哈尔滨理工大学 自动化学院

《单片微型计算机原理及嵌入式系统-II》

项目设计与实训报告

题目：按键驱动指示灯实验

姓名 [redacted]

[redacted]

成绩 _____

完成时间：2023 年 11 月 15 日

一、项目概述（500 字以内，10 分）

在这个实验中，我使用了三个独立按键连接到单片机，作为系统的输入。每个按键按下都对应着单片机的一种工作模式。按键功能分配如下：一号按键按下：有源蜂鸣器模块工作发出声音，再次按下，蜂鸣器停止工作。二号按键按下：指示灯 LED1 的亮灭状态翻转。三号按键按下：指示灯 LED0 显示频率改变。按键检测方法包括两种编程方法，查询法和外部中断法。查询法是使用轮询的方式检测按键状态，实现简单但效率较低。外部中断法是利用 STM32 的外部中断功能，通过硬件触发实现按键检测，提高了系统的响应速度和效率。

通过这个实验，我们将学会如何通过查询法和外部中断法两种编程方法实现按键检测。同时，我们还将了解到如何使用 STM32 单片机的 GPIO 功能驱动指示灯，实现不同工作模式下的状态指示。这不仅帮助我们理解 STM32 单片机的基本原理，还培养了我们在嵌入式系统中进行硬件和软件协同开发的能力。

二、项目的具体目标和指标要求（10 分）

设计子目标：

- 按键功能分配：①一号按键按下时，启动有源蜂鸣器模块工作，再次按下，蜂鸣器停止工作。②二号按键按下时，指示灯 LED1 的亮灭状态翻转。③三号按键按下时，指示灯 LED0 显示频率改变。
- 实现两种按键检测方法：①查询法：通过循环查询按键状态的编程方法，实现按键的检测和相应功能。②外部中断法：通过外部中断初始化，配置中断触发条件，实现按键的异步检测和相应功能。
- 各个工作模式都工作正常：能正常控制有源蜂鸣器模块的工作状态；正常控制 LED1 的亮灭状态翻转；正常控制 LED0 显示频率即亮度的改变。

欲达到的指标要求：

- 熟练掌握 STM32F103RCT6 的外部中断初始化方法。
- 能够编写查询法和外部中断法的代码，实现按键的检测和相应功能。
- 理解 GPIO 的配置方法，能够驱动指示灯 LED0 和 LED1 的状态改变。
- 实现的系统具有稳定性，按键操作能够正确触发相应的功能，无明显的系统延迟。
- 编写清晰、模块化的代码，便于理解和维护。

三、项目的具体内容（62 分）

（一）详细分析设计项目的各项内容，具体应包括以下几方面：

1、原理分析

本次流水灯实验采用库函数法来对相应的寄存器进行配置。通过调用 stm32 官方公司编写好的库函数，能够快速的对实验所需要的寄存器进行相应的配置，从而完成本次实验的目标。对手中正点原子 Mini STM32F103RCT6 进行硬件分析，用库函数对其进行初始化配置和控制，来完成实验。

- 按键检测原理：查询法的原理是通过轮询的方式读取按键的状态，常用于简单应用。在每个循环中，检测按键引脚的电平，判断按键是否被按下。外部中断法的原理是，利用 STM32 的外部中断功能，将按键引脚连接到外部中断引脚，当按键触发状态变化时，触发外部中断，执行相应的中断服务程序。

- GPIO 驱动指示灯原理：使用 STM32 的 GPIO 外设配置指示灯的引脚模式（推挽输出、开漏输出等），通过设置引脚的电平实现对指示灯的控制。
- 蜂鸣器模块工作原理：有源蜂鸣器模块通常包含振荡电路和放大电路，通过单片机输出高低电平来控制蜂鸣器的振荡频率，从而产生不同的声音。
- 系统工作流程分析：一号按键按下，触发外部中断或查询法检测，根据按键状态控制有源蜂鸣器的开启和停止。二号按键按下，通过外部中断或查询法检测，改变 LED1 的亮灭状态。三号按键按下，通过外部中断或查询法检测，改变 LED0 的显示频率。
- 编程方法分析：查询法编程的编程思路是使用循环结构，不断读取按键状态，判断按键是否按下，执行相应的操作。外部中断法编程的编程思路是利用外部中断初始化，配置中断触发条件，编写中断服务程序，实现按键的异步检测和相应功能。

2、方案论证

- 按键检测方法的比较：查询法编程简单直观，但效率相对较低，可能存在轮询延迟。外部中断法响应速度快，但配置稍复杂，占用一定资源。
- 功能模式选定：由实验要求可知，需要对三个按键设置不同的工作模式。在本实验里，我设定一号按键为控制有源蜂鸣器模块工作和停止，设定二号按键为控制指示灯 LED1 的亮灭状态翻转，设定三号按键为控制指示灯 LED0 显示频率改变。

3、功能模块设计和算法流程

本次实验根据实验要求采用了两种按键检测编程方法，一种是查询法检测，一种采用的是外部中断按键检测。先进行外部中断和外设引脚初始化，在主循环里，通过查询法或等待外部中断的方式，不断检测用户按键操作。如果是用查询法检测按键状态，则循环读取一号、二号、三号按键引脚的状态，并判断按键状态是否发生变化。如果是外部中断法检测按键状态，则配置好定时器的设置，等待外部中断触发即可。这里设置一个变量 t，用来标志着哪个按键被按下了。后续根据这个变量的值，去执行执行相应的工作模式。整个实验的原理图和算法流程图如下图 1 和图 2 表示。根据实验内容可以把把整个实验分成了以下几个模块，①查询法按键检测模块②外部中断按键检测模块③LED 控制模块④有源蜂鸣器控制模块⑤系统初始化模块。接下来将分别介绍各个模块。

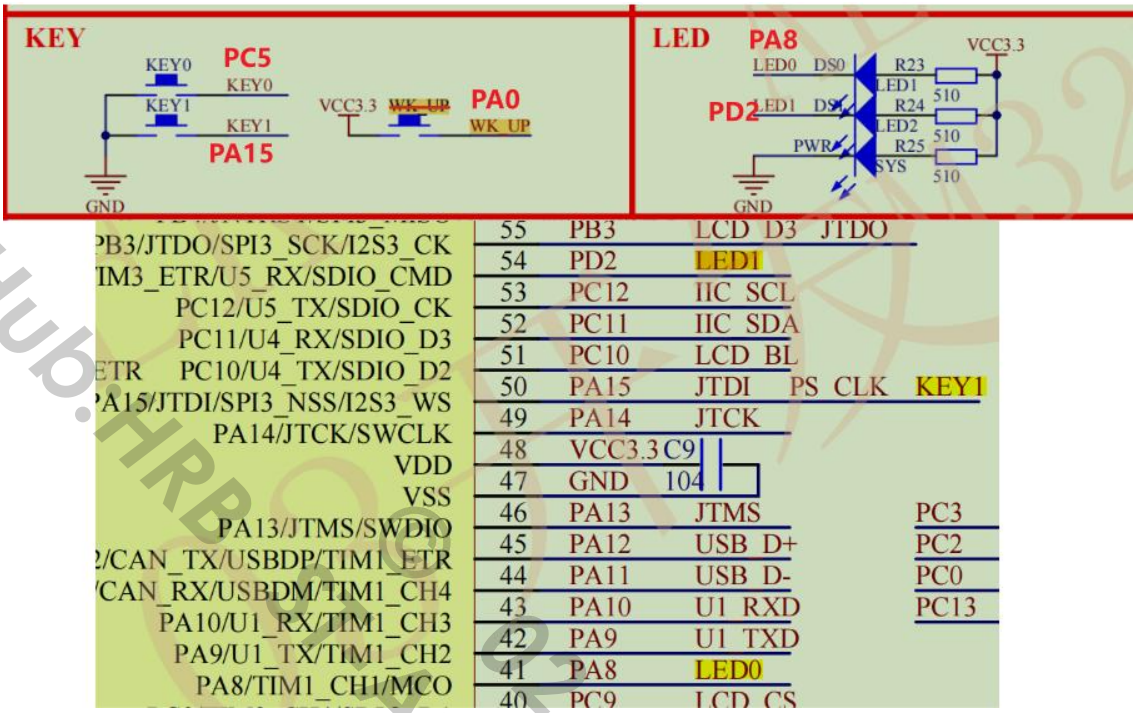

图 1　按键驱动指示灯实验原理图

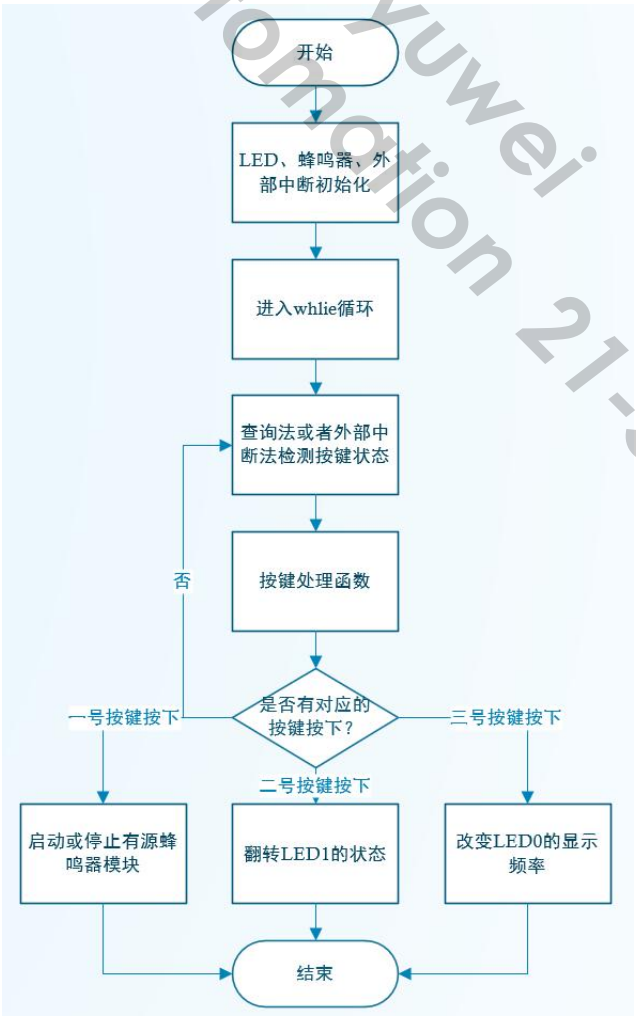

图 2　按键驱动指示灯整体流程图

①查询法按键检测模块

该模块实现的功能是通过轮询的方式检测按键的状态。通过循环读取一号、二号、三号按键引脚的状态。判断按键状态是否发生变化。如果发生变化，调用相应的按键处理函数。相应的代码如下：

```
u8 KEY_Scan(u8 mode)
{
 static u8 key_up=1;//按键按松开标志
 if(mode)key_up=1;  //支持连按
 if(key_up&&(KEY0==0||KEY1==0||WK_UP==1))
 {
  dela _ms( 0);//去抖动
  key_up=0
  if(KEY0==0)retu   KEY0_PRES;
  else if(KEY1= 0)retu n KEY1_PRES;
  else if(WK_UP==1)return WKUP_PRES;
 }else if(KEY0==1&&KEY1= 1&&WK_UP==0)key_up=1;
 return 0;// 无按键按下
}
```

②外部中断按键检测模块

该模块实现的功能是通过外部中断检测按键状态。主要包括两个部分，一是对定时器 1 进行初始化设置，并设定输出捕获模式，二是中断服务程序的编写。算法流程图如图 3 所示。

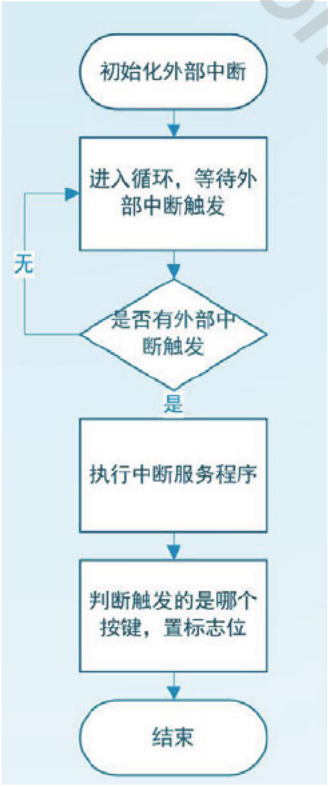

图 3 外部中断按键检测模块流程图

该模块会先初始化外部中断，设置触发条件，如上升沿触发，中断优先级等。再启用外部中断，使能中断控制器。进入主程序循环，等待外部中断触发，等待期间，系统可以执行其他任务，不会因为等待中断而阻塞。当有一号、二号、三号按键之一被按下，外部中断触发。控制权跳转到相应的中断服务程序。

③LED 控制模块

该模块实现的功能是翻转 LED1 亮灭状态和 LED0 显示频率的改变，其中翻转 LED1 亮灭状态的部分比较简单，直接取反 LED1 对应的 GPIO 引脚的高低电平即可。LED0 显示频率的改变也就是改变 PWM 的占空比，首先初始化定时器 TIM1 和 LED0 对应的引脚，然后配置定时器 1 通道 1 的 PWM 输出引脚，定义了一个变量 led0pwmval 用来暂时存储自动重装载值，按键改变的也就是这个变量值，然后再通过 TIM_SetCompare1 给到对应的 PWM 输出引脚上，这样就可以实现 LED0 显示频率的改变了。对应的流程图如下图 4 所示。

图 4 LED 初始化模块流程图

④有源蜂鸣器控制模块

该模块实现的功能是控制有源蜂鸣器的启停和振荡频率。在本实验中我选择的是有源蜂鸣器高电平触发模式，对应模块图如下图 5 所示。该模块逻辑代码比较简单，其使用流程为初始化蜂鸣器引脚为输出模式，再根据系统状态，设置蜂鸣器引脚的电平状态和振荡频率。

图 5 有源蜂鸣器控制模块流程图

⑤系统初始化模块

该模块实现的功能是初始化系统涉及到的外设引脚，包括 LED、蜂鸣器和按键的 GPIO 引脚，如果使用外部中断法作为按键检测的话，则还需要初始化外部中断，配置中断引脚和触发条件。

4、实验实训步骤

①硬件连接

将 STM32F103RCT6 开发板连接到电脑，确保开发环境已经配置好。连接有源蜂鸣器模块，三个独立按键，以及两个指示灯(LED0、LED1)到 STM32 开发板的相应引脚。由于开发板上已经集成了部分外设，所以针对本实验只需要额外连接有源蜂鸣器即可。

②编写程序和编译下载

a. 编写实验的初始化配置代码，将相应的 GPIO 引脚配置为输出模式。确保 GPIO 初始化代码正确，能够控制外设正常工作。

b. 编写两种按键检测编程方法，由于查询法更为简单，可以先编写用查询法实现的代码。如果是外部中断法，则需要写好对应的外部中断服务函数。

c. 分别编写三种工作模式并测试。最后是边编写并调试，这样方便后续查找错误。

d. 完善按键触发事件，根据实验要求，在按键触发事件中添加相应的处理代码，如控制蜂鸣器、改变 LED 频率等。

e. 编译代码，确保没有语法错误和警告。将编译后的程序通过 USB 线下载至 STM32F103C8T6 单片机。

③测试与调试

通上电，观察运行实验，观察按键触发效果和指示灯状态变化。若出现问题，进行调试，检查电路连接和代码逻辑，逐步修正错误。

④实验报告撰写

a. 记录电路连接图和代码。

b. 分析实验过程中遇到的问题和解决方法。

c. 总结实验成果，进行技术和经验上的反思。

5、调试方法和过程记录

①查询法按键检测调试

首先，确保按键引脚和 GPIO 初始化正确。使用 LED 指示灯来指示测试信息，

确保程序正常运行。我在测试的时候编写了一段按键按下就让 LED1 亮的测试代码，程序烧录成功后，观察按键状态变化，注意防抖动的处理是否有效。在按下按键时，确认相应的处理代码是否执行。下图 6 是测试结果截图。

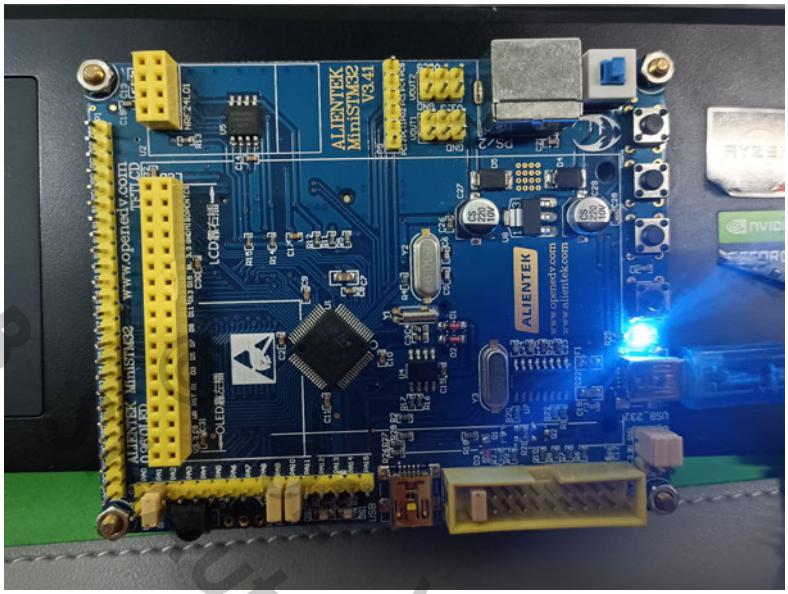

图 6 查询法按键检测调试结果

②外部中断按键检测调试

首先，检查外部中断的引脚配置是否正确。确保外部中断的优先级设置正确，不会影响其他中断的正常运行。在外部中断回调函数中加入调试语句，确认是否正确触发。观察按键状态变化，确保外部中断能够正确响应。我使用了 Keil uVision5 软件自带的 Debug 模式进行调试，在外部中断回调函数中设置断点，通过调试器逐步执行程序，观察外部中断的触发情况。按下按键看看是否跳转至中断服务函数。调试过程的截屏如下图 7 所示。

图 7 外部中断按键检测调试结果

6、综合调试和运行仿真

在测试完各个功能模块后，接下来是将他们综合在一起，实现能够通过按键触发不同的功能，从而改变指示灯和有源蜂鸣器的状态。利用搭建好的电路，使用下载器将程序烧录进 STM32 里面，测试整体按键驱动指示灯效果，观察实验现象，并记录整体运行情况。

按键分布如下图 8 所示：

图 8 按键分布图

首先测试按键 1 的效果，可以发现按下按键 1，有源蜂鸣器模块开始发出声音。再次按下按键 1，有源蜂鸣器停止发声。效果如图 9 所示。

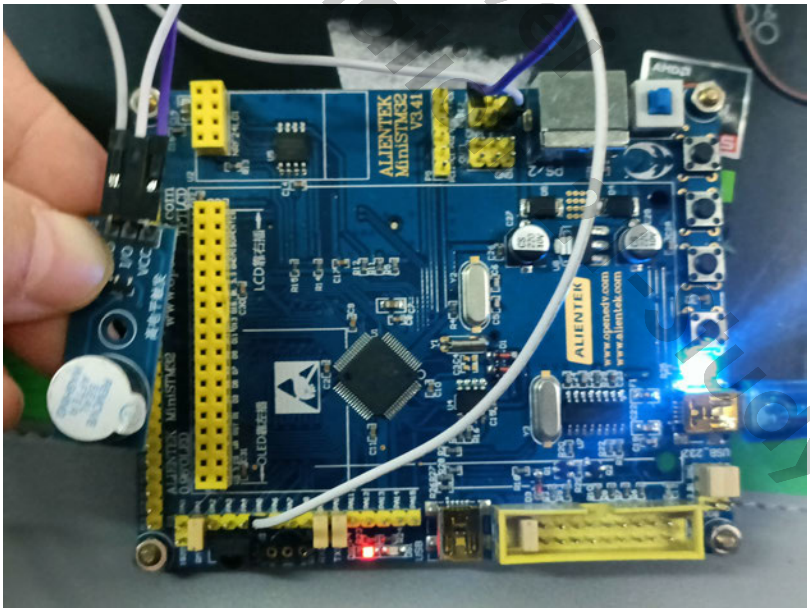

图 9 按键 1 测试结果

然后测试按键 2 的效果，可以发现按下按键 2，指示灯 LED1 的亮灭状态翻转（从亮变成灭，或从灭变成亮）。效果如图 10 所示。

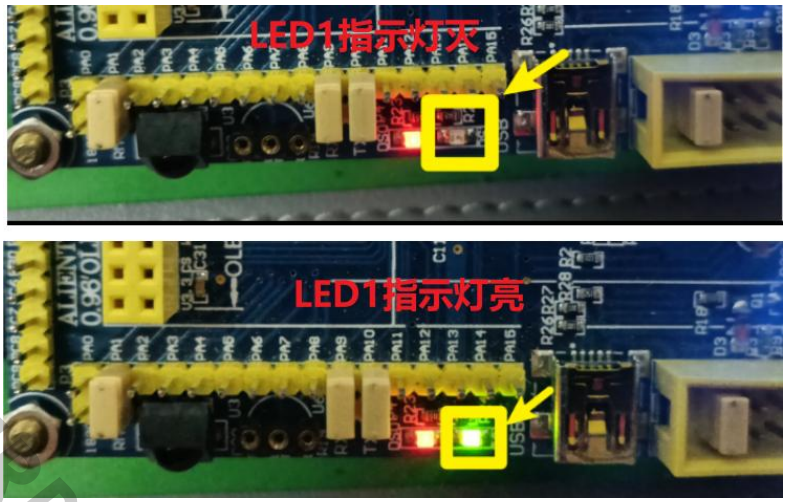

图 10 按键 2 测试结果

再测试按键 3 的效果，可以发现按下按键 3，指示灯 LED0 的显示频率发生改变，可能是越来越亮或越来越暗。效果如图 11 所示。可以发现通过按下按键 3，改变了 LED0 的占空比，可以发现亮度更大了。

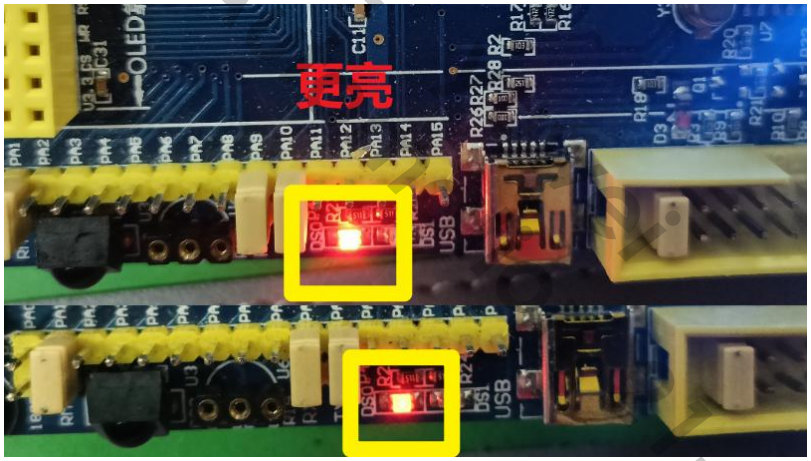

图 11 按键 3 测试结果

7、实验结论与结果分析

实验结论：

- 按键检测

通过查询法和外部中断法两种方式，我成功实现了对三个独立按键的检测。使得按键的状态变化能够被准确捕捉，系统能够正确识别按键的按下和松开动作。

- 指示灯控制

实现了对个指示灯的控制，包括 LED0、LED1。按键 2 按下时，LED1 的亮灭状态翻转。按键 3 按下时，LED0 的亮灭状态频率发生改变。

- 蜂鸣器控制

按键 1 按下时，成功启动有源蜂鸣器模块，发出声音。再次按下按键 1，蜂鸣器停止发声。

结果分析：

- 按键检测方式选择

查询法和外部中断法各有优缺点，查询法简单直观但可能会浪费系统资源，而外部中断法更加实时且高效。我们需要结合实际应用需求选择合适的按键检测方式。

- GPIO 控制指示灯

我通过正确配置 GPIO 引脚，实现了对指示灯的精确控制。LED 的状态变化符合实验要求，满足按键触发不同功能的要求。

- 蜂鸣器控制

有源蜂鸣器成功地被控制启停，实现了按键触发声音的功能。这样的控制方式在实际嵌入式系统中可以用于报警、提醒等应用场景。

- 综合实验效果

实验最终成功实现了一个基本的按键驱动指示灯的小系统。可以通过按下不同的按键，触发不同的功能，包括控制指示灯和有源蜂鸣器。系统稳定可靠，按键触发功能响应迅速。

（二）**本项目是否涉及工程与社会、环境与可持续发展、职业规范等方面的考虑，若有，请简单说明。**具体含义如下：

- 工程与社会

通过实现按键驱动指示灯的小系统，能够在一定程度上解决特定应用场景中的问题，例如在嵌入式控制系统中实现按键触发不同功能。这有助于提高嵌入式系统的用户友好性和交互性。

- 环境和可持续发展

在实现指示灯和有源蜂鸣器的控制时，可以考虑优化代码，以降低系统功耗，从而在一定程度上节约能源，符合可持续发展的理念。

- 职业规范

在整个实验项目中，强调对 STM32 单片机按键检测、外部中断、GPIO 等基础知识的掌握，同时也提及了工程伦理、法规和文化因素，培养了对人文社会科学的素养。强调在项目设计和应用中，理解并遵守工程职业道德和规范，履行责任。这包括对项目的透明度和可维护性的考虑，以及对用户安全和隐私的保护。

四、项目总结与体会（12 分）

（1）项目的总体完成情况。

项目在整体上完成得较为顺利。成功实现了按键驱动指示灯的实验，包括按键检测、指示灯控制和蜂鸣器的功能。系统能够按照预期响应按键输入，达到了设计的基本要求。

（2）项目对（二）中各项具体目标和指标要求的达成情况。

按键检测： 实现了查询法和外部中断法两种按键检测方式，确保了按键的准

确捕捉。

指示灯控制： 成功控制了两个指示灯的状态，包括 LED0 的频率变化和 LED1 的翻转。

蜂鸣器控制： 实现了按键触发有源蜂鸣器的启停，使其能够在需要时发声。

（3）项目中尚存在的问题和改进方向。

防抖动机制： 按键检测部分可能存在抖动，可以进一步引入软件或硬件防抖动机制，以提高按键检测的稳定性。

功耗优化： 在实际应用中，可以进一步优化代码，降低系统功耗，特别是在需要长时间运行的场景下。

多任务系统： 随着系统功能的增加，可能需要考虑使用多任务系统，以更好地组织和管理各个功能模块。

（4）团队及成员的体会

通过这个实验，我深化了对 STM32 单片机的理解，特别是在 GPIO 配置、中断处理和嵌入式系统设计方面的知识。实际应用这些知识，使我更加熟练地运用在具体的项目中。在实验项目过程中，我遇到了一些实际问题，如按键的抖动、外部中断的配置等。通过分析和讨论，我学到了解决问题的方法，这对我的问题解决能力和实际应对能力都有了提升。在实验中，我意识到一个好的嵌入式系统不仅要有良好的功能实现，还需要考虑系统的稳定性和体验。因此，在编写代码时，我更加注重代码的健壮性，以确保系统在各种情况下能够正常运行。通过这个项目，我意识到嵌入式系统领域的知识更新很快，需要保持对新技术的学习和掌握。这种持续学习的精神将对我的职业发展产生积极影响。

五、参考文献（6 分）

[1] 孙雪蕾,范月圆,陈蓉.“定时器中断的应用”信息化教学设计案例[J].镇江高专学报,2023(1):101-103.

[2] 曹振华,吴健,刘靖阳.多按键与指示灯混编系统设计与实现[J].现代计算机,2015(19):65-69.

[3] 王慧丽,张勇强.蜂鸣器故障的分析及应用[J].电子技术与软件工程,2014(9):263-263.

附录 2-实验 2 代码

```c
#include "Device/Include/stm32f10x.h"   // Device header
#include "led.h"
#include "delay.h"
#include "sys.h"
#include "key.h"

void TIM1_PWM1_Init(u16 arr,u16 psc)
{
  GPIO_InitTypeDef GPIO_InitStructure;
  TIM_TimeBaseInitTypeDef  TIM_TimeBaseStructure;
  TIM_OCInitTypeDef  TIM_OCInitStructure;

  RCC_APB2PeriphClockCmd(RCC_APB2Periph_TIM1, ENABLE);//
   RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA , ENABLE);  //使能GPIO 外设时钟使能


    //设置该引脚为复用输出功能,输出TIM1 CH1 的PWM 脉冲波形
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_8; //TIM_CH1
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;  //复用推挽输出
  GPIO_InitStructure.GPIO_Speed = GPIO_Spe_d_5_MHz;
  GPIO_Init(GPIOA, &GPIO_InitStructure);


  TIM_TimeBaseStructure.TIM_Period = arr; //设置在下一个更新事件装入活动的自动重装载寄存器周期的值  80K
  TIM_TimeBaseStructure.TIM_Prescaler =psc; //设置用来作为TIMx 时钟频率除数的预分频值  不分频
  TIM_TimeBaseStructure.TIM_ClockDivision = 0; //设置时钟分割:TDTS = Tc _tim
  TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;  //TIM 向上计数模式
  TIM_TimeBaseInit(TIM1, &TIM_TimeBaseStructure); //根据TIM_TimeBaseInitStruct 中指定的参数初始化 TIMx 的时间基数单位

  TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM2; //选择定时器模式:TIM 脉冲宽度调制模式 2
  TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable; //比较输出使能
  TIM_OCInitStructure.TIM_Pulse = 0; //设置待装入捕获比较寄存器的脉冲值
  TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High; //输出极性:TIM 输出比较极性高
  TIM_OC1Init(TIM1, &TIM_OCInitStructure);  //根据 TIM_OCInitStruct 中指定的参数初始化外设 TIMx

  TIM_CtrlPWMOutputs(TIM1,ENABLE); //MOE 主输出使能

  TIM_OC1PreloadConfig(TIM1, TIM_OCPreload_Enable);  //CH1 预装载使能

```

```
  TIM_ARRPreloadConfig(TIM1, ENABLE); //使能 TIMx 在 ARR 上的预装载寄存器

  TIM_Cmd(TIM1, ENABLE);  //使能 TIM1
}
void LED_Init(void)
{

  GPIO_InitTypeDef  GPIO_InitStructure;

  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA|RCC_APB2Periph_GPIOD, ENABLE);  //使能 PA,PD
端口时钟

  GPIO_Init tructure.GPIO_Pin = GPIO_Pin_8;     //LED0-->PA.8 端口配置
  GPIO_InitStructur  GPIO_Mode = GPIO_Mode_Out_PP;    //推挽输出
  GPIO_InitStructure.G IO_Speed = GPIO_Speed_50MHz;   //IO 口速度为 50MHz
  GPIO_Init(GPIOA, &GPIO_InitStructur );      //根据设定参数初始化 GPIOA.8
  GPIO_SetBits(GPIOA,GPIO_Pin_8);       //PA.8 输出高

  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2;       //LED1-->PD.2 端口配置, 推挽输出
  GPIO_Init(GPIOD, &GPIO_InitStructu e);       / 推挽输出 , IO 口速度为 50MHz
  GPIO_SetBits(GPIOD,GPIO_Pin_2);       //PD.2 输出高
}
u8 KEY_Scan(u8 mode)
{
  static u8 key_up=1;//按键按松开标志
  if(mode)key_up=1;  //支持连按
  if(key_up&&(KEY0==0||KEY1==0||WK_UP==1))
  {
   delay_ms(10);//去抖动
   key_up=0;
   if(KEY0==0)return KEY0_PRES;
   else if(KEY1==0)return KEY1_PRES;
   else if(WK_UP==1)return WKUP_PRES;
  }else if(KEY0==1&&KEY1==1&&WK_UP==0)key_up=1;
  return 0;// 无按键按下
}
void BEEP_init(void)
{
  //BEEP-----PA5
  GPIO_InitTypeDef  GPIO_InitStructure;

  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);  //使能 PA,PD 端口时钟

  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_5;     //LED0-->PA.5 端口配置
```

```
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;    //推挽输出
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;   //IO 口速度为 50MHz
  GPIO_Init(GPIOA, &GPIO_InitStructure);      //根据设定参数初始化 GPIOA.5
  GPIO_ResetBits(GPIOA,GPIO_Pin_5);       //PA.5 输出低
}
u8 t=0;
 int main(void)
 {

 u16 led0pwmval=0;
 u8 dir=1;
 u8 beep_fl g=0;
 BEEP_init();
 delay_init();       //延时函数初始化
 LED_Init();         //初始化与 LED 连接的硬件接口
 KEY_Init();         //初始化与按键连接的硬件接口
 LED0=0;     //点亮 LED
 TIM1_PWM1_Init(899,0);
 while(1)
 {
  t=KEY_Scan(0);  //得到键值
  switch(t)
  {
   case KEY0_PRES:    //指示灯显示频率改变
  if(dir)led0pwmval+=50;
  else led0pwmval-=50;
   if(led0pwmval>300)dir=0;
  if(led0pwmval==0)dir=1;
  TIM_SetCompare1(TIM1,led0pwmval);
    break;
   case KEY1_PRES:   //指示灯亮灭状态翻转
    LED1=!LED1;
    break;
   case WKUP_PRES:
    beep_flag=!beep_flag;
    break;
   default:
    delay_ms(10);
  }
  if(beep_flag==1)
  {
   GPIO_SetBits(GPIOA,GPIO_Pin_5);
   beep_flag=0;
  }
```

```
  else
  {
   GPIO_ResetBits(GPIOA,GPIO_Pin_5);    //PA.5 输出低
   beep_flag=1;
  }
 }
}
 //WAKE_UP--PA0
void EXTI0_IRQHandler(void)
{
 if(EXTI_GetFlagStatus(EXTI_Line0)!=RESET)
 {
  t=3;
  EXTI_ClearITPendingBit(EXTI_Line0);
 }
 }
 //KEY1--PA15
 void EXTI15_10_IRQHandler (void)
 {
 if(EXTI_GetFlagStatus(EXTI_Line15)!=RESET)
 {
  t=2;
  EXTI_ClearITPendingBit(EXTI_Line15);
 }
 }
 //KEY0--PC5
 void EXTI9_5_IRQHandler(void)
 {
 if(EXTI_GetFlagStatus(EXTI_Line5)!=RESET)
 {
  t=1;
  EXTI_ClearITPendingBit(EXTI_Line5);
 }
 }
```