MOTOROLA

APR5/D
Rev. 1

Implementation of PID Controllers

on the

Motorola DSP56000/SPS/DSP56001 Digital Signal Processors

Motorola's High-Performance DSP Technology dsp

Motorola Digital Signal Processors

Implementation of PID Controllers on the Motorola DSP56000/DSP56001

by
Jay Stokes and Guy R. L. Sohie
Digital Signal Processor Operation

Table of Contents

Table of Contents

Illustrations

Illustrations

Illustrations

Illustrations

SECTION 1

Introduction

"The DSP56000/ DSP56001 offers the control designer the advantages of digital electronics in a low-cost package with the processing power to handle sampling rates up to a megahertz for simple algorithms."

The purpose of this application note is to show how the Motorola DSP56000/DSP56001 digital signal processor (DSP) may be used to solve real-time digital control problems. This application note will concentrate on implementing some general control algorithms including proportional-integral-derivative (PID) controllers and notch filters.

In the past, many real-time control systems have been restricted to analog electronics; however, analog components have several problems. Device parameters are dependent upon age, temperature, power supply voltages, and manufacturing lot. In addition, analog filters and controllers exhibit low noise immunity, require periodic tuning, and offer little flexibility when altering coefficients.

These problems can be solved by implementing the control structure using a microprocessor, microcontroller, or DSP. Digital controllers and filters are not dependent upon age or environmental factors, and software can be easily modified. In addition, when using a fast microprocessor, microcontroller, or DSP, many difficult control structures can be implemented in software without additional hardware. Another benefit of digital control is high noise immunity.Once the

observed variables are digitized, virtually no additional external noise can be added to the system. Digital control algorithms tend to be more precise than their analog counterparts.

In the past, designers used analog electronics to implement real-time control systems because microcontrollers were simply too slow to handle sampling rates of more than a few kilohertz. Other solutions built around powerful microprocessors were too expensive to be practical. The DSP56000/DSP56001 offers the control designer the advantages of digital electronics in a low-cost package with the processing power to handle sampling rates up to a megahertz for simple algorithms.

The DSP56000/DSP56001 is both a high-speed microcontroller and a powerful DSP. The DSP56001 has 512 24-bit words of program RAM, which can be downloaded upon reset from a single 2K x 8 EPROM or a host processor. For high-volume products, the DSP56000 offers 3.75K 24-bit words of program ROM, which can be factory programmed for a stand-alone system. Except for the program memory, the DSP56000and DSP56001 are identical. The DSP56000/DSP56001 includes two separate memory spaces for data, which can be simultaneously accessed during a single instruction cycle. Another feature of the DSP56000/DSP56001 is a hardware multiply/accumulator for implementing computationally intensive real-time control algorithms. The DSP56000/DSP56001 can multiply

two 24-bit numbers, add the 48-bit result to a 56-bit accumulator, and access both memories in one instruction cycle. For fast input/output (I/O) operations, the DSP56000/DSP56001includes three on-chip peripherals; the host interface (HI), the synchronous serial interface (SSI), and the serial communications interface (SCI). When the SCI is not being used for communication purposes, its baud rate generator functions as a general-purpose timer. Depending on how these peripherals are configured, the DSP56000/DSP56001 has up to 24 general-purpose I/O lines available. All these features allow the DSP56000/DSP56001 to be used in many computationally intensive real-time control applications including: disk drives, engine control, computer-controlled suspension, active noise cancellation, and robotics.

Some basic analog control concepts and PID controllers are discussed in **SECTION 2 Classical Analog Controls**. Specifically, this section studies the effects of varying the controller gain and adding derivative and integral terms in a closed-loop feedback system. A general transfer function is derived in the Laplace domain for a continuous-time PID controller. Notch filters are often used in control systems to eliminate the effects of mechanical resonances.The general transfer function for many types of filters, including notch filters, is shown to be equivalent to the transfer function of PID controllers. After transforming this analog transfer function into the digital domain, several implementation issues are considered with respect to the DSP56000/DSP56001. Finite-length register effects, including

coefficient quantization, overflow, and roundoff noise of 16- and 24-bit implementations, are then calculated. Finally, system issues concerning how the DSP56000/DSP56001 would fit into a embedded control system are studied. Examples of generating pulse-width modulated (PWM) outputs and three-phase outputs are given. Also, a simple scheme for measuring velocity is presented. ■

SECTION 2

Classical Analog Controls

"For the system to be stable, the poles of the closed-loop transfer function must lie in the left half of the s-plane."

A block diagram of a general analog control system in the Laplace domain is shown in Figure 2-1. The forward transmission path transfer function is given by G(s); H(s) represents the feedback path transfer function. Typically, the feedback element contains a sensor or transducer that measures a physical parameter, such as velocity or temperature, and converts this measurement into a voltage. Therefore, H(s) represents the gain of the transducer.

The basic idea in many control problems is to make the controlled output, c(t), follow the reference input, r(t), in the frequency range of interest. Therefore, the desired transfer function is:

$$T(s) = \frac{C(s)}{R(s)} = 1 \qquad \text{Eqn. 2-1}$$

where: R(s) and C(s) are the Laplace transforms of r(t) and c(t), respectively

If r(t) varies with time, the problem is commonly referred to as the tracking problem; otherwise, if the input remains constant, it is called a regulator problem. Figure 2-1 represents a closed-loop feedback system since the controlled variable is measured and used to control the system. If C(s) is not measured,

H(s)=0, and the system is open-loop. The problem of making c(t) follow r(t) is usually compounded by disturbances in the forward transmission path and the feedback path. Neglecting any disturbances in the system, the error between the input and the output of the system is given by E(s)=R(s)-H(s)C(s). However, C(s)=G(s)E(s). Solving these two equations, the closed-loop transfer function is found to be:

$$T(s) = \frac{C(s)}{R(s)} = \frac{G(s)}{1 + G(s)H(s)} = \frac{(s - z_1)\ldots(s - z_m)}{(s - p_1)\ldots(s - p_n)} m < n \qquad \text{Eqn. 2-2}$$

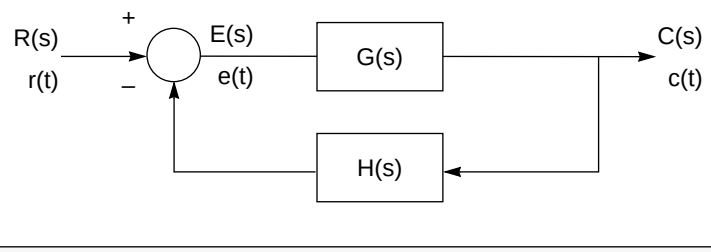


Figure 2-1 General Analog Control System

The poles of the closed-loop transfer function are found by solving the characteristic equation:

$$1 + G(s)H(s) = 0 \qquad \text{Eqn. 2-3}$$

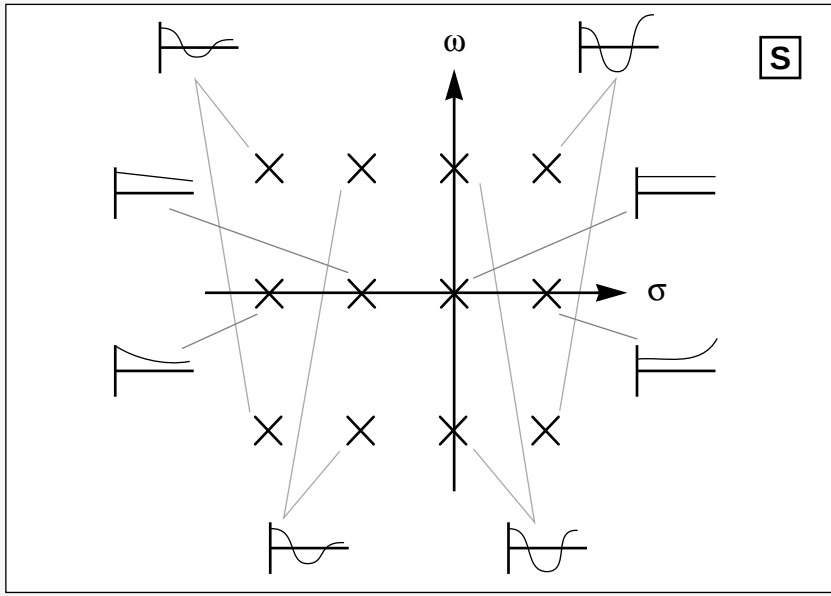


Figure 2-2 Transient Response of a Second-Order System for Various Pole Locations in the s-Plane

Figure 2-2 shows how the location of the poles of a second-order closed-loop transfer function affects the output. For the system to be stable, the poles of the closed-loop transfer function must lie in the left half of the s-plane. In an unstable system with a bounded input, the output becomes unbounded as a result of the poles lying in the right half of the s-plane.

The specifications for a control problem are often given in the time domain as opposed to the frequency domain. These specifications usually include a

certain transient response and steady-state error for a specific input such as a step or ramp function. The transient response specifications for a step input are shown in Figure 2-3 and include the rise time (t_r), percent overshoot, and settling time (t_s). The rise time is defined as the time for the output to rise from 10% to 90% of the step input. The settling time is the time required for the output to settle within a given percentage of the input step function. A typical number for this percentage is 5%. The steady-state specification in the time domain for a step input is also shown in Figure 2-3. The final deviation from the desired step input as $t \rightarrow \infty$ is called the steady-state error (e_{ss}) or offset. To improve the step response, the rise time, percent overshoot, and steady-state error should all be minimized. ■

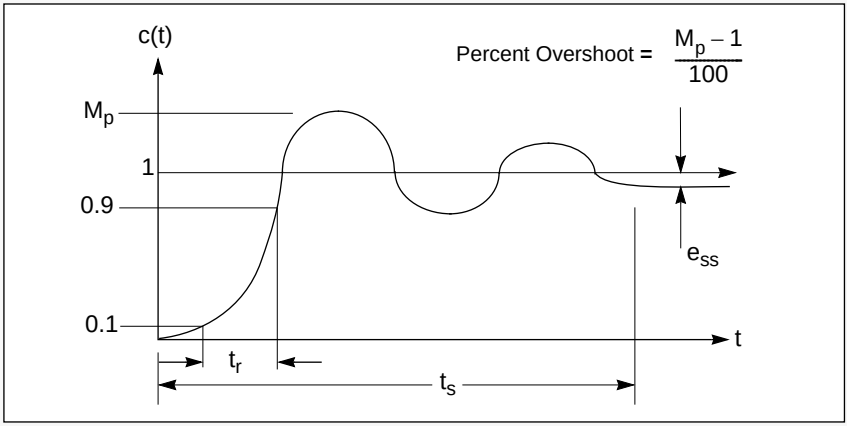


Figure 2-3 Typical Step-Response Characteristics Include Rise Time (t_r), Settling Time (t_s), Percent Overshoot, and Steady-State Error (e_{ss})

SECTION 3

Controllers

"By increasing the gain of the P controller, the rise time of the system can be decreased, allowing the output to follow the input faster."

One way to improve the step response of a control system is to add a controller to the feedback control system in Figure 2-1. The block diagram of a control system, including a controller, $G_c(s)$, is found in Figure 3-1. In this figure, the controller has been added to the forward transmission path although it could have been placed in the feedback path. The error signal from the summing junction, E(s), is the input to the controller, and U(s) is the output of the controller as well as the input to the plant. For this system with unity feedback gain, H(s)=1, and the transfer function is:

$$\frac{C(s)}{R(s)} = \frac{G_c(s)G_p(s)}{1 + G_c(s)G_p(s)} \qquad \text{Eqn. 3-1}$$

where: $G_c(s)$ = the transfer function of the controller
$G_p(s)$ = the transfer function of the plant to be controlled

Again, the purpose of the controller is to make the output of the system follow or track the input such that:

$$T(s) = \frac{G_c(s)G_p(s)}{1 + G_c(s)G_p(s)} \approx 1 \qquad \text{Eqn. 3-2}$$

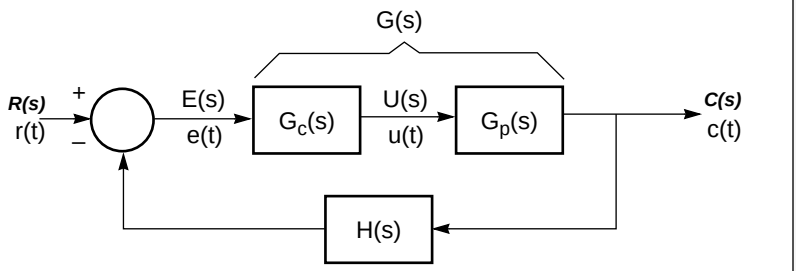


Figure 3-1 General Feedback Control System with Compensation

3.1 Increasing the Gain to Reduce the Rise Time

One way to make the output follow the input is to increase the gain of the controller such that $G_c(s)G_p(s)>>1$ for frequencies of interest. In this case, Eqn. 3-1 reduces to:

$$\frac{C(s)}{R(s)} \approx \frac{G_c(s)G_p(s)}{G_c(s)G_p(s)} = 1 \qquad \text{Eqn. 3-3}$$

A controller consisting of only a large gain is called a proportional controller (P). Consider what happens when a P controller is used in Figure 3-1. If c(t) is less than the r(t), a positive error signal, e(t), results. After this positive error voltage has been amplified by the gain of the P controller, the output of the system is increased to track the input signal.

Likewise, if c(t) is greater than r(t), then a negative error voltage is applied to the plant, and the output of the system is reduced. The speed at which the output can respond to the error signal is dependent upon the magnitude of the gain of the P controller. By increasing the gain of the P controller, the rise time of the system can be decreased, allowing the output to follow the input faster.

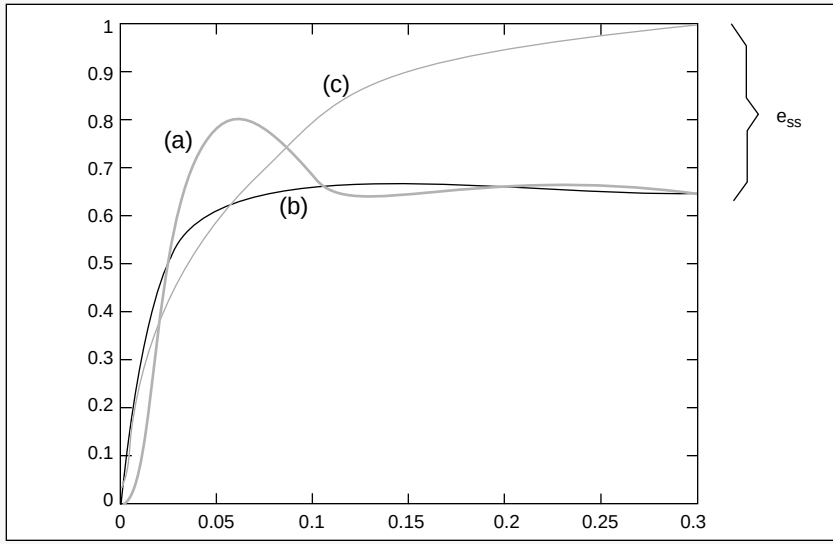


Figure 3-2 Step Response of a Closed-Loop Feedback System Shown for a (a) P Controller, (b) PD Controller, and (c) PID Controller

However, a problem is associated with simply increasing the gain of the P controller. The overshoot of the output is increased as the gain of the P controller is increased, causing a damped oscillatory

output. This type of output is illustrated in the (a) graph of Figure 3-2. If the gain is increased further, the system will become critically stable at some point, and the output will oscillate. When this occurs, the poles of the closed-loop transfer function (see Eqn. 2-2) lie on the jω–axis in the s-plane (see Figure 2-2). Increasing the gain past the critically stable point causes the system to become unstable, and the output increases without bound. In addition, a constant gain amplifies the high-frequency noise in addition to the lower frequency bandwidth of the control system. This high noise amplification can greatly distort the lower frequency control signals.

3.2 Adding a Derivative Term to Reduce Overshoot

One way to reduce the rise time without increasing the percent overshoot is to add a derivative term to the P controller. The transfer function of a proportional-plus-derivative (PD) controller is:

$$G_c(s) = K_p + K_d s \qquad \text{Eqn. 3-4}$$

Figure 3-3 (from Reference 7) shows a typical step response of a closed-loop system without compensation. The output of the system, c(t), the error signal, e(t), and the derivative of the error signal, de(t)/dt, are shown. The de(t)/dt graph represents

the slope of the error signal and provides information about how the error signal is changing with respect to time. For a system with zero overshoot, the derivative of the error signal should not oscillate about the x-axis (see Figure 3-3(c)) but should approach the x-axis asymptotically. A derivative term has been added to the P controller in the (b) graph of Figure 3-2. As Figure 3-2 illustrates, the rise time of the PD controller is as fast as the P controller, but the output does not exhibit any overshoot. By including the derivative term in $G_{c(}s)$, the controller can estimate future values of the error signal and can compensate accordingly.

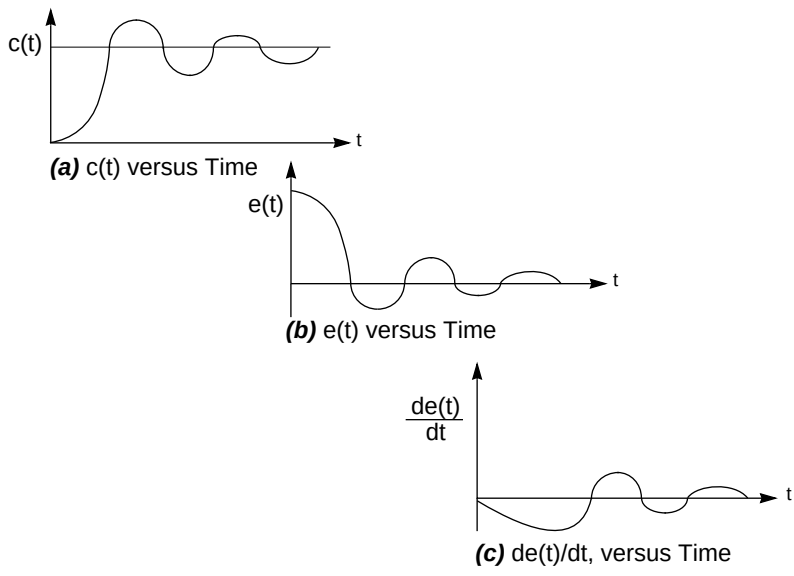


Figure 3-3 Output, c(t), Error, e(t), and the Derivative of the Error, de(t)/dt, Versus Time

In addition to improving the transient response, a controller is added to a feedback system to decrease the steady-state error. If the steady-state error is constant, the contribution of the derivative term is zero since the derivative of a constant is zero. However, if the steady-state error is time-varying, the derivative term can be used to reduce this offset. Again, the derivative of the steady-state error predicts future values of the error and can be used to reduce e_{ss}. The equation for the steady-state error is given by:

$$e_{ss} = \lim_{t \to \infty} e(t) = \lim_{s \to 0} sE(s) \qquad \text{Eqn. 3-5}$$

The error term is dependent upon the input to the system as well as the system itself (see Figure 3-1). Depending on the input, many systems with a P or PD controller will exhibit some steady-state offset. If the controlled variable ever matches the reference value exactly, c(t)=r(t), then the error signal, e(t), equals zero. For a constant zero input to a P or PD controller, the output of the controller, u(t), also equals zero. This condition forces the output of the plant, c(t), to zero. Of course, this deviation of the output would then change the error signal, and the controller would again try to readjust the input to the plant to match r(t). Eventually, the output of the plant reaches an equilibrium state that is offset from the desired step input. This offset is called the steady-state error. In Figure 3-2, the output of a system with either a P or PD controller exhibits an offset of approximately 0.375 for a unit-step input.

Another problem associated with the PD controller is that it functions as a high-pass filter. Therefore, the PD controller amplifies high-frequency noise, which reduces the stability of the overall system.

3.3 Adding an Integral Term to Eliminate Steady-State Error

For the controlled output, c(t), to exactly match the reference input, r(t), the system must have zero steady-state error. One way to eliminate the e_{ss} is to add an integral term to the P controller. The transfer function for a PID controller is:

$$G_c(s) = K_D s + K_p + \frac{K_i}{s} = \frac{K_d s^2 + K_p s + K_i}{s} \quad \text{Eqn. 3-6}$$

Adding an integral term gives the controller the ability to remember the past. For the P or PD controller, the steady-state error can not be driven to zero since a zero input to the controller forces a zero input to the plant. The integral term allows the PID controller to have a nonzero output for a zero input. This integral term is similar to a capacitor holding a charge. By remembering the correct input to the plant, u(t), which corresponds to the matching of c(t) to r(t), the PID controller is able to constantly drive the plant, allowing the controlled output to exactly match the reference input. Therefore, the

integral term allows zero steady-state error. The effects of adding an integral term to the P and PD controllers is shown in graph (c) of Figure 3-2. The output of the PID reaches unity and has no steady-state error.

Two problems are introduced by adding the integral term. Since the integrator adds another pole to the closed-loop transfer function, the stability of the system may be reduced. Also, the integral term acts as a low-pass filter and tends to reduce the transient response of the system. Graph (c) in Figure 3-2 shows how the transient response of this system is slowed by the addition of the integral term.

As previously mentioned, the derivative portion of the PID controller amplifies high-frequency noise. To solve this problem, two or more poles must be added at higher frequencies. Additional zeros may be needed to obtain a specific frequency response for the controller. Depending on the number of zeros and poles added, the resulting controller has the generalized transfer function:

$$G_c(s) = \frac{\beta(0) + \beta(1)s^{-1} + \beta(m)s^{-m}}{1 + \alpha(1)s^{-1} + \ldots + \alpha(n)s^{-n}} m < n \qquad \text{Eqn. 3-7}$$

■

SECTION 4

Notch Filters

"...the techniques used to implement PID controllers can also be used to implement notch filters."

Notch filters are often added to control systems to negate the effects of a mechanical resonance at a specific frequency. Suppose a motor has a frequency response such as the one illustrated in Figure 4-1(a). When the motor reaches a speed corresponding to f_r, it becomes unstable and begins to vibrate due to the mechanical resonance. These mechanical vibrations tend to shorten the life of the motor. By filtering the output with a notch filter as shown in Figure 4-1(b), the gain of the resonance can be reduced. The resultant system's frequency response is shown in Figure 4-1(c). The transfer function of a notch filter, $G_f(s)$, has a form equal to the transfer function of the controller shown in Eqn. 3-7. Therefore, the techniques used to implement PID controllers can also be used to implement notch filters. ■

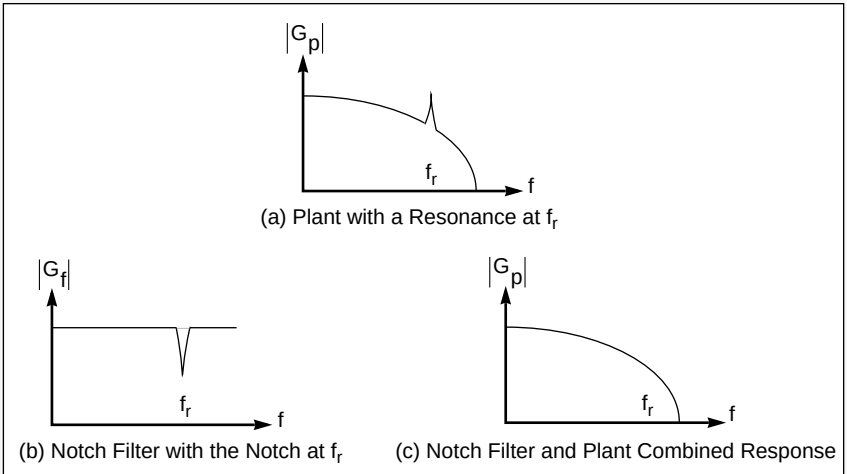


Figure 4-1 Frequency Response

SECTION 5

Control in the Digital Domain

"Using an idea similar to the bilinear transform, a technique involving a pseudo-discrete plane, called the w-plane..., allows the engineer to design with classic analog techniques."

Analog control is a well-defined discipline. Many methods for designing analog controllers in the time and frequency domains are found in Reference 7. Some of the more common design techniques include root locus, Routh-Hurwitz, Nyquist plots, and Bode diagrams. These design tools try to determine the location of the poles of the closed-loop transfer function without directly solving the characteristic Eqn. 2-3. One way to design digital controllers and filters is to transform an analog design to the z-domain. Some of the more common transforms include input invariance methods and the bilinear transform (see References 4, 5, 6, 9, and 11). Input invariance methods sample the output of a continuous-time system for a given continuous-time input. The types of inputs typically considered include the impulse, step, and ramp functions although the impulse response is not particularly relevant in control applications. A discrete-time system is then calculated, which, when driven with the discrete-time version of the input, yields a discrete-time output matching the sampled output of the continuous-time system. The major problem with input invariance methods is that aliasing occurs due to the sampling process.

Another method of transforming an existing analog design to the z-domain is to use the bilinear transform given by:

$$s = \frac{2}{T}\left(\frac{1-z^{-1}}{1+z^{-1}}\right) \qquad \text{Eqn. 5-1}$$

The bilinear transform compresses the entire $j\omega$ axis in the s-plane to the frequency range bounded by $\pi/T/2 \leq \omega \leq \pi/T$, where T represents the sampling period. Since sampling is not used in this transform, the output will not exhibit aliasing, but the frequency response of the system is distorted due to the compression of the $j\omega$ axis. Using an idea similar to the bilinear transform, a technique involving a pseudo-discrete plane, called the w-plane (see References 6 and 10), allows the engineer to design with classic analog techniques.

An alternative to these analog-based controllers and filters is to design directly in the z-domain. Digital-filter theory removes many of the restrictions imposed in analog-filter theory. However, the design of digital controllers directly in the z-domain is not as developed as digital-filter theory. Although some simple digital design techniques such as deadbeat control exist, most digital control books present state feedback methods, which use the state variable description of the system and are beyond the scope of this application note.

Disregarding state feedback control, a simple PID controller, deadbeat controller, or digital filter is described by the following equation:

$$G_c(z) = \frac{\gamma(0) + \gamma(1)z^{-1} + \ldots + \gamma(n)z^{-m}}{1 + \delta(1)z^{-1} + \ldots + \delta(n)z^{-n}} \quad m < n$$

Eqn. 5-2

■

SECTION 6

Implementation of Digital Controllers

"Since the architecture of the DSP56000/ DSP56001 allows efficient implementation of digital controllers and filters, the computational delay associated with digital signal processing is minimized."

The architecture of the DSP56000/DSP56001 allows discrete-time controllers and filters to be implemented efficiently and accurately. To reduce roundoff noise caused by finite-length registers, digital controllers and filters are often implemented in cascaded or parallel first- and second-order sections. Figure 6-1 shows a sixth-order controller implemented in cascade sections and parallel sections. The parallel structure is useful in multiprocessing schemes but can be more sensitive to coefficient quantization noise, which is discussed in **7.1 Coefficient Quantization**. The transfer function for a digital controller implemented in cascaded biquad sections is:

$$G_c(z) = g\prod_{i=1}^{n}\frac{b_i(0)+b_i(1)z^{-1}+b_i(2)z^{-2}}{1+a_i(1)z^{-1}+a_i(2)z^{-2}} \qquad \text{Eqn. 6-1}$$

Implementation of the overall system gain, g, is included in **SECTION 7.4 Implementation of the Gain, g**. Typically, these second-order sections are implemented using either direct form I or direct form II realizations as shown in Figure 6-2. For each section, direct form I requires more memory and more instruction cycles to implement than direct form II but may be less sensitive to roundoff noise.

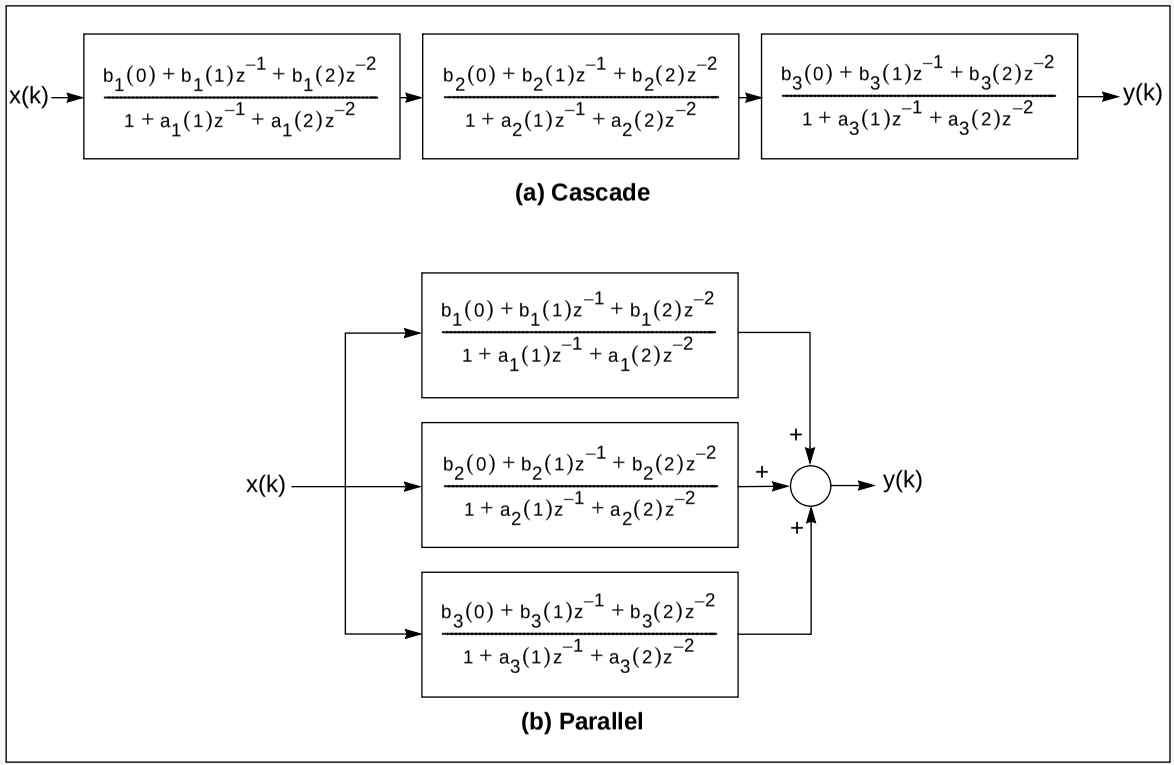
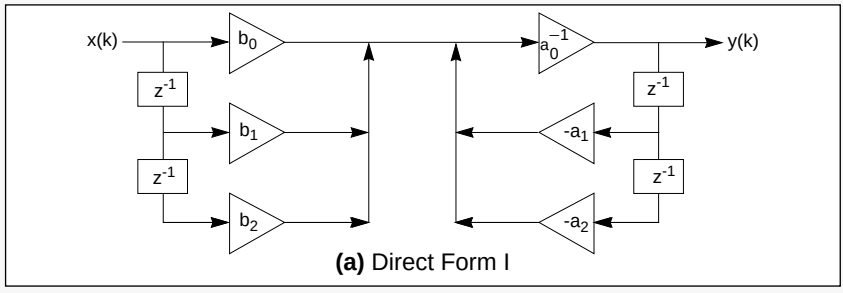
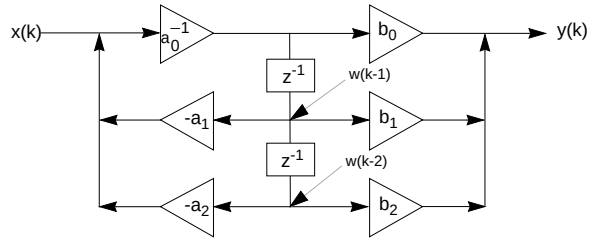


Figure 6-1 Cascade and Parallel Implementations of Digital Filters and Controllers



(a) Direct Form I



(b) Direct Form II

Figure 6-2 Biquad Sections

Figure 6-3 shows a sixth-order filter or controller implemented in cascaded direct form I biquad sections. In practice, the redundant z^{-1} terms in Figure 6-3 can be removed. The resulting structure is shown in Figure 6-4 and is composed of cascaded direct form II sections with an all-zero section at the beginning and an all-pole section at the end. Therefore, in the remainder of this application note, only direct form II implementations will be considered. Depending on the numerator and denominator coefficients in Eqn. 6-1, several different implementations of the second-order direct form II must be considered.

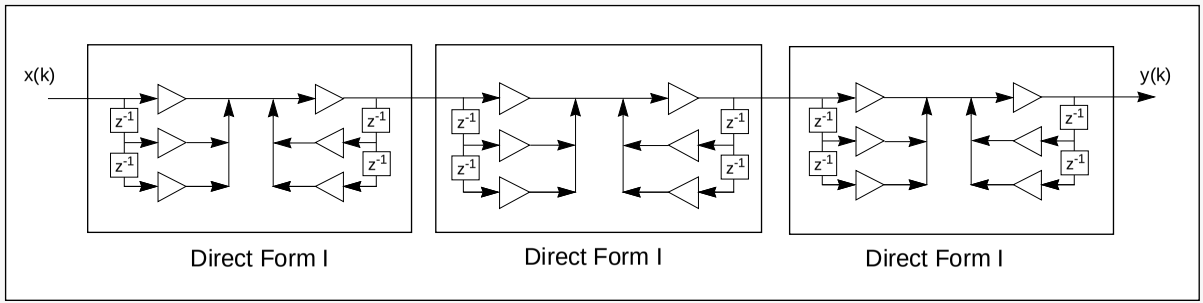


Figure 6-3 Sixth-Order Controller Implemented in Cascaded Direct Form I Biquad Sections

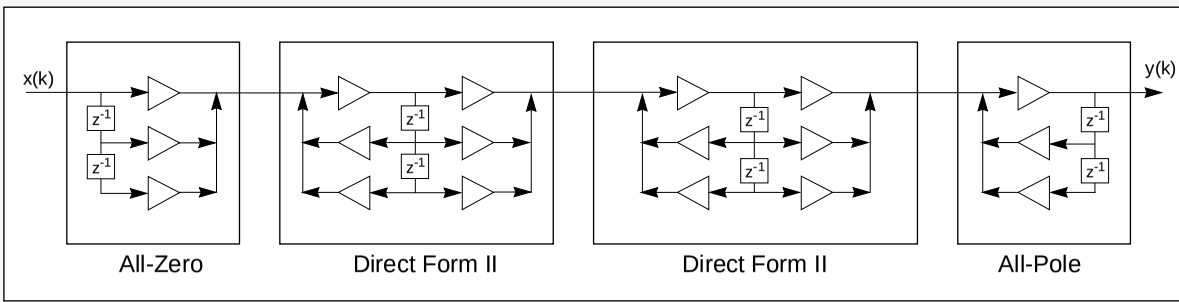


Figure 6-4 Figure 6-3 Sixth-Order Controller Transformed into Cascaded Direct Form II Sections with an All-Zero Section and an All-Pole Section

6.1 The Magnitudes of $a_i(1)$ and $a_i(2)$ are Less than Unity

The most general second-order section occurs when the magnitudes of $a_i(1)$ and $a_i(2)$ are less than one and the structure includes a direct path, $b_i(0)$. For a second-order structure to be implemented efficiently, all of the filter coefficients, $a_i(k)$ and $b_i(k)$, must be fractional since the DSP56000/DSP56001 uses a fractional data representation for all arithmetic logic unit (ALU) operations. If magnitudes of one or more of the numerator coefficients, $b_i(k)$, are greater than one, then all the numerator coefficients of the biquad section must be divided by the smallest power of two, which makes the magnitude of all of the numerator coefficients less than one. As a result, this factor of two should be incorporated into an input scaling factor, other biquad section's numerator coefficients, or the output scaling factor. To illustrate how a simple PID controller could be implemented in direct form II sections using the DSP56000/DSP56001, the following transfer function will be considered:

$$G_C(z) = \left(\frac{0.6 + 0.4z^{-1} - 0.25z^{-2}}{1 - 0.8z^{-1} + 0.6z^{-2}}\right)\left(\frac{0.8 + 0.6z^{-1} + 0.2z^{-2}}{1 + 0.4z^{-1} + 0.3z^{-2}}\right)\left(\frac{0.8 + 0.64z^{-1} + 0.9z^{-2}}{1 + 0.7z^{-1} + 0.5z^{-2}}\right)$$

Eqn. 6-2

Referring to Figure 6-2 (b), the intermediate values to be calculated and stored are designated as w(k-1) and w(k-2). Often, w(k-1) and w(k-2) are called the

internal nodes of the structure. The recursive set of equations describing this structure is:

$$w(k) = \frac{1}{a_0}(x(k) - a(1)*w(k-1) - a(2)*w(k-2)) \qquad \text{Eqn. 6-3}$$

$$y(k) = b(0)*w(k) + b(1)*w(k-1) + b(2)*w(k-2) \qquad \text{Eqn. 6-4}$$

$$w(k-2) = w(k-1) \qquad \text{Eqn. 6-5}$$

$$w(k-1) = w(k) \qquad \text{Eqn. 6-6}$$

In the assembly language in Figure 6-5, parallel A/D and D/A are assumed to be located in external peripheral space at locations Y:$FFFE and Y:$FFFF, respectively. See Reference 3 for explicit details of the architecture of the DSP56000/DSP56001. The remainder of the equate definition referenced in Figure 6-5 is reproduced in **APPENDIX A Listing of 'declare.dat'**. Figure 6-6 shows a memory map for the data and coefficients. The internal nodes and coefficients for each biquad section are stored in on-chip X and Y data RAM, respectively. This type memory map allows data to be continuously moved on both the X and Y data buses during each multiply to set up the data ALU registers for the following multiply.

```
       include 'declare.dat'

A_D         equ    $FFFE     ;location of A/D in Y memory
D_A         equ    $FFFF     ;location of D/A in Y memory
numsec      equ    3         ;number of cascaded biquad sections

;************************************************************************
; X memory locations
;************************************************************************
;
            org    x:$0
data        dc     0         ;cascade section 1 w(k-2)
            dc     0         ;cascade section 1 w(k-1)
            dc     0         ;cascade section 2 w(k-2)
            dc     0         ;cascade section 2 w(k-1)
            dc     0         ;cascade section 3 w(k-2)
            dc     0         ;cascade section 3 w(k-1)

;************************************************************************
;
; Y memory locations
;************************************************************************
;
            org    Y:$0
coef
            dc      .6       ;a(2) — cascade section 1
            dc    -.8        ;a(1) — cascade section 1
            dc    -.25       ;b(2) — cascade section 1
            dc      .4       ;b(1) — cascade section 1
            dc      .6       ;b(0) — cascade section 1
            dc      .3       ;a(2) — cascade section 2
            dc      .4       ;a(1) — cascade section 2
            dc      .2       ;b(2) — cascade section 2
            dc      .6       ;b(1) — cascade section 2
            dc      .4       ;b(0) — cascade section 2
            dc      .5       ;a(2) — cascade section 3
            dc      .7       ;a(1) — cascade section 3
            dc      .9       ;b(2) — cascade section 3
            dc      .64      ;b(1) — cascade section 3
            dc      .8       ;b(0) — cascade section 3

;************************************************************************
;
; Fast Interrupt Service Routines
;************************************************************************
;
            org    p:reset   ;Reset service routine
            jmp    main

            org    p:irqa    ;interrupt request a service routine
            movep  y:A_D,a   ;a=x(k)
            nop              ;unused second word of fast interrupt
```

Figure 6-5 DSP56000/DSP56001 Assembly Language Program that Implements a Sixth-Order PID Controller (sheet 1 of 2)

```
;*************************************************************************
;
; Initialization
;*************************************************************************
;
    org      p:main
    move     #data,r0                ;r0 points to states
    move     #5,m0                   ;r0 is modulo 6
    move     #coef,r4                ;r4 points to filter coefficients
    move     #14,m4                  ;r4 is modulo 15
    movep    #$007,x:ipr             ;irqa is negative edge triggered, pr 2
    movep    #2,x:bcr                ;2 wait states for A/D and D/A
    move     x:(r0)+,x0  y:(r4)+,y0  ;init data ALU registers
    andi     #$FC,mr                 ;enable all interrupts

;*************************************************************************
;
; PID Compensator Algorithm
;*************************************************************************
;
start
     wait                                                ;wait for input sample

     do      #numsec,enddo
     mac     -x0,y0,a    x:(r0)-,x1    y:(r4)+,y0        ;A=x(k)-a2*s2
     macr    -x1,y0,a    X1,x:(r0)+    y:(r4)+,y0        ;A=x(k)-a2*s2-a1*s1
     mpy     x0,y0,a     a,x:(r0)      y:(r4)+,y0        ;A=b2*b2
     mac     x1,y0,a     x:(r0)+,x0    y:(r4)+,y0        ;A=b2*b2+b1*s1
     macr    x0,y0,a     x:(r0)+,x0    y:(r4)+,y0        ;A=b2*b2+b1*s1+
                                                         ; b0(x(k)-a2*s2-a1*s1)
enddo
     movep   a,y,:D_                                     ;output result to D/A
     jmp     start                                       ;repeat
```

Figure 6-5 DSP56000/DSP56001 Assembly Language Program that Implements a Sixth-Order PID Controller (sheet 2 of 2)

6.2 Initialization

Upon hardware reset, the DSP56000/DSP56001 begins executing instructions at location 0 in program memory in all operating modes except normal expanded, mode 2. In mode 2, the program execution begins at location $E000 in program memory. After reset, the DSP56000/DSP56001 immediately jumps to the beginning of the main routine. The first eight instructions are responsible for initializing the digital controller. In this assembly language pro-

gram, r0 is initialized to point to the table containing the internal nodes for the sixth-order controller. This address register is also modified to be modulo 6 to make the table of internal nodes function as a circular buffer. Likewise, r4 is initialized to point to the beginning of the coefficient table and modified to be modulo 15. With modulo addressing, no instruction cycles are wasted reinitializing address pointers.

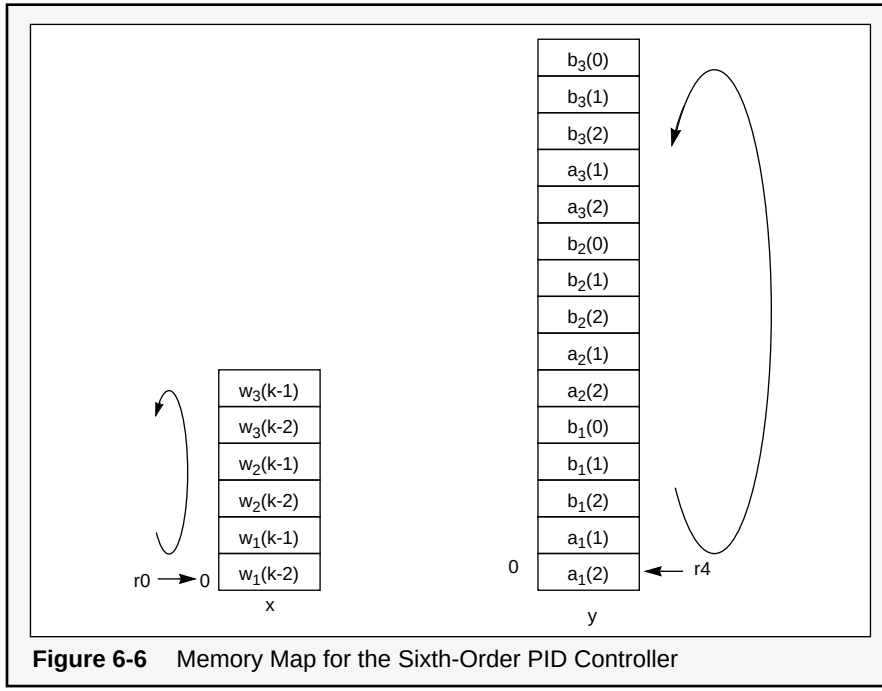


Figure 6-6 Memory Map for the Sixth-Order PID Controller

Next, the interrupt priority register is initialized so that external hardware interrupt A is edge triggered with a priority level of two. Also, the bus control register is programmed to provide two wait states for

the A/D and D/A. Since the filter program can be stored in internal program RAM and only uses internal data RAM, no external wait states are needed for P, X, or Y memory spaces. The next instruction fetches the first section's internal node, $w_1(k-2)$, and the first coefficient, $a_1(2)$. The purpose of this instruction is to initialize the data ALU registers and address pointers for the filtering operation. Finally, the lower two bits of the mode register are cleared to unmask all interrupts with priority levels greater than or equal to zero.

6.3 PID Compensation Algorithm

In this program, the WAIT instruction is used to synchronize the DSP to the A/D. It is assumed that an end-of-conversion pulse from a parallel A/D is used to trigger the DSP's external interrupt, $\overline{\text{IRQA}}$. With the WAIT instruction, the digitized data is input from the A/D using the MOVEP instruction, which is the first instruction of the $\overline{\text{IRQB}}$ fast interrupt. The advantage of using the WAIT instruction is that the DSP56000/DSP56001 goes into a low power-consumption mode until the next sample is ready to be processed. Also, the interrupt allows synchronization between the A/D and the DSP. Note that when using the WAIT instruction, the first instruction of the fast interrupt will require at least eight instruction cycles to be executed. Another possible method of importing and exporting data is to use the SSI for communicating with serial A/Ds and D/As such as the DSP56ADC16 (see Reference 2).

A hardware DO loop is used to implement the cascaded second-order sections of the controller or filter. The DO loop, which allows the programmer to keep the code compact, only requires three instruction cycles to set up and no additional cycles while the loop is executing. Due to the parallel architecture of the DSP56000/DSP56001, each biquad section of this type can be implemented in only five instruction cycles. Before storing intermediate nodes to memory, the value is rounded to 24 bits. This convergent rounding insures that no bias is introduced into the roundoff error in contrast to truncating the result.

6.4 The Magnitude of $a_i(1)$ is Greater than Unity

To insure controller or filter stability, all complex poles of the structure must lie inside of the unit circle in the z-plane. For a second-order section with the denominator defined to be:

$$D(z) = 1 + a_1 z^{-1} + a_2 z^{-2} = \left(1 - p_1 z^{-1}\right)\left(1 - p_2 z^{-1}\right)$$

Eqn. 6-7

where: a_1 represents the negative sum of the poles, $-(p_1+p_2)$

a_2 equals the product of the poles, $p_1\, p_2$

If the poles are complex conjugates, $p_2=p_1^*$. Since the magnitude of both poles must be less than one,

$$|a_2| = |p_1 p_2| < 1 \qquad \text{Eqn. 6-8}$$

and

$$|p_1, p_2| = \left| \frac{-a_1 \pm \sqrt{a_1^2 - 4a_2}}{2} \right| < 1 \qquad \text{Eqn. 6-9}$$

Solving Eqn. 6-9 yields the set of conditions:

$$|a_1| < 1 + a_2 \qquad \text{Eqn. 6-10}$$

Eqn. 6-8 and Eqn. 6-10 define the stability triangle shown in Figure 6-7. For a second-order system to be stable, the values of a_1 and a_2 must lie within the stability triangle. As Figure 6-7 indicates, the magnitude of a_1 can be greater than one and still yield a stable structure. This case presents a minor problem regarding implementation since the fractional arithmetic of the DSP56000/DSP56001 cannot efficiently support nonfractional coefficients. To implement a stable network of this type, a two can be factored out of the denominator.

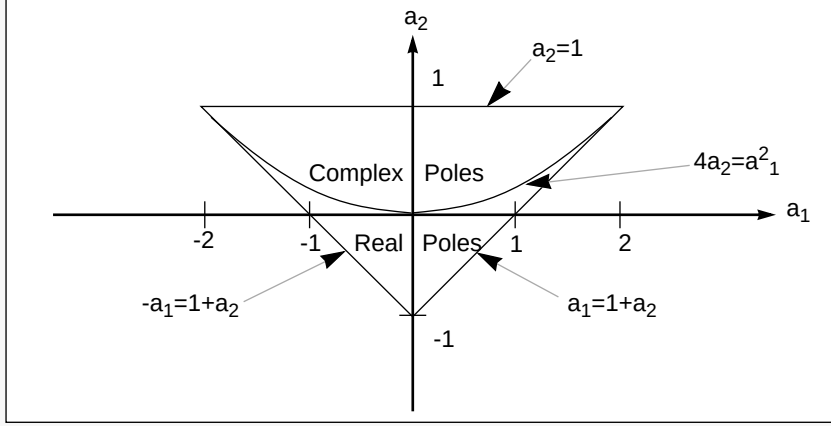


Figure 6-7 Stability Triangle that Illustrates the Stability Requirements for a Second-Order System

The modified signal-flow graph for this type of structure is shown in Figure 6-9. The kernel for implementing this type of biquad section requires six instruction cycles to execute:

```
mac     -x0,y0,a    x:(r0)-,x1    y:(r4)+,y0     ;A=x(k)-a2*s2
macr    -x1,y0,a    x1,x:(r0)+    y:(r4)+,y0     ;A=x(k)-a2*s2-a1*s1
asl     a                                        ;A=2[x(k)-a2*s2-a1*s1]
mpy     x0,y0,a     a,x:(r0)      y:(r4)+,y0     ;A=b2*s2
mac     x1,y0,a     x:(r0)+,x0    y:(r4)+,y0     ;A=b2*s2+b1*s1
macr    x0,y0,a     x:(r0)+,x0    y:(r4)+,y0     ;A=b2*s2+b1*s1+
                                                 ;   2*b0(x(k)-a2*s2-a1*s1)
```

Figure 6-8 Code Kernel that Implements the Modified Direct Form II Biquad Section Shown in Figure 6-9

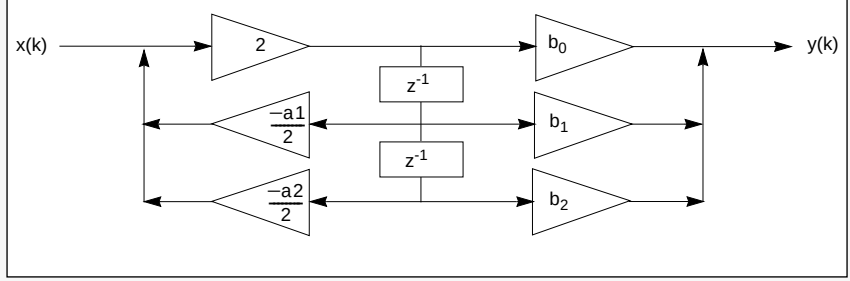


Figure 6-9 Modified Direct Form II Biquad Section

6.5 All $b_i(0)$ Coefficients are 1; $b_i(1)$, $b_i(2)$,$a_i(1)$, and $a_i(2)$ Coefficients are Fractional

The most efficient biquad section that can be implemented on the DSP56000/DSP56001 occurs when $b_i(0)$ is equal to one and $a_i(1)$, $a_i(2)$, $b_i(1)$, and $b_i(2)$ are fractional. The DSP56000/DSP56001 kernel for this type structure requires only four instruction cycles to execute:

```
mac     -x0,y0,a    x:(r0)-,x1    y:(r4)+,y0        ;A=x(k)-a2*s2
macr    -x1,y0,a    x1,x:(r0)+    y:(r4)+,y0        ;A=x(k)-a2*s2-a1*-s1
mac     x0,y0,a     a,x:(r0)+     y:(r4)+,y0        ;A=x(k)-a2*s2-a1*-s1+b2*s2
macr    x1,y0,a     x:(r0)+,x0    y:(r4)+,y0        ;A=x(k)-a2*s2-a1*s1+
                                                    ;        b2*s2+b1*s1
```

Figure 6-10 Code Kernel that Implements the Most Efficient Biquad Section Possible on a DSP56000/DSP56001

6.6 Computational Delay

The time required by the microprocessor to implement a digital controller or filter is called the computational delay. Typically, a digital controller is designed under the assumption that input and output sampling occur simultaneously without regard to the computational delay. The delay required for processing adds negative phase and may reduce the stability of the system. If the computational delay is significant, the controller must be designed to compensate for the additional negative phase. Since the architecture of the DSP56000/DSP56001 allows efficient implementation of digital controllers and filters, the computational delay associated with digital signal processing is minimized. ■

SECTION 7

Finite-Length Register Effects

"Therefore, the 24-bit DSP56000/ DSP56001 may be the only DSP capable of implementing highly precise algorithms."

Digital filters and controllers are typically designed and simulated in high-level computer languages using double-precision floating-point arithmetic. However, once the double-precision coefficients have been derived, a fixed-point simulation of the structure with finite-length registers must also be performed to determine if the filter still meets the design specifications. A fixed-point design includes three sources of error that are negligible in the double-precision floating-point design: coefficient quantization, overflow, and roundoff noise.

7.1 Coefficient Quantization

The first error source to be studied is coefficient quantization. Coefficient quantization occurs because, due to the limited word length of the data in a fixed-point processor, the 48-bit double-precision filter coefficients cannot be accurately represented. A summary of Jackson's explanation (see Reference 5) of the problems associated with coefficient quantization follows.

Factoring the denominator of the second-order bi-quad section:

$$1 + a_1 z^{-1} + a_2 z^{-2} = (1 + -pz^{-1})(1 - p^* z^{-1}) \quad \text{Eqn. 7-1}$$

yields the pole, p, and its conjugate, p*. From this equation, it is apparent that:

$$a_1 = -2\mathrm{Re}(p) \quad \text{Eqn. 7-2}$$

and

$$a_2 = |p|^2 \quad \text{Eqn. 7-3}$$

Quantization of a_1 corresponds to the quantization of the real part of the two poles. A 3-bit quantization of the real portion of the poles is illustrated by the vertical lines in Figure 7-1. On the other hand, a_2 represents the equation of a circle centered at the origin with radius p. The quantization of a_2 is represented by the various circles in Figure 7-1. Possible fixed-point pole locations for 3-bit word length are given by the intersection of the vertical lines and the concentric circles. Due to the spare spacing of possible pole locations near the points $z = \pm 1$, large quantization errors can be introduced in both narrow-band low-pass filters and phase-lag controllers requiring poles in these areas. The result of this quantization error is that it may not be possible to obtain the required response from these types of numerically sensitive structures. In fact, quantized poles may even lie outside of the unit circle, making the controller or filter unstable. Since the DSP56000/DSP56001 is the only 24-bit fixed-point

DSP on the market, it will be much less sensitive to quantization effects than 16-bit devices.

A study of the quantization effects of the biquad-section zeros indicates an important point concerning the robustness of parallel implementations. In the parallel implementation shown in Figure 6-1, the location of all zeros of the overall transfer function are dependent upon all individual numerator and denominator coefficients, a_i, and b_i.This dependency can be easily seen by considering a simple fourth-order parallel system with the following transfer function:

$$G_c(z) = \frac{b_1(0) + b_1(1)z^{-1} + b_1(2)z^{-2}}{1 + a_1(1)z^{-1} + a_1(2)z^{-2}} + \frac{b_2(0) + b_2(1)z^{-1} + b_2(2)z^{-2}}{1 + a_2(1)z^{-1} + a_2(2)z^{-2}}$$

Eqn. 7-4

Finding a common denominator yields:

$$G_c(z) = \frac{b'(0) + b'(1)z^{-1} + b'(2)z^{-2} + b'(3)z^{-3} + b'(4)z^{-4}}{1 + a'(1)z^{-1} + a'(2)z^{-2} + a'(3)z^{-3} + a'(4)z^{-4}}$$

Eqn. 7-5

where:

$$b'(0) = b_1(0) + b_2(0)$$

$$b'(1) = b_1(1) + b_1(0)a_2(1) + b_2(1) + b_2(0)a_1(1)$$

$$b'(2) = b_1(2) + b_1(1)a_2(1) + b_1(0)a_2(2) + b_2(2) + b_2(1)a_1(1) + b_2(0)a_1(2)$$

$$b'(3) = b_1(2)a_2(1) + b_1(1)a_2(2) + b_2(2)a_1(1) + b_2(1)a_1(2)$$

$$b'(4) = b_1(2)a_2(2) + b_2(2)a_1(2)$$

$$a'(1) = a_1(1) + a_2(1)$$

$$a'(2) = a_1(2) + a_1(1)a_2(1)a_2(2)$$

$$a'(3) = a_1(2)a_2(1) + a_1(1)a_2(2)$$

$$a'(4) = a_1(2)a_2(2)$$

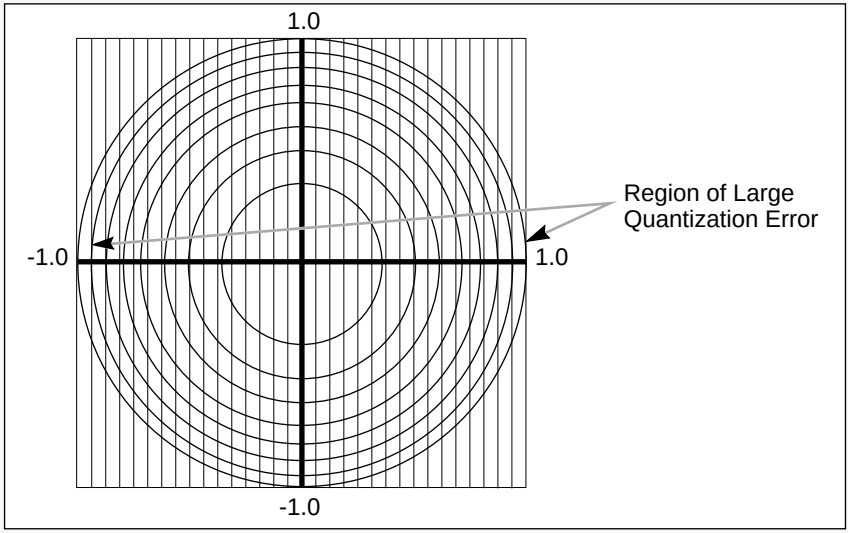


Figure 7-1 Location of the Quantized Poles for a 3-Bit Word Length

This dependency makes the zeros of the parallel implementation highly sensitive to coefficient quantization. Since the stop-band attenuation is increased by moving the zeros closer to the unit circle, filters and controllers with strict specifications should not be implemented using the parallel form. On the other hand, the zeros of the cascaded biquad section are only dependent upon b_1 and b_2, making this form much less sensitive to coefficient quantization.

7.2 Overflow

In **SECTION 6 Implementation of Digital Controllers and Filters**, the equations for the stability triangle were derived, which guarantee stable biquad structures. When implementing these structures in a fixed-point processor such as the DSP56000/DSP56001, even these stable structures may become unstable due to overflow. In classic microprocessor architectures, overflow occurs when the magnitude of an internal node becomes greater than unity and cannot be accurately stored in memory. For example, in Eqn. 6-3, $w(k)=1/a_0(x(k)-a_1\ w(k-1)-a2w(k-2))$. If $|w(k)| > 1$, then the subsequent outputs of the filter will be wrong once w(k) is moved to w(k-1), since |w(k-1)| must be less than unity. Roberts and Mullis present an excellent analysis of overflow in second-order structures (see Reference 11). From an architectural standpoint, overflow could be handled in two ways. First, if overflow is simply ignored and wrap-

around is allowed to occur, then a large positive (negative) value is moved to memory as a large negative (positive) value. This condition, which causes the output to oscillate nonlinearly between large positive and negative numbers, is referred to as overflow oscillation or limit cycles. Roberts and Mullis note that, even for systems with zero input, the output will never converge to zero for various initial values of the internal nodes. Hence, once a filter begins to exhibit overflow oscillations, it may be impossible for the structure to recover. This type of instability is disastrous in digital filters and controllers and must be avoided.

The second way of handling overflow is to use saturation arithmetic. This method involves limiting a number, which is to be stored to memory or output to a D/A, to the maximum positive or negative value represented by the memory or D/A. Even though saturation arithmetic is more costly than the first method, the designers of the DSP56000/DSP56001 chose to implement it to eliminate the large nonlinear errors caused by overflow oscillations. To provide a temporary buffer to protect against overflow, the DSP56000/DSP56001 has 8-bit extension registers in both accumulators to handle intermediate sums within the range $-256 \leq sum < 256$. If an accumulator with a sum between $1 \leq sum < 256$ is moved to memory, the output is limited to the maximum positive value of $1-2^{-23}$. Likewise, if an accumulator has a value between $-256 \leq sum < -1$, the output is limited to the maximum negative value of -1. The “sticky” status bit L indicates whether limiting

has occurred at any time during the execution of an algorithm. With saturation arithmetic, the effects of an overflow are much less severe than if the overflow portion of the sum were truncated.

Two methods of avoiding overflow are available: the input can be scaled, or the structure can be redesigned. Oppenheim and Schafer (see Reference 9) derive a bound for scaling the input to the structure. The value of the j^{th} internal node at time k can be described by:

$$y_j(k) = \sum_{l=-\infty}^{\infty} h_j(l)x(k-1) \qquad \text{Eqn. 7-6}$$

where: x is the input to the filter or controller
h_j is the unit pulse response from the input to the j^{th} internal node
y_j is the value output of the j^{th} node

Considering only the magnitudes of the values in Eqn. 7-6,

$$|y_j(k)| \le |x(k)| \sum_{l=-\infty}^{\infty} |h_j(l)| \qquad \text{Eqn. 7-7}$$

Overflow will occur if the magnitude of $y_j(k)$ exceeds unity. Therefore, to obtain $|y_j(k)| < 1$ and avoid overflow, x_{max} needs to satisfy:

$$x_{max} < \frac{1}{\sum_{l=-\infty}^{\infty} |h_j(l)|} \qquad \text{Eqn. 7-8}$$

for all of the internal nodes. The bound derived in Eqn. 7-8 is very conservative since it assumes that the input to the system is constant at the maximum possible value. Since this type of input rarely occurs, other assumptions can be made about the input, including classifying it as sinusoidal, finite energy, wide sense stationary, or white noise to make the scaling bound less severe. For additional information concerning these bounds, see References 5, 9, 10, and 11.

Scaling of the input data simply describes where the input data is placed in the data word of the DSP. For serial A/Ds, scaling the input equates to shifting the data left (scaling up) or right (scaling down); whereas, for parallel A/Ds, scaling is dependent upon how the data lines of the A/D are connected to the data lines of the DSP. Historically, due to the overwhelming system cost associated with high-resolution A/Ds, system designers where restricted to using low-precision A/Ds for many high-volume products. However, the recent release of the new sigma-delta A/Ds, such as the DSP56ADC16, makes 16-bit input data quite inexpensive. For a DSP with 16-bit word length, the scaling of the data corresponds to the loss of the lower significant bits depending on the scale factor. On the other hand, the 16 bits of data from an A/D could be shifted down to the lower 16 bits of the 24-bit data word of the DSP56000/DSP56001. This shift would cause no loss in input-data accuracy and would provide room for eight bits of algorithm growth without the possibility of overflow or limiting.

The second method of eliminating overflow or limiting is by redesigning the filter or controller. The transfer function shown in Eqn. 5-2 describes only the relationship between the input and the output of the network. An infinite number of different implementations will yield the same transfer function. Roberts and Mullis (see Reference 11) show how to represent these different structures with state variable descriptions (SVDs). An SVD not only gives the I/O relationship, but also describes the exact internal structure of the filter or controller. Roberts and Mullis give a method for translating a structure having the potential to overflow or limit to a different structure with the same transfer function which will not overflow or limit. The method given for translating one SVD to another SVD is based on a series of orthogonal matrix transformations. For additional information on the SVD of filters and controllers, see Reference 5.

7.3 Roundoff Noise

The third type of noise inherent in the implementation of digital filters and controllers is roundoff noise. When multiplying two n-bit numbers together, a 2n-bit result is produced. If this 2n-bit number is then multiplied by another n-bit number, the result is 3n bits in length. If this pattern continues, infinite-length registers and multipliers are needed. Since this solution is not practical, the DSP56000/DSP56001 convergently rounds all 48-bit multiplication results to 24 bits before storing to memory.

This final rounding creates an added source of noise within the biquad section coincident with every move to memory. Oppenheim and Schafer (see Reference 9) derive the following expression for the roundoff noise power at the output:

$$\sigma_f^2 = \frac{2^{-2B}}{12}\frac{1}{2\pi j}\oint_{|z|=1} H(z)H(z^{-1})z^{-1}dz = \frac{2^{-2B}}{12}\sum_{n=-\infty}^{\infty}|h[n]|^2$$

Eqn. 7-9

where: $\oint$ is the Cauchy integral

H(z) is the transfer function

h(n) is the impulse response function from the noise source to the output

B represents the number of bits in the word length

The location of the roundoff noise source for direct form I is shown in Figure 7-2(a). The noise is added when the 48-bit result is rounded to 24 bits for storage in the internal nodes on the right side of the signal-flow graph. The roundoff noise for direct form I only depends upon the location of the poles of the second-order system. Therefore, a closed-form solution of the roundoff noise power can be derived for direct form I. In Reference 9, the roundoff noise power for a direct form I biquad section is shown to be:

$$\sigma_f^2 = \frac{2^{-2B}}{12}\left(\frac{1+r^2}{1-r^2}\right)\frac{1}{r^4 - 2r^2\cos 2\theta + 1}$$

Eqn. 7-10

where: r and θ correspond to the radius and angle of the complex pole pair, respectively.

Figure 7-3 shows a three-dimensional graph of Eqn. 7-10 in the z-plane. The figure is constructed to show the relative magnitudes of the singularities. In reality, as the poles approach the unit circle, the output noise power due to roundoff approaches infinity. The figure shows that the output noise power due to roundoff approaches infinity much faster at the z=1 points because of double singularity. This makes the direct form I structures very sensitive to roundoff noise for phase-lag controllers and narrow-band low-pass and high-pass filters having poles near the z=1 point.

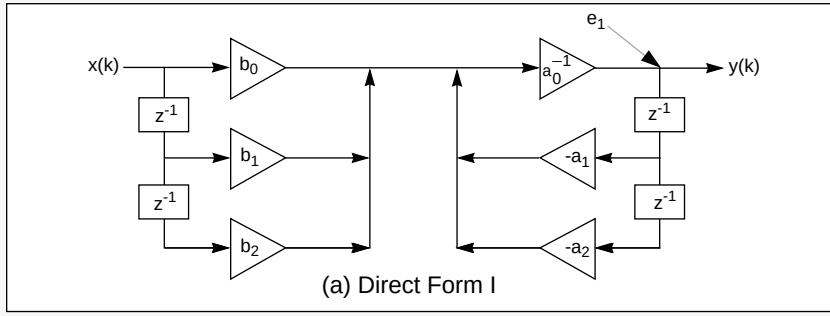


Figure 7-2 Roundoff Noise Sources for Direct Form Implementations

Direct form II has two sources of roundoff noise, which are shown in Figure 7-2(b). Since the first noise source is fed back through the poles as well as fed forward through the zeros, the roundoff noise depends upon the location of both the zeros and the poles. Therefore, no direct comparison can be made between direct forms I and II for the general case. For both direct forms I and II, the word length of the processor is important in determining the structure's output noise power due to roundoff. In fact, Eqn. 7-9 indicates that the roundoff noise power for the 24-bit DSP56000/DSP56001 is 65,536 times less than that for a 16-bit DSP.

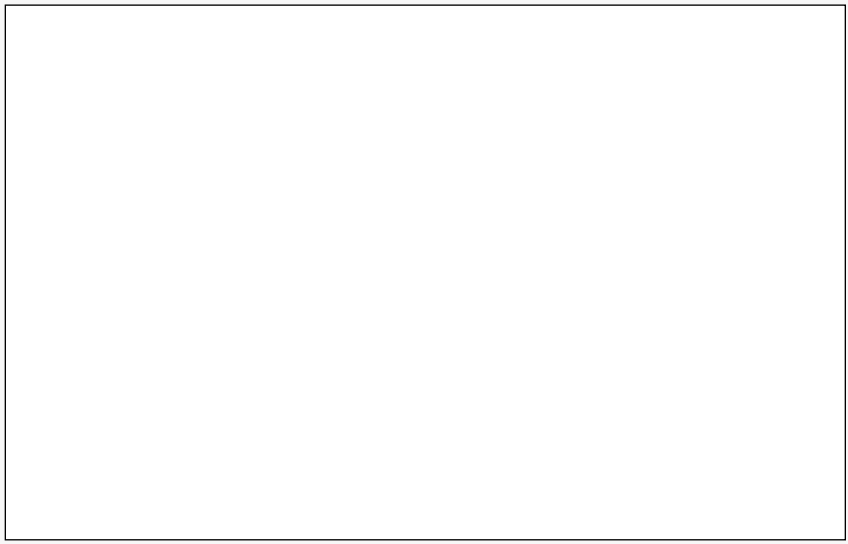


Figure 7-3 Output Noise Power for a Direct Form I Biquad Section Caused by Internal Roundoff Noise

Therefore, the 24-bit DSP56000/DSP56001 may be the only DSP capable of implementing highly precise algorithms.

As previously mentioned, the method presented by Roberts and Mullis (see Reference 11) to eliminate overflow can also be used to reduce roundoff noise. By using a set of orthogonal transformations, a new second-order structure can be found, which possesses the minimum roundoff noise of all possible SVDs for a given transfer function.

7.4 Implementation of the Gain, g

After an analog controller or filter is transformed into the digital domain, Eqn. 6-1 contains a constant gain factor, g. Oppenheim and Schafer (see Reference 9) consider the effect of implementing this gain factor in a cascaded biquad network. The gain factor can be included in the input scaling stage, the numerator coefficients of one or more of the biquad sections, or the output scaling stage. The implementation of this gain factor is dependent upon how it affects the roundoff noise versus the possibility of overflow. Rules are also given for the ordering of the zero and pole pairs based on the effect of roundoff noise in a cascaded structure. ■

SECTION 8

System Considerations

"An alternative to using the HI, SSI, and SCI is to configure the ports as general-purpose I/O pins."

An illustration of the DSP56000/DSP56001 functional signal groups is shown in Figure 8-1. These different groups allow the DSP56000/DSP56001 to function well in a digital control system. Three on-chip peripherals are provided: an 8-bit parallel host MPU/DMA interface, an SCI, and an SSI. Also, depending on which of the on-chip peripherals are used, up to 24 general-purpose I/O pins are available.

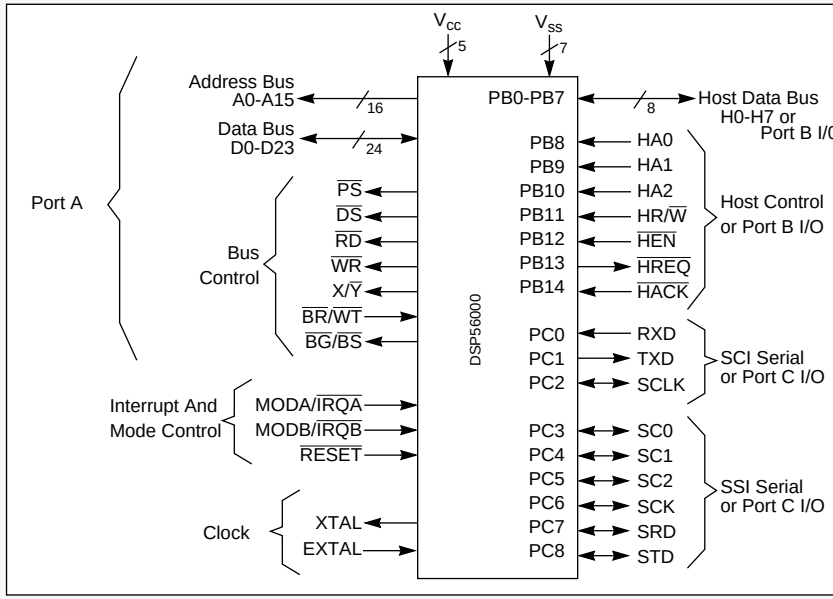


Figure 8-1 DSP56000/DSP56001 Functional Signal Groups

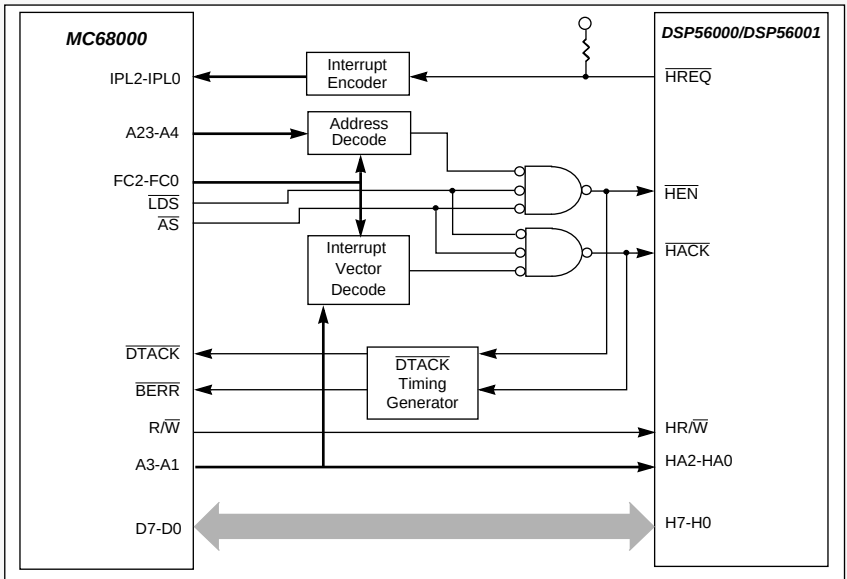


Figure 8-2 Connecting the DSP56000/DSP56001 to the MC68000 via the Host Port

8.1 Host Interface Port

The HI is a dedicated 8-bit parallel port used to connect the DSP56000/DSP56001 to a host microprocessor or DMA controller. Figure 8-2 and Figure 8-3 show examples of using the HI to connect the DSP56000/DSP56001 to an MC68000 and MC68HC11, respectively. The host processor might be responsible for high-level tasks such as monitoring front-panel switches and initializing various control sequences in the DSP. One nice feature of

the HI is the host command option. If desired, the host processor can execute any one of 32 interrupt vectors in the DSP56000/DSP56001. The possible functions that the host can execute are listed in Figure 8-4. Loading a value into the command vector register causes the DSP to execute the corresponding fast interrupt.

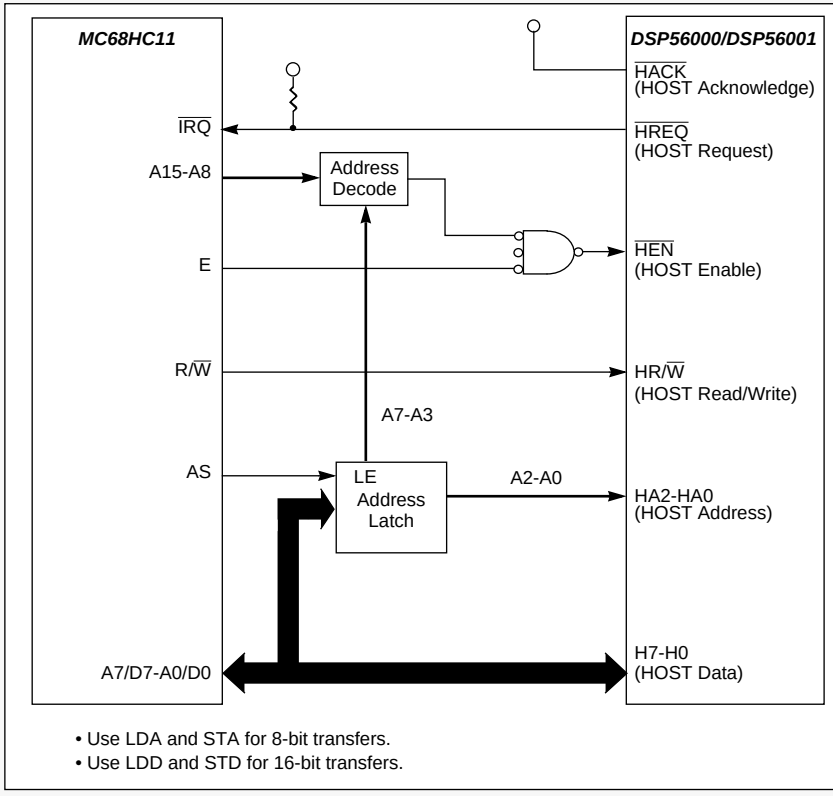
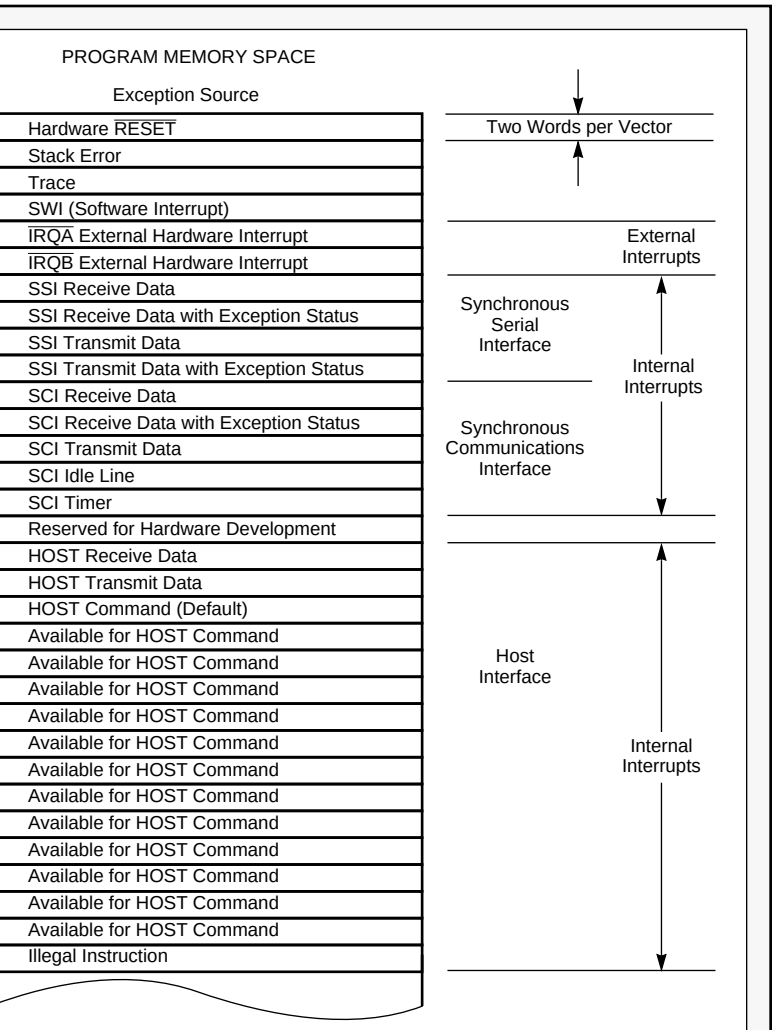


Figure 8-3 Connecting the DSP56000/DSP56001 to the MC68HC11 via the Host Port



: 1. The vector table contains instructions, not addresses.
2. Each vector location can contain:
Two one-word instructions
One two-word instruction

3-4 DSP56000/DSP56001 Fast Interrupt Vectors

Twelve host command vectors are available for general-purpose programming. If any of the remaining predefined interrupts are not used, such as the software interrupt or the SCI idle interrupt, they can also be used as general-purpose command vectors. These host command vectors give much flexibility to the system designer. The DSP56000/DSP56001 program in Figure 8-5 provides one example of the power of the HI.

In Figure 8-5, the host command vectors are used to alter PID coefficients in real time without recompiling the routine or halting the DSP56000/DSP56001. These parameters could be input to the host from a front panel or some software menu format. First, the host processor writes the new PID coefficient to TXH, TXM, and TXL registers on the host processor side of the HI. Next, the host must write the appropriate host command number to the DSP command vector register (CVR). For example, to alter b0, the value $92 must be written to the CVR, which sets the HC bit and causes the DSP56000/DSP56001 to initiate a host command. Once the DSP recognizes that a host command vector is pending, it jumps to location $24 and executes the corresponding fast interrupt.

8.2 SCI Port

The SCI provides a port for asynchronous serial communication to other DSPs, microprocessors, etc., either directly or via modems. It includes facilities for

communicating by using standard asynchronous bit rates and protocols as well as high-speed synchronous data transmission. The asynchronous port includes a multidrop mode for master/slave operation with wakeup on idle line and wakeup on address capability. In some control applications, the SCI port provides an inexpensive method for communicating with the DSP56000/DSP56001 via a terminal and a simple monitor program.

If the baud rate generator in the SCI port is not used for asynchronous communication timing, it can be configured as a general-purpose timer. Figure 8-6 shows how to configure the SCI programmable timer using the SCI clock control register. The system clock is initially divided by 2. Then, depending upon the values stored in the 12 CD bits, the clock is divided by a number from 1 to 4096. A prescale bit allows the programmer to further divide the clock by a factor of 8.

Finally, the clock is divided by a factor of 2 and a factor of 16. The resolution of SCI timer, Δt, is $32t_i \leq \Delta t \leq 1{,}048{,}576t_i$, where t_i is the instruction cycle time of the DSP. For the 27-MHz DSP56000/DSP56001, the resolution of the timer is $2.4\mu s \leq \Delta t \leq 77.6$ ms. In future revisions of the DSP56000/DSP56001, the final divide-by-16 block in Figure 8-6 will be optional, further reducing the resolution of the timer to the minimum interrupt rate of 6t. For this future revision, the minimum resolution of the timer will be 444 ns at 27 MHz.

```
       include 'declare.dat'

input  equ  $fffe           ;address of A/D
output equ  $ffff           ;address of D/A

;*************************************************************************
;X Memory Declaration
;*************************************************************************
       org  x:0
b_0    ds   1               ;b(0) coefficient
b_1    ds   1               ;b(1) coefficient
b_2    ds   1               ;b(2) coefficient
b_1    ds   1               ;a(1) coefficient
b_2    ds   1               ;a(2) coefficient

;*************************************************************************
;Fast Interrupt Definitions
;*************************************************************************

       org         p:reset                  ;loc of RESET service routine
       jump        main                     ;begin at main

       org         p:hc1                    ;location of HC 1 service routine
       movep       x:hrx,x:b_0              ;update b0 coefficient
       nop                                  ;second word of fast interrupt

       org         p:hc2                    ;location of HC 2 service routine
       movep       x:hrx,x:b_1              ;update b1 coefficient
       nop                                  ;second word of fast interrupt

       org         p:hc3                    ;location of HC 3 service routine
       movep       x:hrx,x:b_2              ;update b2 coefficient
       nop                                  ;second word of fast interrupt

       org         p:hc4                    ;location of HC 4 service routine
       movep       x:hrx,x:a_1              ;update a1 coefficient
       nop                                  ;second word of fast interrupt

       org         p:hc5                    ;location of HC 5 service routine
       movep       x:hrx,x:a_2              ;update a2 coefficient
       nop                                  ;second word of fast interrupt

;*************************************************************************
;Main Routine
;*************************************************************************

       org         p:main                   ;location of main routine
       movep       #$0A00,x:ipr             ;enable host interface interrupts
       movep       #0,x:bcr                 ;no wait states
       movep       #$04,x:hcr               ;enable host command interrupt
       movep       #1,x:pbc                 ;turn on host interface
       andi        #$FC,mr                  ;enable all interrupts
start
       movep       y:input,x0               ;get input sample from A/D
       jsr         filter                   ;filter the sample
       movep       a,y:output               ;send the output to D/A
```

Figure 8-5 Program Language that Alters PID Coefficients in Real Time without Recompiling the Routine or Halting the DSP56000/DSP56001

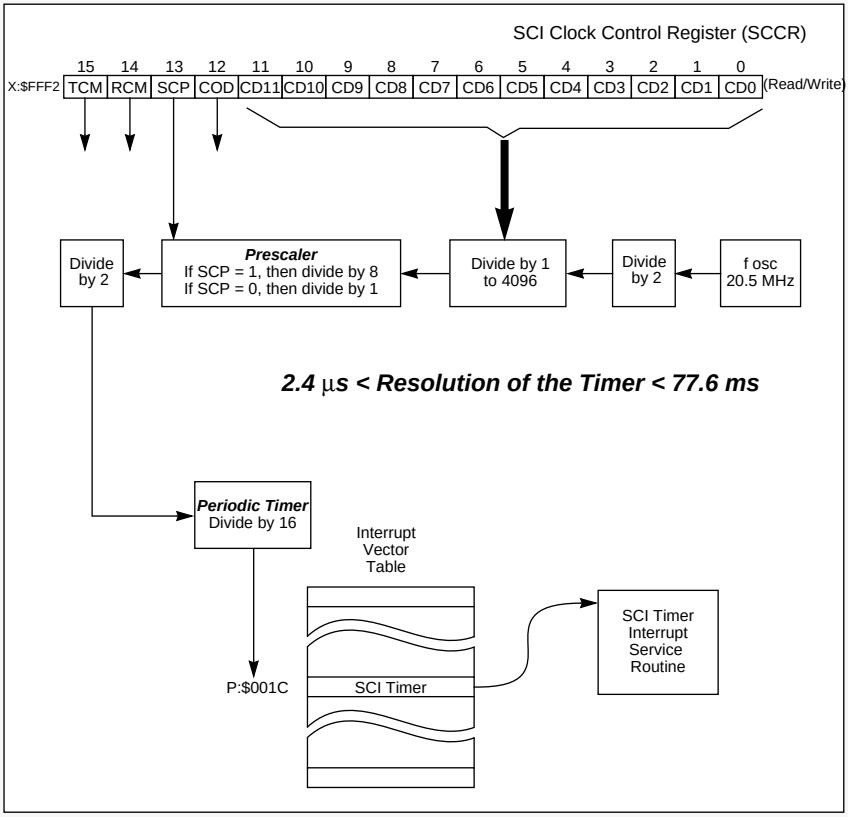


Figure 8-6 Functional Diagram of SCI Programmable Timer

8.3 SSI Port

The SSI can be used to receive and transmit data from serial A/Ds, D/As, and CODECs with little or no glue logic. The SSI is a full-duplex six-pin port that includes a serial transmit pin, a serial receive pin, a

serial clock pin, and three frame sync/output flag pins. The port is full duplex, and the programmable frame sync can be used to gate the transmitted/received data. The SSI also allows the DSP56000/DSP56001 to communicate serially with multiple processors in a network. The network mode allows up to 32 different time slots for communication with other processors or I/O devices.

8.4 General-Purpose I/O Pins

An alternative to using the HI, SSI, and SCI is to configure the ports as general-purpose I/O pins. A port control register is associated with both ports B and C, allowing the port pins to be selected as either general-purpose I/O pins or dedicated peripheral pins. All 15 pins of port B must function as either I/O pins or as the HI. However, all individual pins of port C can be configured separately. For instance, if the serial transmitter of the SSI is not used, PC8 can be configured as an I/O pin while PC3-PC7 perform their assigned SSI functions. A port pin selected as a general-purpose I/O pin is accessed through the corresponding port data register. Data written to the port data register is latched.

One possible use of the 15 port B I/O pins would be in disk drives. Figure 8-7 indicates how the DSP56000/DSP56001 might be used in a hard disk

drive system. Data is transmitted from the host computer via an SCSI bus or some other standard format. A hard disk controller, such as the MC68HC99, decodes the data and performs the appropriate error detection and correction. The data is then transmitted into the disk drive by some standard serial interface, such as ESDI or ST-506.

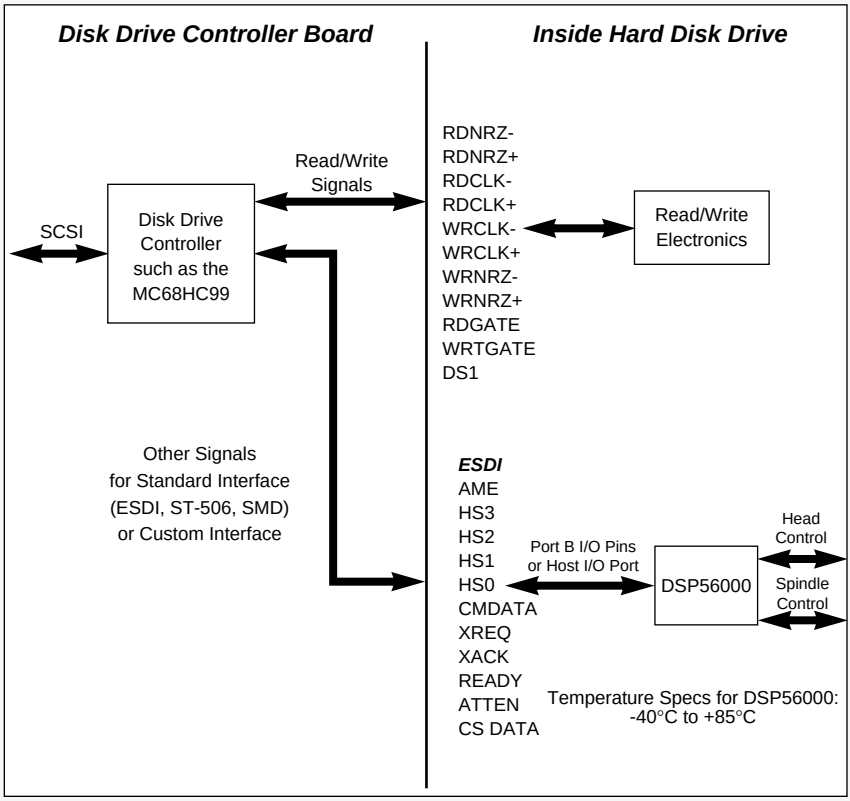


Figure 8-7 Connecting the DSP56000/DSP56001 to the MC68HC99 Disk Drive Controller via the Port B I/O Lines

Analog electronics handle the actual reading and writing of data to the disk; whereas, the DSP56000/DSP56001 decodes the high-level addressing information to correctly position the head assembly at the desired track location.

In addition to controlling the head, the DSP56000/DSP56001 would also be responsible for regulating the speed of the spindle. Unused I/O pins would be able to provide fault detection signals to the head and spindle motors, including stuck rotor shutdown. Since the temperature specifications for the DSP56000/DSP56001 are -40°C to +85°C, the chip should not have any problems operating in the sealed hard disk environment.

8.5 External Interrupts

The DSP56000/DSP56001 provides two external hardware interrupts, which can be programmed as either negative edge triggered or level sensitive. One possible use of the interrupts would be to provide velocity feedback. In many cases, velocity information is used in motor control. For instance, an optical sensor could be used to recognize one or more equally spaced notches in the spindle of a disk drive. The rotational velocity of the spindle is proportional to the amount of time between consecutive notches.

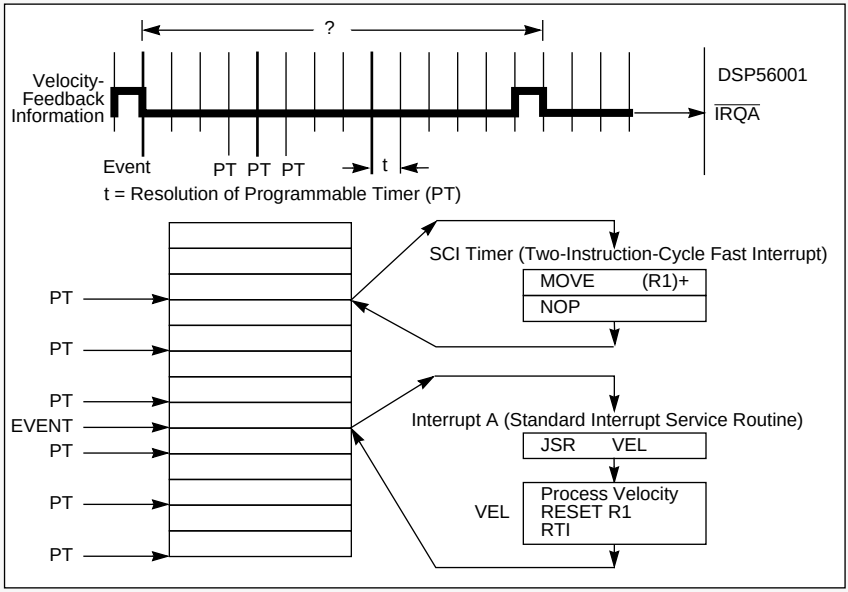


Figure 8-8 Using External Interrupts to Provide Velocity-Feedback Information

Figure 8-8 shows how the external interrupts, in conjunction with the SCI timer, can be used to provide velocity feedback. In Figure 8-8, velocity-feedback information from the optical sensor is input directly into $\overline{\text{IRQA}}$. It is desired to determine the amount of time between the two consecutive pulses. If the SCI timer is not being used for data communications, it is free for indicating the time between notches. An address register (R1) is used as the velocity counter. Once the SCI timer times out, the velocity counter is incremented in the SCI timer fast interrupt. The counter is continuously incre-

mented until the next notch triggers the $\overline{\text{IRQA}}$ fast interrupt. Once this interrupt occurs, the velocity timer (R1) is read, and the difference between the value of the counter for this notch and the previous notch indicates the velocity of the spindle. In effect, this example shows how the SCI timer can be used to provide a readable timer in software. Also, the SCI serial clock pin (SCLK) is available as a general-purpose I/O pin. The DSP56000/DSP56001 software for calculating the velocity in this example is shown in Figure 8-9.

```
        include 'declare.dat

;*************************************************************************
;
;Interrupt Service Routine Definitions
;*************************************************************************
;
        org         p:reset       ;RESET interrupt service routine
        jmp         main          ;go to main

        org         p:irqa        ;IRQA interrupt service routine
        jsr         velocity      ;update velocity information

        org         p:timer       ;SCI timer interrupt service routine
        move        (r0)+         ;update velocity counter
        nop                       ;unused second instruction

;*************************************************************************
;
;Main Routine
;*************************************************************************
;
        org         p:main        ;main routine
        movep       #$C007,x:ipr  ;set SCI,IRQA to priority level 2
                                  ;IRQA is negative edge triggered
        movep       #$0,x:bcr     ;no wait states
        movep       #$0,x:sccr    ;fastest possible timer rate
        movep       #$2000,x:scr  ;timer interrupt enable
        move        #0,r1
        andi        #$FC,mr       ;unmask all interrupts

        jmp         *

velocity
;       process velocity information
        move        #0,r0
        rti
```

Figure 8-9 Measuring the Disk Drive Spindle Velocity Using IRQA and the SCI Timer

Since the SCI timer is automatically reset and continues to run after it times out, the amount of time required to process the fast interrupt does not affect the overall calculation.

8.6 Generating Pulse-Width Modulated Outputs Using the SCI Timer

General-purpose I/O pins can be changed into PWM outputs with the aid of the SCI timer and modulo addressing. The DSP56000/DSP56001 assembly language program shown in Figure 8-10 is used to generate the PWM output shown in Figure 8-11.

The program pulse-width modulates the three general-purpose I/O pins, PC2-PC0. Instead of programming a duty cycle for each pin, the program increments through a table of values that control the voltage levels of each I/O pin. The SCI timer is used to generate periodic interrupts. The interrupt priority registered is configured to enable the SCI timer. When the timer times out, the fast interrupt associated with the SCI timer is executed. This interrupt service routine moves a new data value to the port C data register to toggle the appropriate PWM outputs. By executing an infinite loop of NOPS, the programmer is able to make the PWM outputs synchronous. As with all interrupts, the latency between when the interrupt is asserted and when

the first instruction of the fast interrupt is executed is dependent upon what instructions are being fetched, decoded, and executed at the time the interrupt is received. The reason for continuously executing NOPs is to guarantee that the pipeline is always filled with single-cycle instructions, allowing synchronous PWM output. If NOPs are not used, the PWM outputs exhibit a small jitter of a few instructions cycles, depending on which instructions are in the pipeline.

```
        include 'declare.dat'

        org         x:0                     ;PWM output waveform definit
data    dc          %111                    ; |_     |      |
        dc          %011                    ;  _|    |_     |
        dc          %101                    ; |_      _|    |_
        dc          %001                    ;  _|    |        |
        dc          %110                    ; |_     |         |
        dc          %010                    ;  _|    |_        |
        dc          %100                    ; |_       |      |
        dc          %000                    ;   |      |      |

        org         p:reset
        jmp         main                    ;upon reset, jmp to main

        org         p:timer                 ;alter I/O pin voltages
        jmp         x:(r0)+,x:pcd           ;unused second instr
        nop

        org         p:main                  ;main program
        movep       #$C000,x:ipr            ;enable sci interrupts
        move        #date, r0               ;initialize data pointer
        movep       #7,m0                   ;cir buffer is modulo 8
        movep       #7,x:pcddr              ;PC2-0 are outputs
        movep       #$2000,x:scr            ;enable sci timer
        movep       #0,x:sccr               ;minimum resolution for timer
        andi        #$FC,mr`                ;enable all interrupts

here
        do          #10000,end_do           ;infinite do loop of nops
        nop                                 ;waiting for next timer inter

end_do
        jmp         here
```

Figure 8-10 Generating Three PWM Signals on the General Purpose I/
Using the SCI Timer and Modulo Addressing

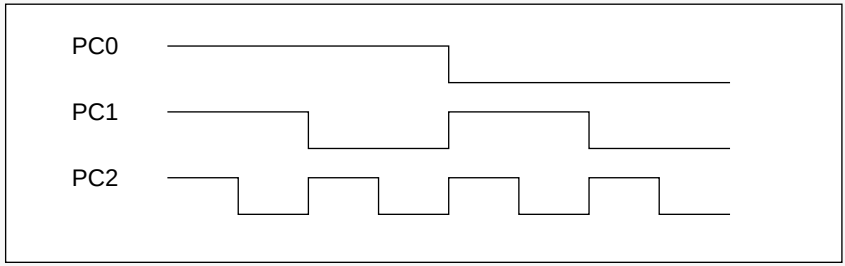


Figure 8-11 Outputs Generated from the PWM Example

8.7 Generating Three-Phase Out-puts Using Modulo Addressing

Modulo addressing can be used to provide a unique solution to the generation of the commutator signals for a three-phase motor. Figure 8-12 illustrates how three address registers point into a table containing a digitized version of the required waveform. In this example, a sine wave is used to control the motor, but any waveform can be stored in the waveform table. The speed of the motor is changed by varying the distance between the three address registers. When the end of the table is reached, the modulo addressing causes each of the individual address registers to wrap around and point to the beginning of the table again. The end result is a continuous,

three-phase, periodic output that can be customized for different motor types.

The DSP56000/DSP56001 software for implementing the three-phase output is shown in Figure 8-13.

■

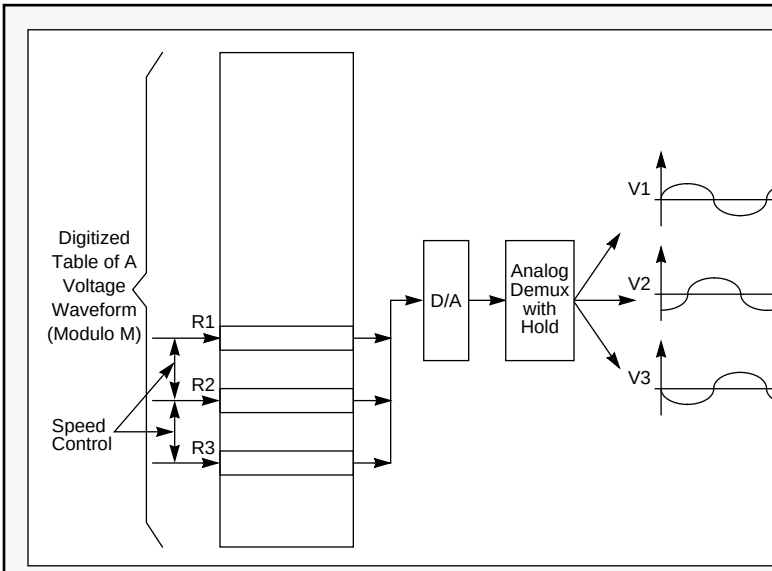


Figure 8-12 Generating Three-Phase Output Using Modulo Addressin

```
        include 'declare.dat'

D_A1    equ     $FFFD               ;address of first D_A
D_A2    equ     $FFFE               ;address of second D_A
D_A3    equ     $FFFF               ;address of third D_A

        org     p:reset             ;hardware reset int service routine
        jmp     main

        org     p:timer             ;SCI timer interrupt service routine
        jsr     output              ;output three phase voltages to D_As

        org     p:main              ;location of main routine
        movep   #$FC#C,x:ipr        ;unmask edge triggered interrupt
                                    ;      and wait for interrupt to occur.
        movep   #2,x:bcr            ;I/O wait states = 2
        ori     #4,omr              ;enable the data ROMs
        move    #$100,r1            ;r1 points to beginning of sine wave
        move    #255,m1             ;r1 is modulo 256
        move    #$140,r2            ;r2 is 90° out of phase6
        move    #255,m2             ;r2 is modulo 256
        move    #$180,r3            ;r3 is 180° out of phase
        move    #255,m3             ;r3 is modulo 256
        move    #$FC,mr             ;enable all interrupts

        jmp     *

output

        movep   y:(r1)+,y:D_A1      ;output phase 1


;repeat nops if delay is needed for multiplexed D_A

        movep   y:(r2)+,y:D_A2      ;output phase 2


;repeat nops if delay is needed for multiplexed D_A

        movep   y:(r3)+,y:D_A3      ;output phase 3

        rti
```

Figure 8-13 Generating Three-Phase Signals for Motor Control Using the SCI Timer and Sine Wave Table

SECTION 9

Conclusion

"All these features give the DSP56000/ DSP56001 the power to solve many of the world's difficult embedded control problems."

The unique architecture of the DSP56000/ DPS56001 enables it to function as a powerful microcontroller as well as a DSP. The dual data spaces allow a highly parallel implementation of basic control algorithms such as PID controllers and notch filters. The kernel of the biquad section can be executed in four to six instruction cycles. The increased throughput allows higher sampling rates and reduces the amount of negative phase added to the overall system due to the computational delay.

The reduced effects of coefficient quantization enable the DSP56000/DSP56001 to implement numerically sensitive algorithms such as phase-lag controllers and narrow-band low-pass filters having poles near the z= ±1points. The 56-bit accumulators, which provide 8-bit extension registers in conjunction with saturation arithmetic, allow the DSP to avoid overflow conditions and limit cycles. In addition, the output noise power due to roundoff noise of the 24-bit DSP56000/DSP56001 is 65,536 times less than that for 16-bit DSPs and microcontrollers.

Finally, the three on-chip peripherals, the HI, the SSI, and the SCI, allow the DSP56000/DSP56001 to function with minimal glue logic in an embedded control system. The two external hardware interrupts in conjunction with the SCI timer can be used to provide velocity feedback. The modulo arithmetic available in the address ALU can be used to efficiently implement PWM and finely tuned, three-phase output voltages. All these features give the DSP56000/DSP56001 the power to solve many of the world's difficult embedded control problems.■

APPENDIX A

Listing of ‘declare.dat’

This appendix contains all of the equates for the locations of the DSP56000/DSP56001's peripheral registers and interrupt service routines.

```
ipr      equ      $FFFF             ;interrupt priority register
bcr      equ      $FFFE             ;port a bus control register
srxh     equ      $FFF6             ;sci high rec register
stxh     equ      $FFF6             ;sci high xmit register
srxm     equ      $FFF5             ;sci middle rec register
stxm     equ      $FFF5             ;sci middle xmit register
srxl     equ      $FFF4             ;sci low rec register
stxl     equ      $FFF4             ;sci low xmit register
stxa     equ      $FFF3             ;sci transmit data address register
sccr     equ      $FFF2             ;sci clock control register
ssr      equ      $FFF1             ;sci status register
scr      equ      $FFF0             ;sci control register
rx       equ      $FFEF             ;ssi rx register
tx       equ      $FFEF             ;ssi tx register
ssisr    equ      $FFEE             ;ssi status register
crb      equ      $FFED             ;ssi control register b
cra      equ      $FFEC             ;ssi control register a
htx      equ      $FFEB             ;host transmit register
hrx      equ      $FFEB             ;host receive register
hsr      equ      $FFE9             ;host status register
hcr      equ      $FFE8             ;host control register
pcd      equ      $FFE5             ;port c data register
pbd      equ      $FFE4             ;port b data register
pcddr    equ      $FFE3             ;port c data direction register
pbddr    equ      $FFE2             ;port b data direction register
pcc      equ      $FFE1             ;port c control register
pbc      equ      $FFE0             ;port b control register
```

Figure A-1 Location of the Peripheral Registers

```
reset     equ     $00   ;reset interrupt service routine
stkerr    equ     $02   ;stack error interrupt service routine
trace     equ     $04   ;trace interrupt service routine
swi       equ     $06   ;software interrupt service routine
irqa      equ     $08   ;irqa interrupt service routine
irqb      equ     $0A   ;irqb interrupt service routine
ssirx     equ     $0C   ;ssi receive interrupt service routine
ssirxex   equ     $0E   ;ssi receive with exception interrupt service routine
ssitx     equ     $10   ;ssi transmit interrupt service routine
ssitxex   equ     $12   ;ssi transmit with exception interrupt service routine
scirx     equ     $14   ;sci receive interrupt service routine
scirxex   equ     $16   ;sci receive with exception interrupt service routine
scitx     equ     $18   ;sci transmit interrupt service routine
sciidle   equ     $1A   ;sci idle interrupt service routine
timer     equ     $1C   ;sci timer interrupt service routine
harddev   equ     $1E   ;hardware development interrupt service routine
hostrx    equ     $20   ;host receive interrupt service routine
hosttx    equ     $22   ;host transmit interrupt service routine
hc1       equ     $24   ;HC 1 interrupt service routine
hc2       equ     $26   ;HC 2 interrupt service routine
hc3       equ     $28   ;HC 3 interrupt service routine
hc4       equ     $2A   ;HC 4 interrupt service routine
hc5       equ     $2C   ;HC 5 interrupt service routine
hc6       equ     $2E   ;HC 6 interrupt service routine
hc7       equ     $30   ;HC 7 interrupt service routine
hc8       equ     $32   ;HC 8 interrupt service routine
hc9       equ     $34   ;HC 9 interrupt service routine
hc10      equ     $36   ;HC 10 interrupt service routine
hc11      equ     $38   ;HC 11 interrupt service routine
hc12      equ     $3A   ;HC 12 interrupt service routine
hc13      equ     $3C   ;HC 13 interrupt service routine
main      equ     $40   ;main routine
```

Figure A-2 Location of the Fast Interrupt Service Routines

■

REFERENCES

1. Astrom, K. and B. Wittenmark, Computer Controlled Systems, Englewood Cliffs, NJ: Prentice-Hall, 1984.

2. DSP56ADC16 Data Sheet, “16-Bit Sigma-Delta Analog-to-Digital Converter,” Motorola, 1989.

3. DSP56000/DSP56001 Digital Signal Processor User's Manual, Motorola, 1989.

4. Hanselmann, H., “Implementation of Digital Controllers—A Survey,” Automatica, vol. 23, no. 1, 1987, pp. 7-32.

5. Jackson, L., Digital Filters and Signal Processing, Boston, MA: Kluwer Academic Publishers, 1986.

6. Kuo, B., Digital Control Systems, New York, NY: Holt, Rienhart and Winston, Inc. 1980.

7. Kuo, B., Automatic Control Systems, Englewood Cliffs, NJ: Prentice-Hall, 1987.

8. Moroney, Issues in the Implementation of Digital Feedback Compensators, Cambridge, MA: The MIT Press, 1983.

9. Oppenheim, A. and R. Schafer, Digital Signal Processing, Englewood Cliffs, NJ: Prentice-Hall, 1975.

10. Phillips, C. and H. Nagle, Digital Control System Analysis and Design, Englewood Cliffs, NJ: Prentice-Hall, 1984.

11. Roberts, R. A. and C. T. Mullis, Digital Signal Processing, Reading, MA: Addison-Wesley, 1987. ■