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：3220103259

姓名：刘侃

指导教师：李宏娟、林志伟、王郑拓

2025 年 6 月 3 日

目录

实验 1 STM32 项目创建实验

一、实验目的

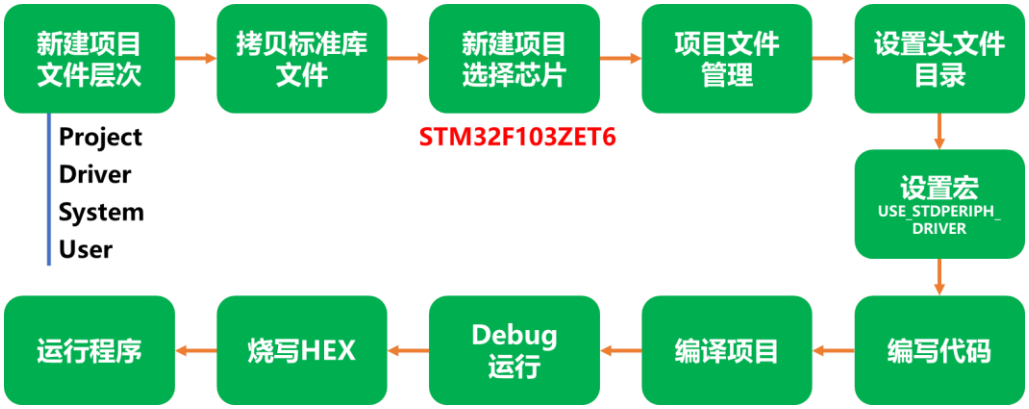
1. 掌握基于 MDK5 的两种 STM32 项目创建方法。
2. 掌握 MDK5 下程序配置、编译、调试、烧写等方法。

二、实验内容

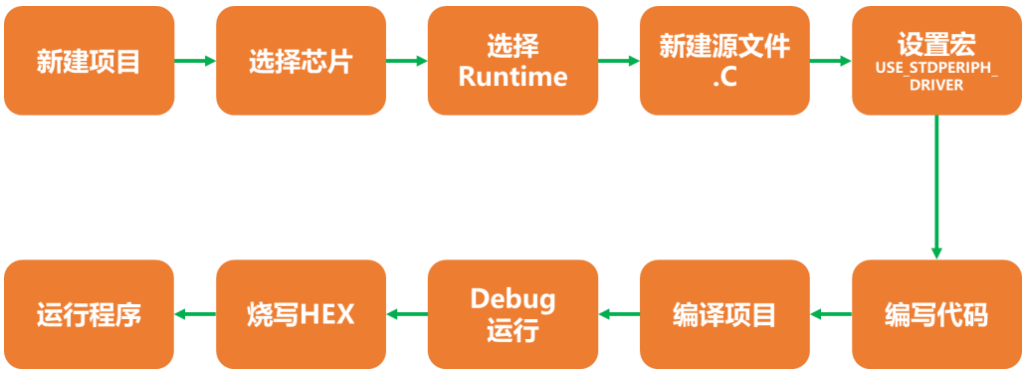
1. 基于标准库的 STM32 项目创建。
2. 基于 MDK5 的 STM32 项目创建。

三、实验步骤

1. 基于标准库的 STM32 项目创建。

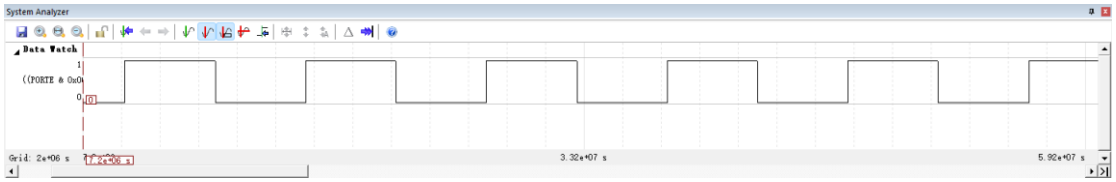


2. 基于 MDK5 的 STM32 项目创建。



四、实验结果

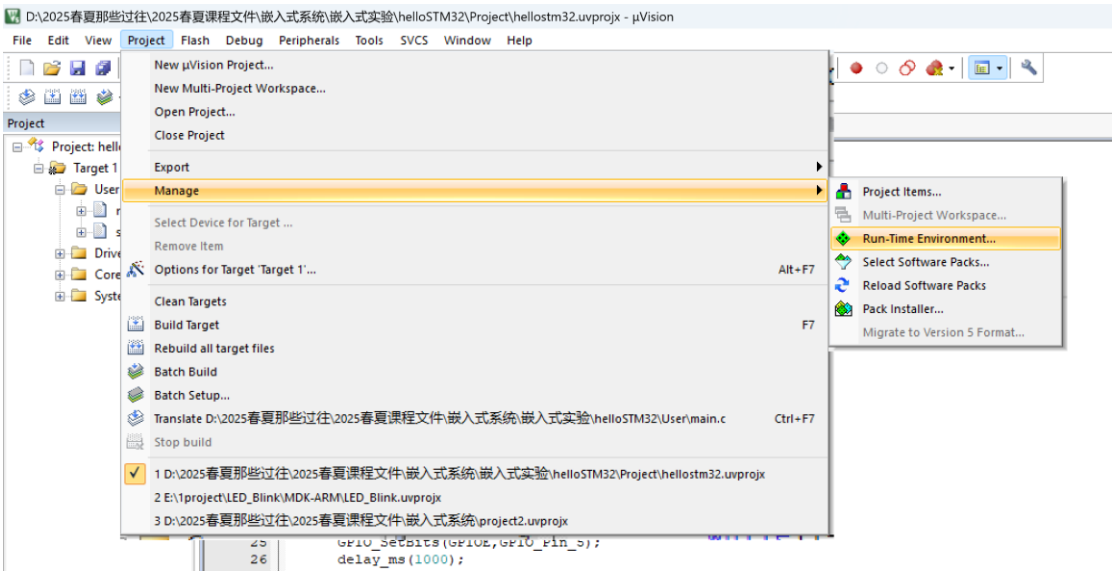
```c
#include "stm32f10x.h"

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;
}
```



（软件是 Keil uVision5，与实验使用的软件有所不同）

五、思考题

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



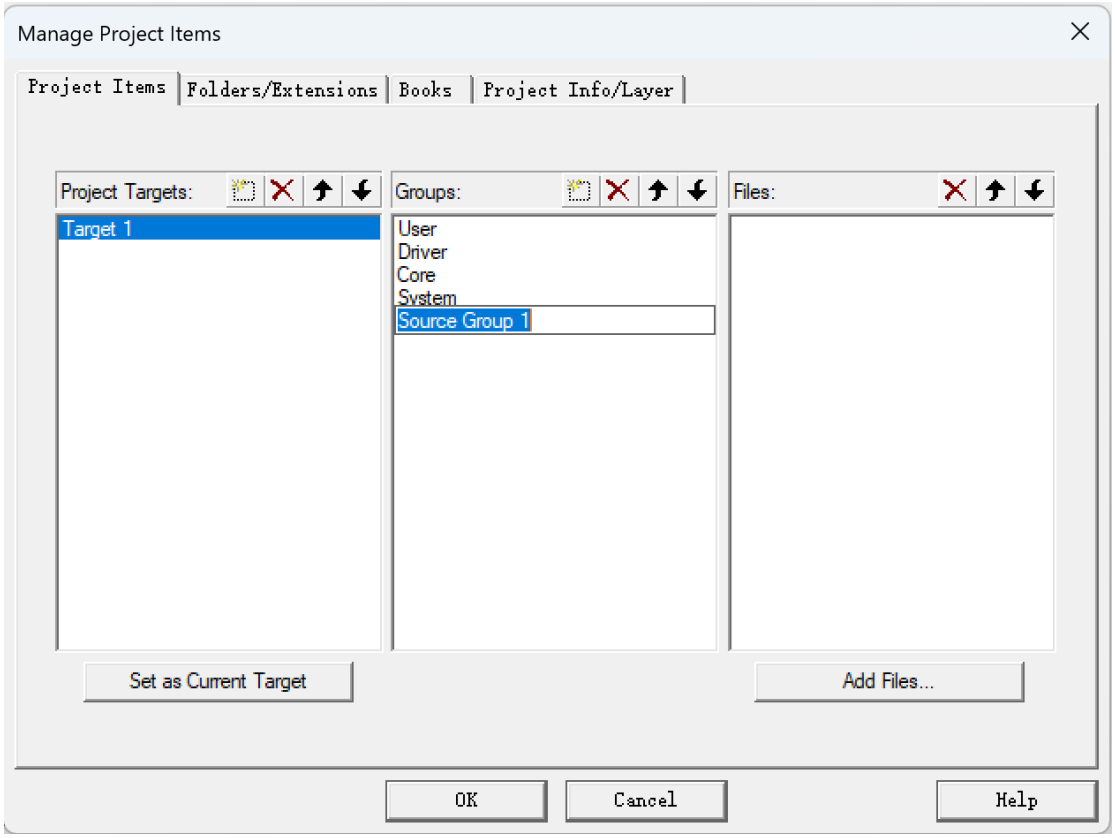
在此界面中修改

2. 在图 1.8 中，如果不去配置 DLL，结果会怎样？为什么？

如果不配置 DLL 文件，调试时将无法正常进行并会报错。

原因分析：Dialog DLL 文件负责为 Keil 提供与芯片匹配的调试会话界面。对于 STM32F103ZE 芯片，必须配置 DARMSTM.DLL（用于调试）和 TARMSTM.DLL（用于目标连接）这两个特定文件。缺少正确的 DLL 配置，Keil 将无法建立与芯片的调试连接，导致调试功能失效。

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



点击“ ”图标，双击“Source Group 1”进行修改。

实验2　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。

该任务可分解为（如图 2.1 所示）：打开 IO 时钟、设置 IO 模式、循环程序（IO 高低电平）。

三、实验步骤

1. 使能 GPIOB、GPIOE

   使能 GPIOB：　RCC->APB2ENR |= (1 << 3)

   使能 GPIOE：　RCC->APB2ENR |= (1 << 6)

2. 设置 PB5 和 PE5 为推挽输出

   PB5:

   ```
   GPIOB->CRL &= 0xFF0FFFFF;
   GPIOB->CRL |= 0x00300000;
   ```

   PE5:

   ```
   GPIOE->CRL &= 0xFF0FFFFF;
   GPIOE->CRL |= 0x00300000;
   ```

3. 设置 PB5 和 PE5 的高低电平循环往复

   PB5 低电平：GPIOB->ODR |= 1 << 5;

   PB5 高电平：GPIOB->ODR &= ~(1 << 5);

四、实验结果

```c
#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;
	GPIOB->CRL |= 0x00300000;
	GPIOE->CRL &= 0xFF0FFFFF;
	GPIOE->CRL |= 0x00300000;

	while (1)
	{

		GPIOE->ODR &= ~(1 << 5);
		delay_ms(500);
		GPIOE->ODR |= 1 << 5;

		GPIOB->ODR &= ~(1 << 5);
		delay_ms(500);
		GPIOB->ODR |= 1 << 5;
	}

	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。

```c
#include "stm32f10x.h"

void delay_ms(int ms)
{
	int i;
	while (ms--)
	{
		i = 7500;
		while (i--);
	}
}
int main()
{
	int j;
	RCC->APB2ENR |= (1 << 3);
	RCC->APB2ENR |= (1 << 6);
	GPIOB->CRL &= 0xFF0FFFFF;
	GPIOB->CRL |= 0x00300000;
	GPIOE->CRL &= 0xFF0FFFFF;
	GPIOE->CRL |= 0x00300000;

	for (j = 1; j <= 5; j++)
	{

		GPIOE->ODR |= 1 << 5;
		GPIOB->ODR |= 1 << 5;
		delay_ms(500);
		GPIOE->ODR &= ~(1 << 5);
		GPIOB->ODR &= ~(1 << 5);
		delay_ms(500);
	}

	while (1)
	{

		GPIOE->ODR &= ~(1 << 5);
		delay_ms(500);
		GPIOE->ODR |= 1 << 5;

		GPIOB->ODR &= ~(1 << 5);
		delay_ms(500);
		GPIOB->ODR |= 1 << 5;
	}

	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。

```c
#include "stm32f10x.h"

void delay_ms(int ms)
{
    int i;
    while (ms--)
    {
        i = 7500;
        while (i--);
    }
}
int main()
{
    int B_ODR = 0x40010C00 + 0x0C;
    int E_ODR = 0x40011800 + 0x0C;
    RCC->APB2ENR |= 1 << 3;
    RCC->APB2ENR |= 1 << 6;

    GPIOB->CRL &= 0xFF0FFFFF;
    GPIOB->CRL |= 0x00300000;
    GPIOE->CRL &= 0xFF0FFFFF;
    GPIOE->CRL |= 0x00300000;

    while (1)
    {
        *((int *)B_ODR) |= 1 << 5;
        *((int *)E_ODR) &= ~(1 << 5);
        delay_ms(500);
        *((int *)B_ODR) &= ~(1 << 5);
        *((int *)E_ODR) |= 1 << 5;
        delay_ms(500);
    }
```

实验3　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。该任务可分解为：打开 IO 时钟、设置 IO 模式、循环程序（IO 高低电平）。

1. 时钟控制函数 RCC_APB2PeriphClockCmd

在本次实验中，参数 RCC_APB2Periph 只需设置 RCC_APB2Periph_GPIOx 即可，其中 x 取值为 A~G，表示使能对应 GPIO 时钟。

此外，参数 NewState 取值为：ENABLE 或 DISABLE，表示使能或关闭使能。

2. 结构体 GPIO_InitTypeDef

```
typedef struct
{
    uint16_t GPIO_Pin;          /*!< Specifies the GPIO pins to be configured.
                                 This parameter can be any value of @ref GPIO_pins_define */

    GPIOSpeed_TypeDef GPIO_Speed; /*!< Specifies the speed for the selected pins.
                                 This parameter can be a value of @ref GPIOSpeed_TypeDef */

    GPIOMode_TypeDef GPIO_Mode;   /*!< Specifies the operating mode for the selected pins.
                                 This parameter can be a value of @ref GPIOMode_TypeDef */
}GPIO_InitTypeDef;
```

三、实验步骤

四、实验结果

```c
#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_ResetBits(GPIOE, GPIO_Pin_5);
	GPIO_Init(GPIOB, &initStructure);
	GPIO_ResetBits(GPIOB, GPIO_Pin_5);

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

		GPIO_SetBits(GPIOB, GPIO_Pin_5);
		delay_ms(1000);
		GPIO_ResetBits(GPIOB, GPIO_Pin_5);
	}

	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。

```c
#include "stm32f10x.h"

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

int main()
{
        int i = 1000;

        GPIO_InitTypeDef initStructure;
        RCC_APB2PeriphClockCm(RCC_APB2Periph_GPIOB, ENABLE);

        initStructure.GPIO_Pin = GPIO_Pin_8;
        initStructure.GPIO_Speed = GPIO_Speed_50MHz;
        initStructure.GPIO_Mode = GPIO_Mode_Out_PP;

        GPIO_Init(GPIOB, &initStructure);
        GPIO_ResetBits(GPIOB, GPIO_Pin_8);
        while (1)
        {
                        GPIO_SetBits(GPIOB,GPIO_Pin_8);
                        delay_ms(i);
                        GPIO_ResetBits(GPIOB,GPIO_Pin_8);
                        delay_ms(i--);
                        if (i == 0)i = 1000;
        }
        return 1;
}
```

2.尝试重写 Delay 函数，使其运行时尽量少占用 CPU 资源。

```c
void delay_ms(uint32_t ms) {
    for (; ms > 0; ms--) {
        for (volatile uint32_t i = 0; i < 7200; i++); // 调整 7200 校准时间
    }
}
```

实验 4 按键输入实验

一、实验目的

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，软件仿真步骤参考：实验 1-调试运行章节。

三、实验步骤

本实验用到的硬件资源有：

1） 指示灯 DS0、DS1

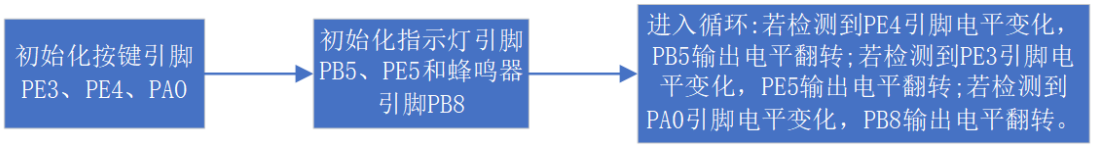
2） 蜂鸣器

3） 3 个按键：KEY0、KEY1 和 KEY_UP。

DS0 连接在 PB5 上,DS1 连接在 PE5 上，蜂鸣器连接在 PB8 上，按键 KEY0 连接在 PE4 上，KEY1 连接在 PE3 上，KEY_UP 连接在 PA0 上。

KEY0 和 KEY1 是低电平有效的，而 KEY_UP 是高电平有效的，并且外部都没有上下拉电阻，所以，需要在 STM32F1 内部设置上下拉。

软件流程图：



四、实验结果

```c
#include "stm32f10x.h"

#define LED0_PIN  GPIO_Pin_5
#define LED0_PORT GPIOB
#define LED1_PIN  GPIO_Pin_5
#define LED1_PORT GPIOE
#define BEEP_PIN  GPIO_Pin_8
#define BEEP_PORT GPIOB
#define KEY0_PIN  GPIO_Pin_4
#define KEY0_PORT GPIOE
#define KEY1_PIN  GPIO_Pin_3
#define KEY1_PORT GPIOE
#define WK_UP_PIN GPIO_Pin_0
#define WK_UP_PORT GPIOA

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

void LED_Init(void) {
    GPIO_InitTypeDef GPIO_InitStructure;

    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB | RCC_APB2Periph_GPIOE, ENABLE);

    GPIO_InitStructure.GPIO_Pin = LED0_PIN;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(LED0_PORT, &GPIO_InitStructure);

    GPIO_InitStructure.GPIO_Pin = LED1_PIN;
    GPIO_Init(LED1_PORT, &GPIO_InitStructure);

    GPIO_SetBits(LED0_PORT, LED0_PIN);
    GPIO_SetBits(LED1_PORT, LED1_PIN);
}

void BEEP_Init(void) {
    GPIO_InitTypeDef GPIO_InitStructure;

    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);

    GPIO_InitStructure.GPIO_Pin = BEEP_PIN;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(BEEP_PORT, &GPIO_InitStructure);

    GPIO_ResetBits(BEEP_PORT, BEEP_PIN);
}

void KEY_Init(void) {
    GPIO_InitTypeDef GPIO_InitStructure;

    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA | RCC_APB2Periph_GPIOE, ENABLE);

    GPIO_InitStructure.GPIO_Pin = KEY0_PIN | KEY1_PIN;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPU;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(KEY0_PORT, &GPIO_InitStructure);

    GPIO_InitStructure.GPIO_Pin = WK_UP_PIN;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPD;
    GPIO_Init(WK_UP_PORT, &GPIO_InitStructure);
}
```

```
int main()
{
    u8 key = 0;
    u8 beep_status = 0;
    u32 time_count = 0;

    LED_Init();
    BEEP_Init();
    KEY_Init();

    while(1)
    {
        if(GPIO_ReadInputDataBit(KEY0_PORT, KEY0_PIN) == 0) {
            delay_ms(10);
            if(GPIO_ReadInputDataBit(KEY0_PORT, KEY0_PIN) == 0) {

                GPIO_ResetBits(LED0_PORT, LED0_PIN);
                GPIO_ResetBits(LED1_PORT, LED1_PIN);

                while(GPIO_ReadInputDataBit(KEY0_PORT, KEY0_PIN) == 0);
            }
        }
        else if(GPIO_ReadInputDataBit(KEY1_PORT, KEY1_PIN) == 0) {
            delay_ms(10);
            if(GPIO_ReadInputDataBit(KEY1_PORT, KEY1_PIN) == 0) {

                GPIO_SetBits(LED0_PORT, LED0_PIN);
                GPIO_SetBits(LED1_PORT, LED1_PIN);

                while(GPIO_ReadInputDataBit(KEY1_PORT, KEY1_PIN) == 0);
            }
        }
        else if(GPIO_ReadInputDataBit(WK_UP_PORT, WK_UP_PIN) == 1) {
            delay_ms(10);
            beep_status = !beep_status;
            if(beep_status) {
                GPIO_SetBits(BEEP_PORT, BEEP_PIN);
            } else {
                GPIO_ResetBits(BEEP_PORT, BEEP_PIN);
            }
            while(GPIO_ReadInputDataBit(WK_UP_PORT, WK_UP_PIN) == 1);
        }

        delay_ms(10);
    }

    return 1;
}
```

五、思考题

1. 写出 IO 作为输入口的重要寄存器；

   端口模式寄存器：GPIOx_MODER；

   端口上拉/下拉寄存器：GPIOx_PUPDR；

   端口输入数据寄存器：GPIOx_IDR；

   端口配置锁定寄存器：GPIOx_LCKR；

2. 本实验中，PA0、PE3、PE4、PB8 设成哪种输入模式？并分析原因。

   PA0 设置为下拉输入，其余设置为上拉输入。

   PA0 接高电平有效按键（按下时 PA0 接 VCC，未按下时悬空或通过电阻下拉）

   其余引脚接低电平有效按键（按下时接地，未按下时悬空或通过电阻上拉）

3. IO 口作为输入口的使用流程；

   步骤 1：IO 口初始化：使能按键对应 IO 口时钟，调用函数：RCC_APB2PeriphClockCmd()；

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()。

步骤 2：while 循环中读取按键输入 IO 口电平、设置指示灯或蜂鸣器的输出 IO 口电平，用到的函数如下。

读取指定 IO 口电平：

GPIO_ReadInputDataBit()，如 GPIO_ReadInputDataBit(GPIOE,GPIO_Pin_4)

设置指定的数据端口：GPIO_SetBits()，如 GPIO_ SetBits (GPIOE,GPIO_Pin_4)

清除指定的数据端口：GPIO_ResetBits()，如 GPIO_ ResetBits (GPIOE,GPIO_Pin_4)

4. 软件流程图；

   见实验步骤

5. 附程序源代码。

   见实验结果

实验5　外部中断实验

一、实验目的

学习如何将 STM32F1 的 IO 口作为外部中断输入用，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（实验原理）

1、初始化 IO 口为输入：GPIO_Init()，参考实验 4

2、开启 IO 口复用时钟：RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO,ENABLE)；

3、设置 IO 口与中断线的映射关系： GPIO_EXTILineConfig()，

示例：GPIO_EXTILineConfig(GPIO_PortSourceGPIOE,GPIO_PinSource3);

4、初始化线上中断，设置触发条件等：EXTI_Init() 使用范例来说明这个函数的使用，仅供参考使用：

EXTI_InitTypeDef EXTI_InitStructure;

EXTI_InitStructure.EXTI_Line=EXTI_Line4; //设置中断线 4

EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt; //设置模式

EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Falling; //中断为下降沿触发。

EXTI_InitStructure.EXTI_LineCmd = ENABLE; //使能中断

EXTI_Init(&EXTI_InitStructure);//根据 EXTI_InitStruct 中指定的参数初始化外设 EXTI 寄存器

5、配置中断分组（NVIC），并使能中断：NVIC_Init()；

使用范例来说明这个函数的使用，仅供参考使用：

NVIC_InitStructure.NVIC_IRQChannel = EXTI0_IRQn; //使能按键外部中断通道

NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0x02; //抢占优先级 2，

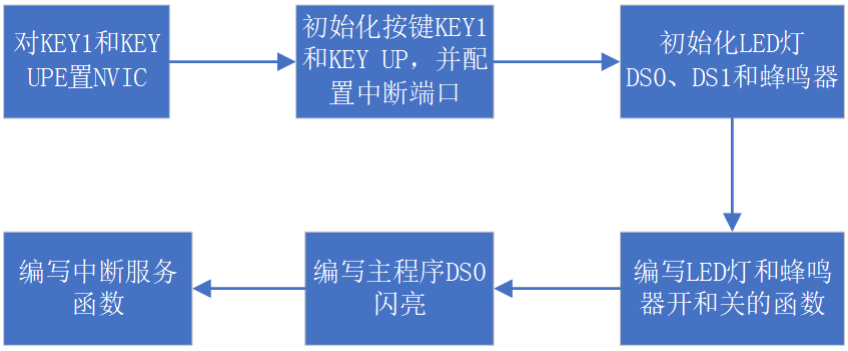
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0x03; //子优先级 3

NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE; //使能外部中断通道

NVIC_Init(&NVIC_InitStructure); //中断优先级分组初始化

6、 编写中断服务函数：EXTIx_IRQHandler()，此函数名不可更改，已在启动文件中声明，不需要再次申明。

7、清除中断标志位：EXTI_ClearITPendingBit()，中断函数执行完后必须清除中断标志位。



四、实验结果

```c
#include "stm32f10x.h"

#define LED0_PIN  GPIO_Pin_5
#define LED0_PORT GPIOB
#define LED1_PIN  GPIO_Pin_5
#define LED1_PORT GPIOE
#define BEEP_PIN  GPIO_Pin_8
#define BEEP_PORT GPIOB
#define KEY1_PIN  GPIO_Pin_3
#define KEY1_PORT GPIOE
#define WK_UP_PIN GPIO_Pin_0
#define WK_UP_PORT GPIOA

void EXTIX_Init(void) {
    EXTI_InitTypeDef EXTI_InitStructure;
    NVIC_InitTypeDef NVIC_InitStructure;
    GPIO_InitTypeDef GPIO_InitStructure;

    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA | RCC_APB2Periph_GPIOE | RCC_APB2Periph_AFIO, ENABLE);

    GPIO_InitStructure.GPIO_Pin = KEY1_PIN;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPU;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(KEY1_PORT, &GPIO_InitStructure);

    GPIO_InitStructure.GPIO_Pin = WK_UP_PIN;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPD;
    GPIO_Init(WK_UP_PORT, &GPIO_InitStructure);

    GPIO_EXTILineConfig(GPIO_PortSourceGPIOE, GPIO_PinSource3);
    GPIO_EXTILineConfig(GPIO_PortSourceGPIOE, GPIO_PinSource4);
    GPIO_EXTILineConfig(GPIO_PortSourceGPIOA, GPIO_PinSource0);

    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);

    EXTI_InitStructure.EXTI_Line = EXTI_Line0;
    EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Rising;
    EXTI_Init(&EXTI_InitStructure);

    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2);

    NVIC_InitStructure.NVIC_IRQChannel = EXTI3_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0x02;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0x03;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);

    NVIC_InitStructure.NVIC_IRQChannel = EXTI0_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0x01;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0x02;
    NVIC_Init(&NVIC_InitStructure);
}
```

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

void LED_Init(void) {
    GPIO_InitTypeDef GPIO_InitStructure;

    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB | RCC_APB2Periph_GPIOE, ENABLE);

    GPIO_InitStructure.GPIO_Pin = LED0_PIN;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(LED0_PORT, &GPIO_InitStructure);

    GPIO_InitStructure.GPIO_Pin = LED1_PIN;
    GPIO_Init(LED1_PORT, &GPIO_InitStructure);

    GPIO_SetBits(LED0_PORT, LED0_PIN);
    GPIO_SetBits(LED1_PORT, LED1_PIN);
}

void BEEP_Init(void) {
    GPIO_InitTypeDef GPIO_InitStructure;

    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);

    GPIO_InitStructure.GPIO_Pin = BEEP_PIN;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(BEEP_PORT, &GPIO_InitStructure);

    GPIO_ResetBits(BEEP_PORT, BEEP_PIN);
}

int main(void) {
    LED_Init();
    BEEP_Init();
    EXTIX_Init();

    while(1) {

        GPIO_WriteBit(LED0_PORT, LED0_PIN, (BitAction)(1 - GPIO_ReadOutputDataBit(LED0_PORT, LED0_PIN)));
        delay_ms(500);
    }
}

void EXTI0_IRQHandler(void) {
    if (EXTI_GetITStatus(EXTI_Line0) != RESET) {

        GPIO_SetBits(BEEP_PORT, BEEP_PIN);
        delay_ms(5000);
        GPIO_ResetBits(BEEP_PORT, BEEP_PIN);

        EXTI_ClearITPendingBit(EXTI_Line0);
    }
}

void EXTI3_IRQHandler(void) {
    if (EXTI_GetITStatus(EXTI_Line3) != RESET) {

        GPIO_ResetBits(LED1_PORT, LED1_PIN);
        delay_ms(5000);
        GPIO_SetBits(LED1_PORT, LED1_PIN);

        EXTI_ClearITPendingBit(EXTI_Line3);
    }
}

EXTI9_5_IRQHandler
{
}
```

五、思考题

1、 简述使用外部 IO 口引脚中断的基本步骤

见实验步骤

2、 如何配置中断优先级

配置中断分组（NVIC），并使能中断：NVIC_Init()；

NVIC_InitStructure.NVIC_IRQChannel = EXTI0_IRQn; //使能按键外部中断通道

NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0x02; //抢占优先级 2，

NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0x03; //子优先级 3

NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE; //使能外部中断通道

NVIC_Init(&NVIC_InitStructure); //中断优先级分组初始化

3、 实验原理、软件流程图

见实验步骤

4、 附程序源代码

见实验结果

实验 6 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。

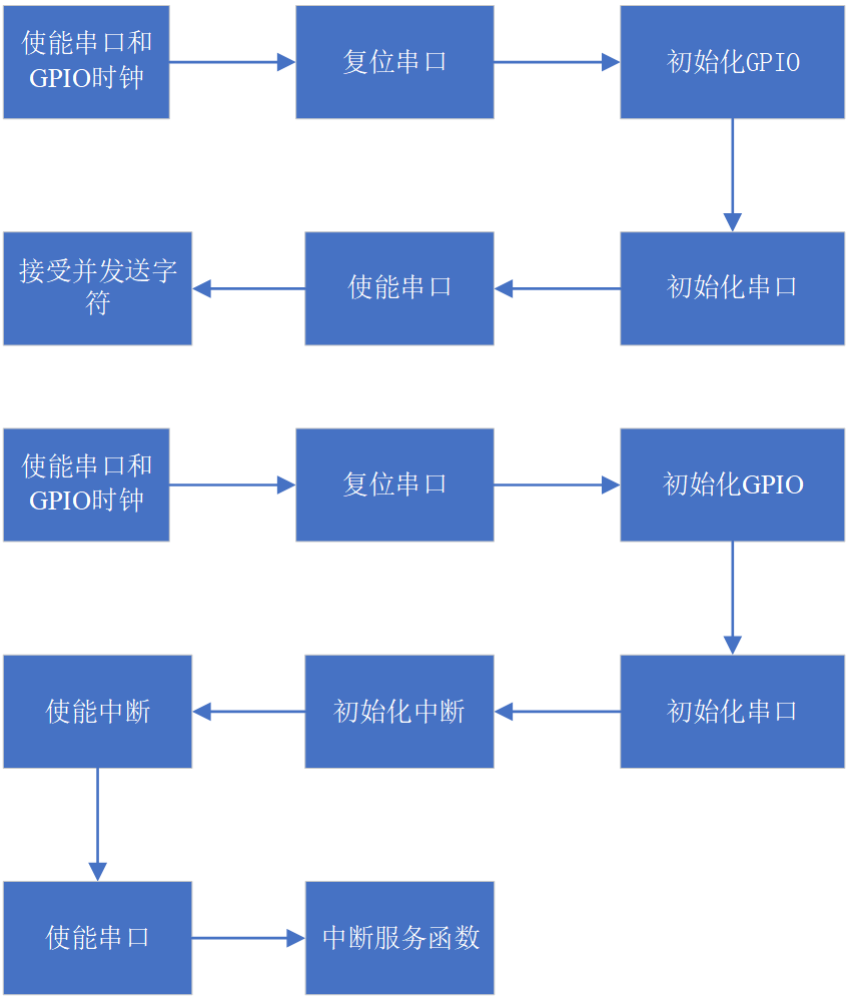
实验二：

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



四、 实验结果

实验 1：

```c
#include "stm32f10x.h"

void USART1_Init(void) {
    GPIO_InitTypeDef GPIO_InitStructure;
    USART_InitTypeDef USART_InitStructure;

    RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1 | RCC_APB2Periph_GPIOA, 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_InitStructure.USART_BaudRate = 115200;
    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) {
    USART1_Init();

    while(1) {
        if(USART_GetFlagStatus(USART1, USART_FLAG_RXNE) == SET) {

            uint16_t data = USART_ReceiveData(USART1);
            USART_SendData(USART1, data);
            while(USART_GetFlagStatus(USART1, USART_FLAG_TXE) == RESET);
        }
    }
}
```

实验 2：

```c
#include "stm32f10x.h"

void GPIO_LED_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_Init(GPIOE, &GPIO_InitStructure);
}

void USART1_Init(void) {
    GPIO_InitTypeDef GPIO_InitStructure;
    USART_InitTypeDef USART_InitStructure;
    NVIC_InitTypeDef NVIC_InitStructure;

    RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1 | RCC_APB2Periph_GPIOA, 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_InitStructure.USART_BaudRate = 115200;
    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 Delay_ms(uint32_t ms) {
    for(uint32_t i = 0; i < ms; i++) {
        for(uint32_t j = 0; j < 7200; j++);
    }
}

int main(void) {
    GPIO_LED_Init();
    USART1_Init();
                GPIO_SetBits(GPIOE, GPIO_Pin_5);
    while(1) {
        GPIO_SetBits(GPIOB, GPIO_Pin_5);
        Delay_ms(200);
        GPIO_ResetBits(GPIOB, GPIO_Pin_5);
        Delay_ms(200);
    }
}

void USART1_IRQHandler(void) {
    if(USART_GetITStatus(USART1, USART_IT_RXNE) != RESET) {
        uint16_t data = USART_ReceiveData(USART1);
        USART_SendData(USART1, data);
        while(USART_GetFlagStatus(USART1, USART_FLAG_TXE) == RESET);
        USART_ClearITPendingBit(USART1, USART_IT_RXNE);
    }
}
```

五、 练习**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；

实现流程见程序流程图，程序流程图和代码见上。

实验结果：上位机发送某字符后，立刻可以接收到回传的该字符。

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；

实现流程见程序流程图，程序流程图和代码见上。

实验结果：上位机发送某字符后，立刻可以接收到回传的该字符，LED0 间隔约 0.2 秒闪烁一次。理论上来说，发送字符中断会使得 LED0 闪烁间隔变长，但是实际情况下，由于处理速度极快，并未观察到 LED0 闪烁间隔变长的现象。

实验 7 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。

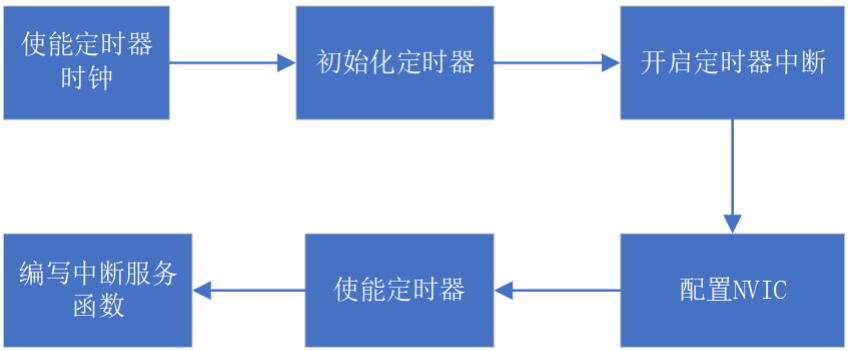
二、 实验内容

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

```c
#include "stm32f10x.h"

void GPIO_LED_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_InitStructure.GPIO_Pin = GPIO_Pin_5;
    GPIO_Init(GPIOE, &GPIO_InitStructure);

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

void TIM3_Init(void) {
    TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure;
    NVIC_InitTypeDef NVIC_InitStructure;

    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3, ENABLE);

    TIM_TimeBaseStructure.TIM_Period = 10000 - 1;
    TIM_TimeBaseStructure.TIM_Prescaler = 7200 - 1;
    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 Delay_ms(uint32_t ms) {
    for(uint32_t i = 0; i < ms; i++) {
        for(uint32_t j = 0; j < 7200; j++);
    }
}

int main(void) {
    GPIO_LED_Init();
    TIM3_Init();
    while(1) {
        GPIO_SetBits(GPIOB, GPIO_Pin_5);
        Delay_ms(200);
        GPIO_ResetBits(GPIOB, GPIO_Pin_5);
        Delay_ms(200);
    }
}

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);
            GPIO_SetBits(GPIOB, GPIO_Pin_8);
        } else {
            GPIO_ResetBits(GPIOE, GPIO_Pin_5);
            GPIO_ResetBits(GPIOB, GPIO_Pin_8);
        }
        TIM_ClearITPendingBit(TIM3, TIM_IT_Update);
    }
}
```

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；

程序流程图和代码见上。

实验结果： LED1 和 BEEP 的状态每 1 秒反转一次：当 LED1 熄灭时，BEEP 响；当 LED1 点亮时，BEEP 不响。LED0 间隔 0.2 秒闪烁一次。

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？

程序一直在执行中断服务函数，LED1 和 LED0 常亮，BEEP 不响（对于不同的程序，也有可能 BEEP 和 LED1 的状态互换）