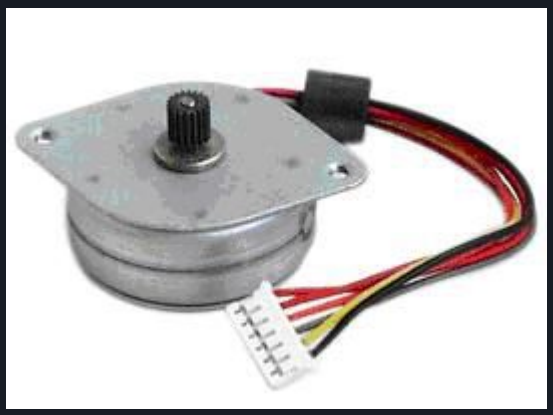
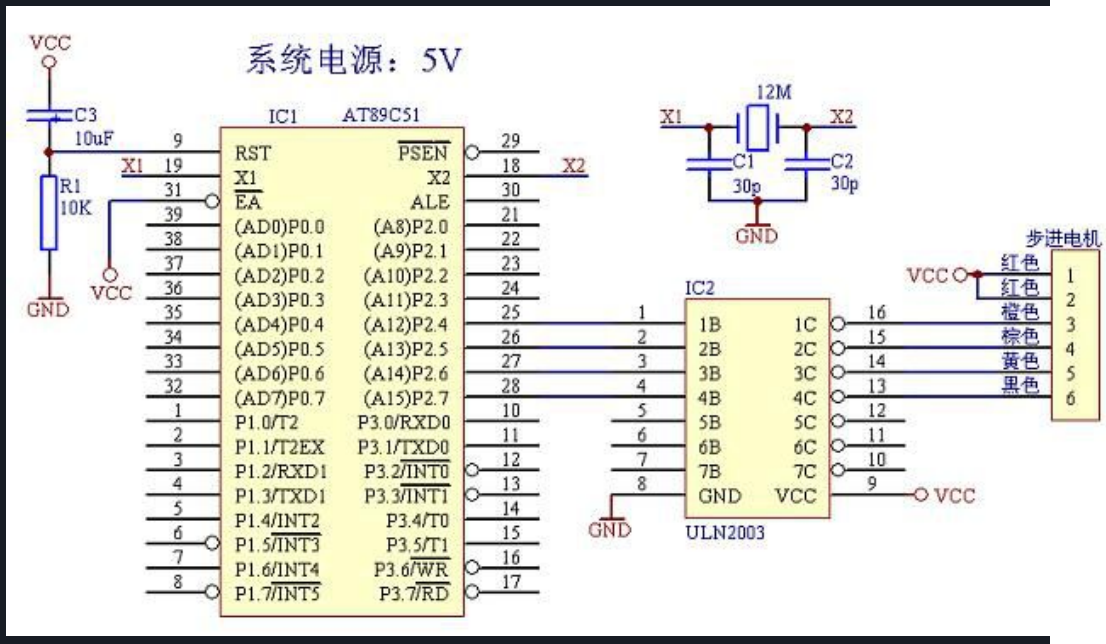
利用 51 单片机驱动步进电机的方法

这款步进电机的驱动电压 12V，步进角为 7.5 度．一圈 360 度，需要 48 个脉冲完成!!!



该步进电机有 6 根引线，排列次序如下：1:红色、2:红色、3:橙色、4:棕色、5:黄色、6:黑色。

采用 51 驱动 ULN2003 的方法进行驱动。



ULN2003 的驱动直接用单片机系统的 5V 电压，可能力矩不是很大，大家可自行加大驱动电压到 12V。

```
;*************************************************************************

;**************************步进电机的驱动*************************************

; DESIGN BY BENLADN911       FOSC = 12MHz       2005.05.19

;-------------------------------------------------------------------------
```

```
; 步进电机的驱动信号必须为 脉冲信号!!! 转动的速度和脉冲的频率成正比!!!

; 本步进电机步进角为 7.5 度 . 一圈 360 度 , 需要 48 个脉冲完成!!!

;-------------------------------------------------------------------------

; A 组线圈对应 P2.4

; B 组线圈对应 P2.5

; C 组线圈对应 P2.6

; D 组线圈对应 P2.7

; 正转次序: AB 组--BC 组--CD 组--DA 组 (即一个脉冲,正转 7.5 度)

;-------------------------------------------------------------------------

;-------------------------正转-------------------------

ORG 0000H

LJMP MAIN

ORG 0100H

MAIN:

MOV R3,#144     正转 3 圈共 144 脉冲

START:

MOV R0,#00H

START1:

MOV P2,#00H

MOV A,R0

MOV DPTR,#TABLE

MOVC A,@A+DPTR

JZ START     对 A 的判断,当 A = 0 时则转到 START

MOV P2,A
```

```
LCALL DELAY

INC R0

DJNZ R3,START1

MOV P2,#00H

LCALL DELAY1

;----------------------------反转-----------------------

MOV R3,#144      反转一圈共 144 个脉冲

START2:

MOV P2,#00H

MOV R0,#05

START3:

MOV A,R0

MOV DPTR,#TABLE

MOVC A,@A+DPTR

JZ START2

MOV P2,A

CALL DELAY

INC R0

DJNZ R3,START3

MOV P2,#00H

LCALL DELAY1

LJMP MAIN

DELAY: MOV R7,#40 步进电机的转速

M3: MOV R6,#248
```

```
DJNZ R6,$

DJNZ R7,M3

RET

DELAY1:          MOV     R4,#20      2S 延时子程序

DEL2:            MOV     R3,#200

DEL3:            MOV     R2,#250

                 DJNZ    R2,$

                 DJNZ    R3,DEL3

                 DJNZ    R4,DEL2

                 RET

TABLE:

DB 30H,60H,0C0H,90H    正转表

DB 00    正转结束

DB 30H,90H,0C0H,60H    反转表

DB 00    反转结束

END
```

51 单片机控制四相步进电机

今天从淘宝网买了一个 EPSON 的 UMX-1 型步进电机，此步进电机为双极性四相，接线共有六根，外形如下图所示：

拿到步进电机，根据以前看书对四相步进电机的了解，我对它进行了初步的测试，就是将 5 伏电源的正端接上最边上两根褐色的线，然后用 5 伏电源的地线分别和另外四根线（红、兰、白、橙）依次接触，发现每接触一下，步进电机便转动一个角度，来回五次，电机刚好转一圈，说明此步进电机的步进角度为 360/(4×5)＝18 度。地线与四线接触的顺序相反，电机的转向也相反。

此步进电机，则只需分别依次给四线一定时间的脉冲电流，电机便可连续转动起来。通过改变脉冲电流的时间间隔，就可以实现对转速的控制；通过改变给四线脉冲电流的顺序，则可实现对转向的控制。所以，设计了如下电路图：

500)this.width=500" border=0> 500)this.width=500" border=0> 500)this.width=500" border=0>

C51 程序代码为：

代码一

```
#include <AT89X51.h>

static unsigned int count;

static unsigned int endcount;

void delay();

void main(void)
```

```
{
  count = 0;
  P1_0 = 0;
  P1_1 = 0;
  P1_2 = 0;
  P1_3 = 0;
  EA = 1;              //允许 CPU 中断
  TMOD = 0x11;   //设定时器 0 和 1 为 16 位模式 1
  ET0 = 1;             //定时器 0 中断允许
  TH0 = 0xFC;
  TL0 = 0x18;      //设定时每隔 1ms 中断一次
  TR0 = 1;           //开始计数
startrun:

  P1_3 = 0;
  P1_0 = 1;
  delay();
  P1_0 = 0;
  P1_1 = 1;
  delay();
  P1_1 = 0;
  P1_2 = 1;
  delay();
  P1_2 = 0;
```

```
    P1_3 = 1;
    delay();
    goto startrun;
  }
  //定时器 0 中断处理
  void timeint(void) interrupt 1
  {
    TH0=0xFC;
    TL0=0x18; //设定时每隔 1ms 中断一次
    count++;
  }
  void delay()
  {
    endcount=2;
    count=0;
    do{}while(count<endcount);
  }
```

将上面的程序编译，用 ISP 下载线下载至单片机运行，步进电机便转动起来了，初步告捷！

不过，上面的程序还只是实现了步进电机的初步控制，速度和方向的控制还不够灵活，另外，由于没有利用步进电机内线圈之间的“中间状态”，步进电机的步进角度为 18 度。所以，我将程序代码改进了一下，如下：

代码二

```
  #include <AT89X51.h>
  static unsigned int count;
  static int step_index;
```

```
void delay(unsigned int endcount);
void gorun(bit turn, unsigned int speedlevel);
void main(void)
{
  count = 0;
  step_index = 0;
  P1_0 = 0;
  P1_1 = 0;
  P1_2 = 0;
  P1_3 = 0;

  EA = 1;             //允许 CPU 中断
  TMOD = 0x11; //设定时器 0 和 1 为 16 位模式 1
  ET0 = 1;            //定时器 0 中断允许
  TH0 = 0xFE;
  TL0 = 0x0C;   //设定时每隔 0.5ms 中断一次
  TR0 = 1;         //开始计数
  do{
    gorun(1,60);
  }while(1);
}
//定时器 0 中断处理
void timeint(void) interrupt 1
{
```

```
    TH0=0xFE;
    TL0=0x0C; //设定时每隔 0.5ms 中断一次
    count++;
}
void delay(unsigned int endcount)
{
    count=0;
    do{}while(count<endcount);
}
void gorun(bit turn,unsigned int speedlevel)
{
    switch(step_index)
    {
    case 0:
      P1_0 = 1;
      P1_1 = 0;
      P1_2 = 0;
      P1_3 = 0;
      break;
    case 1:
      P1_0 = 1;
      P1_1 = 1;
      P1_2 = 0;
      P1_3 = 0;
```

```
    break;
  case 2:
    P1_0 = 0;
    P1_1 = 1;
    P1_2 = 0;
    P1_3 = 0;
    break;
  case 3:
    P1_0 = 0;
    P1_1 = 1;
    P1_2 = 1;
    P1_3 = 0;
    break;
  case 4:
    P1_0 = 0;
    P1_1 = 0;
    P1_2 = 1;
    P1_3 = 0;
    break;
  case 5:
    P1_0 = 0;
    P1_1 = 0;
    P1_2 = 1;
    P1_3 = 1;
```

```
  break;
 case 6:
  P1_0 = 0;
  P1_1 = 0;
  P1_2 = 0;
  P1_3 = 1;
  break;
 case 7:
  P1_0 = 1;
  P1_1 = 0;
  P1_2 = 0;
  P1_3 = 1;
 }
 delay(speedlevel);
 if (turn==0)
 {
  step_index++;
  if (step_index>7)
    step_index=0;
 }
 else
 {
  step_index--;
  if (step_index<0)
```

```
            step_index=7;

          }



        }
```

改进的代码能实现速度和方向的控制，而且，通过 step_index 静态全局变量能“记住”步进电机的步进位置，下次调用 gorun（）函数时则可直接从上次步进位置继续转动，从而实现精确步进；另外，由于利用了步进电机内线圈之间的“中间状态”，步进角度减小了一半，只为 9 度，低速运转也相对稳定一些了。

但是，在代码二中，步进电机的运转控制是在主函数中，如果程序还需执行其它任务，则有可能使步进电机的运转收到影响，另外还有其它方面的不便，总之不是很完美的控制。所以我又将代码再次改进：

代码三

```
#include <AT89X51.h>

static unsigned int count;   //计数

static int step_index;   //步进索引数，值为 0－7

static bit turn;   //步进电机转动方向

static bit stop_flag;   //步进电机停止标志

static int speedlevel; //步进电机转速参数，数值越大速度越慢，最小值为 1，速度最快

static int spcount;     //步进电机转速参数计数

void delay(unsigned int endcount);   //延时函数，延时为 endcount*0.5 毫秒

void gorun();          //步进电机控制步进函数

void main(void)

{

  count = 0;

  step_index = 0;

  spcount = 0;
```

```
    stop_flag = 0;
    P1_0 = 0;
    P1_1 = 0;
    P1_2 = 0;
    P1_3 = 0;
    EA = 1;             //允许 CPU 中断
    TMOD = 0x11; //设定时器 0 和 1 为 16 位模式 1
    ET0 = 1;            //定时器 0 中断允许
    TH0 = 0xFE;
    TL0 = 0x0C;    //设定时每隔 0.5ms 中断一次
    TR0 = 1;          //开始计数
    turn = 0;
    speedlevel = 2;
    delay(10000);
    speedlevel = 1;
    do{
      speedlevel = 2;
      delay(10000);
      speedlevel = 1;
      delay(10000);
      stop_flag=1;
      delay(10000);
      stop_flag=0;
    }while(1);
```

```
}
//定时器 0 中断处理
void timeint(void) interrupt 1
{
  TH0=0xFE;
  TL0=0x0C; //设定时每隔 0.5ms 中断一次
  count++;
  spcount--;
  if(spcount<=0)
  {
    spcount = speedlevel;
    gorun();
  }
}
void delay(unsigned int endcount)
{
  count=0;
  do{}while(count<endcount);
}
void gorun()
{
  if (stop_flag==1)
  {
    P1_0 = 0;
```

```
  P1_1 = 0;
  P1_2 = 0;
  P1_3 = 0;
  return;
}
switch(step_index)
{
case 0: //0
  P1_0 = 1;
  P1_1 = 0;
  P1_2 = 0;
  P1_3 = 0;
  break;
case 1: //0、1
  P1_0 = 1;
  P1_1 = 1;
  P1_2 = 0;
  P1_3 = 0;
  break;
case 2: //1
  P1_0 = 0;
  P1_1 = 1;
  P1_2 = 0;
  P1_3 = 0;
```

```
    break;
  case 3: //1、2
    P1_0 = 0;
    P1_1 = 1;
    P1_2 = 1;
    P1_3 = 0;
    break;
  case 4:   //2
    P1_0 = 0;
    P1_1 = 0;
    P1_2 = 1;
    P1_3 = 0;
    break;
  case 5: //2、3
    P1_0 = 0;
    P1_1 = 0;
    P1_2 = 1;
    P1_3 = 1;
    break;
  case 6: //3
    P1_0 = 0;
    P1_1 = 0;
    P1_2 = 0;
    P1_3 = 1;
```

```
        break;
      case 7: //3、0
        P1_0 = 1;
        P1_1 = 0;
        P1_2 = 0;
        P1_3 = 1;
      }
      if (turn==0)
      {
        step_index++;
        if (step_index>7)
          step_index=0;
      }
      else
      {
        step_index--;
        if (step_index<0)
          step_index=7;
      }

    }
```

在代码三中，我将步进电机的运转控制放在时间中断函数之中，这样主函数就能很方便的加入其它任务的执行，而对步进电机的运转不产生影响。在此代码中，不但实现了步进电机的转速和转向的控制，另外还加了一个停止的功能，呵呵，这肯定是需要的。

步进电机从静止到高速转动需要一个加速的过程，否则电机很容易被“卡住”，代码一、二实现加速不是很方便，而在代码三中，加速则很容易了。在此代码中，当转速参数 speedlevel 为 2 时，可以算出，此时步进电机的转速为 1500RPM，而当转速参数 speedlevel 1 时，转速为 3000RPM。当步进电机停止，如果直接将 speedlevel 设为 1，此时步进电机将被“卡住”，而如果先把 speedlevel 设为 2，让电机以 1500RPM 的转速转起来，几秒种后，再把 speedlevel 设为 1，此时电机就能以 3000RPM 的转速高速转动，这就是“加速”的效果。

在此电路中，考虑到电流的缘故，我用的 NPN 三极管是 S8050，它的电流最大可达 1500mA，而在实际运转中，我用万用表测了一下，当转速为 1500RPM 时，步进电机的电流只有 90mA 左右，电机发热量较小，当转速为 60RPM 时，步进电机的电流为 200mA 左右，电机发热量较大，所以 NPN 三极管也可以选用 9013，对于电机发热量大的问题，可加一个 10 欧到 20 欧的限流电阻，不过这样步进电机的功率将会变小。