浙江工業大學
ZHEJIANG UNIVERSITY OF TECHNOLOGY

DSP 原理及应用

实验 5：PWM 配置及步进电机控制

实验报告

姓　　名：林宇航

班　　级：自动化 1901

学　　号：201906060308

学　　院：信息工程学院

设计日期　2022.4.17

实验五 PWM 配置及步进电机控制

一、实验目的

1. 了解步进电机驱动原理;
2. 了解步进电机的控制原理;
3. 熟悉使用 PWM 控制步进电机的运行。

二、实验主要内容

1. DSP 的初始化;
2. ePWM 模块初始化与配置;
3. 步进电机的驱动程序。

三、实验基本原理

1. 步进电机的驱动：图 1 是单极性步进电机驱动的典型电路，图中的方块为驱动开关。针对 SEED-DEC 中直流电机系统的动作要求，步进电机驱动电路设计思路如下：

1） 电机采用 15V 直流电源供电;
2） 4 路控制信号由 DSP 提供，信号为 CMOS 标准电平，通过排线接入并下拉;
3） 使用达林顿管 TIP31C 代替 IRL549 作为电机驱动开关，基级串接 100 欧电阻减小 MOS 管的寄生震荡;
4） 使用快速二极管 IN4007 完成保护功能，以免电机换向时烧毁电机;

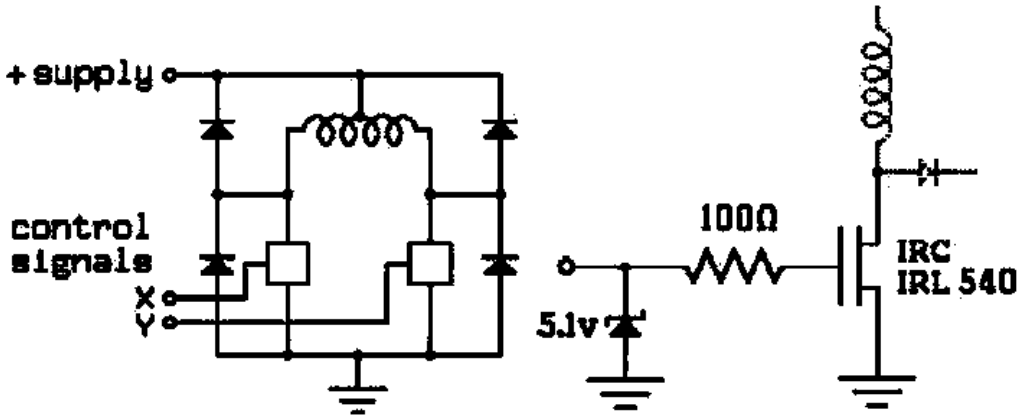


图 1 步进电机驱动电路

2. 步进电机的控制一般分为四相四拍与四相八拍两种方式，其中前者称为全步，后者称为半步。步进电机在这个实验中选择的时 M35SP-7N，其步进角为 7.5°，是一种单极性步进电机。它的结构如图 2：

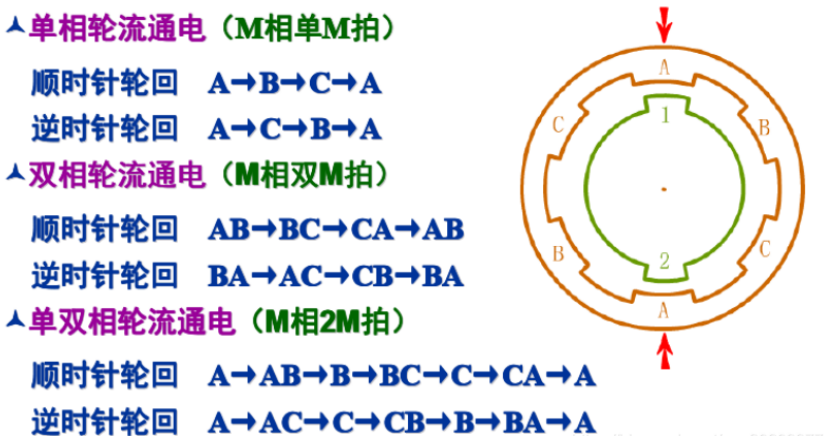


图 2 步进电机结构

四、实验过程和关键程序解读

1. 启动 CCS，进入 CCS 的操作环境，并导入 stepmotor 工程。
2. 加载 stepmotor 工程，添加 NewTargetConfiguration.ccxml 文件
3. 阅读源代码

1）初始化系统控制寄存器与要使用的 GPIO：

```
112 void main(void)
113 {
114     // Step 1. 初始化系统控制寄存器:
115     InitSysCtrl();
116
117     // Step 2. 初始化ePWM1, ePWM2, ePWM3的GPIO:
118     InitEPwm1Gpio();
119     InitEPwm2Gpio();
```

2）关中断、初始化 PIE、初始化 PIE 向量表

```
121     // Step 3. 关中断、初始化PIE、初始化PIE向量表
122     DINT;
123     InitPieCtrl();
124     IER = 0x0000;
125     IFR = 0x0000;
126     InitPieVectTable();
```

3）关 ePWM 时钟，配置后打开时钟，并更新中断向量表

```
128     // Step 4. 关ePWM时钟, 配置后打开时钟, 并更新中断向量表
129     EALLOW;
130     SysCtrlRegs.PCLKCR0.bit.TBCLKSYNC = 0;
131     EDIS;
132     InitEPwm1Example();
133     InitEPwm2Example();
134
135     EALLOW;
136     SysCtrlRegs.PCLKCR0.bit.TBCLKSYNC = 1;
137     PieVectTable.EPWM1_INT = &epwm1_isr;
138     EDIS;
```

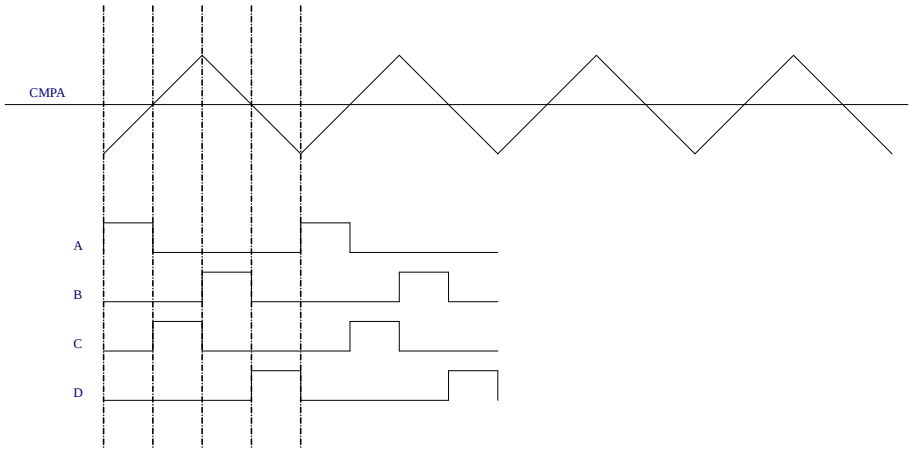
4）ePWM 初始化函数（以 EPwm1 为例）：

```
void InitEPwm1Example()
{
    EPwm1Regs.TBPRD = 0x7fff;                       // 设置周期
    EPwm1Regs.TBPHS.half.TBPHS = 0x0000;            // 相位偏置为0
    // Setup TBCLK
    EPwm1Regs.TBCTL.bit.CTRMODE = TB_COUNT_UPDOWN; // 增减计数
    EPwm1Regs.TBCTL.bit.PHSEN = TB_DISABLE;
    EPwm1Regs.TBCTL.bit.PRDLD = TB_SHADOW;          // 禁用影子寄存器
    EPwm1Regs.TBCTL.bit.SYNCOSEL = TB_CTR_ZERO;

    EPwm1Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;     // 过零时重新加载寄存器
    EPwm1Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;
    EPwm1Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO;
    EPwm1Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO;

    EPwm1Regs.TBCTL.bit.HSPCLKDIV = TB_DIV8;        // 对系统时钟进行分频作为计数器时钟
    EPwm1Regs.TBCTL.bit.CLKDIV = TB_DIV8;           //
    // Setup compare
    EPwm1Regs.CMPA.half.CMPA = 0x3fff;              // 设置对比寄存器
    // EPwm1Regs.CMPB = 0x3333;
    // Set actions

    EPwm1Regs.AQCTLA.bit.ZRO = AQ_SET;              // PWM1A计数过零时产生一个动作：置位
    EPwm1Regs.AQCTLA.bit.CAU = AQ_CLEAR;            // PWM1A增计数过比较寄存器时产生一个动作：复位
    EPwm1Regs.AQCTLB.bit.PRD = AQ_SET;              // PWM1B计数经过一个周期时产生一个动作：置位
    EPwm1Regs.AQCTLB.bit.CAD = AQ_CLEAR;            // PWM1B减计数过比较寄存器时产生一个动作：复位
```

EPWM2 的其他配置与 1 相同，不用的在于一些事件产生的动作不同：

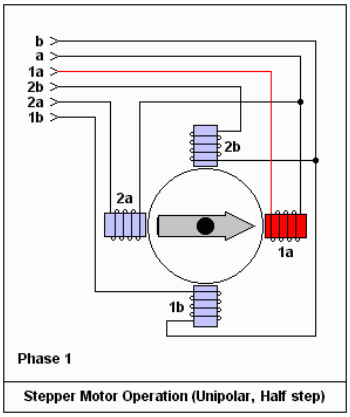
```
    EPwm2Regs.AQCTLA.bit.PRD = AQ_CLEAR;          // PWM2A计数经过一个周期时产生一个动作：复位
    EPwm2Regs.AQCTLA.bit.CAU = AQ_SET;            // PWM2A增计数过比较寄存器时产生一个动作：置位
    EPwm2Regs.AQCTLB.bit.ZRO = AQ_CLEAR;          // PWM2B计数过零时产生一个动作：复位
    EPwm2Regs.AQCTLB.bit.CAD = AQ_SET;            // PWM2B减计数过比较寄存器时产生一个动作：置位
```

其产生的 pwm 波为



线圈按照 ACBD 的次序导通

步进电机接线示意图（2b-A、1b-B、1a-C、2a-D）：



则步进电机顺时针旋转

4. 按照老师要求修改源代码

1）改变步进电机的转速

在使用直流电机时，通常是用占空比来调节转速的，但是在步进电机中，是通过改变 PWM 的频率来调整的，因为在一个 PWM 周期中，步进电机改变的相位是一样的，所以 PWM 频率越高，改变相同相位就越快，所以转速也越快，程序中我们只需改变 TBPRD 的值即可。

2）控制步进电机转一圈

控制电机旋转角度可以转换成控制 PWM 输出的周期数，所以对 pwm 配置周期中断，每经过一个周期产生一个中断，在中断服务函数中计数，当计数值大于多少时停止 pwm 输出即可，由于缺少资料，不知道步进电机的减速比是多少，所以计数值只能通过试凑来得到，经过多次实验，12 比较接近一圈：

```
205     EPwm1Regs.ETSEL.bit.INTSEL = 1;                 //中断选择
206     EPwm1Regs.ETSEL.bit.INTEN = 1;                  //中断使能
207     EPwm1Regs.ETPS.bit.INTPRD = 1;                  //中断周期
208     EPwm1Regs.ETCLR.all = 0x0F;                     //清楚中断标志
```

考虑到步进电机转动有惯性，所以在停下的时候保持某个状态，但经过后来的调试发现停在某个状态可能导致那个状态不是下一个周期的起始位置，在反复转一圈的时候产生固定错位，所以停止状态（InitEPwmSTOP()函数）就把 pwm 所有输出置低。

```
232     EPwm1Regs.AQCTLA.bit.ZRO = AQ_CLEAR;
233     EPwm1Regs.AQCTLA.bit.CAU = AQ_CLEAR;
234     EPwm1Regs.AQCTLB.bit.PRD = AQ_CLEAR;
235     EPwm1Regs.AQCTLB.bit.CAD = AQ_CLEAR;

257     EPwm2Regs.AQCTLA.bit.PRD = AQ_CLEAR;
258     EPwm2Regs.AQCTLA.bit.CAU = AQ_CLEAR;
259     EPwm2Regs.AQCTLB.bit.ZRO = AQ_CLEAR;
260     EPwm2Regs.AQCTLB.bit.CAD = AQ_CLEAR;
```

PWM 中断服务函数：

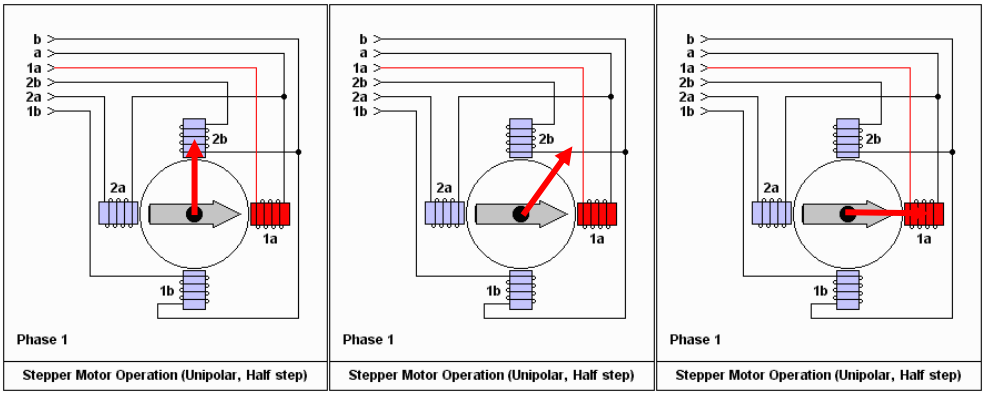
```
294 interrupt void epwm1_isr(void)
295 {
296     EPwm1TimerIntCount++;
297
298     if(EPwm1TimerIntCount > 12)
299     {
300         InitEPwmSTOP();
301         EPwm1TimerIntCount =0;
302     }
303     // 清除中断标志位
304     EPwm1Regs.ETCLR.bit.INT = 1;
305
306     // 产生中断应答
307     PieCtrlRegs.PIEACK.all = PIEACK_GROUP3;
308
309 }
```

同时为了让步进电机重复转一周，方便观察，我还开启了定时器中断，在固定时间间隔清除步进电机停止标志位与周期计数器：

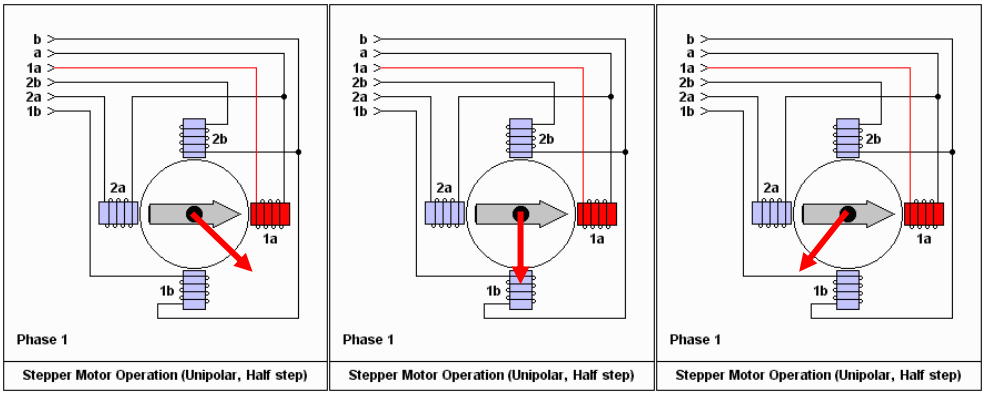
```
311 interrupt void ISRTimer2(void)
312 {
313     CpuTimer2.InterruptCount++;
314     if(CpuTimer2.InterruptCount % 2 == 0)
315     {
316         CpuTimer2.InterruptCount = 0;
317         InitEPwm2Example();
318         InitEPwm1Example();
319     }
320 }
```

3）修改步进电机的步距角

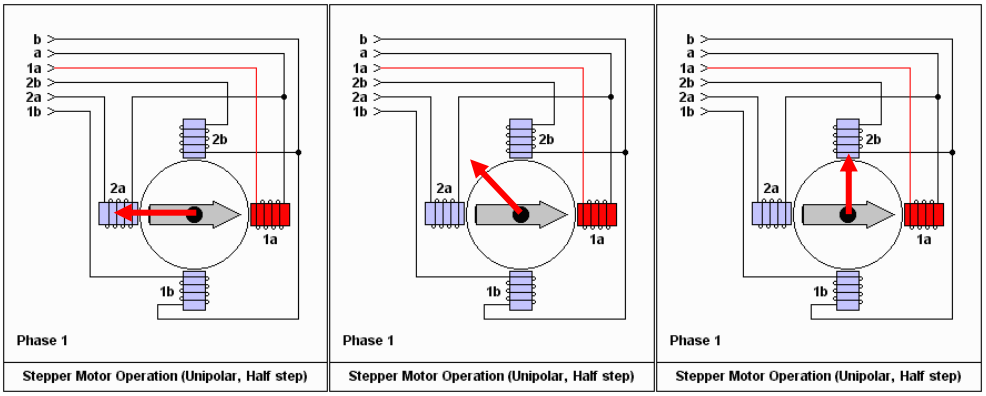
在上面的实验中，步进电机的状态有 4 种，为了减小其步距角从而增加精度，通过矢量控制的方式来引入另外 4 个状态(2b,1a,1b,2a)：



一(1,0,0,0)　　二(1,1,0,0)　　三(0,1,0,0)

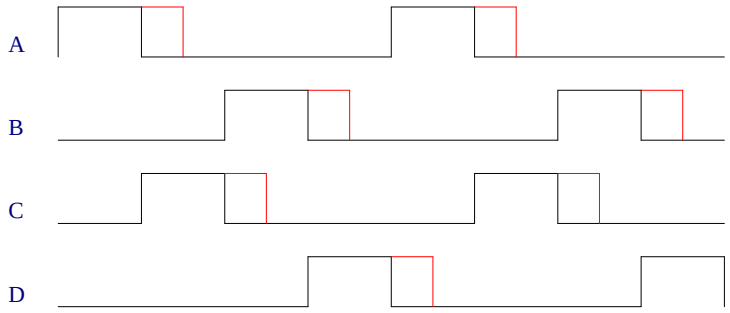


四(0,1,1,0)　　五(0,0,1,0)　　六(0,0,1,1)



七(0,0,0,1)　　八(1,0,0,1)　　一(1,0,0,0)

需要配置的 pwm 波形：



五、实验总结与思考

通过本次实验，我对步进电机的控制方式有了更进一步的了解，在以往的实验中，对于 4 相的步进电机，以为只需要 IO 口的电平按某种规律切换即可，切换频率决定了步进电机的转速，这个想法没错，但是频率不能做得太高，而且频率越高占用单片机的资源就越多。这个思想完全是由于如今现成的库函数过于简单，使用者完全不管底层实现，导致在使用者看来，一个模块的用处就只有这几种，就如 PWM 模块来说，我们做小车的一般就只拿来输出个方波，殊不知设计单片机 pwm 模块的人考虑得十分周全，就如本次实验利用 pwm 的产生模式来代替了软件调整 IO 电平的转换，效率十分的高。