实验四 MVC 程序设计思想

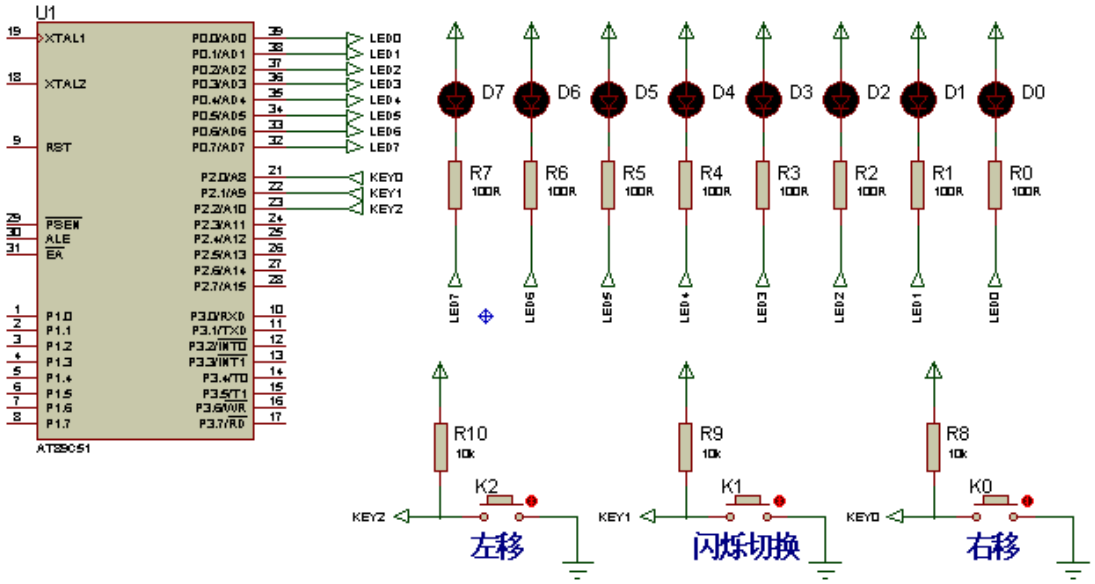
【相关知识】

1.MVC 介绍

MVC 是模型(model)-视图(view)-控制器(controller)的缩写，一种软件设计典范，用一种业务逻辑(Control)、数据、界面显示分离的方法组织代码，将业务逻辑聚集到一个部件里面，在改进和个性化定制界面及用户交互的同时，不需要重新编写业务逻辑。

现在你们还不能很好的理解，详细介绍写在代码分析一栏中。

【实验电路】



【元件清单】

元器件名称	说明
AT89C51	主控芯片
SHIPRES100R	100 欧电阻
SHIPRES10K	10K 欧电阻
LED-YELLOW	黄色 LED 灯
BUTTON	按键

【参考代码】

示例 1 三个按键控制八个 LED

```
#include "reg51.h"

void Key_Control(int keyValue);
int Key_Scan(void);
void delay_ms(int t);

int ledValue = 0xfe;
int flickerFlag = 0;

int main()
{
	while(1)
	{
		Key_Control(Key_Scan());

		P0 = ledValue;
		delay_ms(50);

		if(flickerFlag != 0)
		{
			P0 = 0xff;
			delay_ms(50);
		}
	}
}

void Key_Control(int keyValue)
{
	/* 根据捕捉的键值解析出按下的键，并作出相应的处理 */
	switch(keyValue)
	{
		/* 1111 1110 P2.0 口对应按键按下 K0 */
		case(0xfe):
			if(ledValue != 0xfe)
			{
				ledValue = ~(~(ledValue) >> 1);
			}
			break;

		/* 1111 1101 P2.1 口对应按键按下 K1 */
		case(0xfd):
```

```
                        flickerFlag = ~flickerFlag;
                        break;

                /* 1111 1011 P2.2 口对应按键按下 K2 */
                case(0xfb):
                        if(ledValue != 0x7f)
                        {
                                ledValue = ~(~(ledValue) << 1);
                        }
                        break;
        }
}

int Key_Scan(void)
{
        /* 保存按键键值的变量，默认值 0xff 表示没有按键按下 */
        int keyValue = 0xff;

        /* 用于松手检测的累加变量 */
        int i = 0;

        /* 这里在 1ms 前后检测两边是为了滤除按键抖动产生的尖峰脉冲 */
        if(P2 != 0xff)
        {
                delay_ms(1);

                /* 如果 1ms 前后检测都是低电平的话，就说明是真的有按键按下 */
                if(P2 != 0xff)
                {
                        /* 真的有按键按下，则将按键键值存入 keyValue 中 */
                        keyValue = P2;

                        /* 这里是松手检测，在 20ms 内按键没有放开，程序会一直停在此处 */
                        /* 倘若 50ms 期间，松开了按键，则会跳出此 while 循环 */
                        /* 也就是说：按键没有松开的话，程序不会去做其他的事情 */
                        /* 当然也有松手检测的超时时间，就是我们设置的 20ms */
                        while((i < 20) && (P2 != 0xff))
                        {
                                delay_ms(1);
                                i++;
                        }
                }
        }
```

```
        /* 返回按键的键值，通过解析这个键值，就能知道是哪一个按键按下 */
        return keyValue;
}

void delay_ms(int t)
{
        int i, j;
        for(i = t; i > 0; i--)
                for(j = 1000; j > 0; j--);
}
```

【代码分析】

使用 MVC 模式编写的代码，就可以将代码分离开来。

第一部分：模型变量。

```
int ledValue = 0xfe;
int flickerFlag = 0;
```

模型变量是将实际要表征的状态使用模型变量进行封装建模。使用定义的模型变量就可以表征出所有可能的状态。

比如说本实验中，`ledValue` 保存的是当前是哪一个 LED 亮，`flickerFlag` 则表示的是当前是否闪烁。通过这两个模型变量就可以表示出所有可能的状态。

`ledValue = 0x01; flickerFlag = 1;` 表示的是 D0 闪烁。

`ledValue = 0x01; flickerFlag = 0;` 表示的是 D0 常亮。

`ledValue = 0x04; flickerFlag = 0;` 表示的是 D2 常亮。

`ledValue = 0x20; flickerFlag = 0;` 表示的是 D5 常亮。

第二部分：视图层。

```
while(1)
{
        P0 = ledValue;
        delay_ms(50);

        if(flickerFlag != 0)
        {
                P0 = 0xff;
                delay_ms(50);
        }
}
```

视图层的作用就是取出模型变量的值，将其表征的状态呈现在显示设备上。本实验的显示设备是 LED 灯。

第三部分：控制层。

```
void Key_Control(int keyValue)
{
	/* 根据捕捉的键值解析出按下的键，并作出相应的处理 */
	switch(keyValue)
	{
		/* 1111 1110 P2.0 口对应按键按下 K0 */
		case(0xfe):
			if(ledValue != 0xfe)
			{
				ledValue = ~(~(ledValue) >> 1);
			}
			break;

		/* 1111 1101 P2.1 口对应按键按下 K1 */
		case(0xfd):
			flickerFlag = ~flickerFlag;
			break;

		/* 1111 1011 P2.2 口对应按键按下 K2 */
		case(0xfb):
			if(ledValue != 0x7f)
			{
				ledValue = ~(~(ledValue) << 1);
			}
			break;
	}
}
```

控制层的任务也很简单，他仅仅只需要去捕捉外部的控制信号，根据捕捉到的控制信号去改变模型变量的值，他并不需要去管最后效果怎么呈现，因为视图层已经搞定了。

【实验任务】

任务就是认真阅读代码。理解一下 MVC 模式。

模型最关键，建好模型，View 只使用模型值，Control 只改变模型值。View 和 Control 互不干扰。