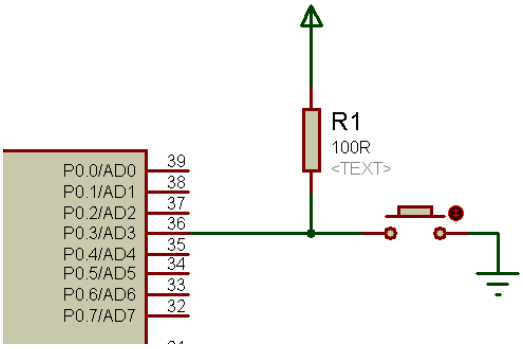
实验三 通过单片机的 I/O 捕捉外部电平信号

【相关知识】

1. 按键电路设计

一般设计按键电路的时候都会接一个上拉电阻。就像下图。

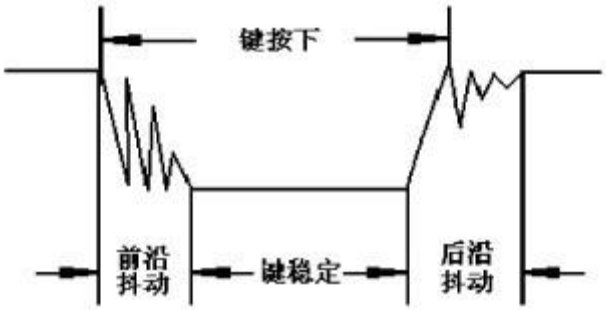


这里为什么要加一个上拉电阻，在这里的作用是给 I/O 口一个默认的电平。如果没有的上拉的话，按键按下之后，端口电平变为了低电平之后，它并不会自动变为高电平，那就意味着以后都不能捕捉到外部的低电平了。或者说认为按键其实一直处于按下状态。

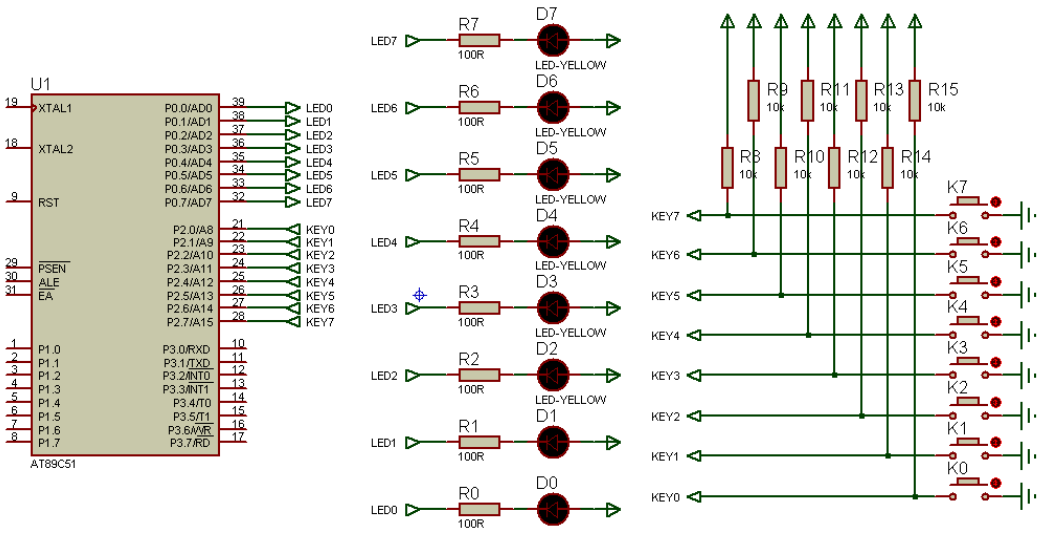
不过需要注意的是，对于 51 单片机而言，P1、P2 和 P3 在内部已经弱上拉。

2. 按键抖动现象

通常的按键所用开关为机械弹性开关，当机械触点断开、闭合时，由于机械触点的弹性作用，一个按键开关在闭合时不会马上稳定地接通，在断开时也不会一下子断开。因而在闭合及断开的瞬间均伴随有一连串的抖动，为了不产生这种现象而作的措施就是按键消抖。



【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 一个按键控制一个 LED

实验效果：按键按下 led 状态反转，原来亮就灭，原来灭就亮。

```
#include "reg51.h"

sbit button = P2^0;
sbit led = P0^0;

int i;

int main()
{
	while(1)
	{
		if(0 == button)                    // 按键延时消抖
```

```
            {
                for(i = 100; i > 0; i--);
                if(0 == button)
                {
                    led = ~led;
                }

                while(0 == button);        // 按键松手检测
            }
        }
}
```

【代码分析】

这里定义了两个位变量，其中我们用 P2.0 口捕捉按键的键值，用 P0.0 口控制 LED。

对于捕捉按键键值也是很简单的，直接读取 button 的值就可以了，这里所谓的直接读取 button 的值是体现在 `if(0 == button)` 这句代码这里，这里是不是将 button 的值取出来，然后和 0 比较。我们通过直接读取 button 变量的值来读取 P2.0 口的电平状态。

然后中间有一个这个结构：

```
if(0 == button)
{
    for(i = 100; i > 0; i--);
    if(0 == button)
    {
        /* 按键点击事件 */
    }
}
```

这里其实就是按键消抖的一种方式，通过延时一段时间二次捕捉 I/O 口的电平，若都为地电平则说明是真正的按下。因为毕竟主函数中的死循环一直在检测 P2.0 口是否是低电平，可能有的时候移动了一下硬件电路，然后就产生了一个抖动，然后因为捕捉的频率确实太频繁了，一抖动就认为是按键按下。因此通过一段很微小的时间之后再次捕捉，如果还是低电平，则说明是真正的按下。

之后还有一个松手检测的程序，`while(0 == button);` 这句代码大家想一下，如果没有这句代码的话会怎样？其实我们可以是做实验的话完全是可以删除这句代码然后在重新仿真一遍，自己去体验一下具体的现象。

如果没有这句话，则由于 `while(1)` 执行的非常快，那么你原本只想按一下的，到最后你的按键点击事件的代码去被执行了很多次，因为只要你按下，`while(1)`就成立，成立就会再次执行按键点击事件。

然而加上 `while(0 == button);` 这一句之后，只有放手后才会退出点击事件，避免了点击事件的重复执行。

【实验任务】

使用 8 个按键控制 8 个 LED 灯，按键每按一次 LED 状态就翻转一次。