NANOMUSCLE

Application Note

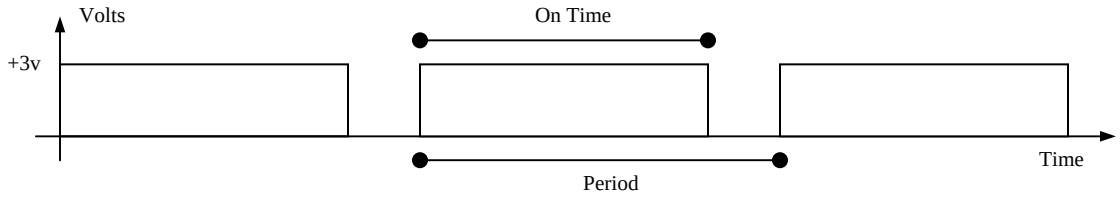
NM-AN03 – Speed Control using NanoMuscles

Rev 1.7 – 29 February 2004

Applicable NanoMuscle Families: RS Rotary

In their standard form all NanoMuscles can provide simple full-range motion by simply applying power to the N1/N2 pins via a 2-transitor drive circuit. The rate at which this motion will occur will vary dependant on, among other things, the load, friction in any linkage, voltage level and ambient temperature. Although for many applications a simple "move as fast as you can" approach is satisfactory, other applications require that the speed of motion can be controlled. A typical requirement is for a balanced, symmetric cycle of rotation and return of the NanoMuscle, independent of any variable operating conditions – typically voltage level and ambient temperature.

The speed of rotation can be controlled in a NanoMuscle using a Pulse Width Modulation (PWM) signal to the drive circuit rather than simply pulling it high. The speed of motion will be governed by the duty cycle of this PWM signal.



$$\text{Duty Cycle} = \frac{\text{On Time}}{\text{Period}}$$

The larger the duty cycle, the faster the NanoMuscle will move. Within certain constraints the actual period of the PWM does not have any affect. Unlike the control of some other devices, NanoMuscles can operate at relatively low periods (typically down to around 100 Hz). Many micro-controllers provide PWM support on-chip, so the generation of this signal is usually as simple as loading the PWM circuit.

However, due to the relative low period requirements of NanoMuscles it is quite feasible to provide a software only PWM signal using either instruction timing or by using a regular timer. In fact, given a NanoMuscles relative immunity to period length, this software does not even need to keep to a strict timing scheme. The psuedo-code for such a simple soft PWM is shown below:

```
Global Period = Period length count
Global OnTime = On Time length count

Static Counter = 0
SoftPWM()
{
      If Counter == OnTime
            Enable Drive Circuit
      Else
            If Counter >= Period
                  Disable Drive Circuit
                  Counter = 0
            EndIf
      EndIf
      Counter++
}
```

Setting Period = 10 and OnTime = 5 in the globals above would cause the NanoMuscle to be driven at half speed, provided the SoftPWM() function is called frequently enough (at least Period * Minimum PWM Period, i.e. 10 * 100 Hz = 1KHz).

Now that we have a way of controlling the speed, the second issue is how do we decide on a value for the duty cycle that will match the desired physical speed? Since there are many variables that affect overall speed it would not be feasible to model the entire system in the micro-controller. Instead we need to have some kind of measurement of the actual speed and use a control loop to adjust the duty cycle to cause the NanoMuscle to home in on the desired speed. There are two ways in which we can monitor the actual NanoMuscle stroke speed:

Using external sensors	In this case a movement sensor is tracked by the micro-controller and the PWM duty cycle is adjusted accordingly.
Using end-stop timing	Assuming that the NanoMuscle is cycling between its end-stops, then by monitoring the time per stroke the micro-controller can calculate any adjustments to the PWM duty cycle.

The first case of external sensors might typically use an optical encoder to generate a bit stream, the rate of which can be tracked against the desired rate, and the duty cycle adjusted accordingly. However, this requires additional hardware to be included in the device. If this is not possible (or un-economical) then end-stop timing can be used. In this scenario the micro-controller tracks the time taken for the NanoMuscle to go through each cycle by watching the C0 and C100 pins that are available on the NanoMuscle driver circuit. This time can then be compared with the desired time. This is essentially similar to using external sensors, albeit that it would take at least one full cycle before the desired speed might be reached.

However, before an algorithm can be developed for either case, there are a number of issues that must be taken into account due to the performance characteristics of the SMA wire used to power NanoMuscles.

- An initial burst of power is needed to get the NanoMuscle to start to move
- The duty cycle adjustment to compensate for the difference between actual and desired speed is different for the rotate and return cycles.

So the pseudo code for a simple algorithm that adjusts the PWN duty cycle using end-stop sensing is shown below. This example assumes the NanoMuscle is being driven by the NM 1076 Driver Board (refer to Application Note *AN08 - Using the NM 1076 Driver Board* for more information).

```
global RotateOnTime = Initial Approximation
global DesiredTime = IdealHalfCycleTime

Contract()
{
     /* First let's get the NanoMuscle to start moving */

     Enable Drive Circuit (Set CTRL to LOW)
     While C0 Pin is HIGH
       Wait
     Endwhile

     /* Now drive to full rotation */

     StartTime = Now
     Enable Drive Circuit with PWM with RotateOnTime as
       the Duty Cycle
     Wait for C100 Pin to go HIGH
     EndTime = Now

     /* Now adjust OnTime so that next time around we are more accurate
     */

     ActualTime = EndTime – StartTime
     RotateOnTime = RotateOnTime + ((ActualTime – DesiredTime) * Kc)
}
```

The constant K_c will determine how quickly the system homes in on the desired speed and to avoid unstable fluctuations a value of 0.5 has been shown to provide good results.

In most schemes the goal is to modify the power of the rotate cycle so that it matches the return cycle (since the return cycle will vary with environmental temperature). To achieve this, the return cycle simply needs to be timed and then this time used to in place of `DesiredTime` for the next rotate cycle:

```
Extend()
{
     /* Time the return cycle */

     StartTime = Now
     Disable Drive Circuit (Set CTRL to HIGH)
     Wait until C0 goes HIGH
     EndTime = Now

     /* Now adjust DesiredTime so that next time around we are more
     accurate */

     DesiredTime = EndTime – StartTime
}
```

In reality, most production toys that use the above scheme provide a balance between speed and symmetry. Typically they will cap the DesiredTime so that it does not go above some threshold so that the whole motion does not seem too slow.

A number of improvements can be made to the above algorithms to fine-tune the results:

- The time taken for the NanoMuscle to start (in the Contract cycle) can be measured and used to bias the first cycle to try and get a more accurate initial speed
- The PWM duty cycle can be adjusted in mid-cycle if the desired time has already been missed
- Timeouts can be added to ensure the wait loops do not become indefinite.
- In rare cases the ideal Desired Time is actually much greater than both the rotate or un-powered return time, then you can also provide power to the NanoMuscle to slow down the return cycle.