实验五 简易计算器设计

一、 实验目的

1. 掌理解任务管理的基本原理，掌握 μCOS-II 中任务管理的基本方法；
2. 掌握 μCOS-II 中任务间通信的一般原理和方法；
3. 掌握嵌入式系统中 LCD 与键盘控制的一般方法。

二、 实验内容

1. 设计多个应用任务，用其中一个任务控制 LED 灯的状态；
2. 设计多个应用任务，用其中一个任务读取键盘键值，键盘的响应用中断实现；
3. 实现一个简易的计算器。

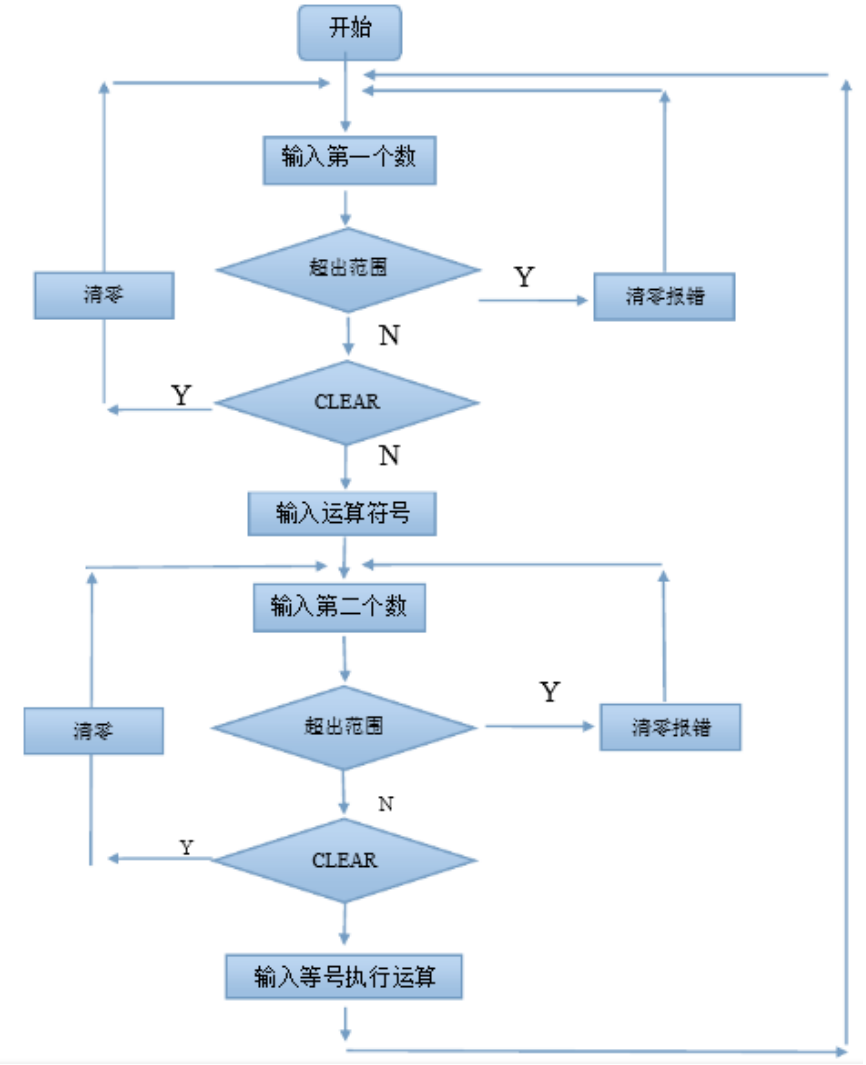
   Task1：键盘

   Task2：流水灯

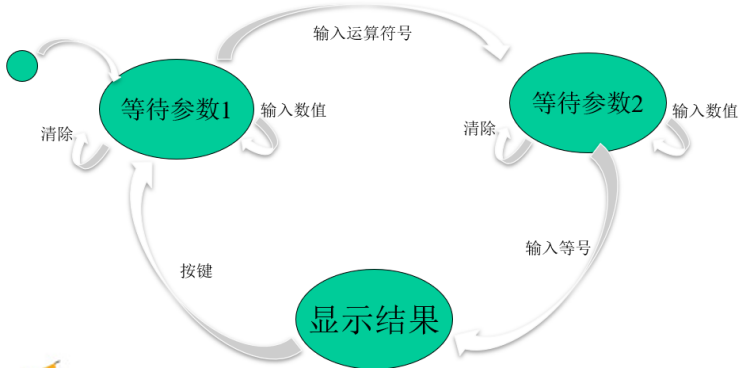
   Task3：LCD 显示。

三、 实验原理

1. 简易计算器工作原理



2. 有限状态机工作原理



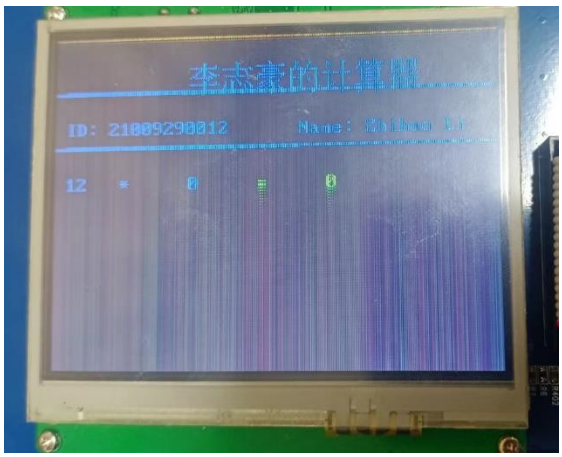
四、 实验步骤

1. 验证示例源码
   - 拷贝整个实验例程源码目录到本地磁盘自己的工作目录下；
   - 使用 μ Vision IDE for ARM 通过 ULINK2 仿真器连接实验板，打开实验例程目录 04-uCOS\3.1_LED_test 子目录下的 ucos2.Uv2 例程，编译链接工程；
   - 将程序下载到实验平台的 NorFlash 中，观察实验结果；
   - 打开实验例程目录\04-uCOS\3.3_keyboard_test 子目录下的 ucos2.Uv2 例程，编译链接工程；
   - 下载调试，观察结果。
2. 设计实现一个简易的计算器
   - 拷贝示例实验源码工程 3.3_keyboard_test；
   - 添加 LCD 驱动程序（参考 11_LCD_Test）；
   - 设计多任务。至少 LCD 显示用一个任务，键盘解析用一个。

五、 实验结果

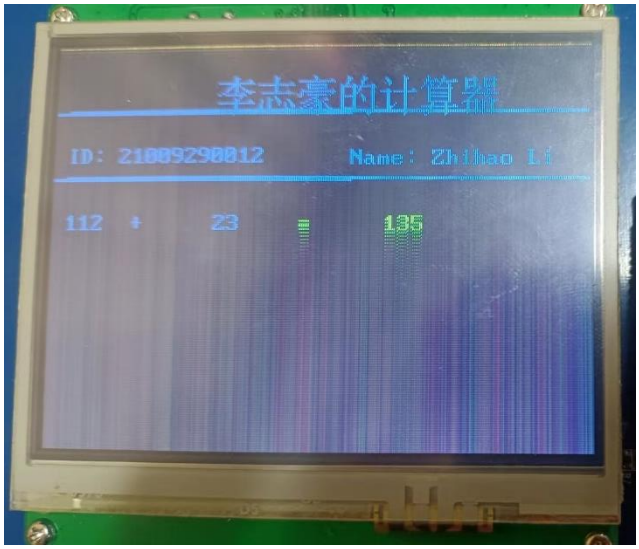
在本次实验中，我自主设计了计数器的 LCD 显示页面，计算表达式的输出，加减乘除法的实现，除法中零除错误的判定，页面重置功能，负数运算的实现，并通过用户键盘读入输入的字符，进行中断控制，最终完成简易计算器的设计，以下是部分实验结果：

- 十进制数与 0 的乘法



- 负数减法运算的实现

- 加法运算的实现



- 除法运算中除零的判定

六、 程序说明

本实验程序基于给定的示例工程 keyboard_test，进行自主设计修改，程序的核心代码如下：

1. 计算器任务分解

为了能够更完美地完成计算器任务，本实验采用多任务管理的机制，对实验要求的三项任务进行并发操作，并将计算器分解为键盘输入取数任务，计算任务，液晶屏任务，保证这三项任务之间以同步关系实现相互通信。

2. 多任务信号量创建与任务初始化

在这一步骤中，我首先创建了用于同步键盘中断的信号量 kbd_sem，初值为 0，在 keyboard_test 中一旦检测到有键盘输入，便对 kbd_sem 进行 V 操作。

信号量 num1_sem 与 num2_sem 用于同步第一源操作数和第二源操作数是否已经输入完毕，show_sem 信号量用于同步 LCD 显示任务，具体工作原理分析见步骤三。

```
OS_EVENT *UART_sem;
OS_EVENT *kbd_sem;
OS_EVENT *num1_sem;
OS_EVENT *num2_sem;
OS_EVENT *InterruptSem;
OS_EVENT *show_sem;

/* allocate memory for tasks' stacks */
#define STACKSIZE 256

/* Global Variable */
OS_STK Stack1[STACKSIZE];
OS_STK Stack2[STACKSIZE];
OS_STK Stack3[STACKSIZE];
OS_STK ShowStack[STACKSIZE];
OS_STK CaculStack[STACKSIZE];
OS_STK GetNumStack[STACKSIZE];
OS_STK StackMain[STACKSIZE];

const char Id1 = '1';
const char Id2 = '2';
const char Id3 = '3';
const char Id4 = '4';
const char Id5 = '5';
const char Id6 = '6';
const char Id7 = '7';

/*********************************************************************************************
```

```
* 函数: void TaskStart(void *Id).
* 描述: 任务起始函数.
*********************************************************************************************/
void TaskStart(void *Id)
{
     Init_Timer4();

     /*create the first Semaphore in the pipeline with 1 to get the task started. */
     UART_sem = OSSemCreate(1);
     kbd_sem =  OSSemCreate(0);
     num1_sem =     OSSemCreate(0);
     num2_sem =     OSSemCreate(0);
     show_sem =     OSSemCreate(0);

     /*create the tasks in uC/OS and assign decreasing priority to them */
     OSTaskCreate(Task1, (void *)&Id1, &Stack1[STACKSIZE - 1], 2);
     OSTaskCreate(Task2, (void *)&Id2, &Stack2[STACKSIZE - 1], 3);
     OSTaskCreate(Task3, (void *)&Id3, &Stack3[STACKSIZE - 1], 4);
     OSTaskCreate(GetNumTask, (void *)&Id4, &GetNumStack[STACKSIZE - 1], 5);
     OSTaskCreate(CaculTask, (void *)&Id5, &CaculStack[STACKSIZE - 1], 6);
     OSTaskCreate(ShowTask, (void *)&Id6, &ShowStack[STACKSIZE - 1], 7);

     OSTaskDel(OS_PRIO_SELF);   // Delete current task
}
```

```
/*********************************************************************************
* name:     keyboard_test
* func:     test keyboard
* para:     none
* ret:      none
* modify:
* comment:
*********************************************************************************************/
void keyboard_test(void)
{
    UINT8T ucChar;

//  uart_printf("\n Keyboard Test Example\n");
//  keyboard_init();
//  while(1)
```

```
16.     {
17.         if(g_nKeyPress==1)
18.        {
19.         g_nKeyPress = 0;
20. //      while(g_nKeyPress == 0);
21.         iic_read_keybd(0x70, 0x1, &ucChar);                    // get data from
    Key(register of ZLG7290)
22.         if(ucChar != 0)
23.         {
24.             ucChar = key_set(ucChar);           // key map for Emsbc2410
25.             if(ucChar < 10) ucChar += 0x30;
26.             else if(ucChar < 16) ucChar += 0x37;
27.             if(ucChar < 255)
28.             {
29.                 uart_sendstring("press key  ");
30.                 uart_sendstring(&ucChar);
31.                 uart_sendstring("\r\n");
32.                 PRESS_KEY = ucChar;
33.             }
34.             if(ucChar == 0xFF)
35.             {
36.                 uart_sendstring(" press key FUN (exit now)\n\r");
37.                 return;
38.             }
39.             PRESS_KEY = ucChar;
40.             OSSemPost(kbd_sem);
41.          }
42.        }
43.     }
44.
45. }
```

3. 简易计算器的多任务同步

在这一步骤中，我将实现计算任务的三个子任务之间的通信管理，即利用信号量机制实现子任务之间的同步关系。

- 在 GetNumTask 中，首先对 kbd_sem 信号量进行 P 操作，检测是否有用户键盘中断输入，如果存在键盘中断输入，则检测输入的字符，在本次实验中，由于键盘中并未提供除法的实现，我们指定输入字符为“D”表示除法操作。因此首先进行除法运算符的转换，再进行检测操作数与运算符，每次处理完毕后都对 show_sem 进行 V 操作，即通知 LCD 显示任务可以再次刷新屏幕；
- 对操作数 1 和操作数 2 输入的处理，是通过分别检测运算符和“=”来实现输入结束判断的，在这里同时进行了 num1_sem,num2_sem 的 V 操作，以便及时通知计算任务进行执行。
- 在计算任务中，首先进行 num1_sem,num2_sem 的 P 操作，检测输

入操作数是否完毕，在这里我们有一个特殊处理，当 num2_sem 完毕时，我们认为运算符也已经输入完毕，因此不再额外进行操作符的判定。计算任务处理完毕后，再次对 show_sem 进行 V 操作，通知输出任务将数据结果进行输出。

- 除法除 0 的判定

  在计算任务中，我对运算符首先进行判定，如果是除法，检测第二源操作数是否为 0，若为 0 则标记除零异常。在输出任务中，检测除零异常标志，如果存在异常，将结果输出为“Error”；如果不存在异常，则正常输出运算结果。

```
/*********************************************************************************
* 函数: void Task3(void *Id).
* 描述: show char.
*********************************************************************************/
void ShowTask(void *Id)
{
    int cnt = 1;
    while(1)
    {
        OSSemPend(show_sem, 0, &err);
        if (SHOW_CHAR == 'C'){
           Lcd_Clear(0);
           continue;
        }
        else{
           Lcd_DspHz24(100, 20, 0x1f, "李志豪的计算器");
           Lcd_Draw_HLine(10, 500, 40, 0x1f, 2);
           Lcd_DspAscII8X16(10,60,0x1f, "ID: 21009290012");
           Lcd_DspAscII8X16(500,60,0x1f, "Name: Zhihao Li");
           Lcd_Draw_HLine(10, 500, 80, 0x1f, 2);
           if(errorInfo == 1)
               Lcd_DspAscII8X16(10*cnt,90,0x1f, SHOW_CHAR);
           else
               Lcd_DspAscII8X16(10*cnt,90,0x1f, "Error");
           cnt += 1;

           Lcd_DspHz24(100, 200, 0x1f, "我的计算器");
        }
    }
}

/*********************************************************************************
* 函数: void Task3(void *Id).
* 描述: show char.
*********************************************************************************/
```

```
void GetNumTask(void *Id)
{   int num = 0;
    while(1)
    {
         OSSemPend(kbd_sem, 0, &err);
         if (PRESS_KEY == 'D'){
              PRESS_KEY = '/';
         }
         else if(PRESS_KEY >= '0' && PRESS_KEY <= '9')
         {
            num = num * 10 + PRESS_KEY - '0';
         }
         else if (PRESS_KEY == '+' || PRESS_KEY == '-
  ' || PRESS_KEY == '*' || PRESS_KEY == '/'){
             Num1 = num;
             opt = PRESS_KEY;
             OSSemPost(num1_sem);
             num = 0;
         }
         else if (PRESS_KEY == '='){
             Num2 = num;
             OSSemPost(num2_sem);
             num = 0;
         }
         SHOW_CHAR = PRESS_KEY;
         OSSemPost(show_sem);
    }
}
/*********************************************************************************
* 函数: void Task3(void *Id).
* 描述: show char.
*********************************************************************************/
void CaculTask(void *Id)
{
    int rs = 0;
    char c[20];
    int i;
    while(1){
        OSSemPend(num1_sem, 0, &err);
        OSSemPend(num2_sem, 0, &err);
        switch(opt){
            case '+':
                rs =  Num1 + Num2;
                break;
```

```
            case '-':
                rs =  Num1 - Num2;
                break;
            case '*':
                rs =  Num1 * Num2;
                break;
            case '/':
                if (Num2==0)
                    errorInfo = 1;
                else
                    rs =  Num1 / Num2;
                    break;

        }
        uart_sendstring("  The result = ");
        sprintf(c,"%d",rs);
        uart_sendstring(c);
        uart_sendstring("\r\n");
        for(i = 0; i < strlen((const char*)c); i++){
            SHOW_CHAR = c[i];
            OSSemPost(show_sem);
         }
         SHOW_CHAR = '\n';
         OSSemPost(show_sem);
    }
}
```

七、 心得体会

在这次综合性实验中，基于 μCOS-II 实现了信号量机制控制的多任务的管理，并通过这种管理机制，实现了一个简易的计算器。这次实验经历，将课堂上讲的理论知识综合运用到了实践当中，锻炼了我编写多任务嵌入式程序的能力，为后续职业发展打下了坚实的基础。