断服务函数

编写主程序

**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 IO 口引脚中断的基本步骤

* 1. 开启复用时钟、IO口时钟。调用函数： RCC\_APB2PeriphClockCmd();
  2. 配置IO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IO口与中断线的映射 GPIO\_EXTILineConfig();
  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

1. 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);

1. 使能中断。

NVIC\_InitStructure.NVIC\_IRQChannelCmd = ENABLE; // 使能中断通道

NVIC\_Init(&NVIC\_InitStructure);

1. 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：

图形用户界面

AI 生成的内容可能不正确。

图片包含 文本

AI 生成的内容可能不正确。

实验二：

图片包含 应用程序

AI 生成的内容可能不正确。 文本

AI 生成的内容可能不正确。

文本, 图标

AI 生成的内容可能不正确。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--)

    {

    i=7500;

    while(i--);

    }

}

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。

1. 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。

1. 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。

1. 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）。

1. 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

文本

AI 生成的内容可能不正确。

图片包含 文本

AI 生成的内容可能不正确。

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\_Interrupt\_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. 如果系统中有多个中断，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。