Freescale Semiconductor

Order this document by: AN1213/D

16-Bit DSP Servo Control With the MC68HC16Z1

By David Wilson

INTRODUCTION

This application note discusses digital filter implementation of Proportional, Integral, Differential (PID) control algorithms. The implementation takes advantage of the control-oriented digital signal processing capabilities of the Freescale M68HC16 family of modular microcontrollers.

Microcontrollers have come a long way in the past two decades. Once relegated strictly to computer applications, these devices have steadily encroached on domains previously dominated by analog technology. Closed-loop control systems are among the most recent bastions to fall. Control systems based on digital processing of measured values are inherently less sensitive to changes in temperature and to aging than systems implemented with analog circuitry. In addition, digital system performance can be changed by developing new software rather than by physically altering a PC board. In fact, many emerging controller technologies, such as adaptive control, would be economically unattainable if not for digital processing capabilities.

MICROCONTROLLERS AND DIGITAL SIGNAL PROCESSORS

After the decision to use a digital solution has been made, a designer must evaluate system requirements to determine what type of device is best suited for the job. The decision often comes down to a choice between a standard microcontroller or a digital signal processor. Although design constraints vary, all digital signal processors are designed specifically to perform algebraic sum of products calculations at high speeds.

Standard microcontrollers are best suited for applications that require relatively little real-time control, and also require the controller to perform other tasks, such as running an operating system or user interface. Digital signal processors, on the other hand, are generally used when a control algorithm is real-time intensive, and other system tasks can be handled by a master processor. There is thus a price-performance gap between general-purpose controllers and specialized, dedicated DSP engines.

THE M68HC16 FAMILY

The M68HC16 family of modular microcontrollers bridges the gap between standard microcontrollers and digital signal processors. The CPU16 module provides a rich instruction set as well as dedicated control-oriented DSP capability, while other system modules provide a variety of interfacing options. The high level of functional integration in M68HC16 devices reduces the amount of external hardware necessary to achieve a complete system solution.

The M68HC16 family bridges another gap by providing a migration path from the M68HC11 family of 8-bit controllers to the M68300 family of 32-bit modular controllers. Many M68HC16 and M68300 modules, such as the general-purpose timer (GPT), queued serial module (QSM), and system integration module (SIM), are identical. Use of the SIM provides both families with a common external bus interface.

The CPU16 programming model is similar to that of the M68HC11 CPU, and the CPU16 instruction set is upwardly code-compatible with that of the M68HC11 CPU. However, the CPU16 also provides significant additional capabilities. With dedicated multiply and accumulate registers and 18 instructions added specifically for DSP support, M68HC16 family devices are general-purpose microcontrollers capable of performing DSP operations, not just DSP engines with a few additional embedded-control features. This is an important distinction because most digital signal processors do not have bit manipulation capability, multiple interrupt vectors, or a flexible software stack. Although M68HC16 devices do not perform multiply and accumulate operations as rapidly as some dedicated digital signal processors, they are ideally suited for applications such as motion control systems.

PID CONTROLLER BASICS

PID controllers offer some distinct advantages over other control topologies; but nothing is free —as in all design processes, there must be trade-offs. **Figure 1** is a block diagram of an analog PID control structure.

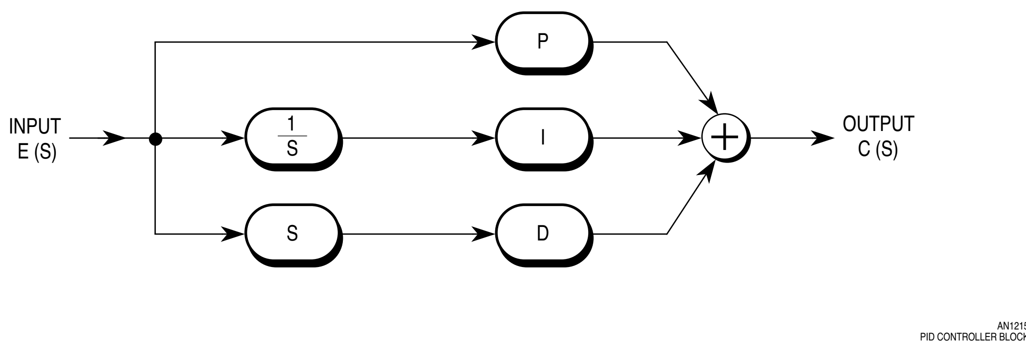

Figure 1 PID Controller Block Diagram

PID TRANSFER FUNCTION

The transfer function (as a function of s) is :

$$\frac{C(s)}{E(s)} = \frac{Ps + I + Ds^2}{S} \qquad \text{(EQ 1)}$$

where:

C(s) is the output of the PID section

E(s) is the input to the PID section (usually servo error)

P is the multiplier for the servo error

I is the multiplier for the integral of the servo error

D is the multiplier for the derivative of the servo error

s is the Laplace complex frequency variable.

The previous equation shows that the PID controller has a pole at s = 0, and two zeros :

$$s = \frac{-P \pm \sqrt{P^2 - 4DI}}{2D} \qquad \text{(EQ 2)}$$

The two zeros are real-valued when $4DI \geq P^2$. A Bode plot of the PID transfer function with real-valued zeros reveals that one of the zeros is used to brake the 20 dB/decade descent associated with the integrator, and the other is used to provide a 20 dB/decade rise and positive phase lead required to stabilize the system.

TRANSFER FUNCTION TERMS

Each of the transfer function terms affects system performance.

The P Term

The **Proportional** term is the most subtle and perhaps most misunderstood term in the PID algorithm. The P term amplifies the error signal by a constant amount. However, P is not in series, but in parallel with I and D, which implies that P cannot be used to scale the transfer function amplitude at all frequencies. Instead, the P term interacts with the I and D terms to determine the placement of the zeros in the controller open-loop transfer function. **Figure 2** shows a root locus solution to the numerator of Equation 1 as P is varied with respect to I and D.

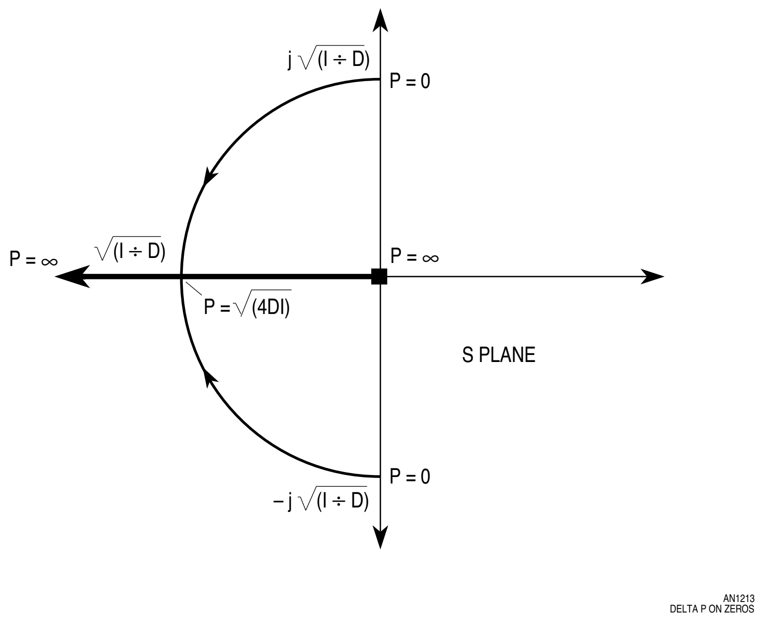

Figure 2 Effect of Varying P on Zeros

The I Term

The **Integral** term gives the servo loop that inflexible, stubborn feel. Since the I term adjusts the amount of integrated error mixed to the output of the filter, any I value other than zero implies that **no** steady state error can be tolerated by the servo loop. In other words, given sufficient time, a PID control loop will eventually servo the output to the exact value of the commanded input.

In the frequency domain, the I term also affects placement of the zeros, as shown in **Figure 3**. For I = 0, one of the zeros is at s = –P/D, and the other zero is at s = 0, which means that it will cancel the integrator pole at s = 0. This makes sense intuitively since the integrator is turned off if I equals zero. As I increases, the servo loop becomes “snappier”, i.e., it responds more quickly to steady state error.

It appears that adding an integrator to the servo loop would be a panacea for motion control headaches. However, while adding an integrator does address steady state error, it can also have a negative impact on system dynamics. The effect is most easily seen in the time domain. Consider a linear PID system that performs servo control. Initially, the controlled motor is at rest, with zero position error. Torque is applied to the motor shaft, changing its position and holding it in the new position. The control system senses a steady-state error and tries to return the shaft to the commanded position. Since the example system is linear, control voltage continues to increase as a result of integrated error. While the increasing control voltage could cause the motor to overheat, this is not the only detrimental effect. If the applied torque is suddenly removed while integrator output is large, the motor shaft will spin past the desired shaft position while control voltage is “dumped”. Eventually, a zero steady-state condition is achieved, but in an underdamped (and potentially unacceptable) manner. Because this situation is similar to winding up a spring and then letting it go, the term “wind-up” is used to describe it. Techniques to mitigate wind-up are discussed later in this note.

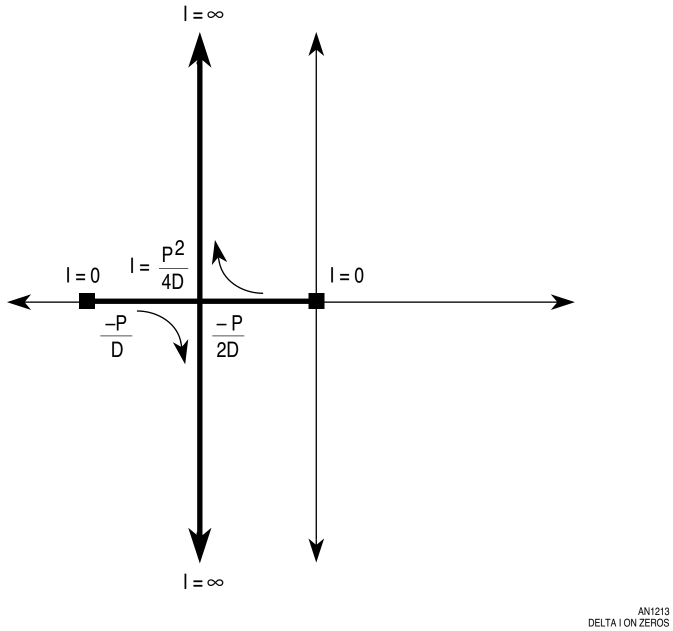

Figure 3 Effect of Varying I on Zeros

The D Term

The **Derivative** term has its greatest effect on servo-loop damping and stability. As **Figure 4** shows, increasing the value of D from 0 to $P^2/4I$ causes both zeros to move toward –2I/P. As this happens, the higher-frequency zero takes on a value that can provide useful phase lead to offset the phase lag introduced by poles elsewhere in the system.

The design of the derivative portion of a PID controller is critical to system performance. In a position servo, the feedback position signal is differentiated (either directly or indirectly) to create a signal proportional to the output velocity. In systems that use a digital feedback mechanism (such as shaft encoders), velocity information is also quantized, typically in encoder tics per sampling interval. At low velocities, the effect of quantization on system performance is pronounced because each quantization step represents a large portion of velocity signal amplitude. This can cause an audible scraping noise or unnecessary motor heating at low speeds.

A velocity state observer can be used to mitigate low-speed quantization effects. The state observer uses a software model of the load to synthesize a higher-resolution velocity signal. Each sample period, PID controller output is input to the model, and the model generates an estimate of output position. The estimate is compared to actual encoder position to generate an error value, which is used to refine the estimate for the next sample period.

A simpler way to deal with this problem is to calculate the velocity information at a lower sampling frequency, thus increasing the number of encoder tics per sampling interval for a given velocity. A similar technique is discussed in the next section of this note.

The location of the differentiator in the feedback loop also affects performance. In **Figure 1**, the differentiator input is the error signal. Since the commanded input position signal is a component of the error signal, any abrupt change in commanded position is differentiated as if it were feedback position, resulting in a "popping" effect at the filter output. An alternate topology can provide more satisfactory performance, as shown in the next section.

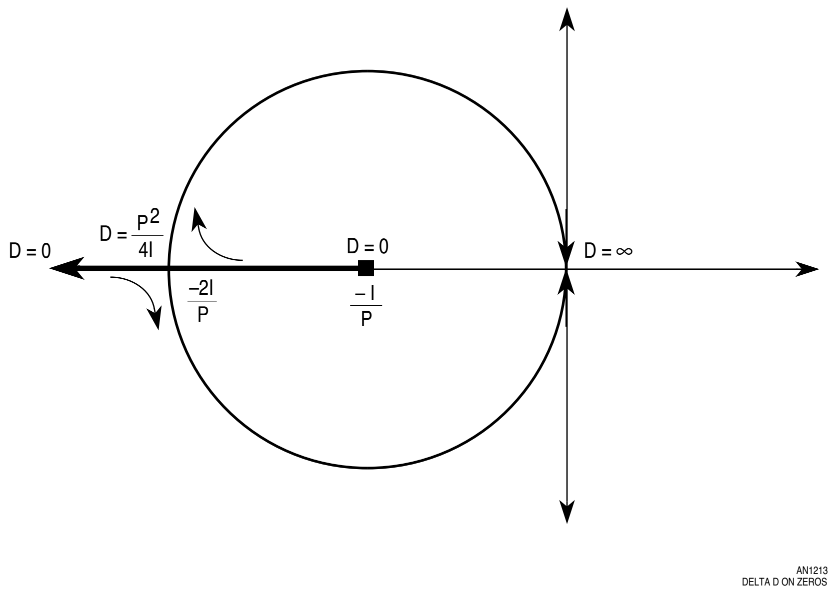

Figure 4 Effect of Varying D on Zeros

DESIGNING A PID FILTER

PID TOPOLOGY

To implement a PID control algorithm on any processor, methods of computing the functions specified by the controller (an integral and a derivative) must be developed. Once these methods are established, the digital PID controller transfer function is calculated in much the same way as the analog version. Unfortunately, because this is a sampled system, the Laplace transform or the s domain cannot be directly used, as in analog PID calculations.

To address this problem, a separate frequency space called the "z" domain has been developed just for sampled systems. Using the z domain, sampled approximations of many common functions can be represented using the variable "z" just as "s" is used to represent linear analog functions. For the sake of simplicity, assume the following definitions are true:

$$\text{INTEGRATOR} = \frac{Tz}{z-1} \qquad \text{(EQ 3)}$$

Where:

z is the complex sampled frequency variable
T is the sampling frequency period, in seconds.

This form is derived from a step-invariant analysis —the filter is constructed by dividing the Z transform of a specified input (a step function) into the Z transform of the desired output for that input (a ramp function). The differentiator is simply the inverse, or:

$$\text{DIFFERENTIATOR} = \frac{z-1}{Tz} \qquad \text{(EQ 4)}$$

Although these are not the only z-domain representations of these functions, they are widely used in control applications. See Reference 1 for more information on this topic.

To mitigate encoder velocity quantization noise, the derivative function is followed by an "n-point averager", which averages velocity information over a range of samples to provide finer resolution. However, this crude low-pass filter also introduces phase lag proportional to *n* that counteracts the desired phase lead generated by the differentiator. To balance these two constraints, *n* is set equal to two, which effectively doubles encoder resolution per sampling interval. The derivative stage transfer function is :

$$\text{VELOCITY} = \left(\frac{z-1}{Tz}\right)\left(\frac{1}{2}+\frac{1}{2z}\right) \qquad \text{(EQ 5)}$$

Figure 5 shows the PID controller and transfer functions of parasitic effects found in the system. All items in **Figure 5** except the power stage, the motor, and the encoder, are implemented by the M68HC16 device.

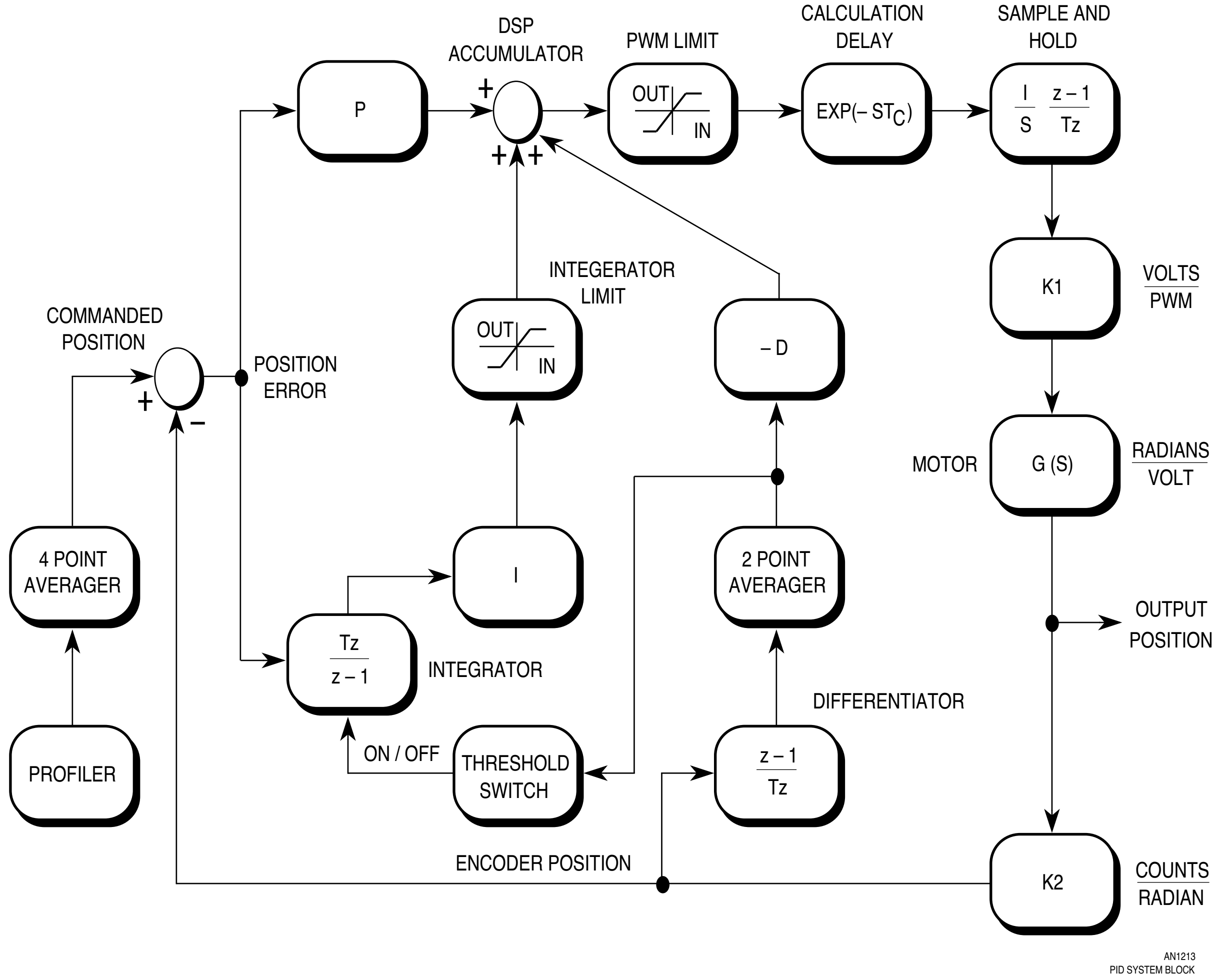

Figure 5 System Block Diagram

The differentiator input is the encoder signal, not the error signal, for reasons discussed in the previous section. The commanded input signal is not differentiated, and the stability of the system is not affected because the overall open-loop transfer function remains the same.

The integrator has several associated features that improve system damping. One is a software switch to enable the integrator only when it is needed. Since this is a position servo, it is assumed that the integrator is not required when the velocity magnitude increases above a specified threshold. This prevents an error signal from being integrated over the entire duration of a change in position, which would require overshoot to dump the error. **Figure 6**, which shows system response to a step function, illustrates the effectiveness of this technique.

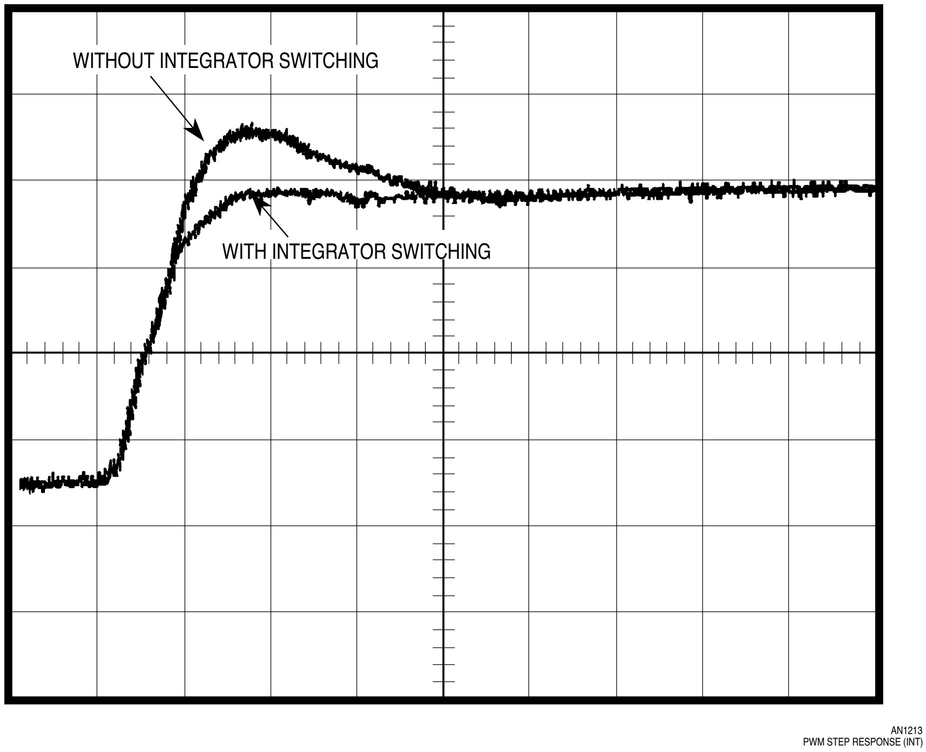

Figure 6 Step Response of System With and Without Integrator Switching

Integrator output magnitude is limited to mitigate the wind-up effect associated with PID integrators. The limit is application-dependent, and should be set to the minimum value required to generate an output sufficient to overcome any anticipated load resistance.

In theory, the sampling process is modeled as a series of impulse functions, which implies a PID filter calculation time of zero. In reality, the amount of time required to execute a digital filter algorithm must be accounted for, since it introduces phase lag into the system. Even though the lag is minuscule at the frequencies of interest in this note, it is included for the sake of completeness, and is given by :

$$G_{cd}(2) = e^{-sTc} \qquad \text{(EQ 7)}$$

Where T_c = Calculation time in seconds

The MC68HC16Z1 GPT module has two PWM ports, one of which is used as an output for the digital filter. The PWM value is updated every sample period, and is latched until the following sample period. This sample and hold function introduces phase lag, which must be considered. There are several mathematical models for a sample and hold; the one selected for this application is :

$$G_{SH}(s, z) = \left(\frac{1}{s}\right)\left(\frac{z-1}{Tz}\right) \qquad \text{(EQ 8)}$$

The commanded position input to the digital filter comes from another software routine called a profiler, which is responsible for generating a series of positions corresponding to a specific velocity profile (a trapezoidal profile is used for this application). Profiler design is beyond the scope of this note, but, because profiler execution time rivals that of the digital filter, a new commanded position is not calculated at each digital filter sampling interval (488 µs). While the CPU16 could easily perform the extra calculations, a recalculation interval equal to four times the sampling interval was judged sufficient. However, lowering the profiler update rate creates another problem. The commanded position input now has a large step change every fourth filter sample. This introduces a frequency component corresponding to profiler update rate at the filter output, which is quite audible in the motor windings. To compensate, profiler outputs are shifted through a 4-point averager running at the same frequency as the PID filter. The averager linearly interpolates or smoothes profiler data at a 4X oversampling rate, which in turn eliminates motor noise. Filter output waveforms shown in **Figure 7** illustrate the effectiveness of this technique.

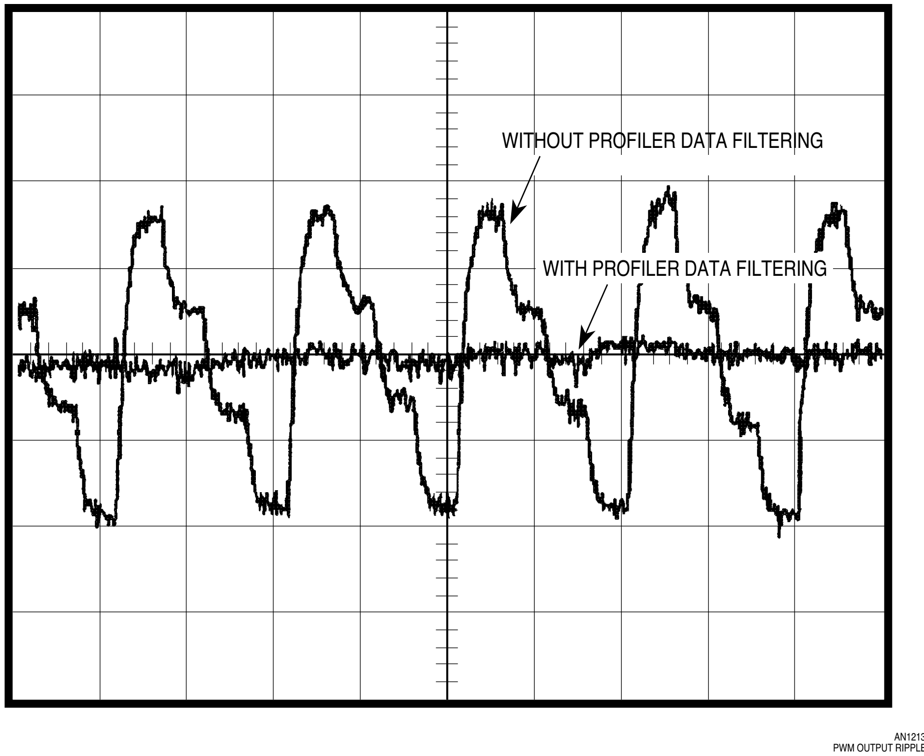

Figure 7 PWM Output Ripple With and Without Profiler Filtration

SELECTING PID COEFFICIENTS

The following points should be taken into consideration when designing a servo.

To assure robust operation and speed, it is generally desirable to have as high a system frequency response as possible.

To obtain adequate damping performance, phase margin (180 degrees minus the phase lag of the open-loop transfer function evaluated at unity gain) should be as large as possible.

To make certain the system can tolerate a large change in gain (e.g., sagging power supply voltage or drifting load parameters) without loss of stability, gain margin (difference in gain between the open-loop transfer function gain evaluated at the point of –180 degrees phase shift, and unity gain) should be as great as possible.

All of these conditions can be observed by generating magnitude and phase frequency plots of the open-loop transfer function G(s,z)H(s,z). The open-loop function can be calculated by breaking the servo loop diagram at a convenient point, then multiplying all individual transfer functions encountered around the loop until arriving back at the break point. However, this procedure yields an equation that is a function of both s and z. Fortunately, there is a mathematical relationship between z and s that allows representation of the equation as a function of s alone.

$$z = e^{sT} \qquad \text{(EQ 9)}$$

Where T is the sampling period in seconds

Assume that the PID controller section is bypassed and calculation delay is zero. By using equation 10 as the transfer function for the motor, and assigning K1 and K2 (Figure 5) values of 0.1875 and 636.62, an open-loop transfer equation as a function of s is obtained.

$$G_{MOTOR}(s) = \frac{\frac{1}{Ke}}{s(1 + st_m)(1 + st_e)} \qquad \text{(EQ 10)}$$

Where:

Ke is the back EMF constant (70.61 mV/(rad/second))
t_m is the mechanical time constant (6.2 ms)
t_e is the electrical time constant (1.62 ms)

Open loop unity gain frequency is approximately 74 Hz, but phase margin is about –20 degrees. The system will oscillate due to poor phase margin —phase compensation is needed. **Figure 8** and **Figure 9** are open-loop transfer function magnitude and phase plots.

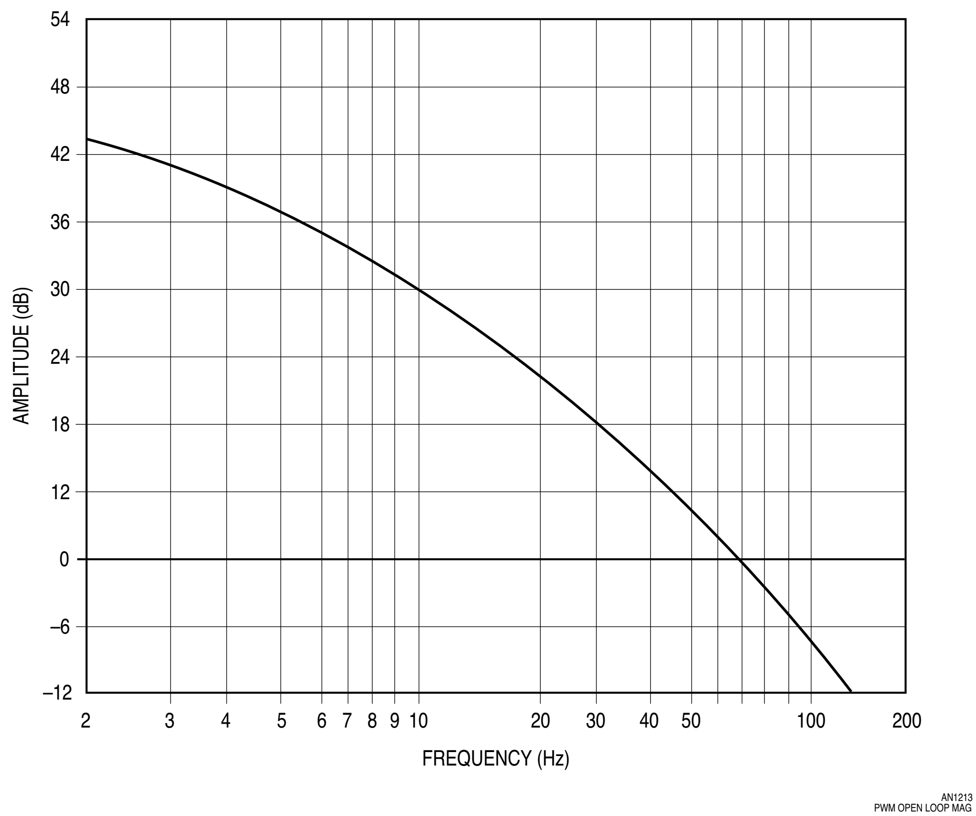

Figure 8 Open-Loop Magnitude Without Compensation

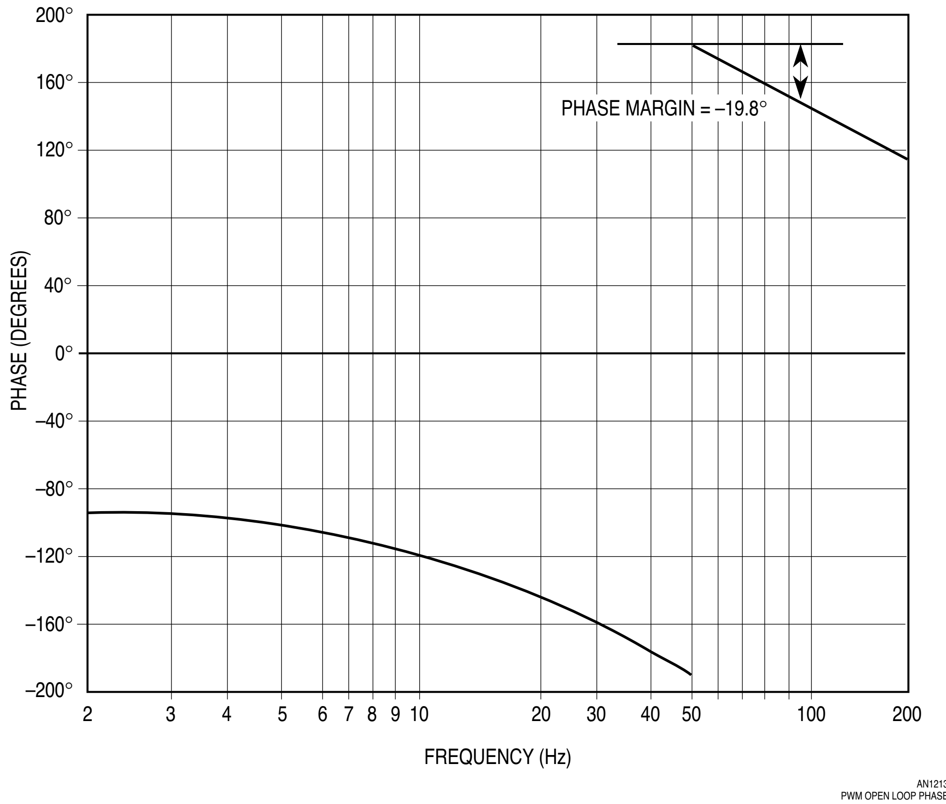

Figure 9 Open-Loop Phase Without Compensation

Values of P, I, and D are iteratively selected using a computer model of the open-loop transfer function that includes the PID controller. An initial T_C value of 30 µs is assumed. Phase margin, gain margin, and frequency response for several sets of terms are calculated, and working values of P = 0.16, I = 5, and D = 1E–3 are selected. Gain margin is –14.4 dB, and phase margin is improved to about 60 degrees, at the expense of lowering open-loop frequency response to 41 Hz. At this frequency, the phase lag generated by the 2-point velocity signal averager is 3.6 degrees. Were a 4-point averager used, phase lag would increase to 10.8 degrees at 41 Hz.

Figure 10 and **Figure 11** are open-loop magnitude and phase plots generated from the working values.

Figure 12 and **Figure 13** are controller transfer function plots generated from the working values. Low-frequency amplification caused by the integrator and high-frequency amplification caused by the differentiator are apparent. Figure 13 shows the positive phase generated by the PID controller.

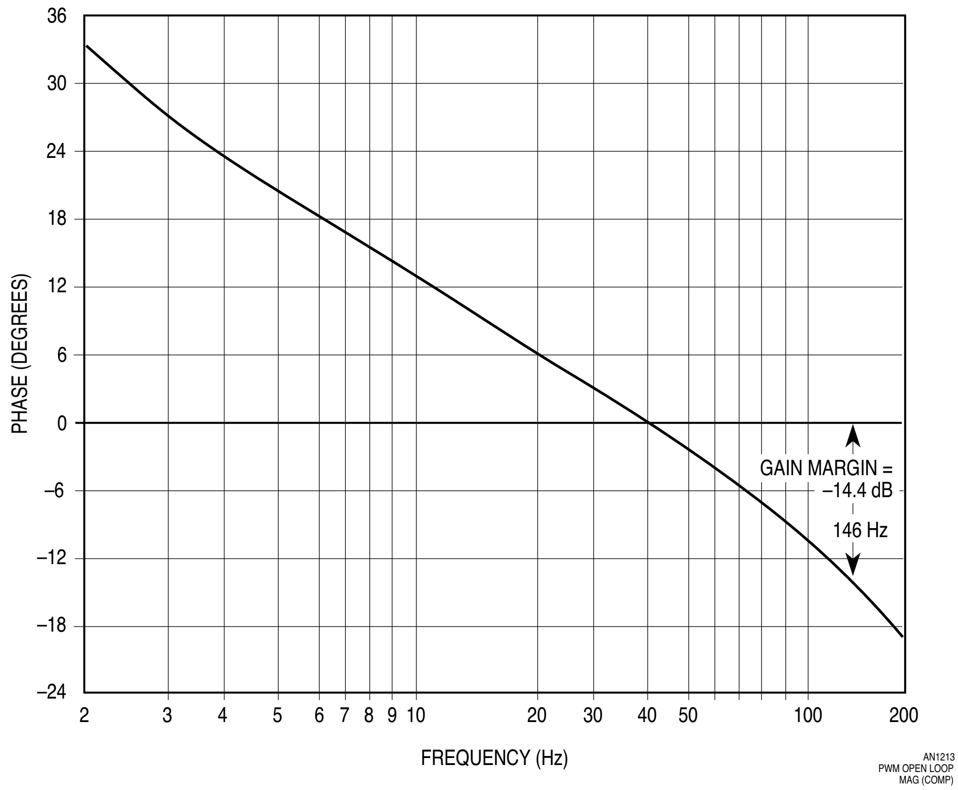

Figure 10 Open-Loop Magnitude With Compensation

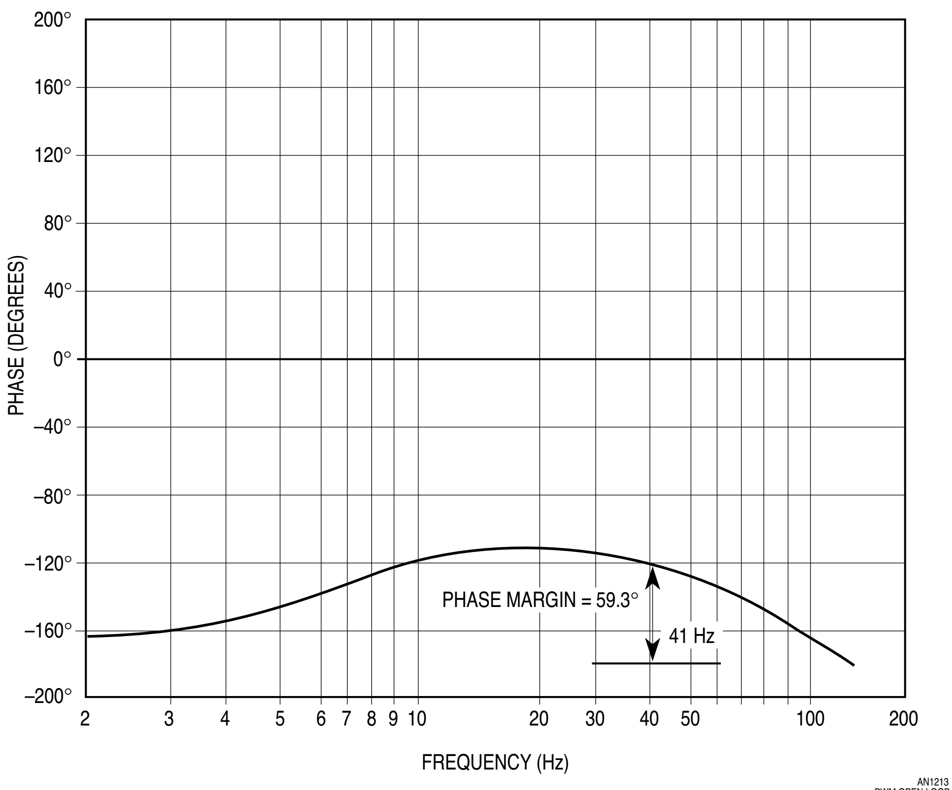

Figure 11 Open-Loop Phase With Compensation

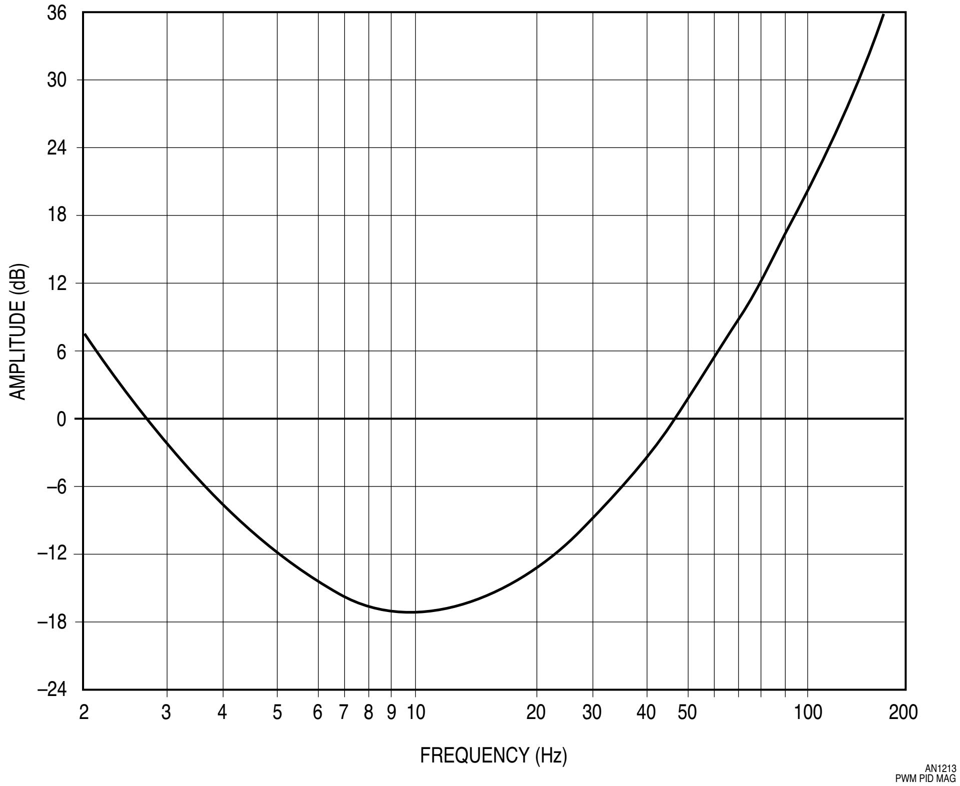

Figure 12 PID Controller Magnitude

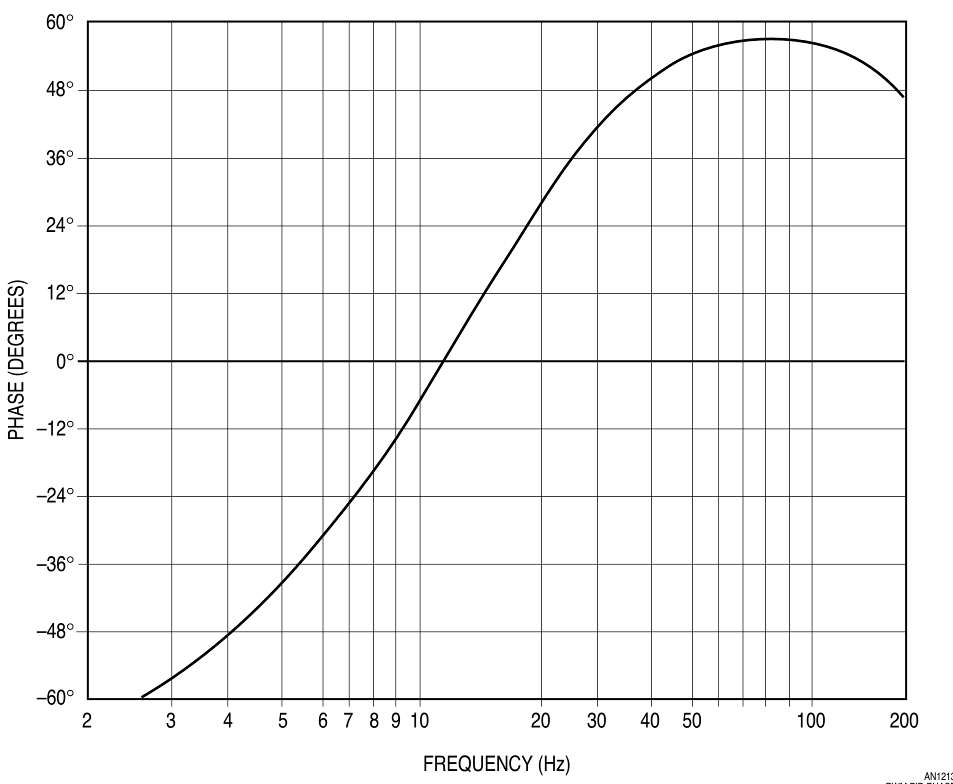

Figure 13 PID Controller Phase

SAMPLED TIME DOMAIN SOLUTION

To realize the controller as a difference equation, solve for the output of each portion of the PID controller.

The P Term

Since P is a constant multiplier, the solution is straightforward :

$$\frac{P(z)}{E(z)} = P \text{ or } P(z) = PE(z) \quad \text{(EQ 11)}$$

where:

P(z) is the output of the proportional stage of the PID controller
E(z) is the error between the commanded position and the feedback position

Performing the transform to the sampled time domain :

$$P(n) = PE(n) \quad \text{(EQ 12)}$$

The notation “n” is used to represent the present sample. Similarly, n+1 indicates the next or future sample, n–1 indicates the previous sample.

The I Term

From Equation 3, we obtain:

$$\frac{Iz}{E(z)} = \frac{ITz}{z-1} \quad \text{(EQ 13)}$$

Where :

I(z) is the output of the integrator stage
T is the sample period (488 µs).

This results in:

$$zI(z) - I(z) = ITz\ E(z) \quad \text{(EQ 14)}$$

Next, the “Shift of Sequence” theorem of the Z transform is used to obtain a solution in the sampled time domain. The theorem is stated as follows:

$$\text{If } Z[x(n)] = X(z), \quad \text{Then } \quad Z[x(n+a)] = z^{a}X(z) \quad \text{(EQ 15)}$$

Where:

Z indicates the Z transform operation.

Applying this theorem in reverse yields:

$$I(n+1) = I(n) + TIE(n+1) \quad \text{(EQ 16)}$$

Now shift the function in time (i.e., n +1 = n) to obtain :

$$I(n) = I(n-1) + TIE(n) \quad \text{(EQ 17)}$$

Simply stated, “the present sample of the integrator output is equal to the previous sample of the integrator output plus T times I times the present sample of the error signal.”

Since output feedback is employed to calculate the next output value, this portion of the filter is classified as an IIR (infinite impulse response) filter.

The D Term

From Equation 5, derive the transfer function for the derivative term:

$$\frac{D(z)}{X(z)} = \left(\frac{D\,(z-1)}{Tz}\right)\left(\frac{1}{2} + \frac{1}{2z}\right) \quad \text{(EQ 18)}$$

where:

X(z) is the feedback position signal
D(z) is the output of the derivative stage.

This equation can be reduced to :

$$2Tz^2D(z) = (Dz^2 - D)X(z) \quad \text{(EQ 19)}$$

Performing the transform to the sampled time domain yields :

$$2TD(n+2) = DX(n+2) - DX(n) \quad \text{(EQ 20)}$$

or

$$D(n+2) = \frac{D}{2T}(X(n+2) - X(n))$$

Shifting in time yields :

$$D(n) = \frac{D}{2T}(X(n) - X(n-2)) \quad \text{(EQ 21)}$$

In other words, the velocity information is derived from X(n), the present position feedback sample, and from X(n–2), the feedback sample made two periods earlier. This yields better quantization results at low speeds.

COMBINING TERMS

From Figure 5, we see that :

$$Y(n) = P(n) + I(n) - D(n) \quad \text{(EQ 22)}$$

Where Y(n) is the PID controller output.

Therefore, combining equations 12, 17, and 21 (EQ 23):

$$Y(n) = \underbrace{PE(n)}_{\text{P Term}} + \underbrace{I(n-1) + TIE(n)}_{\text{I Term}} - \underbrace{\frac{D}{2T}(X(n) - X(n-2))}_{\text{D Term}}$$

Since the proportional term and the integral term both operate directly on E(n), it might appear that these terms could be combined in Equation 23. However, as Figure 5 shows, the integrator must be gated with the velocity term and integrator output must be limited to mitigate wind-up. For these reasons, the P and I terms are kept separate.

Since the terms P, I, D, and T are constants, equation 23 can be rewritten :

$$Y(n) = PE(n) + I(n-1) + aE(n) + b(X(n) - X(n-2)) \quad \text{(EQ 24)}$$

where:

P = 0.16
a = 0.00244
b = –1.0246

CODING THE FILTER

The CPU16 can perform signed and unsigned 16-bit integer multiplication as well as signed and unsigned 16-bit fractional multiplication. Since the PID coefficients are neither all-integer nor all-fraction, the values must be scaled before calculations can be performed, and the same scaling must also be used to correct the filter output. Because CPU16 DSP instructions use fractional notation, each coefficient is divided by 2, so that:

P' = 0.08 or **$0A3D**

a' = 0.00122 or **$0028**

b' = –0.5123 or **$BE6D**

The filter is implemented in an interrupt service routine (ISR) which is called each time the programmable interrupt timer (PIT) in the SIM times out (every 488 µs). The portion of the ISR associated with the PID implementation is listed at the end of this application note. The complete code can be downloaded from Freescale Freeware Data Systems. Modem connection at (512) 891-3733. Internet address (ftp): freeware@aus.sps.mot.com. World-wide web: http://www.freeware.aus.sps.mot.com.

SERVO CONTROL HARDWARE

The MC68HC16Z1EVB provides an excellent platform for this application. Use of CPU16 background debugging mode, the QSM serial communication interface, and the EVB 16-bit DAC are particularly helpful. **Figure 14** is a diagram of the hardware used in the servo system.

The logic to motor interface module (available from Freescale) contains a complementary H-bridge driver (MPM3002) that is used to provide up to 60 volts to a motor load. The board is operated in four-quadrant mode (inverted PWM) —the PWM signal that drives one diagonal transistor pair is an inverted version of the signal that drives the other pair. The PWM interface module prevents excessive heating due to shoot-through current by delaying each enabling PWM edge approximately 2 µs. This provides a switching dead-band between the time one leg is turned off and the other leg is turned on. The PWM interface module also uses the current mirror feedback from the MPM3002 to provide cycle-by-cycle current limiting. For further information about the logic to motor interface module, refer to *Interfacing Microcomputers to Fractional Horsepower Motors* (AN1300).

Operating with four-quadrant PWM implies that the waveform generated by the MC68HC16Z1 must be bipolar (50% PWM corresponds to no voltage on the motor). As the sample ISR code shows, this is accomplished by adding a fixed offset to the digital filter output before it is put in the PWM register. Once the value is in the PWM register, the general-purpose timer (GPT) generates the required PWM signal without further CPU intervention.

The 1000-slot encoder on the motor shaft is processed by the Hewlett-Packard HCTL-2016 quadrature decoder, which accumulates the encoder count on an internal 16-bit up/down counter. One of the 12 MC68HC16Z1 chip-select outputs is programmed to perform the address decoding necessary to access and read the HCTL-2016 counter data. 16-bit data is read on data bus pins DATA[15:8] as two sequential 8-bit values. The 16-bit data word is then used to synthesize a 32-bit absolute position variable, as shown in the beginning of the PID ISR listing.

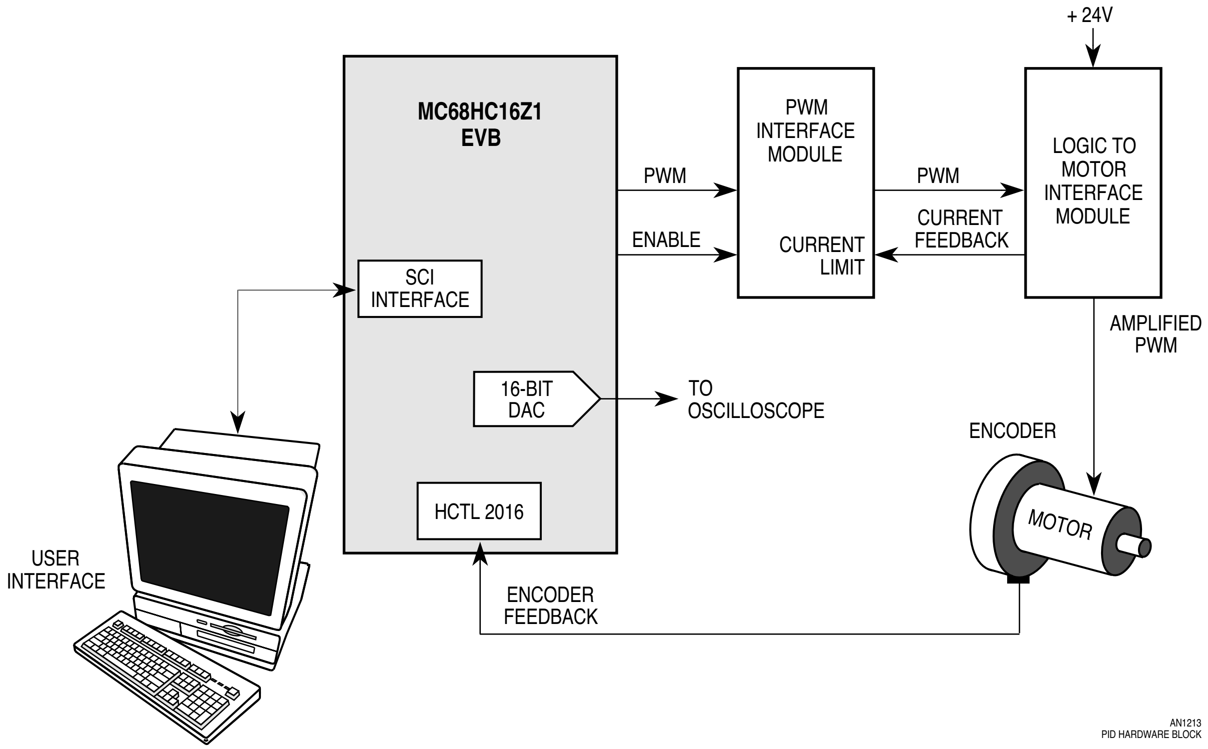

Figure 14 Hardware Block Diagram

A user can control the application via a terminal connected to the serial communications interface (SCI) in the queued serial module (QSM). The MC68HC16Z1 EVB provides a 25-pin D connector for this purpose. To initiate motor movement, issue a **MOVE** command, specifying the final position, the maximum profiler velocity, and the acceleration of the move. The SCI transmits and receives such commands with very little CPU intervention, even during a move.

The MC68HC16 EVB also provides for installation of a Burr-Brown PCM56P 16-bit serial DAC. If the QSM Serial Peripheral Interface (SPI) is used to drive the DAC, 16-bit DAC updates can be provided at approximately 4 µs intervals (SCLK frequency = 4.19 MHz). DAC output can be used to probe portions of the servo loop that are not readily available in analog form, such as shaft encoder position or integrator stage output. The oscilloscope plots in this note were generated by transmitting variable values to the DAC and updating them every pass through the sample ISR (once every 488 µs).

CONCLUSIONS

No matter how thorough the analysis, a working design usually requires some adjustment. The designer must compare actual system function to theoretical expectations, then tune system coefficients to achieve optimum performance.

The D term has a dramatic effect on system stability, and should be the first term adjusted when damping performance is less than satisfactory. **Figure 15** shows how varying D affects system damping, and indicates that the original calculated value for D provides for quick settling time with little or no overshoot.

As mentioned earlier, the I term is used to “servo out” steady-state position error. To demonstrate, a frictional load was applied to the motor shaft, and the system step response was measured with the integrator enabled and disabled. When disabled, the position error was measured to be between 65 –69 encoder counts (approximately 6 degrees). With I set to its default value of 5, the error did not exceed one encoder count (0.09 degrees).

As Equation 5 shows, the PID calculation must be performed quickly, to minimize phase delay introduced into the loop. Calculation time can be measured by connecting an oscilloscope to an I/O pin and running a subroutine that drives the pin high just before encoder sampling, then drives it low it just after the PWM register is loaded with a new value. Measured in this way, calculation time is roughly 30 µs, which translates into 0.44 degrees of phase lag at the open-loop unity gain frequency (41 Hz), a minimal contribution to total system phase lag at that frequency. Also, since less than 10 percent of CPU time is spent controlling axis motion, multiple axes or more sophisticated control algorithms, including state observers, notch filters, and adaptive control, could easily be implemented.

The techniques in this application note allow a designer to customize a controller for an application, rather than be forced to work around pre-packaged servo controls. The capabilities of the M68HC16 family also enable designers to respond quickly to changes, and permit solution of real-time DSP control problems without loss of the user friendliness provided by a complete microcontroller system.

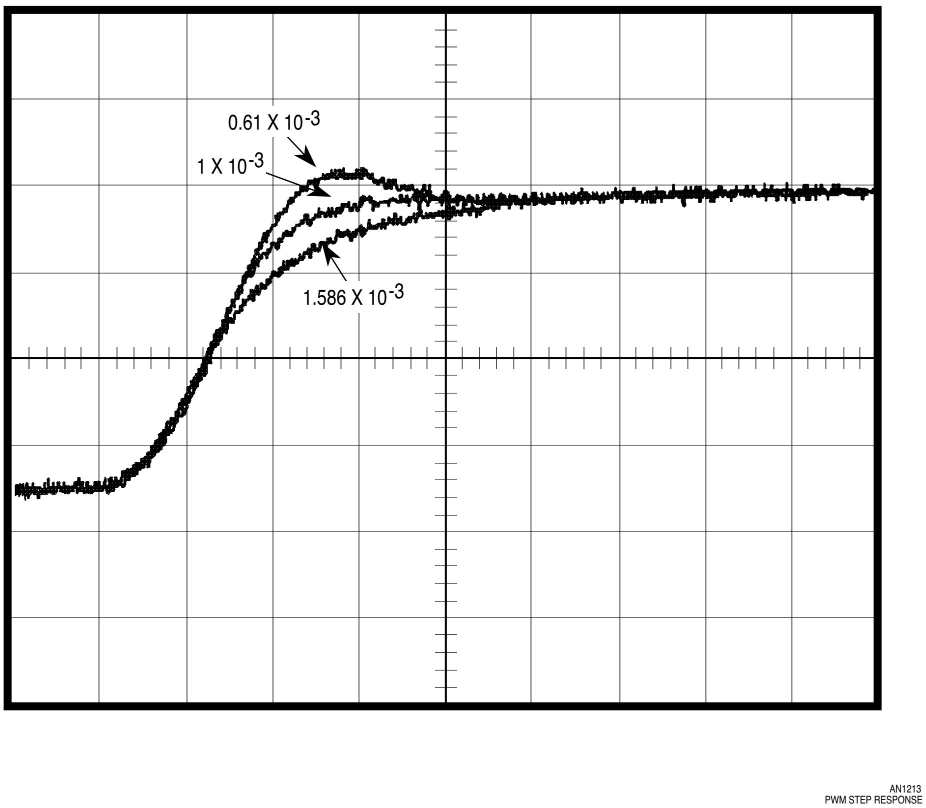

Figure 15 Effect of Varying D on System Damping

REFERENCES

1. Kuo, B.,
   Digital Control Systems,
   Holt, Rinehart and Winston, Inc., 1980
2. Oppenheim, A.V., Schafer, R.W.,
   Digital Signal Processing,
   Prentice-Hall, Inc., 1975

PID INTERRUPT SERVICE ROUTINE LISTING

```
                            * Here we go!
   00062E 3908 FA19                 bset   PORTF0,#test_bit1 ;set test_bit1 output for timing meas.

   000632 37F5 0900                 ldd    enc                      ;read the encoder.
   000636 37FA 0004                 std    Xn+2                     ;update lower word of Xn
   00063A 37F0 0008                 subd   Xn_1+2                   ;subtract old lower word from new one
   00063E BB00                      bmi    enc_neg                  ;IF delta is positive
   000640 2775                      clre                            ; sign extend to AccE ($0000)
   000642 B000                      bra    add_delta                ;ELSE
   000644 3735 FFFF         enc_neg lde    #$FFFF                   ; sign extend to AccE ($FFFF)
                                                                    ;ENDIF
                            add_deltaADDMLONGXn_1
                                    +*Adds a 32-bit value in memory at "location" to
                                    +*the concatenated value in D and E.
   000648 37F1 0008         addd    Xn_1+2
   00064C 3773 0006         adce    Xn_1
   000650 377A 0002                 ste    Xn                       ;Xn now updated.

   000654 2771 003C                 lded   pcommand
                                    SUBMLONGXn
                                    +*Subtract a 32-bit variable in memory at "location" from
                                    +*the concatenated value of the D and E registers.
   000658 37F0 0004         +subd   Xn+2
   00065C 3772 0002         + sbce  Xn
   000660 37FA 0014                 std    En                       ;E(n) = commanded position - x(n)

                            *E(n) must be limited to a 16-bit number
   000664 2776                      tste                            ;check whether E(n) is negative
   000666 BC10                      bge    Epositive                ;IF E(n) is negative
   000668 37B1 8000                 addd   #$8000
   00066C 3733 0000                 adce   #$0000                   ;add $00008000 to E(n)
   000670 BC18                      bge    E_ok                     ; IF result is negative
   000672 37B5 8000                 ldd    #$8000
   000676 37FA 0014                 std    En                       ; E(n) = $8000
   00067A B00E                      bra    E_ok                     ;ENDIF

   00067C                   Epositive                               ;ELSE
   00067C 37B0 8000                 subd   #$8000
   000680 3732 0000                 sbce   #$0000                   ; subtract $00008000 from E(n)
   000684 BD04                      blt    E_ok                     ; IF result is zero or positive
   000686 37B5 7FFF                 ldd    #$7FFF
   00068A 37FA 0014                 std    En                       ; En = $7FFF
                                                                    ; ENDIF
                                                                    ;ENDIF
   00068E                   E_ok
   00068E 2771 0002                 lded   Xn
                                    SUBMLONG Xn_2
                                    +*Subtract a 32-bit variable in memory at "location" from
                                    +*the concatenated value of the D and E registers.
   000692 37F0 000C         +subd   Xn_2+2
   000696 3772 000A         +sbce   Xn_2
   00069A 37FA 0010                 std    Xn1                      ;x`(n) = x(n) - x(n-2)

                            *shift the sampled encoder data

   00069E 2771 0006                 lded   Xn_1
   0006A2 2773 000A                 sted   Xn_2
   0006A6 2771 0002                 lded   Xn
   0006AA 2773 0006                 sted   Xn_1

                            * now perform the digital filter
```

```
109
110 0006AE 37F5 0010          ldd    Xn1
111                           ABS
 1m                           +*Take the absolute value of the D register.
 2m 0006B2 27F6               +tstd
 3m 0006B4 BCFE               +bge   1positive
 4m 0006B6 27F2               +negd
 5m 0006B8                    +1positive:
112 0006B8 37B0 0005          subd   #$0005
113 0006BC BD06               blt    Icalc                 ;IF ABS(X`n) >= $0005
114
115 0006BE 27F5               clrd
116 0006C0 2775               clre
117 0006C2 2773 0018          sted   In_1                  ; I(n-1) = 0 (clear "I" term)
118 0006C6 B040               bra    I_loaded
119
120 0006C8            Icalc                                ;ELSE
121 0006C8 2771 0012          lded   a                     ; a ==> AccE , E(n) ==> AccD
122                                                        ; (E(n) follows a in variable
map)
123 0006CC 3727               fmuls
124                           ADDMLONG In_1
 1m                           +*Adds a 32-bit value in memory at "location" with
 2m                           +*the concatenated value in D and E.
 3m 0006CE 37F1 001A  +addd   In_1+2
 4m 0006D2 3773 0018  +adce   In_1
125 0006D6 2773 0018          sted   In_1                  ; I(n) = I(n-1) + a*E(n)
126                                                        ; Now limit the integrator
127                                                        ; term to 19 bits.
128 0006DA BC10               bge    Ipositive             ; IF I(n) is negative
129 0006DC FC00               addd   #$0000
130 0006DE 3733 0008          adce   #$0008                ; add $00080000 to I(n)
131 0006E2 BC20               bge    get_I                 ; IF result is negative
132 0006E4 27F5               clrd
133 0006E6 3735 FFF8          lde    #$FFF8
134 0006EA 2773 0018          sted   In_1                  ; I(n) = $FFF80000
135 0006EE B018               bra    I_loaded              ; ENDIF
136                                                        ; ELSE
137 0006F0            Ipositive
138 0006F0 37B0 0000          subd   #$0000
139 0006F4 3732 0008          sbce   #$0008                ; subtract $00080000 from
I(n)
140 0006F8 BD0A               blt    get_I                 ; IF result is zero or posi-
tive
141 0006FA 37B5 FFFF          ldd    #$FFFF
142 0006FE 3735 0007          lde    #$0007
143 000702 2773 0018          sted   In_1                  ; I(n) = $0007FFFF
144 000706 B000               bra    I_loaded              ; ENDIF
145                                                        ; ENDIF
146 000708            get_I
147 000708 2771 0018          lded   In_1
148
149 00070C            I_loaded                             ;ENDIF
150
151 00070C 27B1               tedm                         ;AM = I(n-1) + a*E(n)
152 00070E 2771 0014          lded   En                    ;E(n) ==> AccE , p ==> AccD
153                                                        ; (p follows E(n) in variable
table)
154 000712 3727               fmuls
155 000714 3723               aced                         ;AM = I(n-1) + a*E(n) +
p*E(n)
156 000716 2771 000E          lded   b                     ;b ==> AccE, X`(n) ==> AccD
(since
157                                                        ;X`(n) follows "b" in vari-
able map)
158 00071A 3727               fmuls
159 00071C 3723               aced                         ;AM = I(n-1) + a*E(n) +
p*E(n)
```

```
                                                                ; + b*x`(n)

     00071E 27B6                aslm                            ;compensate for data scaling of
                                                                ;PID coefficients (multiply by 2).
     000720 27B4                tmer                            ;transfer result to AccE rounded.
     000722 377A 001C           ste    Yn                       ;Yn is output of filter

                        *Now the pwm value must be limited to an 8-bit value.

     000726 27FB                ted
     000728 BC0A                bge    Ypositive                ;IF Y(n) is negative
     00072A FC7F                addd   #$007F                   ; add $007F to Y(n)
     00072C BC14                bge    get_Y                    ; IF result is negative
     00072E 37B5 FF81           ldd    #$FF81
     000732 37FA 001C           std    Yn                       ; Y(n) = $FF81 (minimum value—
                                                                ; PWM interface module always
                                                                ; needs a PWM edge to do cycle-
                                                                ; by-cycle current limiting)

     000736 B00E                bra    Y_loaded                 ; ENDIF
                                                                ;ELSE
     000738             Ypositive
     000738 37B0 0080           subd   #$0080                   ; subtract $0080 from Y(n)
     00073C BD04                blt    get_Y                    ; IF result is zero or positive
     00073E 37B5 007F           ldd    #$007F
     000742 37FA 001C           std    Yn                       ; Y(n) = $007F (maximum PWM allowed)
                                                                ; ENDIF
                                                                ;ENDIF
     000746             get_Y
     000746 37F5 001C           ldd    Yn
     00074A 37B1 0080   Y_loaded addd  #$0080
     00074E 17FA F926           stab   PWMA                     ;scale PWM so that 50% is zero volts.

   * We are done with PID filter at this point.
```