PIC BLDC 三相电机研读总结

(james14888@gmailcom)

ADC 中断处理

```
void __attribute__((__interrupt__,auto_psv)) _ADC1Interrupt(void)
{
    uint16 u16MotorNeutralVoltage;   uint16 u16MotorPhaseA;
    uint16 u16MotorPhaseB;           uint16 u16MotorPhaseC;

    DesiredSpeed = ADC1BUF0;       //POT
    u16MotorPhaseA = ADC1BUF3;    //--V1
    u16MotorPhaseB = ADC1BUF2;    //--V2
    u16MotorPhaseC = ADC1BUF1;    //--V3
    u16MotorNeutralVoltage = (u16MotorPhaseA + u16MotorPhaseB + u16MotorPhaseC)/3;
    AD1CON1bits.DONE = 0;
    IFS0bits.AD1IF = 0;

    /*************** ADC SAMPLING & BMEF signals comparison ***************/
    if(u16MotorPhaseA > u16MotorNeutralVoltage)   {
        Comparator.PhaseAOutput = 1;
    } else {
        Comparator.PhaseAOutput = 0;
    }
    if(u16MotorPhaseB > u16MotorNeutralVoltage)   {
        Comparator.PhaseBOutput = 1;
    } else {
        Comparator.PhaseBOutput = 0;
    }
    if(u16MotorPhaseC > u16MotorNeutralVoltage)   {
        Comparator.PhaseCOutput = 1;
    } else {
        Comparator.PhaseCOutput = 0;
    }
    u16ComparatorOutputs = (Comparator.PhaseCOutput<<2) & 0x0007;
    u16ComparatorOutputs |= (Comparator.PhaseBOutput<<1);
    u16ComparatorOutputs |= Comparator.PhaseAOutput;
}
```

输出两个参量：

DesiredSpeed 电位器定位的目标速度

u16ComparatorOutputs 与平均电压比较的结果指示

主函数

```
int main(void)
{
    PLLFBD=38;                          // M=40        80MHz ;40MIPS
    CLKDIVbits.PLLPOST=0;               // N2=2
    CLKDIVbits.PLLPRE=0;                // N1=2
    RCONbits.SWDTEN=0;
    while(OSCCONbits.LOCK != 1) {};             // Wait for PLL to lock
    ..........................
    Kps = 3000;
    Kis = 2;
    PILoopControllerOutput = 0;
    timer3value = 0;
    timer3avg = 0;
    ........................
    for(;;)
    {
        while (!S1);                        // wait for S1 button to be hit
        while (S1)                              // wait till button is released
            DelayNmSec(20);

        u16CurrentPWMDutyCycle = 1;
        u16DesiredPWMDutyCycle = 0;
        DesiredSpeed = 0;
        ActualSpeed = 0;
        SpeedError = 0;
        SpeedIntegral = 0;
        PILoopControllerOutput = 0;
        timer3value = 0;
        timer3avg = 0;

        PWM1CON1 = 0x0777;                  // enable PWM outputs
        DelayNmSec(1);
        Flags.RampUpBEMF =1;
        Flags.RunMotor = 1;                 // indicationg the run motor condition
        ......................
        while (Flags.RunMotor)              // while motor is running
        {
            while(Flags.RampUpBEMF)
            {
```

```
                    if(u16CurrentPWMDutyCycle < MIN_DUTY_CYCLE) // 1469 = 100% duty
cycle
                    {
                        u16CurrentPWMDutyCycle++;
                        DelayNmSec(1);
                    }else{
                        // MIN_DUTY_CYCLE = 72;
                        Flags.RampUpBEMF = 0;
                        T3CONbits.TON = 1;
                    }
                }
                if (S1)     // if S5 is pressed
                {
                    PWM1CON1 = 0x0700;          // disable PWM outputs
                    P1OVDCON = 0x0000;          // overide PWM low.
                    Flags.RampUpBEMF = 0;
                    Flags.RunMotor = 0;         // reset run flag
                    u16CurrentPWMDutyCycle = 1;

                    while (S1)                  // wait for key release
                        DelayNmSec(20);
                }
            }//end of motor running loop
        }//end of infinite loop
}//end of main function
```

从主函数可以看出，它的算法控制系数 Kps 为 3000，Kis 为 2；
PWM 的占空比有最小值 MIN_DUTY_CYCLE，为 72。

PWM 中断处理：

```
void __attribute__((__interrupt__,auto_psv)) _MPWM1Interrupt (void)
{
    if(Flags.Direction == CW) {
        if((u16ComparatorOutputs^ADC_XOR[ADCCommState])&
CW_ADC_MASK[ADCCommState]) {
            ADC_bemf_filter |= 0x01;
    } else {
        if((u16ComparatorOutputs^ADC_XOR[ADCCommState])&
CCW_ADC_MASK[ADCCommState]) {
            ADC_bemf_filter |= 0x01;
        }
    }
    if(u16CurrentPWMDutyCycle > 160)     //u16CurrentPWMDutyCycle
```

```
        P1SECMPbits.SEVTCMP = u16CurrentPWMDutyCycle>>1;
else if(u16CurrentPWMDutyCycle>76)
        //P1SECMPbits.SEVTCMP =48;
        P1SECMPbits.SEVTCMP = u16CurrentPWMDutyCycle>>3;
else
        P1SECMPbits.SEVTCMP = 0;
ADC_bemf_filter = ADC_BEMF_FILTER[ADC_bemf_filter];
if (ADC_bemf_filter&0b00000001){
        Commutate();
        Comparator.Commutation = ~ Comparator.Commutation;       //Commutation
        u16CommutationStatus =      Comparator.Commutation;
}
//Ramp-up period to detect the rotor position
        if(Flags.RampUpBEMF)     //
            P1DC1=P1DC2=P1DC3=u16CurrentPWMDutyCycle; //u16CurrentPWMDutyCycle
        if (++ PWMticks > MAX_PWMticks)       // MAX_PWMticks = 1024
            Commutate();
        IFS3bits.PWM1IF = 0;
}
```

PWM 中断处理函数_MPWM1Interrupt( ) 的核心内容是处理换向。一是 PWM 中断计数到达 1024 个调用一次换向 Commutate()，或者是 ADC 输出中三个输出不平衡，又没有被滤波滤掉，这调用一次换向 Commutate()。

PWM 中断的第二个任务就是控制 ADC 采样的时间，当占空比比较大时，采样间隔比较大；但占空比比较小时，采样间隔比较小。

其中：

```
ADC_XOR[8]      =   {0x0000,0x0000,0xFFFF,0x0000,0xFFFF,0x0000,0xFFFF,0x0000}
CW_ADC_MASK[8] = {0x0000,0x0002,0x0001,0x0004,0x0002,0x0001,0x0004,0x0000}
ADC_BEMF_FILTER[64]=
{   0x00,0x02,0x04,0x06,0x08,0x0A,0x0C,0x0E,0x10,0x12,0x14,0x16,0x18,0x1A,0x1C,0x1E
    0x20,0x22,0x24,0x26,0x28,0x2A,0x2C,0x2E,0x01,0x01,0x01,0x36,0x01,0x3A,0x3C,0x3E,
    0x00,0x02,0x04,0x06,0x08,0x0A,0x0C,0x0E,0x01,0x01,0x01,0x16,0x01,0x1A,0x1C,0x1E,
    0x01,0x01,0x01,0x26,0x01,0x2A,0x2C,0x2E,0x01,0x01,0x01,0x36,0x01,0x3A,0x3C,0x3E };
```

问题：

用三个数组，变动下标，就可以实现滤波过滤，算法原理是什么？

换向处理

```
void Commutate(void)
{
```

```
// The code below is uses TMR3 to calculate the speed of the rotor
if (ADCCommState == INDEX)                  // has the same position been sensed?
    if (u8PoleCounter++ >= POLEPAIRS)       // has one mech rev elasped?
    {                                       // if yes then read timer 3
        T3CONbits.TON = 0;    // Stop TIMER3
        timer3value = TMR3;
        TMR3 = 0;
        timer3avg = ((timer3avg + timer3value) >> 1);

        if(!Flags.RampUpBEMF)
            SpeedPILoopController();
        u8PoleCounter = 1;
        T3CONbits.TON = 1;    // Start TIMER3
        }
    //Commutation
    ADC_bemf_filter=0;
    if (++ADCCommState>6)
        ADCCommState=1;
    if(Flags.Direction == CW)
        P1OVDCON = CW_PWM_STATE[ADCCommState];
    else
        P1OVDCON = CCW_PWM_STATE[ADCCommState];
    PWMticks=0;
}
```

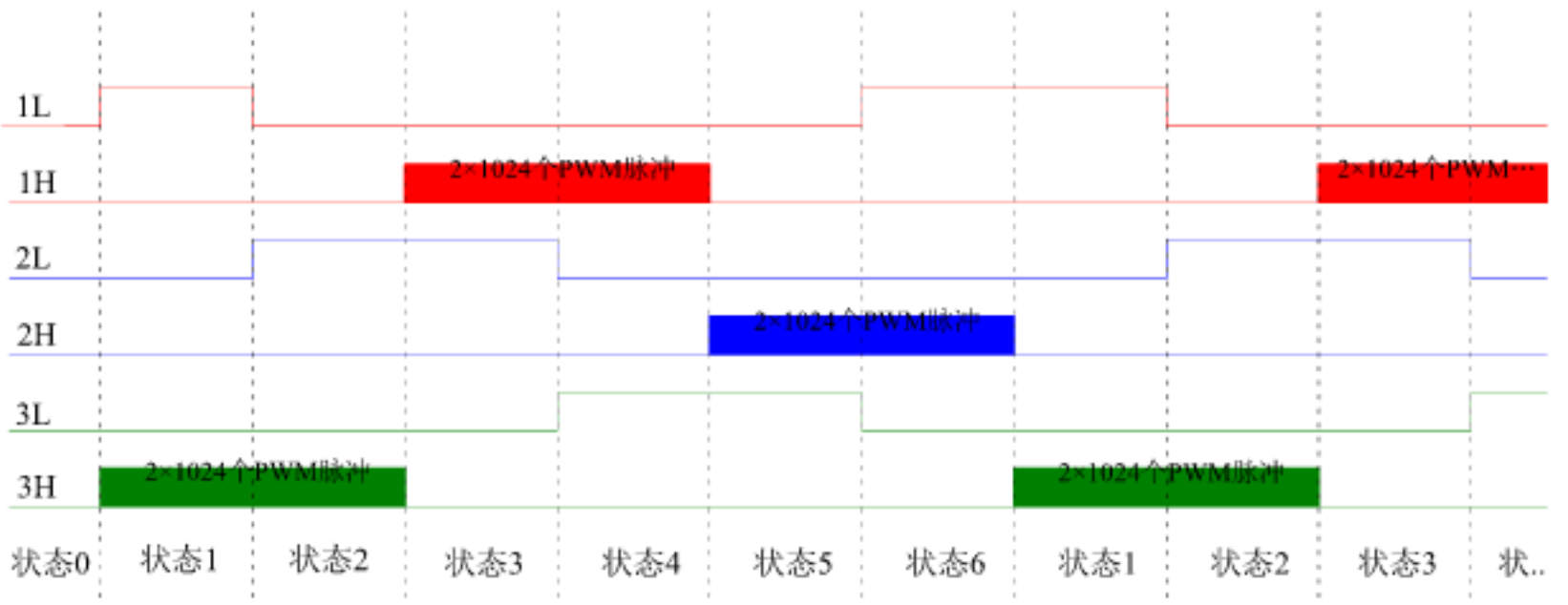
换向处理主要处理两件事情。一件是速度的 PI 算法反馈控制 SpeedPILoopController()；另一件是产生三相控制信号。其中：

1) TIMER3 用来计算电机转速。
2) 数组

CW_PWM_STATE[8] = {0x0000,0x2001,0x2004,0x0204,0x0210,0x0810,0x0801,0x0000};

和 PIC 的 PWM 硬件与本软件共同产生如下的控制信号（其中 PWM 脉冲为 20KHz）：

POVD	1 : I/O 引脚由 PWM 发生器控制 0 : I/O 引脚由 POUT 寄存器控制					POUT	1 : I/O 引脚被驱动为有效状态 0 : I/O 引脚被驱动为无效状态				
POVD						POUT					
3H	3H	2H	2L	1H	1L	3H	3L	2H	2L	1H	1L
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1
1	0	0	0	0	0	0	0	0	1	0	0
0	0	0	0	1	0	0	0	0	1	0	0
0	0	0	0	1	0	0	1	0	0	0	0
0	0	1	0	0	0	0	1	0	0	0	0
0	0	1	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0



PI 控制算法

```
void SpeedPILoopController(void)
{
    //For the HURST motor the Timer3Avg values
    //TIMER3 counts = 9562 AT 7.5% dutycycle (P1DC1 = 109)
    //TIMER3 counts = 714    AT 100% dutycycle (P1DC1 = 1469)
    //RPSelectrical = 19-261Hz
    //RPMelctrical = 1179-15669RPM

    //Normalizing TIMER3 counts to electrical RPM
    ActualSpeed = (SPEEDMULT/timer3avg)*60;
    //ADC POT RANGE 0-1024, 1024*16 = MAXSPEED in electrical RPM

    DesiredSpeed = DesiredSpeed<<4;
    //Calculating Error
    SpeedError = DesiredSpeed - ActualSpeed;
    SpeedIntegral += SpeedError;
    //Dividing PI output by PI controller gain (2exp14)
    PILoopControllerOutput = (((long)Kps*(long)SpeedError +
(long)Kis*(long)SpeedIntegral) >> 14);
    //Divided by 11 to normalize the PI output to PWM PDCx values
    u16DesiredPWMDutyCycle = u16DesiredPWMDutyCycle + (PILoopControllerOutput/11);

    // Max and Min PWM duty cycle limits
    if (u16DesiredPWMDutyCycle < MIN_DUTY_CYCLE){
        u16DesiredPWMDutyCycle = MIN_DUTY_CYCLE;
        SpeedIntegral = 0;
    }
    if (u16DesiredPWMDutyCycle > MAX_DUTY_CYCLE){
        u16DesiredPWMDutyCycle = MAX_DUTY_CYCLE;
```

```
        SpeedIntegral = 0;
    }

    P1DC1 = u16DesiredPWMDutyCycle;
    P1DC2 = u16DesiredPWMDutyCycle;
    P1DC3 = u16DesiredPWMDutyCycle;
    u16CurrentPWMDutyCycle = u16DesiredPWMDutyCycle;
}
```

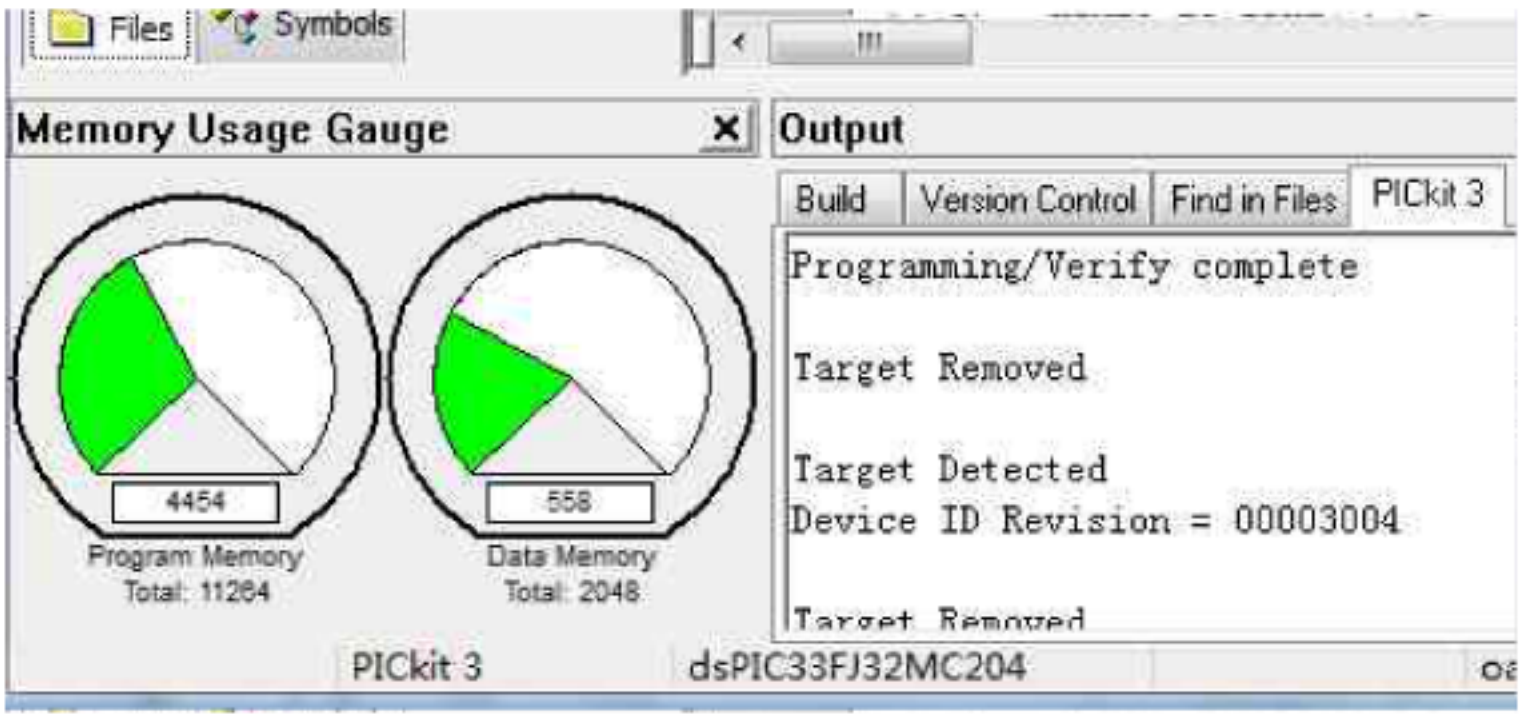
从代码可以看出，实例代码采用的是 PI 控制算法，而非完整的 PID 控制算法。

问题：

关于速度的计算，是利用 PWM 中断间隔和 TIMER3 计数实现。PWM 中断间隔是固定的，再假设三相电压是理想的，即不需要调整的状态下，TIMER3 计数的间隔是 1024×6×2 个 PWM 周期，即每个 TIMER3 统计的值应该是相近的，而不管你的实际转速，这样，它是如何求出实际转速的，亦或代码的的实际转速是不正确的，亦或 TIMER3 的计数与 PWM 占空比有对应关系（如果有，好像是多此一举）？

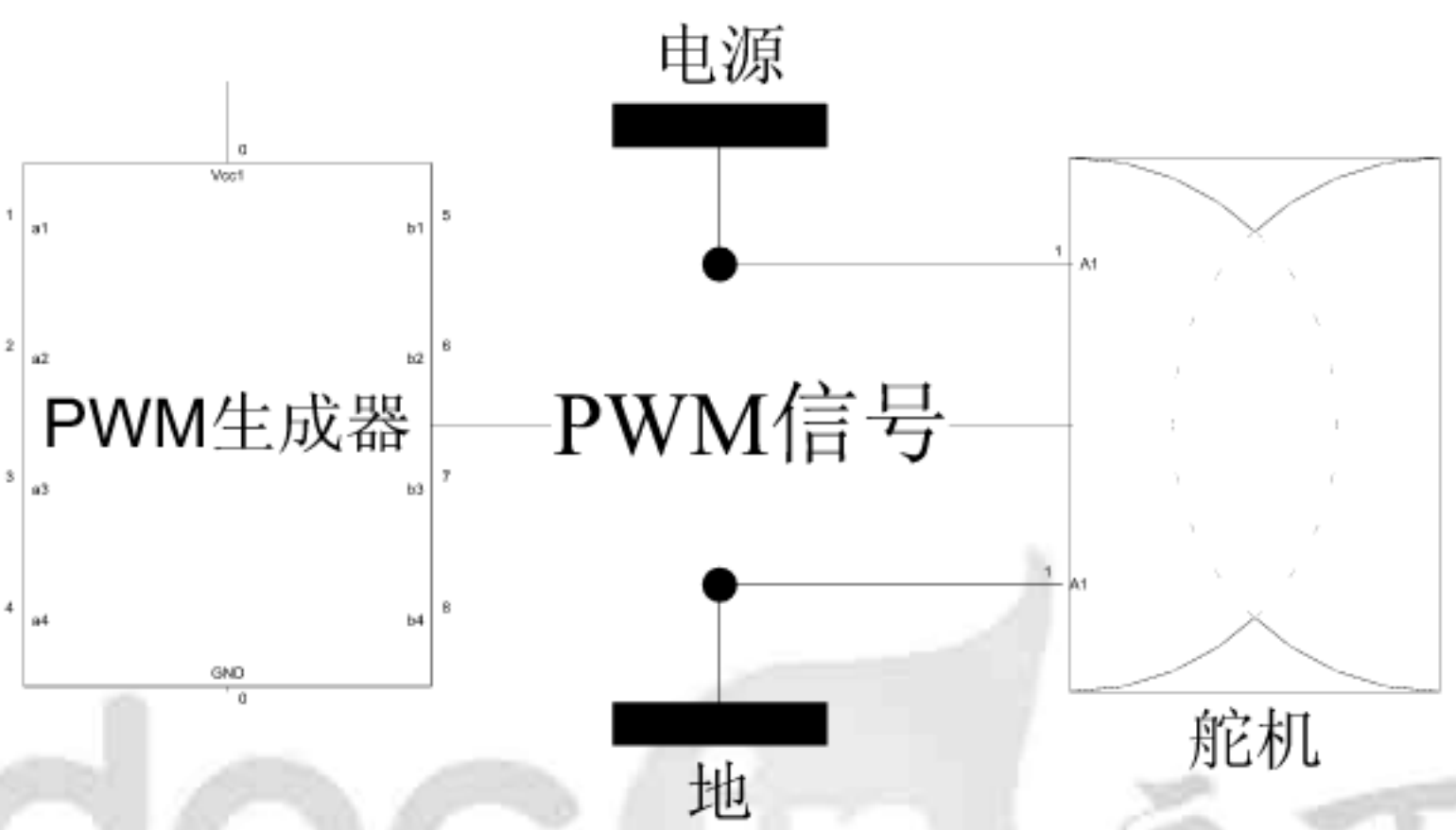
总结：

PIC BLDC 三相电机实例代码是通过控制 PWM 生成器产生特定的 PWM 信号，驱动功率板产生三相电压（电源），从而控制三相电机。与我们目前需要做的正反转电机和舵机需要的时序完全不一样，它所采用的 PI 控制算法也不是我们将用的 PID 算法，它的软件对我们目前没有任何实际意义。我们唯一可用的是它的 PIC 控制板及其硬件功能，可以再其上面写我们的软件测试平台和测试代码。有部分遗憾的是他的 Data Memory 空间很小，我们一次只能编译进部分测试代码，寄希望多编译几个分支吧。



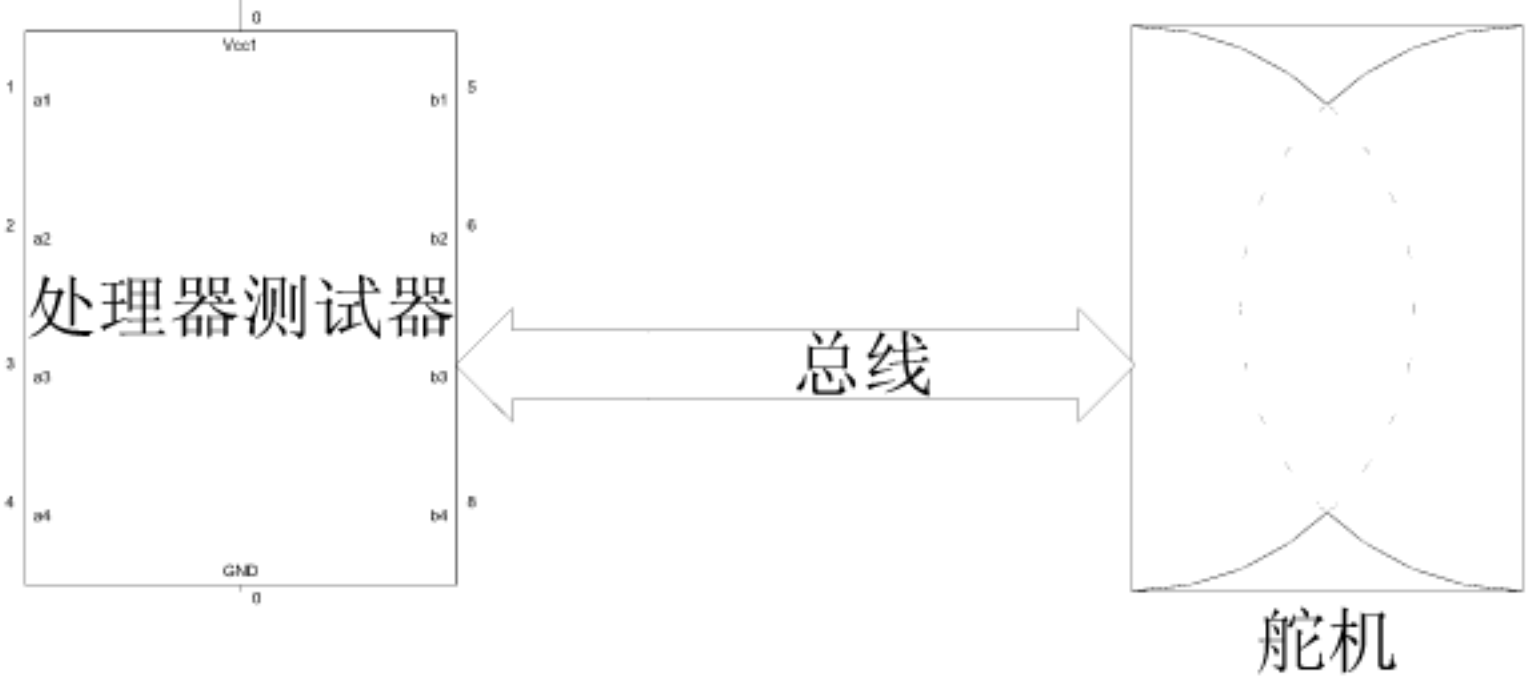
舵机测试框架

传统舵机测试框架



传统舵机模型是灌不同占空比的 50Hz 的 PWM 信号，检测到舵机转的角度。

自定义舵机测试框架



总线：研发过程中的总线是外部总线，可以是 SPI；产品中的总线是 C51 总线。
舵机：包括电机，反馈电位器，ADC，PID 控制器，PWM 生成器。
处理器：运行测试代码。
处理器对舵机中的 PID 控制器，PWM 生成器等进行不同的配置，然后检测舵机的角度。