一、实验目的

1. 了解单片机 I/O 口的工作原理
2. 了解矩阵键盘的工作原理。掌握获取矩阵键盘的输入的方法。
3. 熟悉 51 单片机开发环境。掌握调试方法。

二、实验任务

用八位七段数码管进行流水灯设计。要求

1) 按键改变流水灯显示方式;
2) 要求不少于六种流水灯模式;
3) 需要包含以下几种模式:
   a) 循环左移,
   b) 循环右移,
   c) 数字递增,
   d) 按键改变数字.

三、实验完成效果

完成按键切换流水灯模式的功能，共有六种模式:

1) 循环左移：按下第一行第一列按键，进入循环左移模式。显示效果为 01234567->12345670->23456701->……->70123456->01234567。
2) 循环右移：按下第一行第二列按键，进入循环右移模式。显示效果为 01234567->70123456->67012345->……->12345670->01234567。
3) 数字递增(十进制)：按下第一行第三列按键，进入数字递增模式。显示效果为 00000000->00000001->00000002->……->00000009->00000010->00000011->……
4) 按键置数：按下第二行第一列按键，进入按键置数模式。此时 LED 显示灯仅最后一位亮起，待用户通过矩阵按键输入八位数字后，八位 LED 从左到右显示用户输入的八个数字。此时十六个矩阵按键从左到右、从上到下依次对应 0,1,2,3……D,E,F。
5) 数字递增(十六进制):按下第一行第四列按键，进入数字递增（十六进制）模式。此模式下，若用户先通过 4）按键置数模式输入了八位数字，则递增的初始值即为这八位数字构成的十六进制数；若用户此前未进入按键置数模式，则初始值为 0。以用户输入 0123BCDE 为例，显示效果为: 0123BCDE->0123BCDF->0123BCE0->0123BCE1->……

6) 循环左移（双灯）：按下第二行第三列按键，进入循环左移（双灯）模式。此模式下同一时刻仅有两个七段数码管亮起。显示效果为：(空空空 7 空空空 7)->(空空 7 空空空 7 空)->(空 7 空空空 7 空空)-> (7 空空空 7 空空空)->(空空空 7 空空空 7)。

四、程序流程及原理

4.1 主函数流程图：

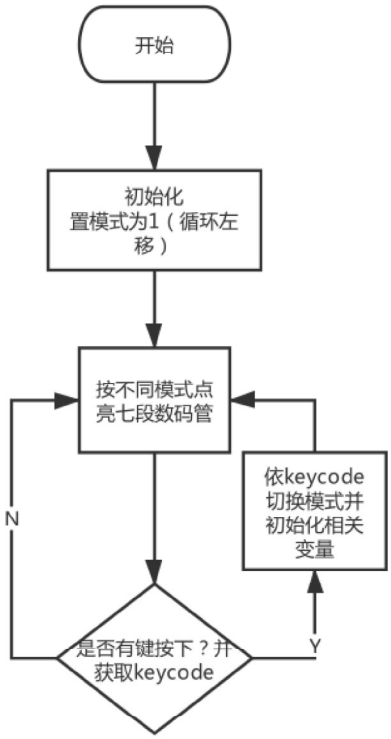

图 1 主函数流程图

主函数主要由初始化部分和循环部分组成。初始化部分包括设置初始循环模式、设置各变量初始值等。循环部分同操作系统的死循环一样，循环部分主要负责以轮询方式调用获取按键的函数和依不同模式点亮七段数码管的函数。

4.2 获取按键流程图及原理：

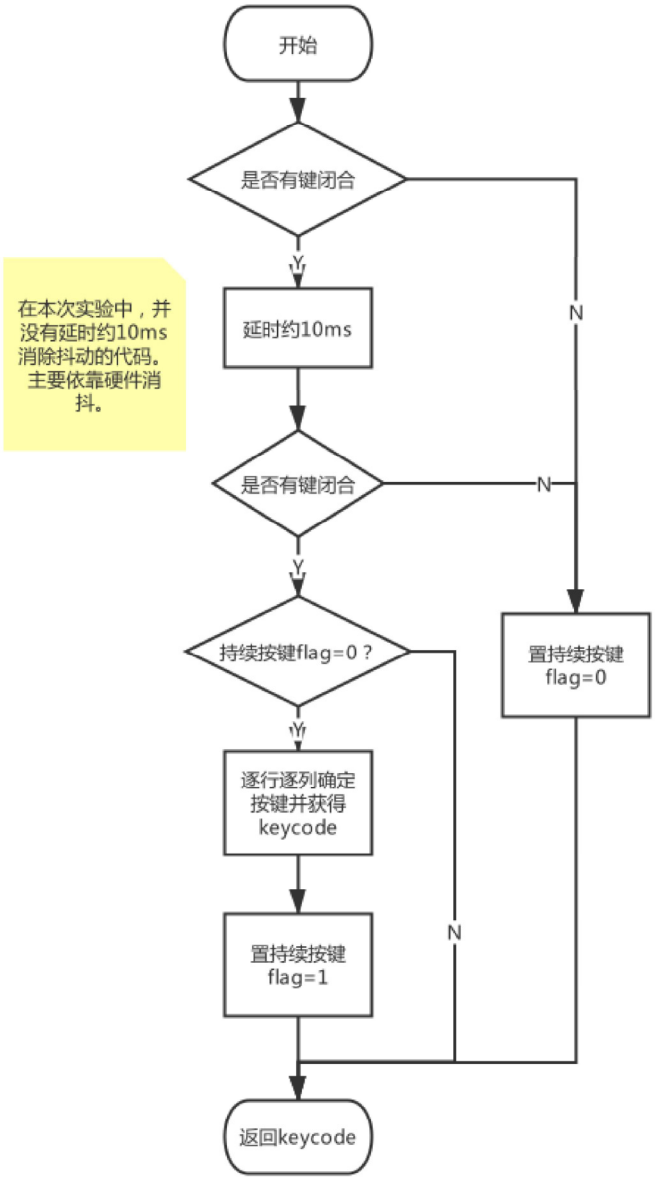

图 2 获取按键流程图

其中获取键码部分要结合矩阵键盘原理，矩阵键盘与 CPU 引脚连接关系类图 3 所示：

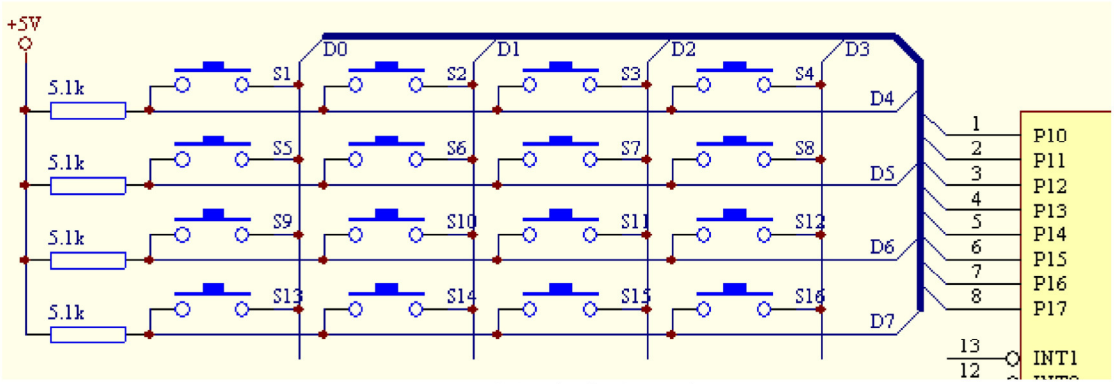

图 3 矩阵键盘电路连接图

则可用如下方法获得键码：

1）使列线 D0 ~ D3 都输出 0，检测行线 D4 ~ D7 的电平。如果 D4 ~ D7 上的电平全为高，则表示没有键被按下。如果 D4 ~ D7 上的电平不全为高，则表示有键被按下。

2）如果没有键闭合，就返回扫描。如果有键闭合，在进行逐列扫描，找出闭合键的键号。先使 D0=0，D1 ~ D3=1，检测 D4 ~ D7 上的电平，如果 D4=0，表示 K1 键被按下；同理，如果 D5 ~ D7=0，分别表示 K5、K9、K13 键被按下；如果 D4 ~ D7=1，则表示这一列没有键被按下。再使 D1=0，D0、D2、D3 为 1，对第二列进行扫描，这样依次进行下去，直到把闭合的键找到为止。

本次试验中使用的矩阵键盘和该电路连接图略有出入，但原理一致：首先确定是否有键闭合，然后逐一扫描以进一步确定是哪一键闭合。

4.3 按不同模式点亮七段数码管的流程图及原理

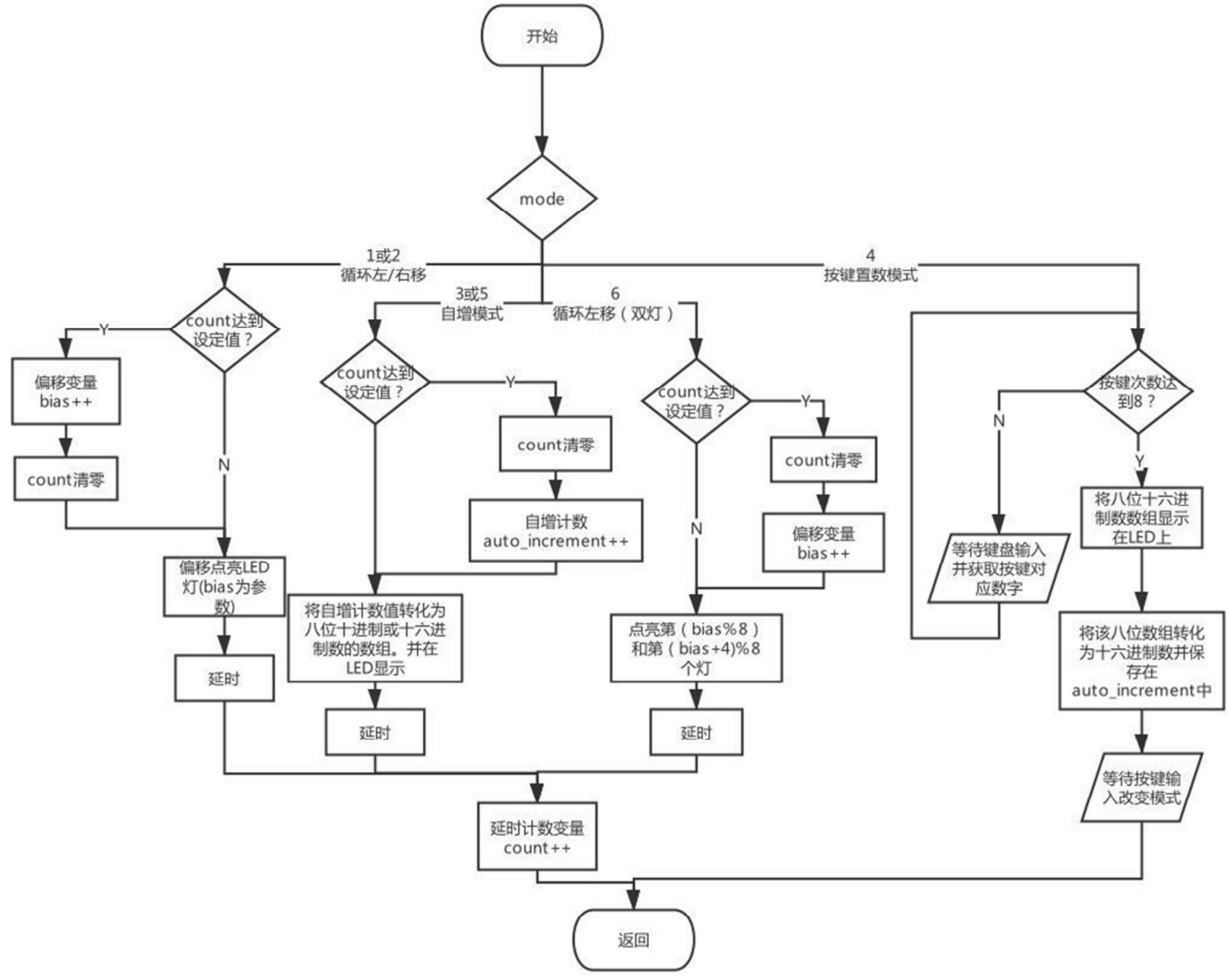

图 4 按不同模式点亮七段数码管流程图

注意：图 4 流程图中并未体现模式切换时相关变量的初始化。

七段数码管工作原理：

七段数码管工作模式分为静态显示和动态显示。静态显示就是单片机将所要显示的数据送出去后，数码管始终显示该数据(不变)，到下一次显示时，再传送一次新的显示数据。静态显示方式的优点是显示的数据稳定，无闪烁，占用 CPU 时间少。其缺点是由于数码管始终发光，功耗比较大。动态扫描显示用分时的方法轮流控制各个数码管的 COM 端，使各个数码管轮流点亮。在轮流点亮数码管的扫描过程中，每位数码管的点亮时间极为短暂。但由于人的视觉暂留现象及发光二极管的余辉，给人的印象就是一组稳定的显示数据。动态显示占用 CPU 较多但当显示位数较多时，动态扫描显示可以节省大量 I/O 管脚。本次试验中只使用动态扫描显示方法。

动态扫描方法是用一个接口电路把所有数码管的 8 个笔划段 a～g 和 dp 同名端连在一起,而每一个数码管的公共极 COM 各自独立地受一条 I/O 线控制。CPU 向字段输出口送出字形码时，所有数码管接收到相同的字形码。但究竟是哪个数码管亮，则取决于 COM 端，COM 端与单片机的 I/O 口相连接，由单片机输出位码到 I/O 控制何时哪一位数码管亮。

由于七段数码管十分常用，此处不再赘述管脚排列、共阴共阳极电路等。

五、资源使用情况：

5.1 存储器使用情况：

Program Size: data=50.0 xdata=0 code=1912

5.2 按键使用情况:

16 个按键全部使用。其中 6 个按键用来切换模式。在置数模式下，全部按键用来做数字输入。

5.3 其他资源使用情况:

并未使用定时器、中断等其他资源。

六、源码：

```
#include <REG51.H>                 /* special function register declarations   */
#include <absacc.h>
#include <stdio.h>


#define uchar unsigned char
unsigned char code led_table[]={                              //七段数码表
    0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f,0x00,0x08,0x40,0x79  };

unsigned char code led_table_16[]={                           //七段数码表，0至F
    0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f,0x77,0x7c,0x39,0x5e,0x79,0x71,0x0
0};

 void delay(unsigned char  p)                 //延时函数
{
    unsigned char  i,j;
```

```
    for(;p>0;p--)
        for(i=11;i>0;i--)      //
            for(j=10;j>0;j--);
}
 unsigned char getkeycode(void)    /*键盘扫描函数，返回获得键码*/
{
    unsigned char line=0x00;         /*行码*/
    unsigned char col=0x00; /*列码*/
    unsigned char scancode=0x01;     /*行扫描码*/
    unsigned char keycode;           /*键号*/

    XBYTE[0x8000]=0xff;
    col=XBYTE[0x8000]&0x0f;       /*从列端口读入四位列码*/
    if (col==0x00){
        keycode=0x00;
    }else
    {
        while((scancode&0x0f)!=0) /*取scancode的低四位，没变为全0，循环*/
        {
            line=scancode;                 /*行号*/
            XBYTE[0x8000]=scancode;      /*给行赋扫描码，第一行为0x01*/
            if((XBYTE[0x8000]&0x0f)==col) /*检测按键所在的行跳出循环*/
                break;
            scancode=scancode<<1;          /*行扫描码左移一位，转下一行*/
        }
        col=col<<4;                       /*把行码移到高四位*/
        keycode=col|line;
    }
    return keycode;
}


unsigned char key_scan(){
    unsigned char xdata *P;
    uchar i,line;
    uchar key_value=0;
    line = 0x01;
    P=0x8000;
    *P=0xFF;                          //赋值FF
    key_value=P0;
    if(key_value){       //说明有按键按下
        delay(10);
        for(i=0;i<4;i++)
        {             //
            P=0x8000;
            *P=line<<i;                           //赋值行值
            P0=0xFF;
            key_value=P0;
            if(key_value)
            return(key_value+i*4);
        }
        return 0;
    }
    else
        return 0;
```

```
}
unsigned char keycode_translation(unsigned char keycode){
    switch(keycode){
        case 0x11:return 0x00;break;                /*第一行第1列*/
        case 0x21:return 0x01;break;               /*第一行第2列*/
        case 0x41:return 0x02;break;
        case 0x81:return 0x03;break;               /*第一行第4列*/

        case 0x12:return 0x04;break;              /*第二行第1列*/
        case 0x22:return 0x05;break;             /*第二行第2列*/
        case 0x42:return 0x06;break;             /*第二行第3列*/
        case 0x82:return 0x07;break;            /*第二行第4列*/
        case 0x14:return 0x08;break;                 /*第三行第1列*/
        case 0x24:return 0x09;break;               /*第三行第2列*/
        case 0x44:return 0x0A;break;               /*第三行第3列*/
        case 0x84:return 0x0B;break;               /*第三行第4列*/

        case 0x18:return 0x0C;break;              /*第四行第1列*/
        case 0x28:return 0x0D;break;             /*第四行第2列*/
        case 0x48:return 0x0E;break;             /*第四行第3列*/
        case 0x88:return 0x0F;break;            /*第四行第4列*/
        default:return 0x00;break;
    }

}

void shift_left_rotate(int bias){
    int i = 0;
    int temp - 0;
    for(i = 0;i<8 ;i++){
        // XBYTE[0x9000] = 0x00;
        temp = (i+bias)%8;
        XBYTE[0x8000] = 0x80>>i;
        XBYTE[0x9000] = led_table[temp];
        delay(10);
    }

}

void main (void)
{
    int i=0;
    int mode = 0;

    int count_divider = 0;
    int count_max = 20;
    int nonsense = 0;
    int j = 0; //mode 1,2
    unsigned char key_input_index = 0; //mode 4
    unsigned char key_input = 0; //mode4
    unsigned char load_values[8]; //mode 4
    unsigned char keycode = 0x00; // mode4
    unsigned char key_input_break_flag = 0; //mode 4
    unsigned char increment_load_init_flag = 0; //mode5
    unsigned char xdata *P;
```

```
unsigned long auto_increment_value = 0; //mode 3
unsigned int  value_for_i = 0; //mode 3
unsigned long remain; // mode 3
int temp;
while (1) {
    i=0;
    count_divider++;
    if(count_divider == count_max)
    {
        if(mode == 1 || mode == 2 || mode == 7){
            j++;
            if(j == 16){
                j = 0;
            }
        }else{
            if(mode == 3 || mode == 5){
                auto_increment_value += 1;
            }
        }
        count_divider = 0;
    }

    for(i=0;i<8;i++)
    {
        delay(1);
        keycode = getkeycode();
        switch(keycode){
            case 0x11:
                mode-1;
                j = 0;
                count_divider = 0;
                break;                  /*第一行第1列*/
            case 0x21:
                mode=2;
                j = 0;
                count_divider = 0;
                break;                /*第一行第2列*/
            case 0x41:
                count_divider = 0;
                auto_increment_value = 0;
                mode=3;
                break;                /*第一行第3列*/
            case 0x81:                              //读取8个数
                mode=4;
                key_input_index = 0;
                key_input_break_flag = 1;
                while(1){
                    if(key_input_index == 8){
                        break;
                    }

                    key_input = getkeycode();
                    if(key_input){
                        if(key_input_break_flag == 0){
                            key_input_break_flag = 1;
                            load_values[key_input_index] =
```

```
keycode_translation(key_input);
                                XBYTE[0x8000] = 0x80>>key_input_index;
                                XBYTE[0x9000] =
led_table_16[load_values[key_input_index]];
                                key_input_index ++;
                            }
                        } else {
                            key_input_break_flag = 0;
                        }
                    }
                    break;              /*第一行第4列*/
                case 0x12:
                    mode=5;
                    auto_increment_value = 0;
                    increment_load_init_flag = 0;
                    break;              /*第二行第1列*/
                case 0x22:mode=6;break;             /*第二行第2列*/
                case 0x42:mode=7;
                    mode=7;
                    j = 0;
                    count_divider = 0;
                    break;             /*第二行第3列*/
                case 0x82:mode=8;break;           /*第二行第4列*/
                case 0x14:mode=1;break;                 /*第三行第1列*/
                case 0x24:mode=2;break;                /*第三行第2列*/
                case 0x44:mode=3;break;                /*第三行第3列*/
                case 0x84:mode=4;;break;                /*第三行第4列*/

                case 0x18:mode=5;break;                /*第四行第1列*/
                case 0x28:mode=6;break;               /*第四行第2列*/
                case 0x48:mode=7;break;               /*第四行第3列*/
                case 0x88:mode=8;break;           /*第四行第4列*/
                default: break;
            }



            switch(mode){
                case 1:
                    shift_left_rotate(j);
                    // 以下内容尚未改成case1中子函数调用的方式。但原理大同小异。
                case 2:
                    temp = (i+16-j)%8;
                    XBYTE[0x9000] = 0x00;
                    XBYTE[0x8000] = 0x80>>i;
                    XBYTE[0x9000] = led_table_16[temp];
                    break;
                case 3:
                    if(i == 0){
                        value_for_i = auto_increment_value % 10;
                        remain = auto_increment_value / 10;
                    }else {
                        value_for_i = remain % 10;
                        remain /= 10;
                    }
                    XBYTE[0x9000] = 0x00;
```

```
                    XBYTE[0x8000] = 0x01<<i;
                    XBYTE[0x9000] = led_table[value_for_i];
                    break;
                case 4:
                    XBYTE[0x9000] = 0x00;
                    XBYTE[0x8000] = 0x80>>i;
                    XBYTE[0x9000] = led_table_16[load_values[i]];
                    break;
                case 5:
                    if(increment_load_init_flag == 0){
                        remain = 1;
                        for(i = 0;i < 8;i++){
                            auto_increment_value += load_values[7-i] * remain;
                            remain *=  16;
                        }

                        increment_load_init_flag = 1;
                    }else{
                    if(i == 0){
                        value_for_i = auto_increment_value & 0x0f;
                        remain = auto_increment_value >>4;
                    }else {
                        value_for_i = remain &0x0f;
                        remain = remain >> 4;
                    }
                    XBYTE[0x9000] = 0x00;
                    XBYTE[0x8000] = 0x01<<i;
                    XBYTE[0x9000] = led_table_16[value_for_i];
                    }

                    break;
                case 7:
                    XBYTE[0x8000] = 0x01<<i;
                    XBYTE[0x9000] = 0x00;
                    if(((j/2)%8 == i) || (j/2+4)%8 == i){
                        XBYTE[0x9000]=    led_table[mode];
                    }
                    break;
                default:
                    XBYTE[0x8000]=    0x01<<i;
                    XBYTE[0x9000]=    led_table[mode];
                break;
            }
        }
    }
}
```