ANALOG DEVICES

AN-227
APPLICATION NOTE

ONE TECHNOLOGY WAY • P.O. BOX 9106 • NORWOOD, MASSACHUSETTS 02062-9106 • 617/329-4700

Digital Control System Design With The ADSP-2100 Family

by Kapriel Karagozyan

INTRODUCTION

The ADSP-2100 Family of Digital Signal Processors is well suited for implementing complex measurement and control algorithms in embedded control systems with high sampling rates. This is mainly due to their computing speed, which is much greater than that of conventional microcontrollers and microprocessors. Typical application areas include servo motor control, process control, robot arm control, disk drive head control, flight control and general servomechanisms.

This application note presents the implementation of several common control algorithms on the ADSP-2100 Family of DSP processors, and presents software and hardware design methods as well as guidelines for designing high speed digital control systems with the ADSP-2100 Family. A table of representative benchmarks for common digital control algorithms can be found at the end of this note.

DIGITAL CONTROL SYSTEMS OVERVIEW

A controller is a system used to control closed-loop feedback systems. It implements algebraic algorithms, such as filters and compensators, in order to regulate, correct, or change the behavior of a controlled system. Controllers can be implemented using analog or digital circuitry. A digital control system is comprised of a digital controller, the controlled plant (or system), and the necessary input/output devices. A general digital control system is shown in Figure 1. Note the analog-to-digital (A/D) and digital-to-analog (D/A) converters that are used to interface the digital controller with the plant (which is a continuous time system). There are several advantages to using a digital controller implementation instead of an analog one. In the case of digital controllers, complex control algorithms can be implemented in software or firmware rather than in special hardware. Digital controller designs and parameters can be changed without affecting the hardware. In digital control systems increased noise immunity is guaranteed and parameter drift is eliminated. Such systems are more reliable, maintainable, and testable. Finally, digital control systems feature reduced size, power, weight and costs.

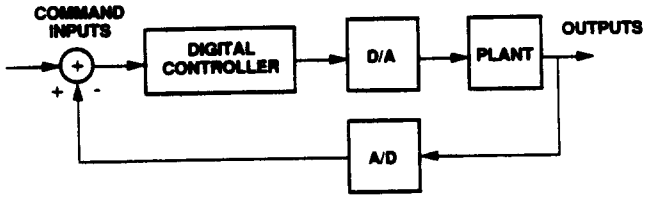


Figure 1. General Digital Control System

Analog Devices' ADSP-2100 Family of Digital Signal Processors has several beneficial features for implementing digital controllers. These features include the following:

- Single-cycle instruction execution
- Three arithmetic function units arranged in parallel
- Single-cycle 16x16-bit multiplications
- Single-cycle ([16x16]+40 bit) multiply-accumulate operations with 40-bit results
- Single-cycle 16-bit additions, subtractions and logical operations
- Single-cycle bit shifts (up to 32 bits at a time)
- Efficient execution of 32-bit (or higher) arithmetic operations
- Efficient modulo addressing for data and coefficient arrays in memory
- No cycle penalty for looped code execution
- Single-cycle access of internal and external memory
- Single or multi-cycle parallel accesses of external peripherals (i.e., A/D, D/A)
- Up to four levels of nested external interrupts
- On-chip interval timer and serial ports
- Low power consumption (CMOS) and power down "idle" mode
- Easy-to-read algebraic assembly language syntax
- Complete set of hardware and software development tools

DIGITAL CONTROL SYSTEM MODEL

Most practical control systems use feedback in their operation. Figure 2 shows a model for a typical closed-loop digital control system. R(z), E(z), U(z) and Y(z) are the z-transforms of the reference input, the error signal, the control signal, and the plant output respectively. G(z) is the transfer function corresponding to the digital controller, while P(z) is the transfer function describing the input-output behavior of the object to be controlled (e.g., plant). This does not imply that the object to be controlled (e.g., a plant) must be a discrete system, but rather that it must be modeled as one. P(z) is also assumed to contain the transfer characteristics of the A/D and the D/A converters that are needed to implement a real system.

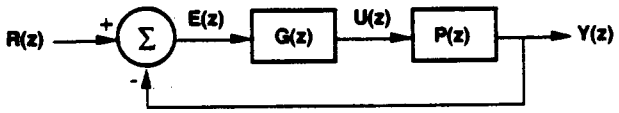


Figure 2. Digital Control System Model

DIGITAL CONTROL SYSTEM HARDWARE IMPLEMENTATION

A digital controller G(z), as shown in Figure 2, must be able to observe and alter certain characteristics of the controlled system. For example, an ADSP-2100 Family-based digital controller can be used to control fast and accurate positioning of an actuator shaft upon an external command R(z). In this case, the output of the controller can be used to alter the amount of current U(z) that is fed into the actuator windings which in turn would move the actuator shaft. In a closed-loop system, the same controller would also need to observe the position of the actuator at all times. This can be achieved by recording the position Y(z) of the shaft at specific intervals and feeding it back to the controller. This would allow the controller to compare the desired shaft position to the actual measured position and make the necessary adjustments in the actuator current. This simple controller example can serve as a starting point for constructing an actual hardware implementation.

The block diagram for a digital control system based on the ADSP-2102 is shown in Figure 3. The ADSP-2102 performs the digital control algorithms by executing instructions from its on-chip program memory ROM. The ROM is also used to store fixed coefficients and scale factors. The processor uses its on-chip data memory RAM and program memory RAM to store incoming data values and other intermediate variables.

The ADSP-2102 accepts up to three external hardware interrupts. In a typical digital control system, the processor operation is interrupt-driven. In the system shown in Figure 3, an external clock (sample clock) drives one of the ADSP-2102 interrupts. The same clock is typically used to initiate A/D conversions at regular intervals. Other interrupts can be set by the host to notify the ADSP-2102 of new commands, expiration of a watchdog timer, etc.

The ADSP-2102 outputs its control current via a D/A converter, whose output is amplified before it is fed into the motor. The processor receives its feedback from a position encoder which can be an optical shaft encoder, a synchro-to-digital converter, a resolver-to-digital converter, or some other circuitry with an

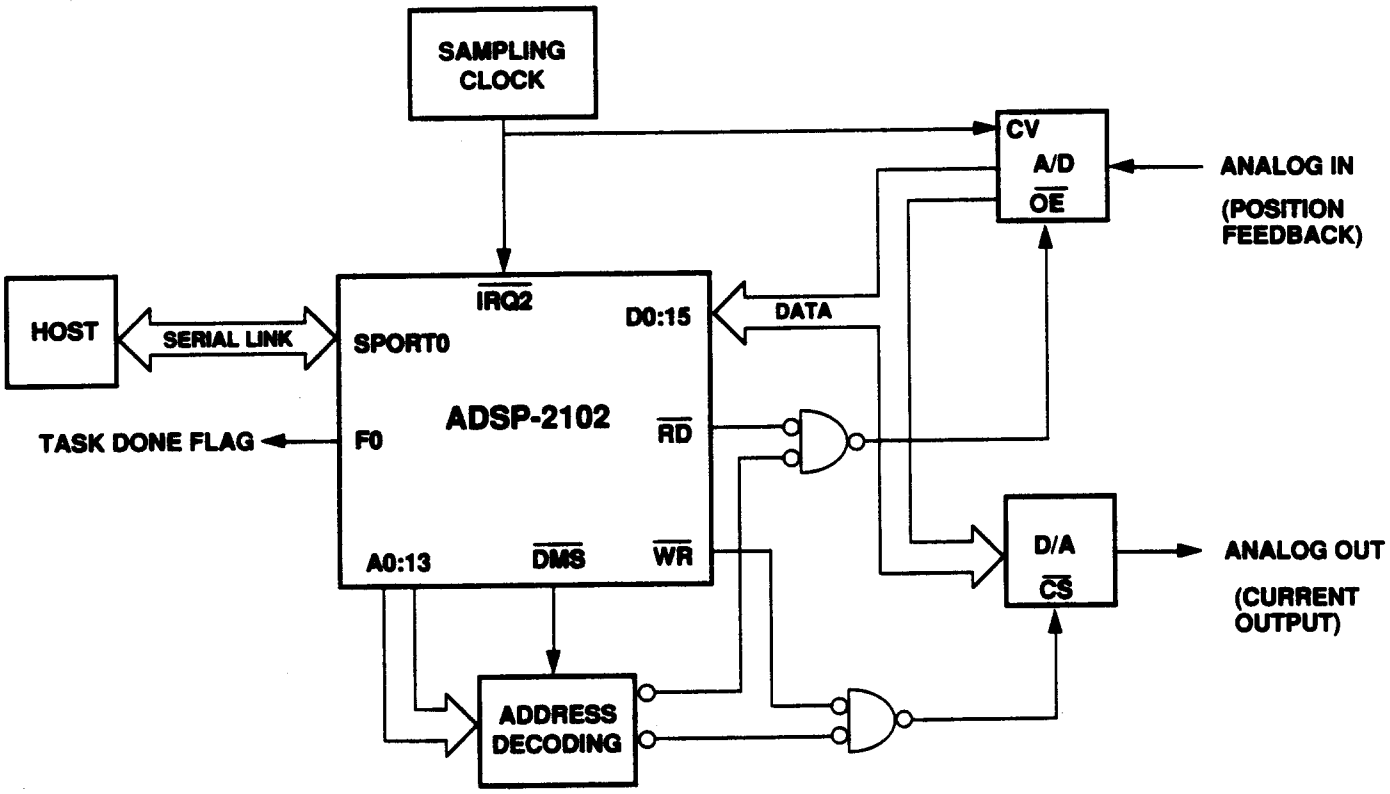


Figure 3. ADSP-2101 Based Actuator Controller

A/D converter. The A/D label is used in the figure since feedback essentially involves an analog-to-digital conversion. Data transfers between the processor and the converters are done over the 16-bit data bus. The converters are mapped into the ADSP-2102's external data memory space. This allows the processor to access them as memory locations. The address decoding circuitry shown in Figure 3 is used to map every converter to a separate data memory location. If the converters have slow data bus interfaces, the processor can extend the duration of the converter access cycles by inserting wait states. The data converters can also be tied to the ADSP-2102's serial ports if it is more convenient to do so. Several serial input and serial output converters are available from Analog Devices, such as the AD766, AD7772, AD7868, and AD7878.

The ADSP-2102 receives its reference position command R(z) from a host processor or an internal software routine that is running concurrently with the shaft positioning program. Figure 3 depicts the case where serial port 0 on the ADSP-2102 is used to exchange commands and results with a host processor. The flag out pin on the ADSP-2102 can be used to notify the host of the completion of a specific task.

The ADSP-2100 Family processors can interface with multiple A/D and D/A converters in order to monitor and control several motors, actuators, or processes. These converters can simply be added as memory-mapped peripherals like the ones shown in Figure 3. Several general purpose and special purpose data converters for digital control applications are available from Analog Devices.

DIGITAL CONTROL SYSTEM SOFTWARE IMPLEMENTATION

The software running in a digital controller system is responsible for executing the control algorithms which are represented by G(z) in the model on Figure 2. Typically, G(z) can be broken into smaller sub-tasks. For example it may be necessary to execute a state estimator, several notch filters and some PID (Proportional, Integral, Derivative) control as a whole function. Generally, a separate portion of the software must manage the input/output operations of the controller with the host and other peripherals. A diagnostic error checking and handling routine is also usually developed, to be run at powerup or at specified intervals during program execution. Finally there is a main manager routine that is responsible for the orchestration of these different subroutines.

The software must be organized in a modular manner in order for the main managing program to call every sub-task as a subroutine. The ADSP-2100 Family Development Software tools encourage modular programming. The subroutines can be written, assembled, and debugged as independent modules which can later be linked with the main manager program. Parameter passing and symbolic coding is supported on the assembler, linker and simulator. An example of a fully coded notch filter algorithm is shown in a later section of this application note. The ADSP-2100 Assembler and Simulator manuals describe the software tools.

Memory management is very straightforward in the ADSP-2100 Family processors. The Data memory (DM) space is typically used for variables and data storage. The incoming A/D samples can be stored in data memory buffers. A large number of variables and intermediate values can also be stored in DM space. The Program memory (PM) space is divided into two sections: the PM instruction space and the PM data space. The instruction space is used to store the programs to be executed.The PM space can also be freely used for additional data and variable storage. This data space is usually used to store filter coefficients and various other tables that may need to be present during program execution. The ADSP-2100 Family processors can read or write to both DM and PM locations in a single instruction cycle and execute an arithmetic operation at the same time. This not only allows classical control algorithms to execute at very high speeds but also allows very efficient implementation of adaptive control algorithms. This is due to the fact that in adaptive control, filter coefficients must be updated periodically with every new incoming sample. These coefficient values can be updated easily in the PM space and can be readily available on the next processing cycle.

The following sections discuss the implementation of first, second, and higher-order control algorithms with the ADSP-2100 Family processors.

DIGITAL PID CONTROLLER DESIGN

The controller G(z) shown in Figure 2 can be designed to vary its output U(z) in relation to the error feedback E(z). A PID (Proportional, Integral, Derivative) controller derives its name from the fact that its output U(z) is a weighted sum of the error signal, its integral, and its derivative. PID controllers are widely-used building blocks in a large variety of servo control applications.

Since analog PID controllers are well understood, it is often desirable to start a digital PID controller design in the continuous domain and then create discrete equivalents. In the continuous time case if E(t) is the error feedback, the PID output U(t) can be expressed as:

$$U(t) = K_p \cdot E(t) + K_d \cdot dE(t)/dt + K_i \cdot \int_0^t E(\tau)d\tau \qquad (1)$$

where K_p, K_d, and K_i are the gains associated with the proportional, derivative, and integral terms, respectively. Equation (1) can be represented in the frequency domain by using Laplace transforms:

$$U(s) = K_p \cdot E(s) + K_d \cdot s \cdot E(s) + (K_i/s)\, E(s) \qquad (2)$$

where it is assumed that the initial conditions are 0. The equations (1) and (2) are graphically represented in Figure 4.

The coefficients Kp, K_d and K_i must be determined during the design process. These coefficients will depend on the desired controller characteristics and will be varied in different systems.

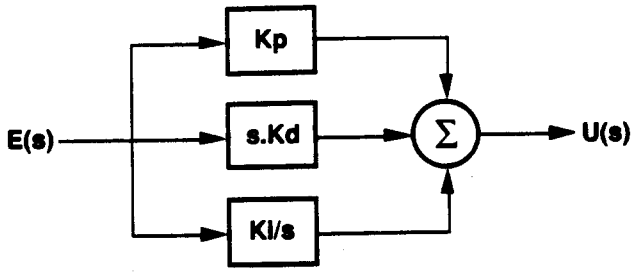


Figure 4. PID Block Diagram

Other types such as PD (Proportional, Derivative) and PI (Proportional, Integral) controllers can also be expressed in a similar manner to the PID relationships in (1) and (2). These controllers lack either the differential or the integral term that exists in the relationships above.

The next step is to derive the discrete equivalent for the controller described by equations (1) and (2). The backward difference is defined as the discrete-time equivalent for the continuous-time derivative of a function. It is obtained by:

$$\Delta f(t) = [f(t) - f(t-T)] / T \qquad (3)$$

where T is the sample period.

The definite sum is defined as the discrete time equivalent for the continuous time integral of a function. It is obtained by:

$$\int_a^b f(\tau) = T [f(a+T) + f(a+2T) + \cdots + f(b)] \qquad (4)$$

where T is the sample period.

By applying the relationships shown in (3) and (4) to the ones shown in (1) and (2) we obtain U(n) and U(z), which are the discrete equivalents of the PID output U(t):

$$U(n) = U(n-1) + A_1 \cdot E(n) + A_2 \cdot E(n-1) + A_3 \cdot E(n-2) \qquad (5)$$

and

$$U(z) / E(z) = [A_1 \cdot z^2 + A_2 \cdot z + A_3] / [z^2 - z] \qquad (6)$$

with

$$A_1 = K_p + K_i \cdot T + K_d / T$$
$$A_2 = -[K_p + 2K_d / T]$$
$$A_3 = K_d / T$$

Transfer functions and difference equations for PD and PI controllers can also be derived in a similar manner. Figure 5 shows the results for these as well as for the PID controllers.

PID CONTROLLER IMPLEMENTATION

An ADSP-2100 Family assembly language subroutine that implements the PID algorithm is shown in Listing 1. There are a number of registers that need to be initialized in order to execute this subroutine. It may be sufficient to do this initialization only once (e.g. on powerup) if other algorithms that are being executed do not need to use these registers. In most typical cases, however, some of these registers may need to be set every time the PID subroutine is called.

The PID routine in Listing 1 takes its input from the AR register. This register must contain the 16-bit error input E(n). E(n) is assumed to be already computed before the PID subroutine is called. The output of the PID algorithm, U(n), is made available in the SR1 register.

	PD Controller	PI Controller	PID Controller
Definitions	$A_1 = K_P + \frac{K_D}{T}$ $A_2 = -\frac{K_D}{T}$	$A_1 = K_P + K_I T$ $A_2 = -K_P$	$A_1 = K_P + K_I T + \frac{K_D}{T}$ $A_2 = -[K_P + 2\frac{K_D}{T}]$ $A_3 = \frac{K_D}{T}$
Transfer Function, $\frac{C(z)}{E(z)}$	$\frac{A_1 z + A_2}{z}$	$\frac{A_1 z + A_2}{z-1}$	$\frac{A_1 z^2 + A_2 z + A_3}{z^2 - z}$
Difference Equation	$U(n) = A_1 E(n) + A_2 E(n-1)$	$U(n) = U(n-1) + A_1 E(n) + A_2 E(n-1)$	$U(n) = U(n-1) + A_1 E(n) + A_2 E(n-1) + A_3 E(n-2)$

Figure 5. PD, PI, and PID Controllers

```
.MODULE        PID_CONTROLLER;

{   This is a PID controller subroutine that executes the following equation:

                                U(n) = B•U(n-1) + A0•E(n) + A1•E(n-1) + A2•E(n-2)

                Calling Parameters:

                    AR=       error input E(n), [E(n) = Y(n) - R(n)]
                    I0 ->     circular delay line buffer for E(n-2), E(n-1) and U(n-1)
                              this delay buffer must be initialized to zero at powerup
                    I4 ->     circular buffer for the scaled coefficients A2, A1, A0, B
                    M0,M4=    1
                    L0 =      3
                    L4 =      4
                    SE=       scaling factor for the coefficients

                Return Value:
                    SR1=   output sample U(n)

                Altered Registers:
                    MX0, MX1, MY0, MR, SR

                Computation Time:
                    ADSP2101/2  :    8 Instruction Cycles
                    ADSP2105/6  :    8 Instruction Cycles
                    ADSP2100/A  :    12 Instruction Cycles

                All coefficients and data values are assumed to be in 1.15 format
}

.ENTRY         PID;

PID:           MX0 = DM(I0,M0), MY0 = PM(I4,M4);
               MR = MX0*MY0 (SS), MX1 = DM(I0,M0), MY0 = PM(I4,M4);
               MR = MR+MX0*MY0 (SS), MY0 = PM(I4,M4);
               MR = MR+AR*MY0(SS), MX0 = DM(I0,M0), MY0 = PM(I4,M4);
               MR = MR+MX0*MY0 (RND), DM(I0,M0)= MX1;
               SR = ASHIFT MR1 (HI), DM(I0,M0) = AR;
               DM(I0,M0) = SR1;
               RTI;

.ENDMOD;
```

Listing 1. PID_CONTROLLER Routine

After the initial design of a digital PID controller, all coefficients must be scaled down by the same factor. This is necessary in order to conform to the 16-bit fixed-point fractional number format as well as to insure that overflows won't occur in the final stage of the multiply-accumulate operations. The scaled down coefficients are the ones that get stored in the processor's memory. The result of the multiply and accumulate operations is eventually scaled up before being output to the controlled system. The choice of a proper scaling factor depends greatly on the design objectives and in some cases it may even be unnecessary. The PID controller coefficients are usually designed with a commercial software package in higher precision arithmetic than 16 bits. System performance deviates from ideal when such high precision PID coefficients are quantized to 16 bits and further scaled down. In systems that require stringent PID specifications, careful simulations of quantization and scaling effects must be performed.

During the initialization for the PID routine, the scaling factor for the coefficients must be stored in the SE register. The index register I0 points to the circular data memory buffer that contains the previous error inputs and the previous PID output. This buffer must be initialized to zero at powerup unless some non-zero initial condition is desired. The index register I4, on the other hand, points to the circular program memory buffer that contains the scaled PID coefficients. These coefficients include a term "B" (for U(n-1)), which is equal to the value " 1/scaling factor ". This value is derived from the fact that the real coefficient for U(n-1) is " 1 " and that it must be scaled down along with the other coefficients. The order that these scaled coefficients are stored in program memory is: A2, A1, A0, B.

The PID core routine fetches the coefficients and data values from memory following the sequence that they have been stored. These values are multiplied and accumulated until all

of them are accessed. Note that both of the address generators are used in parallel with the multiply-accumulator throughout these operations. Finally, the data memory buffer is updated with the new samples and the output is obtained by scaling up the result of the multiplication and accumulation operations.

The PID routine executes in 8 instruction cycles on the ADSP-2101 and ADSP-2105. It executes in 12 instruction cycles on the ADSP-2100 and ADSP2100A. In the case that the initialization registers have been modified by other routines, it may be necessary to execute up to 7 overhead setup cycles before calling the core PID routine.

N'TH ORDER DIGITAL CONTROLLER DESIGN

There are several methods to design high order digital controllers. This section briefly outlines three approaches and cites some references on this topic. The three methodologies are "analog-controller-based digital design", "direct digital design" and "state-space design".

Analog-Controller-Based Digital Design

This is a very common way of designing digital controllers. In this method, an analog controller that satisfies the desired requirements is first created using well established analog design procedures. This controller is then transformed into the digital domain and implemented.

The analog controller design may be performed in the s-plane using common design techniques such as root-locus methods, Bode plots, the Routh-Hurwitz criterion, state variable techniques and other methods. The resulting analog transfer function is then transformed into a digital transfer function in the z-domain. Finally, the z-domain transfer function is inverse-z transformed into a difference equation that can be implemented on a digital processor.

The transformation from the s-domain to the z-domain can be accomplished using various techniques such as the matched pole-zero method, the bilinear (Tustin) transformation the method of mapping differentials, the impulse-invariance method, the step-invariance method, and the zero-order hold technique.

The most commonly used of these methods is the bilinear transformation. This transformation approximates the s-domain transfer function with a z-domain transfer function by use of the substitution:

$$s = (2/T)\,(z-1/z+1) \qquad (7)$$

Analog controllers that are in parallel or cascade maintain their respective structures after going through this transformation. The Tustin transformation maps the stable region of the s-plane exactly into the stable region of the z-plane although the entire jw-axis of the s-plane is stuffed into the 2π-length of the unit circle. Obviously a great deal of distortion takes place in the mapping in spite of the consistency of the stability regions. This distortion can be corrected by using a frequency pre-warping scheme. The pre-warping matches the single most important critical frequency in the analog domain and the digital domain. This method replaces each "s" in the analog transfer function with

$$(\omega 1 / \omega_2)\, s$$

where ω_1 is the frequency to be matched in the digital transfer function and with

$$\omega_2 = (2/T) \tan (\omega_1 T/2) \qquad (8)$$

Bilinear transformation with frequency pre-warping is one of the most commonly used analog based design techniques. The most significant drawback of this type of design is that the digital controller that results from it is only an approximation to the analog one. The analog controller puts an implicit upper bound on the digital controller's performance. More information on the other analog based controller design methods can be found in References 1 and 3.

Direct Digital Design

This method allows us to perform the control system design directly in the digital domain. Thus, the design can be carried out in the z-plane. The approximations and limitations that arise from starting in the s-domain and transforming into the z-domain are eliminated. Conventional design techniques can be used to place the closed-loop poles and zeros exactly where appropriate. Some of these z-domain techniques include the root-locus method, the pole-zero cancellation method and the w-transform. More detailed information on these methods can be found in References 1 and 3.

State-Space Design

The digital controller design methods discussed above are designated as classical design methods. Same design tasks can be accomplished by using a different set of techniques based on the state-space or modern control formulation. Modern control design methodology is especially advantageous when designing controllers for multi-input and multi-output systems. However, single-input and single-output systems that are discussed in this application note can also be efficiently designed using state-space methods. More detailed information on this topic can be found in References 1 and 3.

N'TH ORDER DIGITAL CONTROLLER STRUCTURES

Standard second order (N=2) digital controller implementation is directly analogous to IIR (Infinite Impulse Response) filter implementations. These second order controller blocks can be implemented as biquad second order IIR filter sections. A second order biquad section is shown on Figure 6 and its corresponding transfer function in the z-domain is given by:

$$G(z) = U(z)/E(z) = (B_0 + B_1 \cdot z^{-1} + B_2 \cdot z^{-2})/(1 + A_1 \cdot z^{-1} + A_2 \cdot z^{-2}) \qquad (9)$$

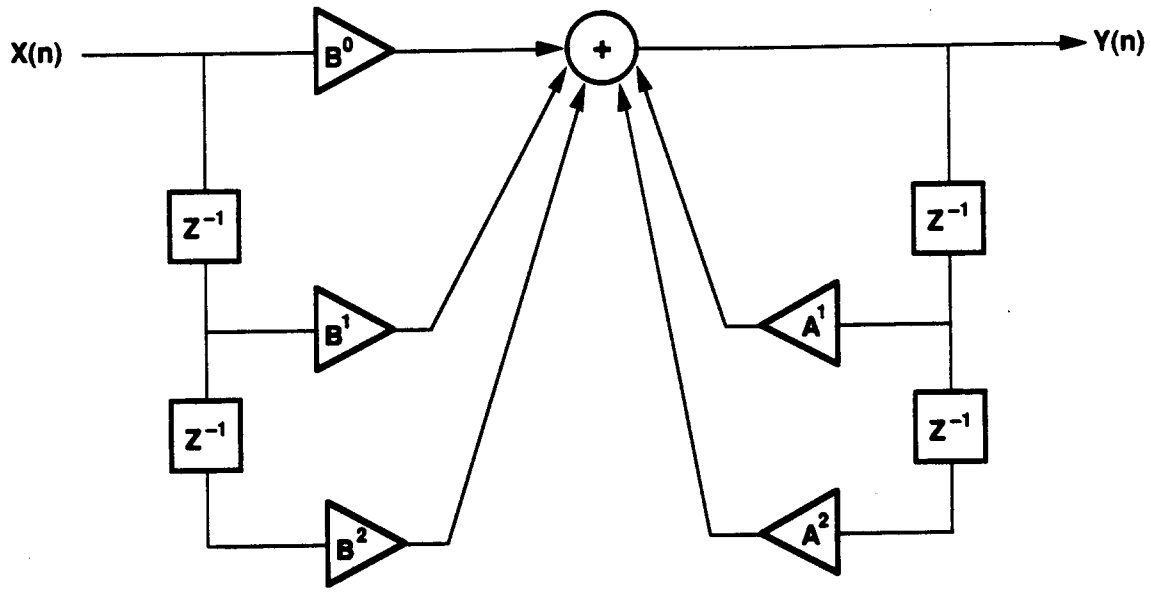


Figure 6. Second Order Biquad Structure

where A_1, A_2, B_0, B_1 and B_2 are coefficients that determine the desired impulse response of the system G(z). Furthermore, the corresponding difference equation for a biquad section is given by:

$$U(n) = B_0 \cdot E(n) + B_1 \cdot E(n-1) + B_2 \cdot E(n-2) - A_1 \cdot U(n-1) - A_2 \cdot U(n-2) \quad (10)$$

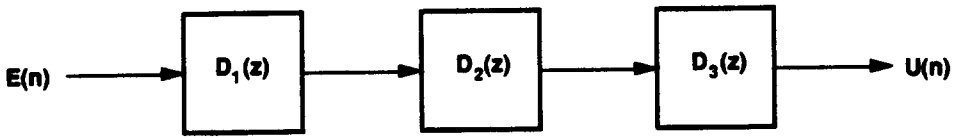
Higher order (N'th order) controllers can be obtained by cascading several biquad sections with appropriate coefficients. An example is shown on Figure 7 where three biquad sections are cascaded to construct the overall G(z) transfer function. Another way to design higher order controllers is to use only one complicated single section. This approach is also called the direct form implementation. The block diagram of a direct form fourth order controller is shown on Figure 8 as an example.

The direct form implementation executes faster but generates larger numerical errors than the biquad implementation. The biquads can be scaled separately and then cascaded in order to minimize the coefficient quantization and the recursive accumulation errors. The coefficients and data in the direct form implementation must be scaled all at once, which gives rise to larger errors. Another disadvantage of the direct form implementation is that the poles of such single stage high order polynomials get increasingly sensitive to quantization errors. The second order polynomial sections (i.e. biquads) are less sensitive to quantization effects.

N'TH ORDER CONTROLLER IMPLEMENTATION

An ADSP-2100 Family assembly language subroutine that implements a high order controller is shown in Listing 2. The subroutine is arranged as a module and is labeled BIQUAD_CONTROLLER. There are a number of registers that need to be initialized in order to execute this subroutine. It may be sufficient to do this initialization only once (e.g. on powerup) if other executed algorithms do not need these registers. In most typical cases, however, some of these registers may need to be set every time the BIQUAD_CONTROLLER subroutine is called. It may sometimes be beneficial, from a modular software point of view, to always initialize all the setup registers as a part of this subroutine.

The BIQUAD_CONTROLLER routine in Listing 2 takes its input from the SR1 register. This register must contain the 16



$$G(z) = \frac{U(z)}{E(z)} = D_1(z) \cdot D_2(z) \cdot D_3(z)$$

Figure 7. Cascased Biquad Sections

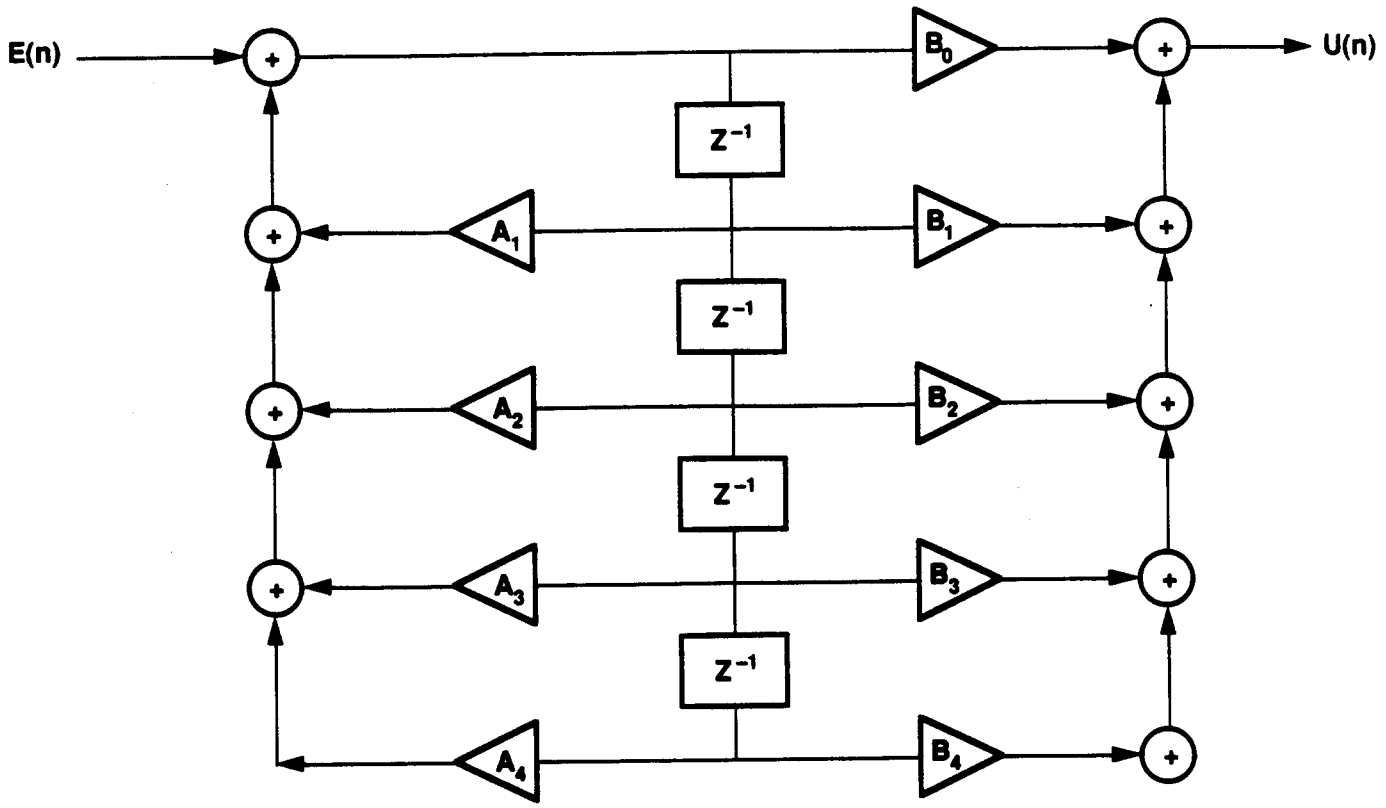


Figure 8. Fourth Order Direct Form Controller

bit error input E(n). E(n) is assumed to be already computed before this subroutine was called. The output of the controller is also made available in the SR1 register.

After the initial design of a high order controller, all coefficients must be scaled down in each biquad stage separately. This is necessary in order to conform to the 16 bit fixed point fractional number format as well as to insure that overflows won't occur in the final multiply-accumulate operations in each stage. The scaled down coefficients are the ones that get stored in the processor's memory. The operations in each biquad are performed with scaled data and coefficients and are eventually scaled up before being output to the next one. The choice of a proper scaling factor depends greatly on the design objectives and in some cases it may even be unnecessary. The controller coefficients are usually designed with a commercial software package in higher precision arithmetic than 16 bits. System performance deviates from ideal when such high precision coefficients are quantized to 16 bits and further scaled down. In systems that require stringent specifications, careful simulations of quantization and scaling effects must be performed.

During the initialization of the BIQUAD_CONTROLLER routine, the index register I0 points to the data memory buffer that contains the previous error inputs and the previous biquad section outputs. This buffer must be initialized to zero at powerup unless some non-zero initial condition is desired. The index register I1 points to another buffer in data memory that contains the individual scale factors for each biquad. The buffer length register L1 is set to zero if the controller has only one biquad section. L1 is initialized with the number of biquad sections in the case of multiple biquads. The index register I4, on the other hand, points to the circular program memory buffer that contains the scaled biquad coefficients. These coefficients are stored in the order : B2, B1, B0, A2, A1 for each biquad. All of the individual biquad coefficient groups must be stored in the same order that the biquads were cascaded in such as: B2, B_1, B_0, A_2, A_1, B_2^*, B_1^*, B_0^*, A_2^*, A_1^*, B_2^{**}...etc The buffer length register L4 must be set to the value given by: (5 x number of biquad sections). Finally, the loop counter register "CNTR" must be set to the number of biquad sections since the controller code will be executed as a loop.

The core of the BIQUAD_CONTROLLER routine starts its execution at the "biquad" label. The routine is organized in a looped fashion where the end of the loop is the instruction labeled "sections". Each iteration of the loop executes the computations for one biquad. The number of loops to be executed is determined by the CNTR register contents. The SE register is loaded with the appropriate scaling factor for the particular biquad at the beginning of each loop iteration. After this operation, the coefficients and the data values are fetched from memory in the sequence that they have been stored. These numbers are multiplied and accumulated until all of the values for a particular biquad have been accessed. The result of the last multiply accumulate is rounded to 16 bits and upshifted by the scaling value. At this point the "biquad" loop is executed again or the controller computations are completed by doing the final update to the delay line. The delay lines for data values are always being updated within the biquad loop as well as outside of it.

```
.MODULE          BIQUAD_CONTROLLER;

{   This is an Nth order cascaded biquad controller subroutine

                 Calling Parameters:

                       SR1= error input E(n),  [ E(n) = Y(n) - R(n) ]
                       I0 -> delay line buffer for E(n-2), E(n-1), Y(n-2), Y(n-1)
                       L0 = 0
                       I1 -> scaling factors for each biquad section
                       L1 = 0  (in the case of a single biquad)
                       L1 = number of biquad sections (for multiple biquads)
                       I4 -> scaled biquad coefficients
                       L4 = 5 x [number of biquads]
                       M0,M4 = 1
                       M1 = -3
                       M2 = 1 (in the case of multiple biquads)
                       M2 = 0 (in the case of a single biquad)
                       M3 = (1 - length of delay line buffer)

                 Return Value:
                       SR1 = output sample U(n)

                 Altered Registers:
                       SE, MX0, MX1, MY0, MR, SR

                 Computation Time (with N even):
                       ADSP-2101/2102:        (8 x N/2) + 5 instruction cycles
                       ADSP-2100/2100A:       (8 x N/2) + 5 + 5 instruction cycles

                 All coefficients and data values are assumed to be in 1.15 format
}

.ENTRY           CONTROLLER;

BIQUAD:          CNTR = number of biquads
                 DO SECTIONS UNTIL CE;
                       SE = DM(I1,M2);
                       MX0 = DM(I0,M0), MY0 = PM(I4,M4);
                       MR = MX0*MY0(SS), MX1 = DM(I0,M0), MY0 = PM(I4,M4);
                       MR = MR+MX1*MY0(SS), MY0=PM(I4,M4);
                       MR = MR+SR1*MY0(SS), MX0 = DM(I0,M0), MY0 = PM(I4,M4);
                       MR = MR+MX0*MY0(SS), MX0 = DM(I0,M1), MY0 = PM(I4,M4);
                       DM(I0,M0) = MX1, MR = MR+MX0*MY0(RND);
SECTIONS:              DM(I0,M0) = SR1, SR = ASHIFT MR1 (HI);
                 DM(I0,M0) = MX0;
                 DM(I0,M3) = SR1;
                 RTS;
.ENDMOD;
```

Listing 2. BIQUAD_CONTROLLER Routine

The controller coefficients must be scaled appropriately so that no overflows occur after the upshifting operation between the biquads. If this is not insured by design, it may be necessary to include some overflow checking between the biquads.

The execution time for an N'th order BIQUAD_CONTROLLER routine can be calculated as follows (assuming that the appropriate registers have been initialized and N is a power of 2):

ADSP2101/2105 : (8 x N/2) + 4 processor cycles
ADSP2100/2100A : (8 x N/2) + 4 + 5 processor cycles

It may take up to a maximum of 11 cycles to initialize the appropriate registers every time the controller is called. But typically this number will be lower in most applications.

NOTCH FILTER EXAMPLE FOR THE ADSP-2100A

A fully coded ADSP-2100A notch filter program is presented in this section. The program executes two cascaded biquad sections and is designed to run standalone on an ADSP-2100A. The processor reads the input samples from a data memory mapped A/D converter and sends the filtered output to a data memory mapped D/A converter. The operation of the ADSP-2100A is interrupt driven. The occurrence the IRQ0 interrupt notifies the processor that there is a new sample ready at the A/D converter output. The ADSP-2100A normally waits in a wait loop, processes the incoming samples upon an interrupt and returns to the wait loop again. This process starts with a reset or powerup and repeats until a powerdown or another reset occurs. The full code of the notch filter is shown in Listing 3.

The program starts with some variable, constant and port declarations. These declarations allow the program to refer to specific memory addresses symbolically. This greatly eases the software maintenance and debugging tasks at the assembly level. In order to run this program, the NOTCH_FILTER assembly module shown in Listing 3 must be first assembled using the ADSP-2100 assembler. It must then be linked with an architecture description file that was built using the ADSP-2100 system builder. This architecture file

```
{THIS IS AN ADSP2100A ASSEMBLY PROGRAM THAT EXECUTES A FOURTH}
{ORDER NOTCH FILTER IN A CASCADED BIQUAD IMPLEMENTATION}

.MODULE/RAM             NOTCH_FILTER;           {The name of the module}

.VAR/PM/CIRC            COEFFICIENTS[10];       {These are the declarations for}
.VAR/DM                 DATA_BUFFER[6];         {data and program memory buffers}
.VAR/DM/CIRC            SCALE_FACTORS[2];

.PORT                   AD_CONVERTER;           {There is one input port and one}
.PORT                   DA_CONVERTER;           {output port in the system}

.INIT                   COEFFICIENTS: <COEFF.DAT>;    {The memory buffers}
.INIT                   DATA_BUFFER: <INITIAL.DAT>;   {are initialized here}
.INIT                   SCALE_FACTORS: <SCALE.DAT>;

                JUMP BIQUAD;                    {Interrupt vector for IRQ0}
                RTI;
                RTI;
                RTI;

                I0 = ^DATA_BUFFER;              {These are the proper initializations}
                L0 = 0;                         {for the index, length and modify}
                I1 = ^SCALE_FACTORS;            {registers to be used}
                L1 = %SCALE_FACTORS;
                I4 = ^COEFFICIENTS;
                L4 = %COEFFICIENTS;
                M0 = 1;
                M1 = -3;
                M2 = 1;
                M3 = -5;
                M4 = 1;
                ICNTL = B#00001;                {Set IRQ0 to be edge sensitive}
                IMASK = B#0001;                 {Enable IRQ0 interrupt only}
WAIT:           JUMP WAIT;                      {Wait for IRQ0 interrupt to occur}
```

```
{The interrupt service routine below executes the two biquad sections of the filter}

BIQUAD:          SR1=DM(AD_CONVERTER);          {Read the A/D converter}
                 CNTR = %SCALE_FACTORS;
                 DO SECTIONS UNTIL CE;
                       SE = DM(I1,M2);
                       MX0 = DM(I0,M0), MY0 = PM(I4,M4);
                       MR = MX0*MY0(SS), MX1 = DM(I0,M0), MY0 = PM(I4,M4);
                       MR = MR+MX1*MY0(SS), MY0=PM(I4,M4);
                       MR = MR+SR1*MY0(SS), MX0 = DM(I0,M0), MY0 = PM(I4,M4);
                       MR = MR+MX0*MY0(SS), MX0 = DM(I0,M1), MY0 = PM(I4,M4);
                       DM(I0,M0) = MX1, MR = MR+MX0*MY0(RND);
SECTIONS:              DM(I0,M0) = SR1, SR = ASHIFT MR1 (HI);
                 DM(I0,M0) = MX0;
                 DM(I0,M3) = SR1;
                 DM(DA_CONVERTER) = SR1;        {Send the filtered output to the D/A}
                 RTI;                           {Return to the wait loop}
.ENDMOD;
```

Listing 3. NOTCH_FILTER Routine

should be specific to the particular hardware configuration that the ADSP-2100A is being built in. The assembly module also needs to be linked with three data files along with the architecture file. The first data file must contain the scale factors, the second one must contain the scaled coefficients and the third one must contain the initial values for the delay taps of the filter.

ADSP-2100 Family Benchmarks For Digital Control Applications

	ADSP-2101 (60 ns)	ADSP-2105 (100 ns)
PID Loop	0.48 µs	0.8 µs
FIR Filter	60 ns/tap	100 ns/tap
IIR Biquad – 16 Bit (2nd Order)	0.72 µs	1.2µs
IIR Biquad – 16 Bit (nth Order)	4N + 4 cycles	4N + 9 cycles
IIR Biquad – 32 Bit (2nd Order)	2.64 µs	4.4 µs
IIR Biquad – 32 Bit (nth Order)	16N + 12 cycles	16N + 22 cycles
256-Point FFT (Complex)	0.405 ms	0.675 ms
Matrix Multiply (3x3 * 3x1)	3.12 µs	5.2 µs
Stochastic Gradient (LMS) N Tap Coefficient Update	2N + 9 cycles	2N + 9 cycles

REFERENCES

(1) Franklin, G.F., J.D. Powell and M.L. Workman 1990. "Digital Control of Dynamic Systems". Reading, MA: Addison-Wesley Publishing Company.

(2) Oppenheim, A. V. and A. Willsky 1983. "Signals and Systems". Englewood Cliffs, NJ: Prentice-Hall, Inc.

(3) Borrie, J. A. 1986. "Modern Control Systems". Englewood Cliffs, NJ: Prentice-Hall Inc.

(4) Kazanzides, P. 1985. "A Microprocessor Based Control System with Robotics Applications". Brown University, LEMS, Providence, RI.