轮 趣 科 技

双路步进电机驱动 D36A
使用手册

推荐关注我们的公众号获取更新资料

版本说明：

版本	日期	内容说明
V1.0	2024/11/22	第一次发布
V1.1	2024/12/4	补充拨码开关部分的具体说明
V1.2	2025/2/6	修改拨码开关的设置说明
V1.3	2025/6/25	修改了驱动详情图，补充 Arduino 例程内容

序言

我们将通过这篇教程与大家一起学习步进电机的原理和控制，并介绍一款步进电机双路驱动模块 D36A。D36A 能同时驱动双路步进电机，驱动上板载拨码开关，可通过拨码开关对步进电机的进行细分控制以及电流控制

目录

1. 步进电机入门基础知识

步进电机是一种把电脉冲信号转换为角位移或线位移的电机。在非超载的情况下，电机的转速、停止的位置只取决于脉冲信号的频率和脉冲数，而不受负载变化的影响，即给电机加一个脉冲信号，电机则转过一个步距角。

1.1 步进电机分类

① 按磁激励方式分类

步进电机按磁激励可分为永磁式、反应式（磁阻式）和混合式。

永磁式步进电机动态性能好，输出力矩较大，能够快速响应电脉冲信号；但其步距角相对较大，通常为 7.5° 或 15° ，这限制了其在一些对精度要求极高的应用场景中的使用。

图 1-1 永磁式步进电机实物图

反应式（磁阻式）结构简单，成本较低，并且步距角可以做得比较小，一般可达 1.2° ，适用于对步距角要求不那么高、但对成本敏感的应用场景；但是其运行时噪音和震动相对较大，动态性能较低，在高速运行或对运行平稳性要求较高的场合可能不太适用。

图 1-2 反应式步进电机实物图

混合式步进电机兼具永磁式和反应式电机的优点，具有力矩大、动态性能好、步距角小、精度高等特点，是目前应用较为广泛的一种步进电机类型，其结构也相对复杂一些，制造难度及其制造成本都相对较高。

图 1-3 混合式步进电机实物图

② 按相数分类

按相数可将步进电机分为单相步进电机、两相步进电机、三相步进电机和五项及以上步进电机。随着相数的增多，步进电机的制作难度以及制作成本都会升高，目前市面上使用最多的是两相步进电机。

两相步进电机在市面上最为常见，使用比较多的有 42 步进电机和 57 步进电机两种，电机内部有两个绕组，通常标记为 A 相和 B 相，通过控制 A 相和 B 相绕组的通电顺序和时间，可以精确地控制电机的转动角度和方向。其步距角较

小，一般可以达到 1.8°，精度较高，并且启动和运行频率相对较高，在速度控制方面表现较好。

图 1-4 二相步进电机实物图

1.2步进电机原理

如图 1-5 所示，不仅电机内部同样由定子跟转子组成，定子上绕有线圈，转子为永磁体。先以单线圈通电为例，简易结构图中的黑色箭头为电流方向，电流从 A 相正极进入（A+），根据右手定则，可以得出红色线圈的磁性，上面的线圈的上面部分为 N 极，靠近转子部分为 S 极，下面的线圈靠近转自部分为 N 极，另一端为 S 极，根据“同性相斥，异性相吸”的原则，转子的 N 极会受到上面线圈的力作用，被其吸引，S 极会受到下面线圈的力作用，被其吸引，这个时候可以确定转子的位置，因为转子的两端均受力，所以这个时候的力矩会比单极性的步进电机要大。

电流切换到从 B 相正极进入，同样的，转子受到线圈产生的磁性的影响，可以将转子固定在一个位置；电流从 A-、B+端进入也是一样的道理，这样不断切换电流方向则可以改变步进电机的转动。

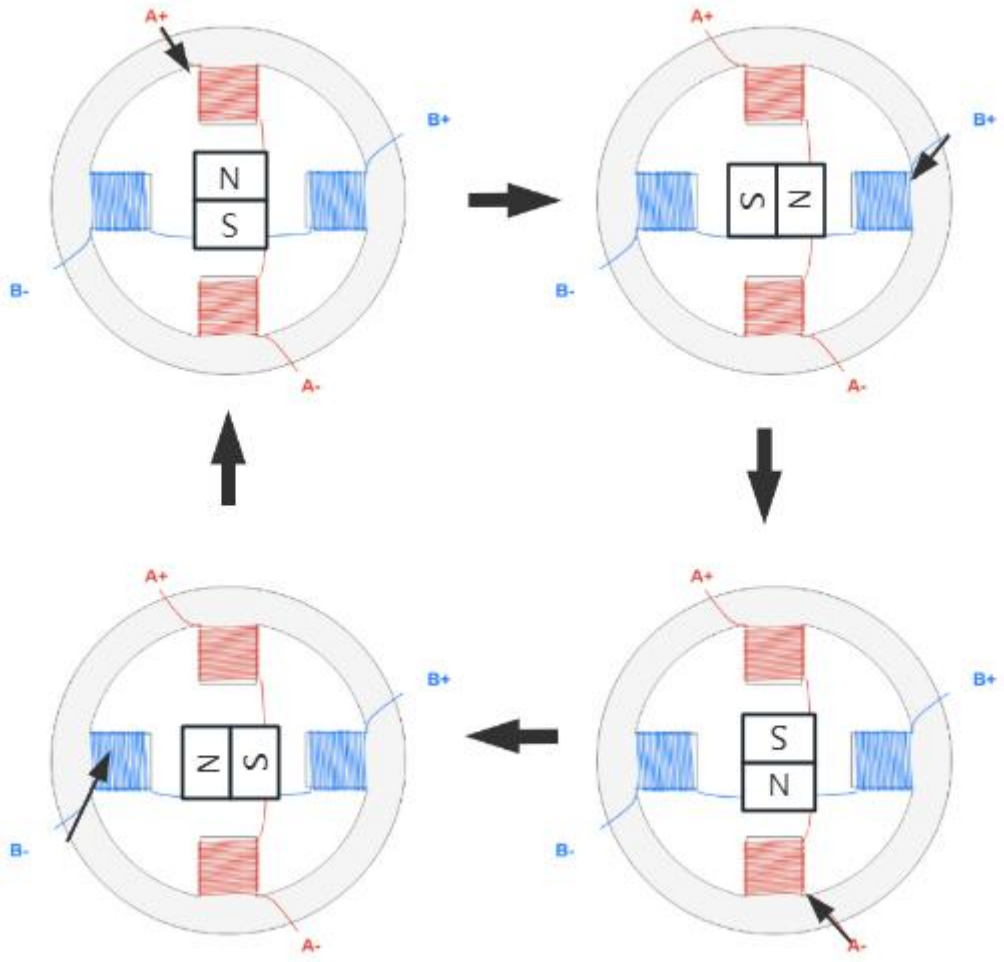

图 1-5 两相混合式步进电机单线圈通电简易结构图

如果将双线圈都通电，如图 1-6 所示，依旧以黑色箭头的方向为电流进入的方向，依次分析转子在各个线圈上受到的力作用，如果电流从 A+、B+进入线圈，假设两个线圈的电流大小完全相同，则 A+、B+两端都会有一个力作用在转子的 N 极上，这个时候对转子进行受力分析，可得转子 N 极的朝向为 45°（以 A+线圈为 0°），转子的 S 极受到 A-、B-的力作用，所以其朝向为 225°，这个时候的转子受到四个方向的力的作用，其力矩会变得更大。

当电流分别从 B+和 A-、A-和 B- 以及 B-和 A+进入，对转子进行受力分析，可以得出转子对每次增加 90°。

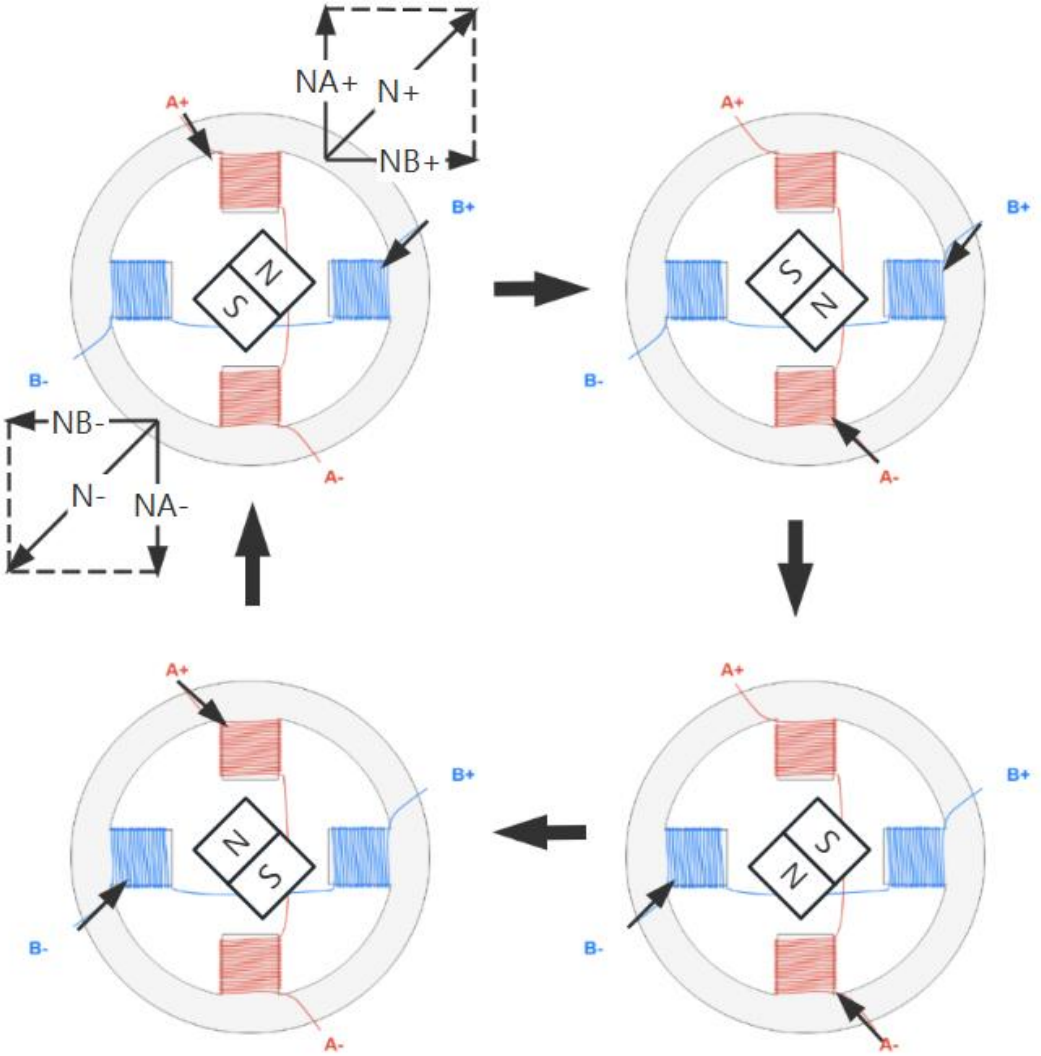

图 1-6 两相混合式步进电机双线圈通电简易结构图

以上的两种情况称之为整步驱动，如果将两者结合起来，则步进电机需要转动八个角度才能转一圈，其称之为半步驱动，如图 1-7 所示。

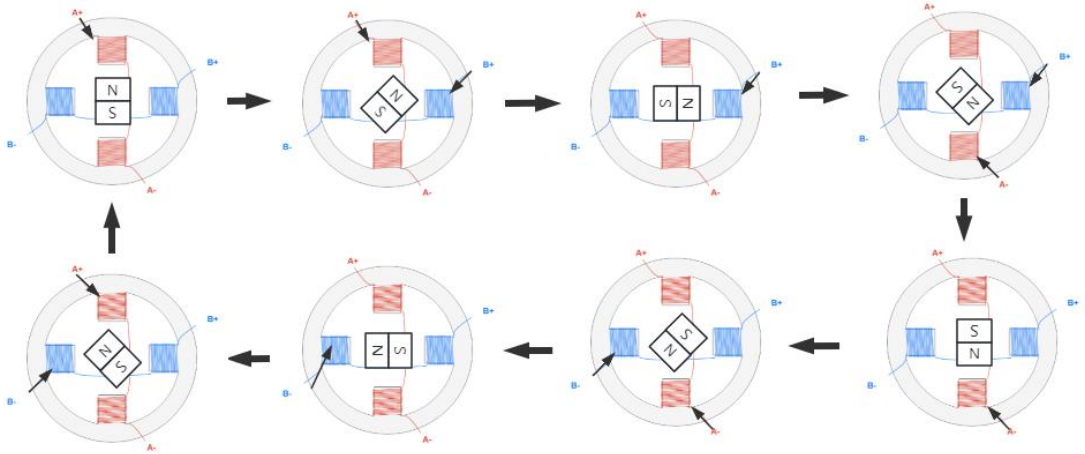

图 1-7 半步驱动示意图

这样看的话步进电机每走一步就需要跨过 45° 的距离，但是实际上步进电机内部转子上会有极齿，转子每次转动都会从上个极齿转动到下一个极齿，这样可以提升步进电机的精度，也就减小了步距角。

1.3细分控制

尽管有了极齿，步进电机的步距角相对也是较大的，当步距角不能满足要求时，则可采用细分驱动器来驱动步进电机，其原理是通过改变 AB 相的电流大小，以改变合成的磁场的夹角，从而将一步变为多步。

以二相混合式步进电机为例，将电机原有的步距角细分成若干个小步的控制方法称为细分控制，一般的二相步进电机步距角为 1.8°，那么想要让电机转动一圈则需要 360° / 1.8° = 200 个脉冲，如果使用细分驱动器，则可以改变步距角，假如细分倍数为 1/2，则原本 1.8° 的步距角变为 1.8° / 2 = 0.9°，这个时候想要让电机转动一圈则需要 360° / 0.9° = 400 个脉冲。

常见的细分倍数有：1/2，1/4，1/8，1/16，1/32，1/64，细分后电机的步距角为：步距角 = 电机原有步距角 * 细分倍数。

1.4电机实物接线图解

以我们的 42 步进电机为例，如图 1-8 所示是一个二相步进电机，可以看到该步进电机是六线的，真正使用的四根线分别为 B-、B+、A-和 A+，将电机与驱动器连接后通过驱动器输出 AB 相脉冲即可控制步进电机。

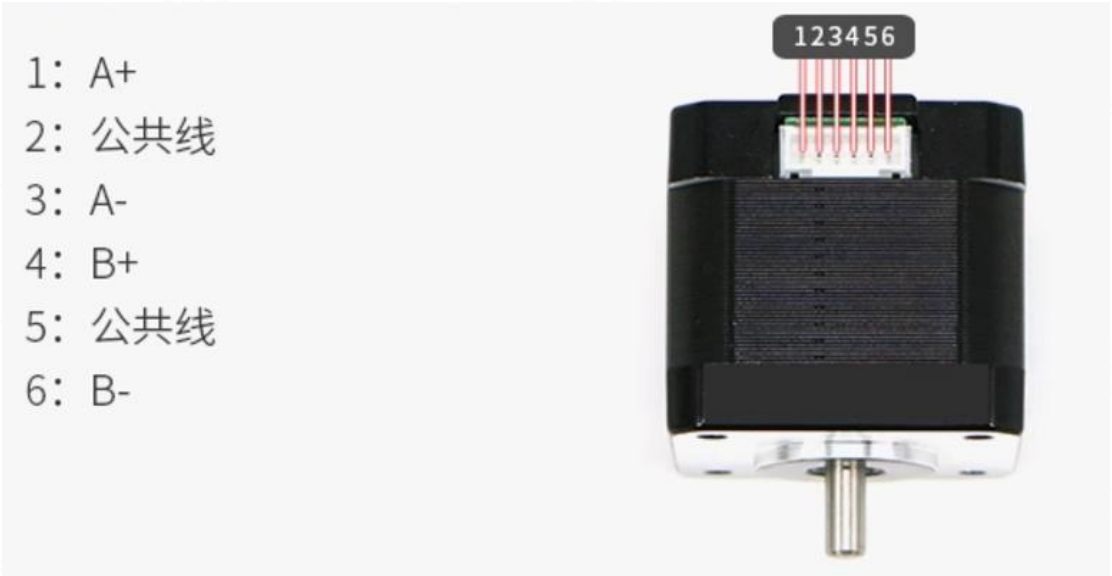

图 1-8 二相步进电机接线图解

2. D36A 双路步进电机驱动模块介绍

2. 1D36A 模块介绍

双路步进电机驱动 D36A 由两个 ATD5984 芯片、散热风扇和电源电路组成。

（1） ATD5984 芯片是一款由杭州中科微设计生产的微特步进电机驱动芯片，其内部集成了译码器，能使双极步进电机以全、半、1/4、1/8、1/16 和 1/32 步进模式工作。

（2） 含有电压测量电路，可以通过 ADC 采集并计算得到电源的电压进行监控。

（3） 引出二相步进电机标准的 4 PIN 接口，可以通过排线连接。

（4） 电源输入接口做了并联输出，可以多个模块级联使用。

（5） 含有电源开关，可以进行开启。

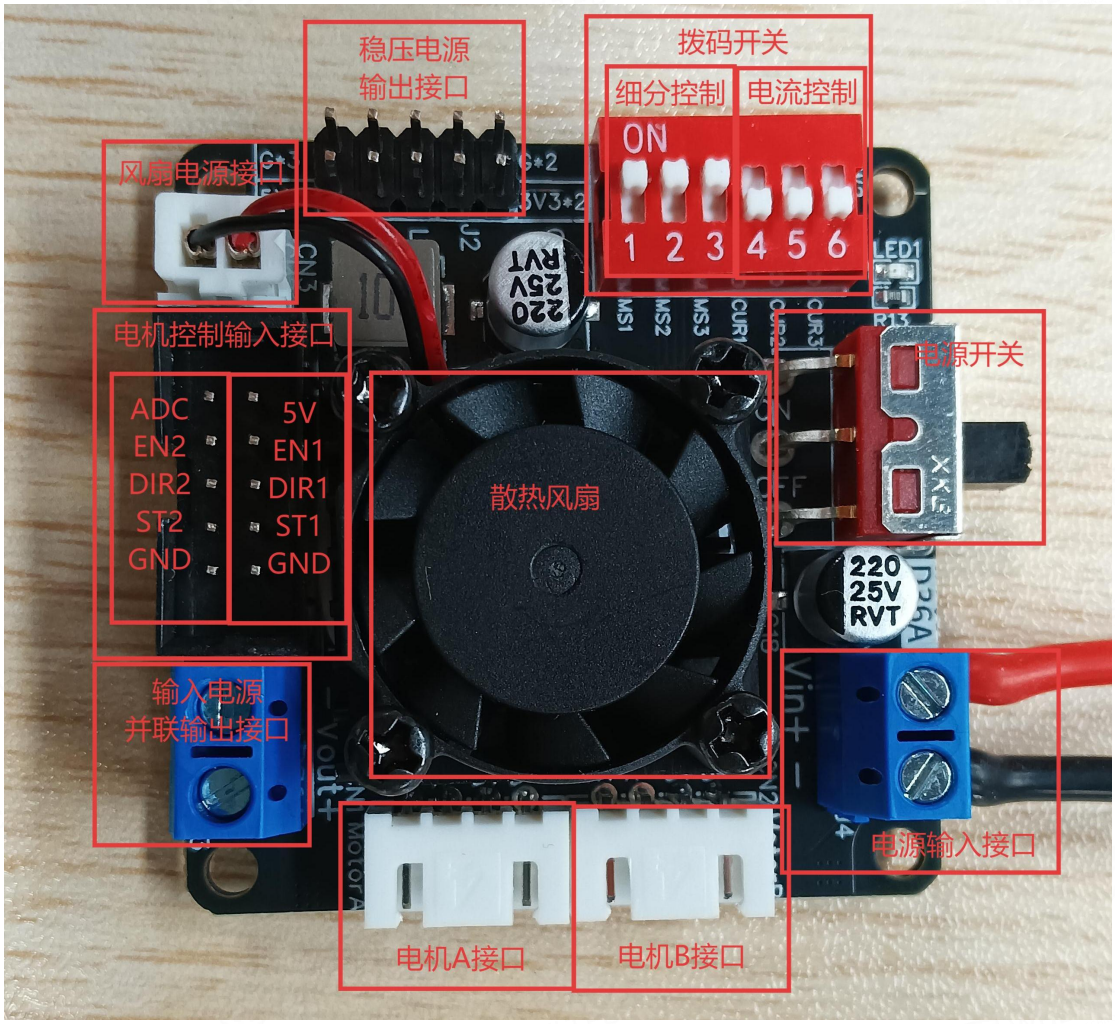

图 2-1 双路步进电机驱动详细图

2.2D36A 接口介绍与控制说明

表 3-1 D36A 接口引脚介绍

引脚	功能	引脚	功能
GND	电源地	GND	电源地
ST2	步进电机 2 STEP 信号上升沿有效，每个上升沿转动一个步距角（PWM）	ST1	步进电机 1 STEP 信号上升沿有效，每个上升沿转动一个步距角（PWM）
DIR2	步进电机 2 方向控制	DIR1	步进电机 1 方向控制
EN2	步进电机 2 使能引脚，低电平休眠高电平使能	EN1	步进电机 1 使能引脚，低电平休眠高电平使能
ADC	读取电源电压的 1/11	5V	5V 电源输出

在控制电机的时候，MS1、MS2、MS3 三个引脚控制步进电机的细分数，DIR 引脚控制步进电机转动方向，STEP 引脚输入控制信号，上升沿有效，所以在使用 ATD5984 控制步进电机的时候，STEP 引脚需要输入 PWM 信号（**具体频率范围查看下文中 3.2 部分内容**），这样才会有上升沿信号的出现，如果只输入高电平或者低电平无法产生上升沿信号，则无法控制步进电机。

2.3细分与电流设置说明

D36A 模块上有 6 个拨码开关可用，其中 123 设置细分，456 设置电流，拨到 ON 为导通，否则是断开。

图 2-2 拨码开关方向示意图

需要注意，细分设置的拨码开关 123 引脚带有上拉电阻，因此，当拨码开关

断开时，MS1、2、3 引脚对应高电平；当拨码开关导通时，MS1、2、3 对应低电平。**请注意，表格 2-2 所示的细分值设置真值表，对应原理图中 MS1、2、3 引脚的高低电平状态。**

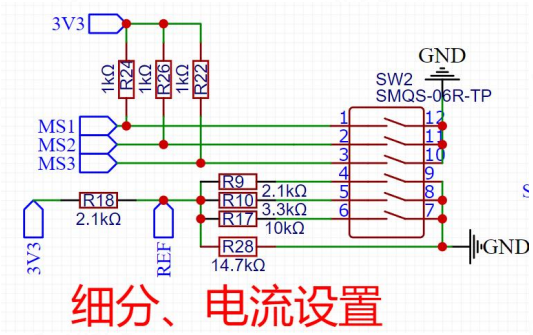

图 2-3 拨码开关部分原理图

MS1、MS2、MS3 三个引脚总共有 8 种排列组合，但是实际有用的只有表 2-2 所示的这 6 种。细分的选择需要根据自身的需求去设定，如果有高精度的需求的话，可以选择细分倍数更小一些，细分之后相对应的步距角则会更小一些，以此来达到高精度的需求。

表 2-2 MS1、MS2、MS3 对应的细分数（此处的 0 对应拨码开关导通，1 对应拨码开关断开）

MS1	MS2	MS3	细分
0	0	0	不细分
1	0	0	2 步细分
1	0	1	4 步细分
0	1	1	8 步细分
1	1	1	16 步细分
0	1	0	32 步细分

不同的步进电机可能会有不同的电机额定电流，所以 D36A 双路步进电机驱动模块使用拨码开关来控制驱动的输出电流，最大可输出 1.44A，其内部主要是通过三个拨码开关分别串联不同阻值的电阻，拨动不同的拨码开关会使等效电阻发生变化，将等效电阻与 2.1kΩ 的电阻进行串联，使用 ATD5984 的 REF 引脚检测等效电阻的分压情况，以此来控制电流的大小，具体的电流大小如表 2-3 所示。

随着电流的增大，步进电机的力矩也会有所增大，但是驱动的发热情况也会随着增大，所以在选择合适的电流值时需要考虑当前所需的力矩以及驱动板的发热情况，可以先将拨码开关都拨到 1，选择最小的电流档，再通过表 2-3 所示的

拨码开关对应的电流值调整拨码开关逐渐增加电流值，调整到力矩适合当前步进电机的电流值为止。

该设计的主要优点就是可以适配不同额定电流值的步进电机，通过拨码开关即可实现输出电流大小的调节，另外需要注意，电流值除了影响发热和力矩还会影响电池供电情况的续航时长，因此选择合适的电流值是一个综合性的工作。

表 2-3 电流与拨码开关对应表（此处的 0 对应拨码开关断开，1 对应拨码开关导通）

CUR1	CUR2	CUR3	等效电流（A）
0	0	0	1.44
0	0	1	1.22
0	1	0	0.93
0	1	1	0.83
1	0	0	0.77
1	0	1	0.70
0	1	1	0.59
1	1	1	0.55

2.4D36A 模块原理图介绍

D36A 模块包含了稳压电路，电机接口，电源输入，与输入电源并联的输出，电源开关，拨码开关、两个 ATD5984 芯片、板载 5V 稳压电路和 3.3V 线性稳压电路。可以直接通过单片机的最小系统对该驱动进行控制，同时可以采集 ADC，对模块的电源进行监测；有了电机接口，可以直接插上电机线，省去了用杜邦线逐根连接的麻烦；与输入电源的并联输出可以使多个该模块级联使用。

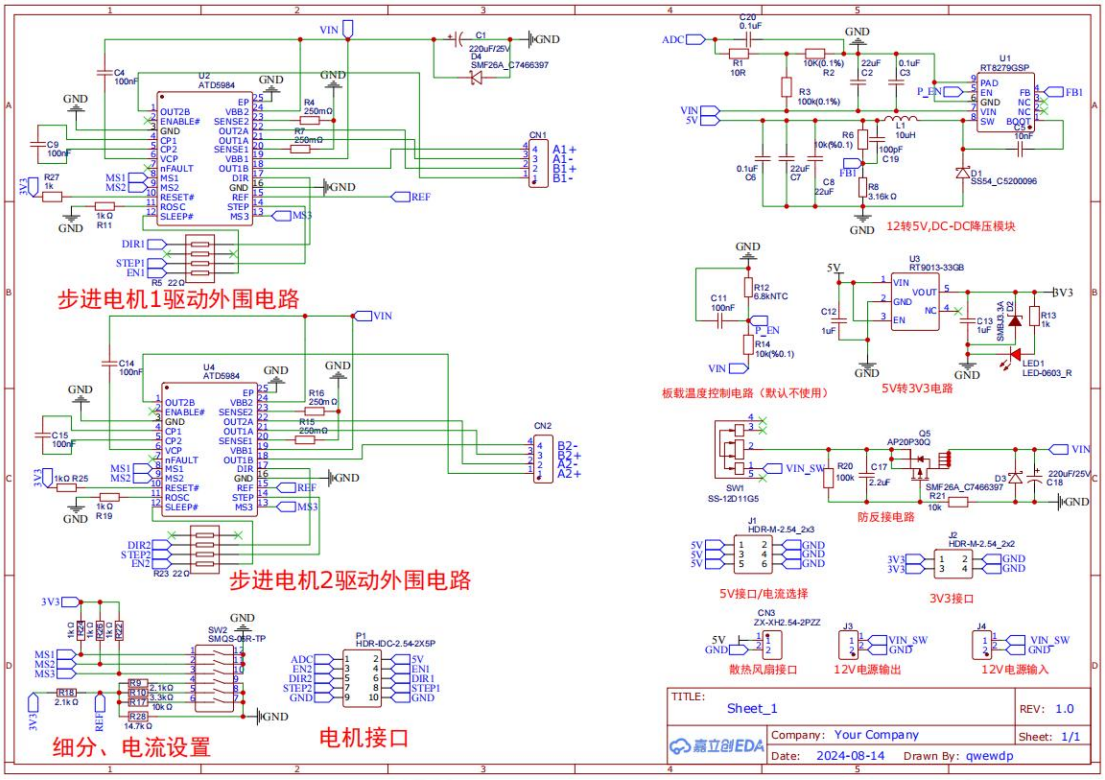

图 2-4 D36A 原理图

2.5注意事项

① D36A 双路步进电机驱动上的拨码开关分为细分控制和电流控制，不管是哪个控制都是一次性设置双路步进电机的。比如，拨码开关设置步进电机的细分倍数为 1/2，则两个步进电机的细分倍数都为 1/2，电流控制也是如此。

② 散热片与电路板之间采用粘胶（导热胶）固定住的，需要注意不要故意用力去破坏或者使其承受重力。

③ ATD5984 芯片在使用过程中会有一定的发热，请勿用手直接触碰，这属于正常现象。

④ 使用 D36A 双路步进电机驱动时需要注意散热风扇，默认是一直转动的，需要避免杜邦线或者别的物品伸进风扇内卡住风扇导致风扇堵转烧坏。

⑤ 接线时注意看好接口的各个引脚的丝印（控制接口的丝印在板子的背面），避免电源接反导致驱动模块烧毁。

3. D36A 模块用户例程使用说明

本章内容主要介绍资料中提供的 STM32F103RCT6 例程与 Arduino 例程，如何配合 D36A 驱动模块使用。

3.1 D36A 与单片机接线说明

表 3-1 D36A 与 STM32F103RCT6 接线表

D36A	STM32F103RCT6	备注
ADC	PC0	
EN1	PD2	步进电机 1
DIR1	PC13	
ST1	PC9	
5V	5V	D36A 提供 5V 给 F103 核心板供电
GND	GND	
EN2	PC12	步进电机 2
DIR2	PC14	
ST2	PC8	
Vin +		外接 12V 电源正极
Vin -		外接 12V 电源负极

表 3-2 D36A 与 Arduino UNO 接线表

D36A	Arduino UNO	备注
ADC	A0	
EN1	3	步进电机 1
DIR1	4	
ST1	5	
5V	5V	D36A 提供 5V 给 Arduino 核心板供电
GND	GND	

EN2	8	步进电机 2
DIR2	7	
ST2	6	
Vin +		外接 12V 电源正极
Vin -		外接 12V 电源负极

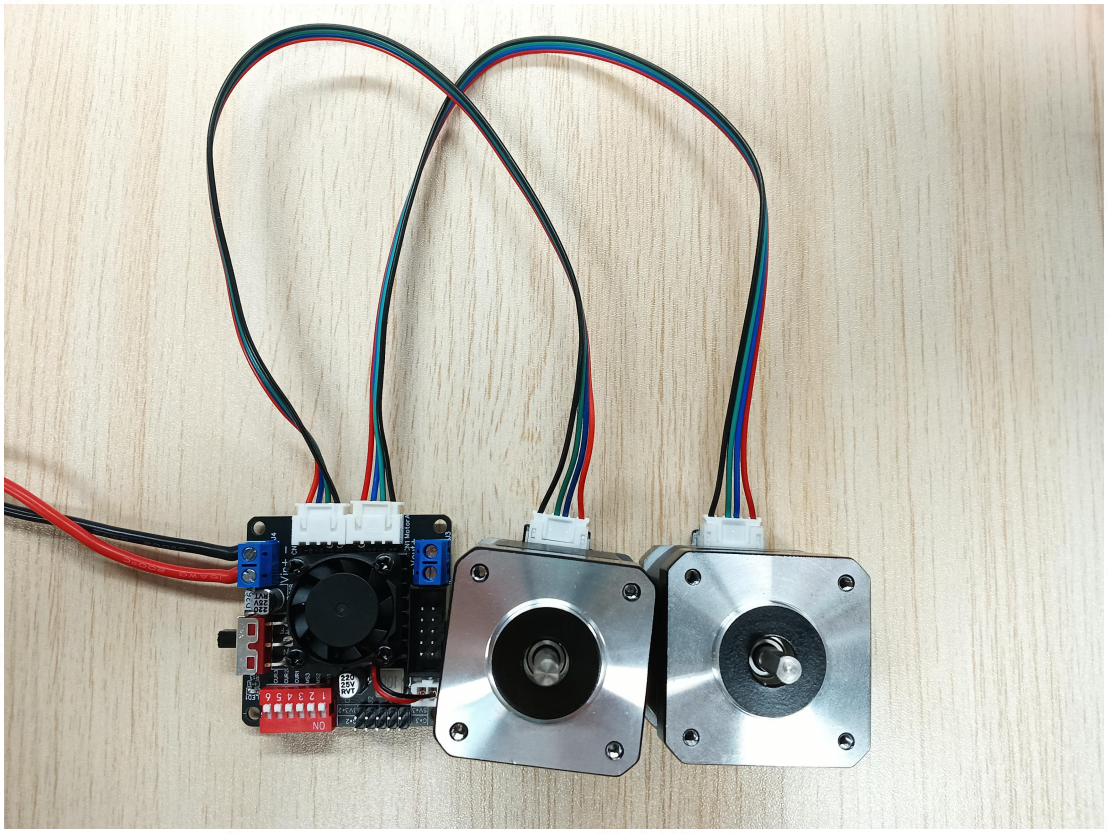

图 3-1 D36A 与步进电机接线图

3. 2STM32F103RCT6 例程源码介绍

从原理图中可以得出，使用 D36A 驱动两个步进电机，除了电源线之外，只需要用到六个引脚，也就是每个步进电机需要三个引脚控制，另外还有一个 ADC 引脚读取电源电压。

```
void ATD5984_Init(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;

    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC, ENABLE);
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOD, ENABLE);
```

```
    GPIO_InitStructure.GPIO_Mode =  GPIO_Mode_Out_PP;
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_12 | GPIO_Pin_13 | GPIO_Pin_14;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOC, &GPIO_InitStructure);

    GPIO_InitStructure.GPIO_Mode =  GPIO_Mode_Out_PP;
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOD, &GPIO_InitStructure);

    GPIO_SetBits(GPIOC, GPIO_Pin_12); //控制步进电机B的ATD5984的SLEEP引脚
    GPIO_SetBits(GPIOC, GPIO_Pin_13);//控制步进电机A的ATD5984的DIR引脚
    GPIO_SetBits(GPIOC, GPIO_Pin_14); //控制步进电机B的ATD5984的DIR引脚
    GPIO_SetBits(GPIOD, GPIO_Pin_2);//控制步进电机A的ATD5984的SLEEP引脚
}
```

控制每个步进电机都需要控制 D36A 驱动上的 EN、DIR 和 ST 三个引脚，ST 引脚为上升沿有效，所以需要提供 PWM 信号，EN 和 DIR 引脚只需要高低电平则可以控制，EN 引脚控制 ATD5984 芯片使能，低电平进入休眠高电平使能，所以可以先将其配置为高电平；DIR 引脚表示步进电机的转动方向，该引脚输入高低电平的区别仅仅只是步进电机转动方向不同而已，所以可以先将这两个引脚都先输出高电平即可。

```
void STEP12_PWM_Init(u16 arr, u16 psc)
{
    GPIO_InitTypeDef GPIO_InitStructure;
    TIM_TimeBaseInitTypeDef TIM_TimeBaseInitStructure;
    TIM_OCInitTypeDef TIM_OCInitStructure;

    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC, ENABLE);
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_TIM8, ENABLE);

    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_8 | GPIO_Pin_9;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOC, &GPIO_InitStructure);

    TIM_TimeBaseInitStructure.TIM_Period = arr;          //自动重装载值
    TIM_TimeBaseInitStructure.TIM_Prescaler = psc;       //预分频值
    TIM_TimeBaseInitStructure.TIM_ClockDivision = 0;     //时钟分割
    TIM_TimeBaseInitStructure.TIM_CounterMode = TIM_CounterMode_Up; //向上
计数
```

```
    TIM_TimeBaseInit(TIM8, &TIM_TimeBaseInitStructure);

    //TIM 脉冲宽度调制模式 2
    TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1;
    //TIM 输出比较极性高
    TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High;
    //比较输出使能
    TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
    //待装入捕获比较寄存器的脉冲值
    TIM_OCInitStructure.TIM_Pulse = 50;


    TIM_OC3Init(TIM8, &TIM_OCInitStructure);            //初始化通道 3
    TIM_OC3PreloadConfig(TIM8, TIM_OCPreload_Enable);   //通道预装载使能

    TIM_OC4Init(TIM8, &TIM_OCInitStructure);            //初始化通道 4
    TIM_OC4PreloadConfig(TIM8, TIM_OCPreload_Enable);   //通道预装载使能

    TIM_ARRPreloadConfig(TIM8, ENABLE);                 //使能预加载

    TIM_CtrlPWMOutputs(TIM8, ENABLE);                   //使能定时器输出
    TIM_Cmd(TIM8, ENABLE);                              //使能定时器
}
```

使用定时器 8 输出 PWM 信号给到 ST 引脚，PWM 信号中每个上升沿都可以让步进电机转动一个步距角，连续输出 PWM 则可以让步进电机转动。

假如不对步进电机进行细分控制，因为原本的步距角为 1.8°，所以转一圈就需要 200 步，也就是 200 个上升沿，假如使用的步进电机的最大空载转速为 420RPM 并且想要设置其转速为最大，通过转换可以得出步进电机 1s 会转 7 圈，也就是 1s 需要 7 * 200 = 1400 步才可以实现 420RPM，因为是一个上升沿步进电机就转动一步，所以这个时候需要将 PWM 信号的频率设置为 1400Hz。

如果想要对步进电机进行细分控制，与不做细分控制的差别只在步进电机转动一圈所需要的步数不同而已。假如要实现 1/32 的细分倍数，则步进电机转一圈需要 6400 步，如果还是想要实现转速最大的话，同样的，1s 依旧转动 7 圈，也就是说 1s 需要转动 7 * 6400 = 44800 步，即 44800Hz。

```
void ADC1_Init(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;
    ADC_InitTypeDef ADC_InitStructure;
    //使能时钟
```

```
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC, ENABLE);
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);

    //设置 ADC 分频因子 6 72M/6=12,ADC 最大时间不能超过 14M
    RCC_ADCCLKConfig(RCC_PCLK2_Div6);

    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOC, &GPIO_InitStructure);

    ADC_InitStructure.ADC_ScanConvMode = DISABLE;       //非扫描模式
    ADC_InitStructure.ADC_ContinuousConvMode = DISABLE; //连续转换
    ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;  //右对齐
    ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None;
//使用软件触发
    ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;  //独立模式
    ADC_InitStructure.ADC_NbrOfChannel = 1;             //只转换规则序列 1
    ADC_Init(ADC1, &ADC_InitStructure);
    ADC_Cmd(ADC1, ENABLE);

    //复位 ADC 校准寄存器
    ADC_ResetCalibration(ADC1);
    //获取 ADC 校准寄存器状态
    while(ADC_GetResetCalibrationStatus(ADC1));
    //启动 ADC 校准
    ADC_StartCalibration(ADC1);
    //获取 ADC 启动校准寄存器状态
    while(ADC_GetCalibrationStatus(ADC1));
}
```

初始化 ADC1 的通道 10，用来采集并计算得出输入电源的电压值。

```
int main(void)
{
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2); //设置中断分组为 2
    delay_init(72);                 //延时初始化

    ADC1_Init();                            //ADC 初始化
    uart_init(115200);                      //串口初始化
    ATD5984_Init();                         //ATD5984 初始化
    STEP12_PWM_Init(65535, 14);             //定时器 8 初始化

    while(1)
    {
        adc_val = Get_adc_Average(10, 5);       //每采集五次取一次平均值
```

```
            voltage = (float)(adc_val*3.3*11/4096); //计算得出电池电压值
            printf("当前电池电压为: %f\r\n", voltage);  //串口打印电压值
        }
    }
```

烧录代码后，按前文所述的接线方式将单片机与 D36A 连接，接上步进电机后给单片机和驱动板上电，可以看到步进电机会一直转动，同时，连接串口助手后可以看到如图 3-2 所示电源电压值的输出，如果需要改变方向则修改代码中控制 DIR 的引脚，如果想要做细分控制则拨动驱动板上的 123 号拨码开关，如果要设置电流大小则拨动驱动板上的 456 号拨码开关。

图 3-2 串口助手输出电源电压值

3.3Arduino UNO 例程源码介绍

同样的，Arduino 也需要控制 D36A 驱动上的 6 个引脚外加一个 ADC 读取电源电压。

```
void setup() {
  pinMode(SLEEP1, OUTPUT);    //控制步进电机 1 的 SLEEP，输出 0 时进入休眠
  pinMode(DIR1, OUTPUT);      //控制步进电机 1 的 DIR，控制步进电机转动方向

  pinMode(SLEEP2, OUTPUT);    //控制步进电机 2 的 SLEEP，输出 0 时进入休眠
  pinMode(DIR2, OUTPUT);      //控制步进电机 2 的 DIR，控制步进电机转动方向

  pinMode(ADC, INPUT);        //初始化 A0 为 ADC 引脚

  digitalWrite(SLEEP1, 1);    //不休眠
  digitalWrite(DIR1, 1);      //顺时针转动
  digitalWrite(SLEEP2, 1);
  digitalWrite(DIR2, 1);

  analogWrite(STEP1, 50);     //输出 PWM 信号
  analogWrite(STEP2, 50);

  Serial.begin(9600);
}
```

初始化函数中对使用到的引脚进行初始化，EN 引脚低电平会进入休眠状态，所以将 EN 配置为高电平；DIR 引脚控制方向，这里也配置为高电平即可；ST 引脚为上升沿有效，所以这里需要提供 PWM 信号。

```
void loop() {
  ADC_Val = Get_ADC_Average(5);                    //每五次取一次 ADC 的平均值
  Voltage_Val = (int)(ADC_Val*5*11*100/1023);      //转换为电源电压值
  Serial.print("当前电池电压为：");
  Serial.print(Voltage_Val/100);
  Serial.print(".");
  Serial.print(Voltage_Val%100);
  Serial.print("V");
  Serial.print("\r\n");
  delay(500);
}
```

loop 函数中每隔 0.5 秒采集五次 ADC 并取平均值，再转换成电源电压值，最后不断的向串口输出当前电池的电压。

代码烧录并接好线后，拨动拨码开关可以对步进电机做细分控制或设置电流

大小，同时，通过打开 Arduino IDE 的串口监视器，可查看到如图 3-3 所示输出电源电压值大小。

```
49    ADC_Val = Get_ADC_Average(5);                    //每
50    Voltage_Val = (long int)(ADC_Val*5*11*100/1023);
51    Serial.print("当前电池电压为: ");
52    Serial.print(Voltage_Val/100);
53    Serial.print(".");
54    Serial.print(Voltage_Val%100);
55    Serial.println("V");
56    delay(500);
57  }
58
```

输出　串口监视器 ×

消息 (按回车将消息发送到"COM20"上的"Arduino Uno")

```
当前电池电压为：12.15V
当前电池电压为：12.15V
当前电池电压为：12.15V
当前电池电压为：12.15V
当前电池电压为：12.15V
```

图 3-3 Arduino IDE 串口监视器输出电源电压值