# 计算机原理与嵌入式系统综合实验

## ---利用两个按键分别控制 LED 闪烁周期的增减并在LCD屏幕显示闪烁周期 实验报告

## 信息科学技术学院 李毅PB22051031

### 第一部分 需求分析

1.利用两个按键分别控制 LED 闪烁周期的增减

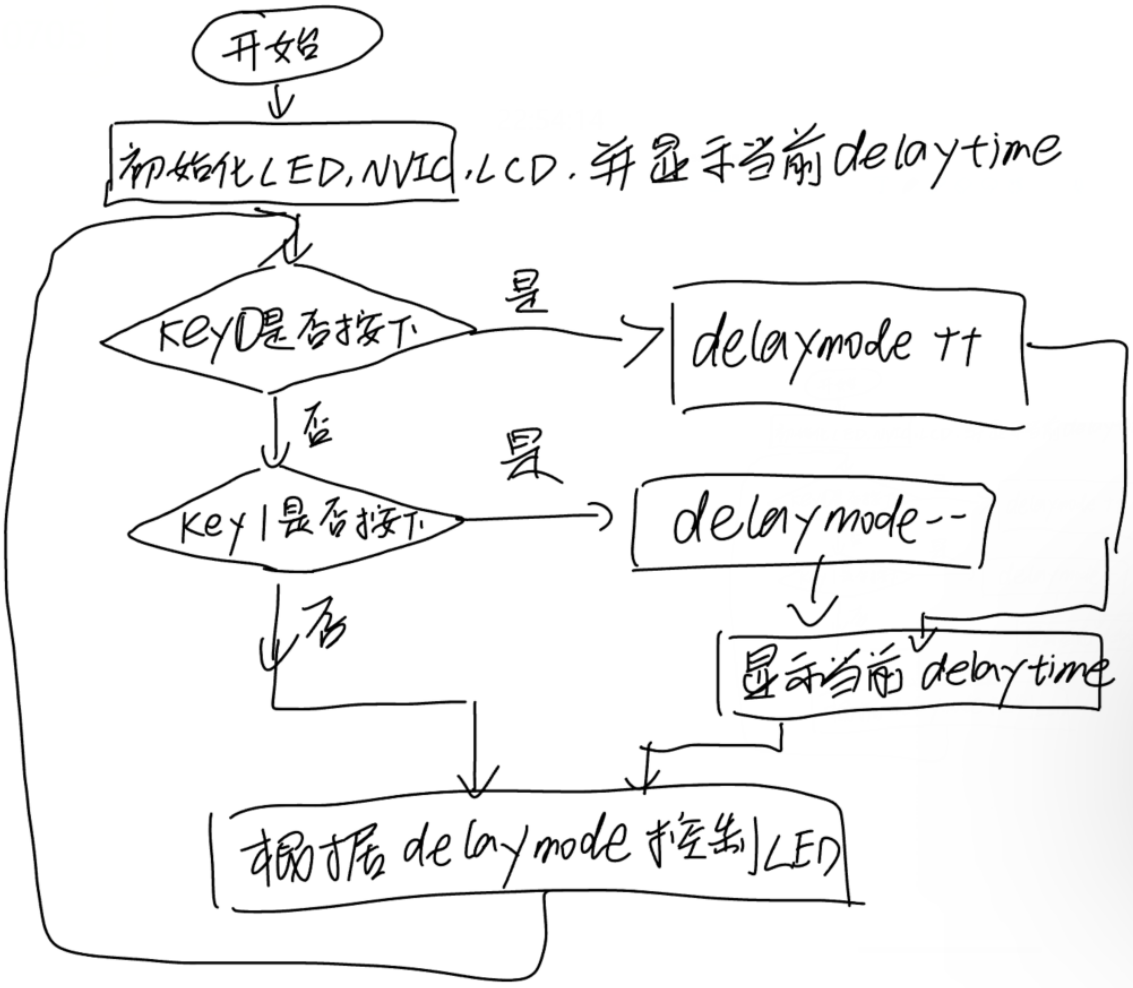
2.在LCD屏幕显示闪烁周期

### 第二部分 功能模块划分

1.利用两个按键分别控制 LED 闪烁周期的增减（使用EXTI模块和GPIO）

2.在LCD屏幕显示闪烁周期（使用LCD模块和GPIO）

### 第三部分 设计流程图



# 第四部分 实现功能核心代码

## 4.1 初始化部分

### 4.1.1 key.c

```
#include "key.h"

void KEY0_Init(void)
{
    GPIO_InitTypeDef  GPIO_InitStructure;
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOE, ENABLE);

    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
    GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
    GPIO_Init(GPIOE, &GPIO_InitStructure);
    GPIO_ResetBits(GPIOE, GPIO_Pin_0);
}

void KEY1_Init(void)
{
    GPIO_InitTypeDef  GPIO_InitStructure;
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOE, ENABLE);

    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
    GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
    GPIO_Init(GPIOE, &GPIO_InitStructure);
    GPIO_ResetBits(GPIOE, GPIO_Pin_1);
}
```

### 4.1.2 exti.c

```
#include "exti.h"

void EXTI0_Init(void)
{
    NVIC_InitTypeDef NVIC_InitStructure;
    EXTI_InitTypeDef EXTI_InitStructure;
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_SYSCFG, ENABLE);
    SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOE, EXTI_PinSource0);
    EXTI_InitStructure.EXTI_Line = EXTI_Line0;
    EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
    EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Rising; //上升沿触发
    EXTI_InitStructure.EXTI_LineCmd = ENABLE;
    EXTI_Init( &EXTI_InitStructure );
    //中断 NVIC 配置
    NVIC_InitStructure.NVIC_IRQChannel=EXTI0_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0x00;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0x02;
    NVIC_InitStructure.NVIC_IRQChannelCmd=ENABLE;
    NVIC_Init(&NVIC_InitStructure);
}
```

```
void EXTI1_Init(void)
{
    NVIC_InitTypeDef NVIC_InitStructure;
    EXTI_InitTypeDef EXTI_InitStructure;
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_SYSCFG, ENABLE);
    SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOE, EXTI_PinSource1);
    EXTI_InitStructure.EXTI_Line = EXTI_Line1;
    EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
    EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Rising; //上升沿触发
    EXTI_InitStructure.EXTI_LineCmd = ENABLE;
    EXTI_Init( &EXTI_InitStructure );
    //中断 NVIC 配置
    NVIC_InitStructure.NVIC_IRQChannel=EXTI1_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0x00;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0x02;
    NVIC_InitStructure.NVIC_IRQChannelCmd=ENABLE;
    NVIC_Init(&NVIC_InitStructure);
}
```

#### 4.1.3 12864.c

```
//12864.c
#include "12864.h"
#include "delay.h"
void LCD_GPIO_Init()
{
  //GPIO 初始化代码
    GPIO_InitTypeDef  GPIO_InitStructure;
  RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOG|
                                RCC_AHB1Periph_GPIOF, ENABLE);

    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
  GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
  GPIO_Init(GPIOG, &GPIO_InitStructure);
    GPIO_ResetBits(GPIOG,GPIO_Pin_1);

    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_15 |GPIO_Pin_14;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
  GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
  GPIO_Init(GPIOF, &GPIO_InitStructure);
    GPIO_ResetBits(GPIOF, GPIO_Pin_15 |GPIO_Pin_14);

    CS=1;
    SID=1;
    SCLK=1;
}
void SendByte(u8 byte)
{
    u8 i;
    for(i = 0;i < 8;i++)
    {
```

```
        if((byte << i) & 0x80) //0x80(1000 0000)
        {
            SID = 1;
        }
        else
        {
            SID = 0;
        }
        SCLK = 0;
        delay_us(5);
        SCLK = 1;
    }
}
void Lcd_WriteCmd(u8 Cmd )
{
    delay_ms(1);
    SendByte(WRITE_CMD); //11111,RW(0),RS(0),0
    SendByte(0xf0&Cmd);
    SendByte(Cmd<<4);
}
void Lcd_WriteData(u8 Dat )
{
    delay_ms(1);
    SendByte(WRITE_DAT);
    SendByte(0xf0&Dat);
    SendByte(Dat<<4);
}
void LCD_Init(void)
{
    delay_ms(50);
    Lcd_WriteCmd(0x30);
    delay_ms(1);
    Lcd_WriteCmd(0x30);
    delay_ms(1);
    Lcd_WriteCmd(0x0c);
    delay_ms(1);
    Lcd_WriteCmd(0x01);
    delay_ms(30);
    Lcd_WriteCmd(0x06);
}
void LCD_Display_Words(uint8_t x,uint8_t y,uint8_t*str)
{
    Lcd_WriteCmd(LCD_addr[x][y]);
    while(*str>0)
    {
        Lcd_WriteData(*str);
        str++;
    }
}
void LCD_Clear(void)
{
    Lcd_WriteCmd(0x01);
    delay_ms(2);
}
```

### 4.2 中断服务函数

```
void EXTI0_IRQHandler(void)
{
    if(EXTI_GetITStatus(EXTI_Line0)!=RESET)
    //判断某个线上的中断是否发生
    {
    if(delaymode<4)
        {
            delaymode++;
        }
        LCD_Display_Words(0,0,"delaytime");
        LCD_Display_Words(1,0,(uint8_t*)delaystring);
        delay_ms(100);
        EXTI_ClearITPendingBit(EXTI_Line0);
    //清除 LINE 上的中断标志位
    }
}

void EXTI1_IRQHandler(void)
{
    if(EXTI_GetITStatus(EXTI_Line1)!=RESET)
    //判断某个线上的中断是否发生
    {
        if(delaymode>0)
        {
            delaymode--;
        }
        LCD_Display_Words(0,0,"delaytime");
        LCD_Display_Words(1,0,(uint8_t*)delaystring);
        delay_ms(100);
        EXTI_ClearITPendingBit(EXTI_Line1);
    //清除 LINE 上的中断标志位
    }
}
```

### 4.3 main函数

```
#include "sys.h"
#include "delay.h"
#include "usart.h"
#include "led.h"
#include "key.h"
#include "exti.h"
#include "12864.h"
#include <string.h>
extern int delaymode;
extern int delaytime;
extern char delaystring[16];
int main(void)
{
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2);
    delay_init(168);
    LED_Init();
    KEY0_Init();
    KEY1_Init();
```

```
    EXTI0_Init();
    EXTI1_Init();
    LCD_GPIO_Init();
    LCD_Init();
    LED0=1;
    strcpy(delaystring,"0100");
    LCD_Display_Words(0,0,"delaytime");
    LCD_Display_Words(1,0,(uint8_t*)delaystring);
    while(1)
    {
        LED0=!LED0;
        switch(delaymode)
        {
            case 0: delaytime=100;
            strcpy(delaystring,"0100");
            break;
          case 1: delaytime=200;
            strcpy(delaystring,"0200");
            break;
            case 2: delaytime=500;
            strcpy(delaystring,"0500");
            break;
            case 3: delaytime=1000;
            strcpy(delaystring,"1000");
            break;
            case 4: delaytime=2000;
            strcpy(delaystring,"2000");
            break;
            default:delaytime=1000;
            strcpy(delaystring,"1000");
            break;
        }

        delay_ms(delaytime);
    }
}
```

## 第五部分 总结

通过综合实验，使我对stm32系统设计开发的方式有了更进一步的认知

## 附：第一次实验记录

1. 观察以下变量存放格式并记录。

```
int main (void)
{
    unsigned int ui_tmp;
    unsigned int ui_a, ui_b, ui_c;
    static int i_tmp; //signed int (32bits)
    static short s16_tmp; //signed short (16bits)
    static float f_tmp; //floating point (32bits)
    static int s[8];
    int k;
    //记录浮点数 IEEE754 规范表示方法
```

```
    f_tmp = -0.5;
    f_tmp = f_tmp + 1;
    //临时变量， 观察 ui_tmp, ui_a, ui_b, ui_c 被保存在哪里
    //观察执行前后 PC 寄存器
    ui_a = 1;
    ui_b = 2;
    ui_c = 0xFF;
    //观察执行后 PSR 寄存器的标志位 Negtive
    ui_c = ui_a - ui_b;
    ui_tmp = ui_c;//观察数的表示方法（ 补码）
    i_tmp = -1;
    i_tmp = i_tmp - 1;
    s16_tmp = -1;
    s16_tmp = -2;
    s16_tmp = s16_tmp - 32766;
    //单步跟踪观察循环体执行过程
    for(k=8; k>0; --k)
    s[k-1] = 0x80000000 + k;
}
```

-0.5 0xBF 00 00 00

0.5 0x3F 00 00 00

临时变量 ui_a:寄存器R2

临时变量 ui_b:寄存器R3

临时变量 ui_c:寄存器R4

临时变量 ui_tmp:寄存器R5

前PC=0x0800044C

后PC=0x08000452

执行后 xPSR=0x8100 0000

-1: 0x FF FF FF FF

-2: 0x FF FF FF FE

-2-32766: 0x 80 00 00 00

2.编写子函数实现统计 unsigned char 型数据中二进制“ 1” 数量的功能。例： unsigned char uc_c=0x78;统计结果为 4， 提示： 可使用 C 语言中“ 位与” 及“ 移位” 功能实现。

```
int main(void)
{
    unsigned char uc_c=0x78;
    unsigned int amount_1=0;
    unsigned char test=0x01;
    while(test!=0x00)
    {
        if((uc_c & test) != 0x00)
        {
```

```
            amount_1++;
        }
        test=test<<1;
    }
}
```

3.使用 CMSIS-CORE 函数实现底层操作并记录读取内容。阅读教材 P278~P282， 特殊寄存器。 单步执行程序中如下代码并记录， 对比ui_tmp 的值和 Register 窗口观察到的数值是否一致， 如图 1-17 所示。

注意： 相关 CMSIS-CORE 函数调用需包含头文件#include "stm32f4xx.h"ui_tmp = **get_FAULTMASK();**

```
//Get Fault Mask register ui_tmp = __get_BASEPRI();

//Get Base Priority register ui_tmp = __get_PRIMASK(); __

//Get Priority Mask Register ui_tmp = __get_CONTROL();

//Get CONTROL Register ui_tmp = __get_MSP();

//Get Main Stack Pointer
```

| 函数 | 值 |
| --- | --- |
| __get_BASEPRI() | 0 |
| __get_PRIMASK() | 0 |
| __get_CONTROL() | 0x 00 00 00 04 |
| __get_MSP() | 0x 20 00 06 60 |