From Continuous-Time Domain to Microcontroller Code

AND90334/D

Introduction

Control theory is one of the many aspects of electronic theory required for power electronic design. With the ever increasing popularity of digital control, it is important to have a good understanding of the basics of digital control. Many textbooks have been written about system modeling and control theory, but what can be difficult to find is a clear explanation of how to take an existing continuous-time model and convert it to something that can actually be programmed into a microcontroller. This is the subject of this application note. As an example, a continuous-time transfer function is implemented in C code, with the essential mathematical theory highlighted along the way.

Overview

The process of transforming a continuous-time transfer function to digital control code is as follows. We use the bilinear transformation to map the transfer function from the complex s-plane to the complex z-plane. After normalizing the denominator of the resulting z-domain transfer function, we rearrange it to solve for the output. Finally, we take the reverse Z-transform to yield a discrete-time difference equation that can be directly implemented in a digital controller.

We start with a continuous-time transfer function. This could be a filter or a control loop compensator (which is also a filter) derived from a small-signal model of the power converter. How to create the continuous-time transfer function is outside the scope of this application note. The goal here is to implement an existing transfer function digitally in a microcontroller.

Consider a general n^{th} order analog transfer function, with subscript *a* indicating analog.

$$H_a(S) = \frac{Y(s)}{X(s)} = \frac{B_0 + B_1 s + B_2 s^2 + \cdots + B_n s^n}{A_0 + A_1 s + A_1 s^2 + \cdots + A_n s^n} \quad \text{(eq. 1)}$$

In equation (1), $H_a\ (s)$ represents a filter transfer function where $Y(s)$ and $X(s)$ represent the output and input respectively, and s is the complex frequency variable $s = \sigma + j\omega$ with ω being the angular frequency. The coefficients *B0, B1, ···Bn* and *A0, A1, ···An* can be real numbers including zero. Such a transfer function can be used to solve an n^{th} order differential equation.

Many transfer functions are first or second order. First and second order filters can be cascaded to create higher order filters. An example of a second order low-pass filter transfer function is given below in equation (2), and this is the example that we will follow through to create sample C code for a microcontroller. In this equation, ω_0 is the natural angular frequency of the system, and ζ (zeta) is the damping factor.

$$H_F(S) = \frac{\omega_0^2}{s^2 + s2\zeta\omega_0 + \omega_0^2} \quad \text{(eq. 2)}$$

With $\zeta = \frac{\sqrt{2}}{2}$ this transfer function is a second-order Butterworth filter with a cutoff frequency at ω_0. (Use $\zeta = \frac{\sqrt{3}}{2}$ for Bessel.) Referring to equation (1), the coefficients for this example are:

$$B_0 = \omega_0^2, B_1 = 0, B_2 = 0, A_0 = \omega_0^2, A_1 = 2\zeta\omega_0, \text{and } A_2 = 1.$$

The Z-Transform

Comparable to how the Laplace transform converts continuous-time signals to the complex frequency s-plane, the z-transform converts a discrete-time sequence of numbers to a complex frequency z-plane representation. Mathematically, the discrete-time numbers can be real or complex, but in reality they represent a sequence of values sourced from the analog-to-digital converter (ADC); and input, stored, and output values of our digital filter.

Consider a discrete number sequence *x(k)* generated from data sampled every *T* seconds, *k = 0, 1, 2, ···*; and where *k(x)* is understood to be *x(kT)*, but the sampling period *T* is dropped for convenience. The z-transform of *x(k)* is defined as a power series of z^{-k} with coefficients equal to *x(k)*. The transform is then

$$X(z) = Z[x(k)] = x(0) + x(1)z^{-1} + x(2)z^{-2} + \cdots \quad \text{(eq. 3)}$$

In equation (3), $Z[\cdot]$ indicates the z-transform. This can be written as

$$X(z) = Z[x(k)] = \Sigma_{k=0}^{\infty} x(k)z^{-k} \quad \text{(eq. 4)}$$

This is the single-sided or unilateral z-transform. We need to know about the z-transform and a couple of its properties in order to take its inverse, resulting in a discrete-time sequence that is easily programmed into a digital controller. The first property is linearity, meaning that we can perform addition and multiplication with z-domain terms. The second property is time shifting (also called real translation).

$$Z(x(k - n)) = z^{-n} \cdot X(z) \quad \text{(eq. 5)}$$

To time-shift back *n* samples, we simply multiply the z-transform term by z^{-n}.

Now consider a discrete-time domain transfer function, with subscript *d* indicating discrete (or digital).

$$H_d(Z) = \frac{Y(z)}{X(z)} = \frac{\beta_0 + \beta_1 s + \beta_2 z^2 + \cdots + \beta_n z^n}{\alpha_0 + \alpha_1 z + \alpha_2 z^2 + \cdots + \alpha_n z^n} \quad \text{(eq. 6)}$$

Our z-domain transfer function could be in this form after performing the bilinear transform on the s-domain transfer function. We must convert it to a form that facilitates taking the inverse z-transfer of the transfer function, and for solving for the discrete-time output *y(k)*. To do this, we divide the top and bottom of equation (1) by $a_n z^n$, where *n* is the order of the transfer function. If we consider $n = 2$ in equation (6) and divide the numerator and denominator by $a_n z^n$, and rename the coefficients, the result is a normalized second order transfer function.

$$H_d(Z) = \frac{Y(z)}{X(z)} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 a^{-2}} \quad \text{(eq. 7)}$$

The transfer function must end up in this normalized form (1 + ⋯ in the denominator) with other terms raised to a negative power of z. This allows you to solve for *Y(z)*. *This is the key to implementing all this theory in a digital controller.* Due to linearity of the z-transform, we can rearrange equation (7).

$$Y(z)\,(1 + a_1 z^{-1} + a_2 z^{-2}) = X(z)\,(b_0 + b_1 z^{-1} + b_2 z^{-2}) \quad \text{(eq. 8)}$$

Finally, solve for *Y(z)*.

$$Y(z) = b_0 X(z) + b_1 X(z) z^{-1} + b_2 X(z) z^{-2} - a_1 Y(z) z^{-1} + a_2 Y(z) z^{-2} \quad \text{(eq. 9)}$$

Recalling the time shifting property, we can take the inverse z-transform directly.

$$y(k) = b_0 x(k) + b_1 x(k-1) + b_2 x(k-2) - a_1 y(k-1) - a_2 y(k-2) \quad \text{(eq. 10)}$$

This is a difference equation (solution to a differential equation in discrete-time) that is easily implemented in a digital controller. For example, *x(k)* is the recently measured value from the ADC, and *x (k − 1)* and *x (k − 2)* are the first and second previous measured values respectively. (Actual values are most likely scaled in some way.) Similarly, *y (k − 1)* is the previous result value, and *y(k − 2)* is from two results prior. So all values needed to compute the present value of *y(k)*, our desired output from the filter, are known.

As a side note, equation (7) is sometimes written with the coefficients of the denominator somewhat arbitrarily negated, probably to save an assembly code instruction in certain microcontrollers. This masks the fact that we are dealing with a difference equation, but the end result is of course the same.

The s and z Planes

Now that we have "seen the answer at the back of the book", we need to map from the s-plane to the z-plane, in preparation to performing the bilinear transform. The exact mapping between them is made through the relationships $z = e^{sT}$ and inversely $s = \frac{1}{T}\ln(z)$. By calculating *z* for various complex values of *s* and a fixed value for *T*, you find that the imaginary axis of the s-plane maps to the unit circle on the z-plane. The left and right halves of the s-plane map to within and without the z-plane unit circle respectively.

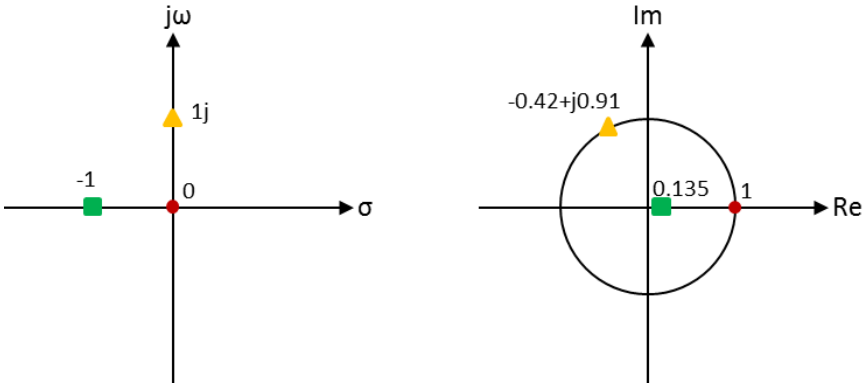

Figure 1. Complex s-plane (a) Complex z-plane (b) with Exact Mapping

As an example, consider a point 0 + j0 in the complex s-plane. It maps to 1 + j0 in the complex z-plane, as indicated by the red circles in Figure 1 (a) and (b). Now consider point 0 + j1 in the s-plane, which maps to −0.42 + j0.91 in the z-plane with the sampling time *T* set to 2 seconds, as indicated by the yellow triangles. This mapping is calculated using Euler's formula $e^{j\omega T} = \cos(\omega T) + j\sin(\omega T)$. Imagine the angular frequency increasing, which would correspond to the yellow triangle moving up along the axis in the s-plane, and rotating counterclockwise along the unit circle in the z-plane. A point in the z-plane can make any number of counterclockwise revolutions as the angular frequency increases, or clockwise revolutions as the frequency decreases. Finally, consider a point at −1 + j0 in the s-plane, which corresponds to 0.135 + j0 in the z-plane, indicated by the green squares in Figure 1 (a) and (b). Since this point is

in left half of the s-plane, it maps to within the unit circle in the z-plane. If this point moves farther left (more negative) along the real axis in the s-plane, the corresponding point in the z-plane moves closer to the origin but never quite reaches it.

The Bilinear Transform

Because the exact mapping $s = \frac{1}{T}ln(z)$ is nonlinear, it is difficult to implement in many digital controllers. An approximate mapping resulting in efficient filter computation is desirable. The bilinear transform is popular for this, especially because it results in unique mapping, and so as with exact mapping, there is no aliasing. Also, a stable analog filter yields a stable digital filter, and the gain (maxima and minima) is preserved in the digital filter. The phase margin of a feedback control loop filter is however reduced due to ADC delay, and there is frequency warping; both of which will be discussed.

The bilinear transform is defined as

$$s \leftarrow \frac{2}{T}\frac{z-1}{z+1} \qquad \text{(eq. 11)}$$

The inverse bilinear transform is

$$z \leftarrow \frac{1+\frac{sT}{2}}{1-\frac{sT}{2}} \qquad \text{(eq. 12)}$$

This can be derived either by trapezoidal integration, or by Taylor series approximation ignoring all but the first-order terms. The bilinear transformation is a first-order approximation, and therefore the mapping between s and z-planes is different from the exact mapping in some important ways.

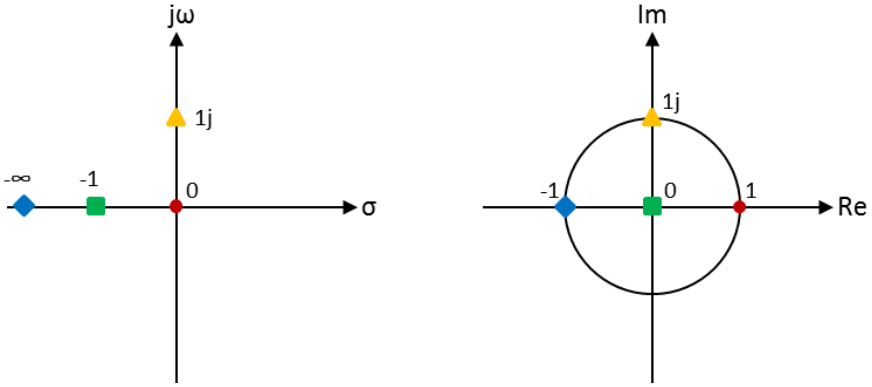

Figure 2. Complex s-plane (a) Complex z-plane (b) with Bilinear Approximation Mapping

Using the inverse bilinear transform, the point 0 + j0 in the s-plane maps to 1 + j0 in the z-plane, as with the exact mapping in Figure 1 (a) and (b). The point 0 + 1j in the s-plane now maps to 0 + 1j in the z-plane; still on the unit circle but with a different rotation. There is no revolution about the origin with the bilinear transform. Instead, increasing the angular frequency moves the yellow triangle in the s-plane of Figure 2 (a) up along the $j\omega$ axis while it approaches −1 + j0 along the unit circle in the z-plane of Figure 2 (b). Points on the $j\omega$ axis in the s-plane still map to the unit circle in the z-plane, and points in the left half of the s-plane still map to within the unit circle of the z-plane, but the frequency response is different. Along the real axes, the point −1 + j0 in the s-plane now maps to the origin of the z-plane, indicated by the green squares in Figure 2 (a) and (b). If a point moves farther left (more negative) along the real axis in the s-plane, the corresponding point in the z-plane moves closer to −1 along the real axis but never quite reaches it, indicated by the blue diamonds in Figure 2. To summarize, plus or minus infinity in frequency converges on −1 + j0 in the z-plane as well as minus infinity along the real axis of the s-plane.

Frequency Response and Warping

To determine the frequency response of in discrete-time, the exponential response is ignored such that the continuous-time complex frequency variable s reduces from $s = \sigma + j\omega$ to simply $s = j\omega$.

$$H_d(z) = H_d\left(e^{j\omega_d T}\right) = H_a\left(\frac{2}{T}\frac{z-1}{z+1}\right) = H_a\left(\frac{2}{T}\frac{e^{j\omega_d T}-1}{e^{j\omega_d T}+1}\right) \qquad \text{(eq. 13)}$$

Equation (13) can be simplified to

$$H_d\left(e^{j\omega_d T}\right) = H_a\left(j\frac{2}{T}\tan\left(\omega_d\frac{T}{2}\right)\right) \qquad \text{(eq. 14)}$$

Equation (14) verifies that points on the unit circle of the z-plane map to points on the $j\omega$ axis of the s-plane, and it shows that the discrete-time to continuous-time frequency mapping of the bilinear transform is

$$\omega_a = \frac{2}{T}\tan\left(\omega_d\frac{T}{2}\right) \qquad \text{(eq. 15)}$$

Solving for ω_d:

$$\omega_d = \frac{2}{T}\arctan\left(\omega_a\frac{T}{2}\right) \qquad \text{(eq. 16)}$$

At this point it is helpful to note that the Nyquist frequency in radians per second is $\omega_{Nyquist} = 2\pi\frac{1}{2T} = \frac{\pi}{T}$. With a sample time of 2 s, the Nyquist frequency is 1.57 rad/s.

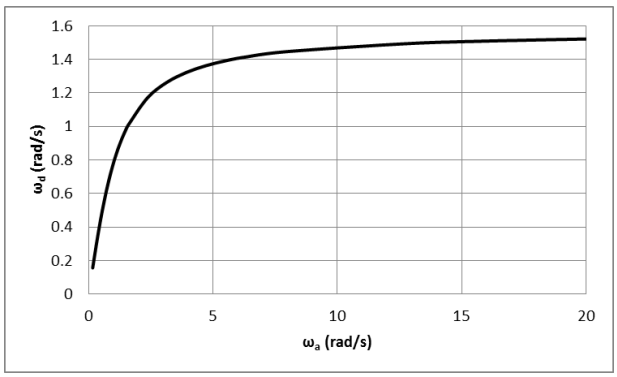

Figure 3. ω_d Versus ω_a with T = 2 s

Figure 3 clearly shows the frequency warping effect of the bilinear transform. At low frequency $\omega_d \cong \omega_a$, but as ω_a increases, ω_d levels off and approaches but never quite reaches the Nyquist frequency, even though ω_a is itself can be far above the Nyquist frequency, in a mathematical sense at least. The Nyquist frequency therefore correlates to the −1 + j0 point on the z-plane of the bilinear transform, see Figure 2 (b).

To use the bilinear transform effectively it is important to understand the error caused by frequency warping. Relating this error to the capability of the digital controller is helpful to know whether the discrete-time response will be acceptable, or if measures should be taken to reduce the error.

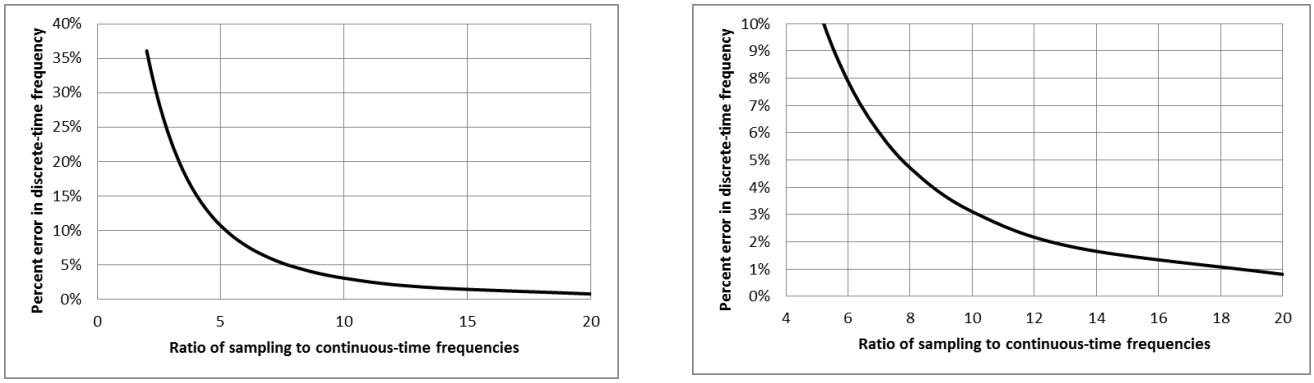

Figure 4. Error Versus Frequency Ratio with (a) Continuous-time Frequency ≤ Nyquist Frequency, and (b) Zoomed in for Clarity

In Figure 4, the error between continuous and discrete-time frequencies is plotted as a function of the ratio of the sampling frequency to the continuous-time frequencies. Figure 4 (a) starts with a ratio corresponding to the continuous-time frequency equal to the Nyquist frequency, and Figure 4 (b) shows a subset of the plot for clarity at lower percentage error. From Figure 4 (b) we see that if the sampling frequency is ten times higher than the continuous-time frequency, then the error in the discrete-time frequency is about 3.1 %. To keep the error within 1 % the sampling frequency must be at least 18 times higher than the continuous-time frequency of interest, which is usually a cutoff frequency for a filter.

Increasing the sampling frequency reduces the warping error at a given frequency. If increasing the sampling frequency is not practical, then it is possible to reduce the warping error at a certain frequency by pre-warping. This can be accomplished while making the bilinear transform by multiplying the substitution for the continuous-time variable s with $\frac{\omega_d}{\omega_a}$, and substituting in from equation (15). The pre-warped bilinear transform is thus

$$S \leftarrow \frac{\omega_d}{\omega_a}\cdot\frac{2}{T}\frac{z-1}{z+1} = \frac{\omega_d}{\frac{2}{T}\tan\left(\omega_d\frac{T}{2}\right)}\cdot\frac{2}{T}\frac{z-1}{z+1} = \frac{\omega_d}{\tan\left(\omega_d\frac{T}{2}\right)}\frac{z-1}{z+1} \quad \text{(eq. 17)}$$

Calculating Discrete-Domain Coefficients

To generate the discrete-time coefficients of the z-domain transfer function of equation (5), a bilinear transform with simplified notation is applied to the s-domain transfer function, with or without pre-warping. This notation is as follows.

$s \leftarrow K\frac{z-1}{z+1}$ where $K \triangleq \frac{2}{T}$ without pre-warping, and $K \triangleq \frac{\omega_0}{\tan\left(\omega_0\frac{T}{2}\right)}$ with pre-warping around angular frequency ω_0.

Substituting this into a general first-order continuous-time transfer function yields

$$H_a(s) = \frac{B_1 s + B_0}{A_1 s + A_0} \rightarrow H_d(z) = \frac{B_1 K \frac{z-1}{z+1} + B_0}{A_1 K \frac{z-1}{z+1} + A_0} = \frac{z(B_0 + B_1 K) + B_0 - B_1 K}{z(A_0 + A_1 K) + A_0 - A_1 K} \quad \text{(eq. 18)}$$

The denominator must be normalized to the form $1 + a_1 z^{-1}$, which is done by dividing the top and bottom of equation (18) by $z(A_0 + A_1 K)$.

$$H_d(z) = \frac{\frac{B_0 + B_1 K}{A_0 + A_1 K} + \frac{B_0 - B_1 K}{A_0 + A_1 K} z^{-1}}{1 + \frac{A_0 - A_1 K}{A_0 + A_1 K} z^{-1}} \quad \text{(eq. 19)}$$

Distinguishing z-domain coefficients with lower-case coefficients, this can be written in the form of equation (7) as $H_d(z) = \frac{b_0 + b_1 z^{-1}}{a_0 + a_1 z^{-1}}$ with $b_0 = \frac{B_0 + B_1 K}{A_0 + A_1 K}$, $a_0 = 1$, and $a_1 = \frac{A_0 - A_1 K}{A_0 + A_1 K}$.

See the appendix for formulas of second, third, and forth-order z-domain coefficients.

To apply pre-warping at a certain specified frequency ω_0, use the $\frac{\omega_0}{\tan(\omega_0 \frac{T}{2})}$ definition for K, otherwise use $K = \frac{2}{T}$. DC response is preserved, whether pre-warping or not. Beware however that while pre-warping aligns the continuous and discrete-time frequency response at the specified frequency, other frequencies are still warped, but differently from that shown in Figure 3. This could be a concern for example with bandpass and band-reject filters.

Example

A design goal is to implement a second-order Butterworth low-pass filter with a cutoff frequency within 5 % of 800 Hz with a microcontroller capable of sampling (performing an analog-to-digital conversion and executing the filter code) at a 10 kHz rate.

The sample frequency is 12.5 times higher than the cutoff frequency, which from Figure 4 (b) means the actual cutoff frequency of the digital filter is within 2 % of the 800 Hz specification. Up to 5 % error is allowed, so no pre-warping is required.

The continuous-time second-order Butterworth transfer function from equation (2) is repeated here for convenience.

$$H_F(s) = \frac{\omega_0^2}{s^2 + s2\zeta\omega_0 + \omega_0^2} \quad \text{with} \quad \zeta = \frac{\sqrt{2}}{2}, \quad \text{and} \quad \omega_0 = 2\pi 800 = 5026.5.$$

Substitute the coefficients , and into the formula for second-order z-domain coefficients from the appendix with since pre-warping is not required. With the sampling frequency equal to 10 kHz, the sampling period T is the inverse of 10 kHz, or 0.0001 seconds.

$$b_0 = \frac{B_0 + B_1 K + B_2 K^2}{A_0 + A_1 K + A_2 K^2} = \frac{\omega_0^2}{\omega_0^2 + 2\zeta\omega_0 \frac{2}{T} + \left(\frac{2}{T}\right)^2} = \frac{25266187}{25266187 + 142172254 + 400000000} = \frac{25266187}{567438441} = 0.044527$$

$$b_1 = \frac{2B_0 - 2B_2 K^2}{A_0 + A_1 K + A_2 K^2} = \frac{2\omega_0^2}{\omega_0^2 + 2\zeta\omega_0 \frac{2}{T} + \left(\frac{2}{T}\right)^2} = \frac{50532375}{567438441} = 0.089053$$

$$b_2 = \frac{B_0 - B_1 K + B_2 K^2}{A_0 + A_1 K + A_2 K^2} = \frac{2\omega_0^2}{\omega_0^2 + 2\zeta\omega_0 \frac{2}{T} + \left(\frac{2}{T}\right)^2} = b_0 = 0.044527$$

$$a_0 = 1$$

$$a_1 = \frac{2A_0 - 2A_2 K^2}{A_0 + A_1 K + A_2 K^2} = \frac{2\omega_0^2 - 2\left(\frac{2}{T}\right)^2}{\omega_0^2 + 2\zeta\omega_0 \frac{2}{T} + \left(\frac{2}{T}\right)^2} = \frac{50532375 - 800000000}{567438441} = -1.320791$$

$$a_2 = \frac{A_0 - A_1 K + A_2 K^2}{A_0 + A_1 K + A_2 K^2} = \frac{\omega_0^2 - 2\zeta\omega_0 \frac{2}{T} + \left(\frac{2}{T}\right)^2}{\omega_0^2 + 2\zeta\omega_0 \frac{2}{T} + \left(\frac{2}{T}\right)^2} = \frac{25266187 - 142172254 + 400000000}{567438441} = 0.498898$$

The C code to implement the filter is shown below. The define statements and data structure definition are typically placed in a header file, included in a source code file with the instantiation of the filter data structure and the filter code. This code is intended for a floating point microcontroller, such as TMS320F28335 or TMS320F28069 from Texas Instruments. Notice how simple the actual C code is. Such code compiles into very efficient executable code.

```
// IIR "Butterworth" filter coefficients, 800 Hz corner freq, 10 kHz sample rate
#define BW_b2  0.044527
#define BW_b1  0.089053
#define BW_b0  0.044527
#define BW_a2  0.498898
#define BW_a1 -1.320791
typedef struct  // Second order filter structure
{
  float x[3], y[3]; //y[0] stores the "clean" output value
}filter2struct;
volatile filter2struct filter2;
// ***** IIR "Butterworth" filter, 800 Hz corner freq, 10 kHz sample rate
filter2.x[0] = ADCbuf[0]; // ADC buffer contains input data to filter
filter2.y[0] = BW_b2 * filter2.x[2]
             + BW_B1 * filter2.x[1]
             + BW_b0 * filter2.x[0]
             - BW_a2 * filter2.y[2]
             - BW_a1 * filter2.y[1];
filter2.x[2] = filter2.x[1];
filter2.x[1] = filter2.x[0];
filter2.y[2] = filter2.y[1];
filter2.y[1] = filter2.y[0]; // y[0] stores the "clean" output value
```

ADC Delay and Phase Lag

If we represent the phase lag introduced by the ADC and other delays as a zero-order hold, then the phase lag caused by delays can be expressed as

$$\theta_{lag} = \omega \cdot \frac{\delta t}{2} \qquad \text{(eq. 20)}$$

In equation (20), θ_{lag} is the phase shift in radians, δ is frequency in radians per second, and δt is the delay time, meaning the delay time between sampling and updating the PWM duty. Sandwiched within the delay are the ADC sequence, filter computation, and other processing overhead. Some gain and phase margin are required in feedback control systems to ensure reliable operation in the presence of perturbation (noise) and mismatches between system modeling and reality. The frequency of interest for ω is when the gain of the open-loop transfer function (plant plus filter) drops to unity (0 dB on a Bode plot). A desirable range for phase margin at this frequency is 45 to 60 degrees.

If the ADC sampling is initiated by the pulse width modulation (PWM) interrupt service routing (ISR) in a microcontroller, and filter computations are made in the PWM ISR, then the PWM ISR frequency is the sampling frequency, and its inverse is the sampling period *T*, which is equal to δt in equation (20). This is actually a worst-case scenario, for which the phase lag is $\theta_{lag} = \omega_{crossover} \cdot \frac{T}{2}$. For example, suppose the sampling frequency is 10x the unity-gain crossover frequency, then the phase lag introduced by the sampling-to-update delay would be $\varphi_{lag} = 2\pi f_{crossover} \cdot \frac{T}{2} = 2\pi \frac{1}{10T} \cdot \frac{T}{2} = \frac{\pi}{10} = 0.314$ rad, or 18 degrees. There are techniques and features for microcontrollers that can reduce the delay compared to this example. The point is that additional phase margin must be added before performing the bilinear transform to account for delays within the digital feedback control system. Alternatively, these delays can be accounted for in a discrete-time model of the control system, which is beyond the scope of this application note.

References

[1] Brian Douglas' YouTube channel Control Systems Lectures
[2] Wikepedia Bilinear Transform page
[3] Feedback Control Systems by Charles L. Phillips, Royce D. Harbor, Prentice Hall 1988
[4] Mark Hagan, Vahid Yousefzedeh; "Applying Digital Technology to PWM Control-Loop Designs", Texas Instruments seminar

Appendix – Formulas for Calculation of Z-Domain Coefficients

First Order

$$H_a(s) = \frac{B_1 s + B_0}{A_1 s + A_0} \rightarrow H_d(z) = \frac{B_1 K\frac{z-1}{z+1} + B_0}{A_1 K\frac{z-1}{z+1} + A_0} = \frac{(B_0 + B_1 K)z + B_0 - B_1 K}{(A_0 + A_1 K)z + A_0 - A_1 K} = \frac{\frac{B_0+B_1K}{A_0+A_1K} + \frac{B_0-B_1K}{A_0+A_1K}z^{-1}}{1 + \frac{A_0-A_1K}{A_0+A_1K}z^{-1}}$$

$$H_d(z) = \frac{b_0 + b_1 z^{-1}}{a_0 + a_1 z^{-1}}$$

$b_0 = \frac{B_0 + B_1 K}{A_0 + A_1 K}$	$a_0 = 1$
$b_1 = \frac{B_0 + B_1 K}{A_0 + A_1 K}$	$a_1 = \frac{A_0 + A_1 K}{A_0 + A_1 K}$

$$y(k) = b_0 x(k) + b_1 x(k-1) - a_1 y(k-1)$$

Second Order

$$H_a(s) = \frac{B_2 s^2 + B_1 s + B_0}{A_2 s^2 + A_1 s + A_0} \rightarrow H_d(z) = \frac{B_2\left(K\frac{z-1}{z+1}\right)^2 + B_1 K\frac{z-1}{z+1} + B_0}{A_2\left(K\frac{z-1}{z+1}\right)^2 + A_1 K\frac{z-1}{z+1} + A_0} =$$

$$\frac{\left(B_0 + B_1 K + B_2 K^2\right)z^2 + \left(2B_0 - 2B_2 K^2\right)z + B_0 - B_1 K + B_2 K^2}{\left(A_0 + A_1 K + A_2 K^2\right)z^2 + \left(2A_0 - 2A_2 K^2\right)z + A_0 - A_1 K + B_2 K^2} =$$

$$\frac{\frac{B_0+B_1K+B_2K^2}{A_0+A_1K+A_2K^2} + \frac{2B_0-2B_2K^2}{A_0+A_1K+A_2K^2}z^{-1} + \frac{B_0-B_1K+B_2K^2}{A_0+A_1K+A_2K^2}z^{-2}}{1 + \frac{2A_0-2A_2K^2}{A_0+A_1K+A_2K^2}z^{-1} + \frac{A_0-A_1K+A_2K^2}{A_0+A_1K+A_2K^2}z^{-2}}$$

$$H_d(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{a_0 + a_1 z^{-1} + a_2 z^{-2}}$$

$b_0 = \frac{B_0 + B_1 K + B_2 K^2}{A_0 + A_1 K + A_2 K^2}$	$a_0 = 1$
$b_1 = \frac{2B_0 - 2B_2 K^2}{A_0 + A_1 K + A_2 K^2}$	$a_1 = \frac{2A_0 + 2A_2 K^2}{A_0 + A_1 K + A_2 K^2}$
$b_2 = \frac{B_0 - B_1 K + B_2 K^2}{A_0 + A_1 K + A_2 K^2}$	$a_2 = \frac{A_0 + A_1 K + A_2 K^2}{A_0 + A_1 K + A_2 K^2}$

$$y(k) = b_0 x(k) + b_1 x(k-1) - b_2 x(k-2) - a_1 y(k-1) - a_2 y(k-2)$$

Third Order

$$H_a(s) = \frac{B_3s^3 + B_2s^2 + B_1s + B_0}{A_3s^3 + A_2s^2 + A_1s + A_0} \rightarrow H_d(z) = \frac{B_3\left(K\frac{z-1}{z+1}\right)^3 + B_2\left(K\frac{z-1}{z+1}\right)^2 + B_1K\frac{z-1}{z+1} + B_0}{A_3\left(K\frac{z-1}{z+1}\right)^3 + A_2\left(K\frac{z-1}{z+1}\right)^2 + A_1K\frac{z-1}{z+1} + A_0}$$

$$H_d(z) = \frac{b_0 + b_1z^{-1} + b_2z^{-2} + b_3z^{-3}}{a_0 + a_1z^{-1} + a_2z^{-2} + a_3z^{-3}}$$

$b_0 = \frac{B_0 + B_1K + B_2K^2 + B_3K^3}{A_0 + A_1K + A_2K^2 + A_3K^3}$	$a_0 = 1$
$b_1 = \frac{3B_0 + B_1K - B_2K^2 - 3B_3K^3}{A_0 + A_1K + A_2K^2 + A_3K^3}$	$a_1 = \frac{3A_0 + A_1K - 2A_2K^2 - 3A_3K^3}{A_0 + A_1K + A_2K^2 + A_3K^3}$
$b_2 = \frac{3B_0 - B_1K - B_2K^2 + 3B_3K^3}{A_0 + A_1K + A_2K^2 + A_3K^3}$	$a_2 = \frac{3A_0 - A_1K - A_2K^2 + 3A_3K^3}{A_0 + A_1K + A_2K^2 + A_3K^3}$
$b_3 = \frac{B_0 - B_1K + B_2K^2 - B_3K^3}{A_0 + A_1K + A_2K^2 + A_3K^3}$	$a_3 = \frac{A_0 - A_1K + A_2K^2 - A_3K^3}{A_0 + A_1K + A_2K^2 + A_3K^3}$

$$y(k) = b_0x(k) + b_1x(k-1) + b_2x(k-2) + b_3x(k-3) - a_1y(k-1) - a_2y(k-2) - a_3y(k-3)$$

Fourth Order

$$H_a(s) = \frac{B_4s^4 + B_3s^3 + B_2s^2 + B_1s + B_0}{A_4s^4 + A_3s^3 + A_2s^2 + A_1s + A_0} \rightarrow H_d(z) = \frac{B_4\left(\frac{z-1}{z+1}\right)^4 + B_3\left(K\frac{z-1}{z+1}\right)^3 + B_2\left(K\frac{z-1}{z+1}\right)^2 + B_1K\frac{z-1}{z+1} + B_0}{A_4\left(K\frac{z-1}{z+1}\right)^4 + A_3\left(K\frac{z-1}{z+1}\right)^3 + A_2\left(K\frac{z-1}{z+1}\right)^2 + A_1K\frac{z-1}{z+1} + A_0}$$

$$H_d(z) = \frac{b_0 + b_1z^{-1} + b_2z^{-2} + b_3z^{-3} + b_4z^{-4}}{a_0 + a_1z^{-1} + a_2z^{-2} + a_3z^{-3} + a_4z^{-4}}$$

$b_0 = \frac{B_0 + B_1K + B_2K^2 + B_3K^3 + B_4K^4}{A_0 + A_1K + A_2K^2 + A_3K^3 + A_4K^4}$	$a_0 = 1$
$b_1 = \frac{4B_0 + 2B_1K - 2B_3K^3 - 4B_4K^4}{A_0 + A_1K + A_2K^2 + A_3K^3 + A_4K^4}$	$a_1 = \frac{4A_0 + 2A_1K - 2A_3K^3 - 4A_4K^4}{A_0 + A_1K + A_2K^2 + A_3K^3 + A_4K^4}$
$b_2 = \frac{6B_0 - 2B_2K^2 + 6B_4K^4}{A_0 + A_1K + A_2K^2 + A_3K^3 + A_4K^4}$	$a_2 = \frac{6B_0 - 2B_2K^2 + 6B_4K^4}{A_0 + A_1K + A_2K^2 + A_3K^3 + A_4K^4}$
$b_3 = \frac{4B_0 - 2B_1K + 2B_3K^3 - 4B_4K^4}{A_0 + A_1K + A_2K^2 + A_3K^3 + A_4K^4}$	$a_3 = \frac{4A_0 - 2A_1K + 2A_3K^3 - 4A_4K^4}{A_0 + A_1K + A_2K^2 + A_3K^3 + A_4K^4}$
$b_4 = \frac{B_0 - B_1K + B_2K^2 - B_3K^3 + B_4K^4}{A_0 + A_1K + A_2K^2 + A_3K^3 + A_4K^4}$	$b_4 = \frac{A_0 - A_1K + A_2K^2 - A_3K^3 + A_4K^4}{A_0 + A_1K + A_2K^2 + A_3K^3 + A_4K^4}$

$$y(k) = b_0x(k) + b_1x(k-1) + b_2x(k-2) + b_3x(k-3) + b_4x(k-4) - a_1y(k-1) - a_2y(k-2) - a_3y(k-3) - a_4y(k-4)$$

ADDITIONAL INFORMATION

TECHNICAL PUBLICATIONS:
Technical Library: www.onsemi.com/design/resources/technical−documentation
onsemi Website: www.onsemi.com

ONLINE SUPPORT: www.onsemi.com/support
For additional information, please contact your local Sales Representative at www.onsemi.com/support/sales