实验二 使用单片机的 I/O 实现流水灯电路

【相关知识】

1.数字电路基础

以前我们接触到的电压都是具体的电压值，比如说 0.1V、3.3V、2.5V 这样的。但是对于数字电路元件来说，它们只能识别数字信号 0 和 1，那什么是数字信号呢？

我们把 0-0.8V 之间的电压值认为是数字信号 0，也就我们所谓的低电平。把大于 2.0V 以上的电压值认为是数字信号 1，高电平。

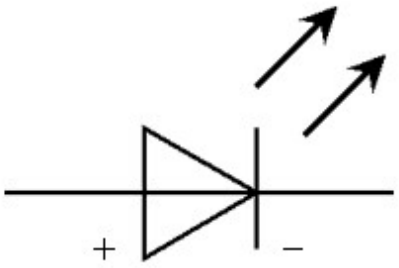
标准 TTL 电平为，输入高电平最小 2V，输出高电平最小 2.4V,典型值 3.4V。输入低电平最大 0.8V，输出低电平最大 0.4V，典型值 0.2V。

2.进制转换

十进制	十六进制	二进制
00	0	0000
01	1	0001
02	2	0010
03	3	0011
04	4	0100
05	5	0101
06	6	0110
07	7	0111
08	8	1000
09	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

3.发光二极管

发光二极管简称为 LED。左边是实际元件，右边是电路符号。



二极管具有单向导电性，电流只能从 + 到 – ，并且二极管的导通电压为 0.7V，发光二极管是一个特殊的二极管，在导通时还会发光。

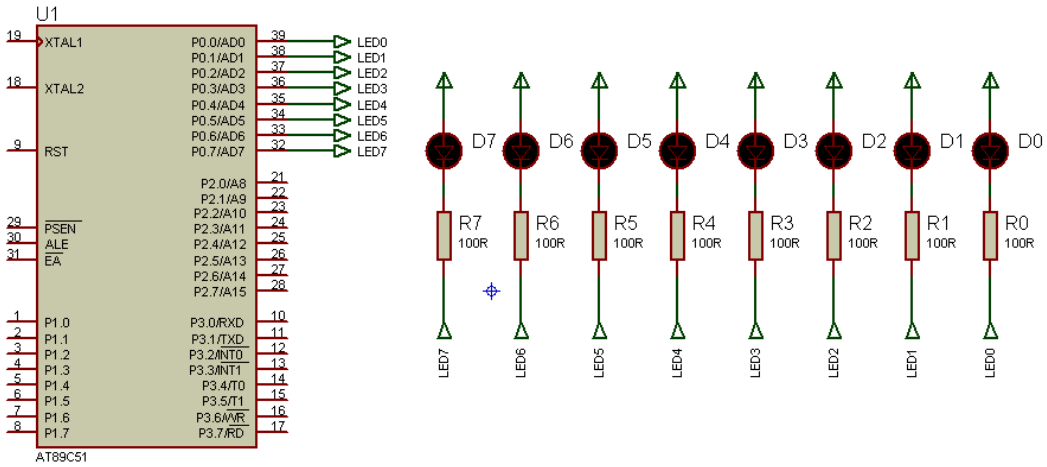
也就是说，当 LED 的 + 极比 – 极电压高 0.7V 时，就可以点亮小灯。不能将二极管直接接在 3.3V 和 GND 之间，二极管的的耐压值是 0.7V，所以一般都会加一个限流电阻(串联一个 100R 电阻)。

4.单片机 C 语言的新数据类型

sfr 特殊功能寄存器也是一种扩充数据类型，占用一个内存单元，值域为 0～255。利用它可以访问 51 单片机内部的所有特殊功能寄存器。如用 sfr P1 = 0x90 这一句定 P1 为 P1 端口在片内的寄存器，在后面的语句中我们用以用 P1 = 255（对 P1 端口的所有引脚置高电平）之类的语句来操作特殊功能寄存器。

sbit 可录址位也是 C51 中的一种扩充数据类型，利用它可以访问芯片内部的 RAM 中的可寻址位或特殊功能寄存器中的可寻址位。如先前我们定义了 sfr P1 = 0x90; 因 P1 端口的寄存器是可位寻址的，所以我们可以定义 sbit led = P1^1;意思是定义 led 为 P1 中的 P1.1 引脚。同样我们可以用 P1.1 的地址去写，如 sbit led = 0x91; 这样我们在以后的程序语句中就可以用 led 来对 P1.1 引脚进行读写操作了。通常这些可以直接使用系统提供的预处理文件（如 reg51.h），里面已定义好各特殊功能寄存器的简单名字，直接引用可以省去一点时间，当然你也可以自己写自己的定义文件，用你认为好记的名字。

【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
LED-YELLOW	黄色 LED 灯

【参考代码】

示例 1 同时控制八个 LED 闪烁

```
#include "reg51.h"          // 51 相关寄存器的定义

int main()
{
    int i;                  // 用于循环的循环变量

    while(1)                // 进入死循环，单片机程序是一直运行的
    {
        P0 = 0x00;          // 点亮
        for(i = 1000; i > 0; i--);
        P0 = 0xff;          // 熄灭
        for(i = 1000; i > 0; i--);
    }
}
```

示例 2 只控制单个 LED 闪烁

```
#include "reg51.h"

sbit led = P0^0;            // 定义一个位变量

int main()
{
       int i;

       while(1)
       {
              led = 0;
              for(i = 1000; i > 0; i--);
              led = 1;
              for(i = 1000; i > 0; i--);
       }
}
```

【代码分析】

打开 `reg51.h` 头文件，如何打开?

鼠标停在 `reg51.h` 上，右键 -> open document "reg51.h"。

```
#ifndef __REG51_H__
#define __REG51_H__

/*  BYTE Register  */
sfr P0   = 0x80;
sfr TCON = 0x88;
sfr TMOD = 0x89;
sfr IE   = 0xA8;
sfr IP   = 0xB8;

/*  TCON  */
sbit TF1  = 0x8F;
sbit TR1  = 0x8E;
sbit TF0  = 0x8D;
sbit TR0  = 0x8C;

#endif
```

这个文件里面有很多的 sfr 和 sbit 结构的变量，他们的值都是 51 内部的寄存器地址值。也就是说，我们可以通过操作这些变量去操作 51 内部的寄存器。

上述实验代码中就用到了 led 来操作 P0 寄存器的位 0，而 led 来操作 P0 寄存器的位 0 又是直接控制着 P0.0 这个 I/O 口。因此可以实现代码小灯。

```
sbit led = P0^0;              // 定义一个位变量
```

如果直接操作 P0 则是同时操作硬件 P0.0-P0.7 这 8 个 I/O 口。

```
P0 = 0x00;
```

【实验任务】

任务 1. 使用 P2.1 口控制 LED 灯，要求高电平点亮，实现 LED 灯的闪烁。

任务 2. 使用 P0 口控制 8 个 LED 灯，要求低电平点亮，实现跑马灯的效果。即每次只有一个灯亮，且 8 个灯是依次的亮一下。