Control System Compensation and Implementation with the TMS32010

APPLICATION REPORT: SPRA009

Authors:
Charles Slivinsky
Department of Electrical Engineering
University of Missouri – Columbia

Jack Borninski
Digital Signal Processing – Semiconductor Group

Digital Signal Processing Solutions
1989

TEXAS INSTRUMENTS

TRADEMARKS

TI is a trademark of Texas Instruments Incorporated.

Other brands and names are the property of their respective owners.

CONTACT INFORMATION

US TMS320 HOTLINE	(281) 274-2320
US TMS320 FAX	(281) 274-2324
US TMS320 BBS	(281) 274-2323
US TMS320 email	dsph@ti.com

Control System Compensation and Implementation with the TMS32010

Abstract

This application report describes the design of a digital control system and its implementation with the TMS32010. Because of the increased availability and lower cost of suitable digital hardware, microprocessors such as the TMS32010 are increasingly being used to implement algorithms for the control of feedback systems. This report provides an example that uses the design of a digital compensator. Other control applications are possible using the TMS32010, such as computer disk control, laser print-head control, robotic control, automobile-engine system control, flight control, and autopilot control systems.

Product Support on the World Wide Web

Our World Wide Web site at www.ti.com contains the most up to date product information, revisions, and additions. Users registering with TI&ME can build custom information pages and receive new product updates automatically via email.

INTRODUCTION

Algorithms, software, and hardware for designing and implementing a digital control system using the Texas Instruments TMS32010 signal-processing microprocessor are presented in this application report. Microprocessors, such as the TMS32010, are increasingly being used to implement algorithms for the control of feedback systems. The major factors contributing to this trend are increased availability and lower cost of suitable digital hardware.

Current and potential applications include servo motor control, process control, robot arm and disk head controllers, and temperature and pressure controllers. Military and aerospace applications include stabilized platforms, flight control and autopilot systems, inertial reference systems, and general servomechanisms.

In meeting control system requirements, designers face many alternatives. Cost, size, weight, power, and reliability decisions are typically application dependent. This report highlights the tradeoffs in developing algorithms, software, and hardware for a digital control system.

DIGITAL CONTROL SYSTEMS

General Considerations

A digital controller is a signal processing system that executes algebraic algorithms inherent to the control of feedback systems (i.e., compensator and filter algorithms). Together with the plant (system to be controlled) and signal-acquisition circuitry, the digital controller makes up a digital control system such as the one shown in Figure 1.

Note that the system requires analog-to-digital (A/D) converters for the external (command) inputs and for the state-variable feedback inputs to the digital controller. The system also requires a digital-to-analog (D/A) converter for the control outputs to the plant.

The advantages of the digital control approach over the analog approach are:

1. Ability to implement advanced control algorithms with software rather than special-purpose hardware
2. Ability to change the design without changing the hardware
3. Reduced size, weight, and power, along with low cost
4. Greater reliability, maintainability, and testability
5. Increased noise immunity.

Microprocessor Selection and System Development Cycle

Choosing an appropriate microprocessor is an important factor in efficiently implementing a digital control design. A class of special-purpose (as opposed to general-purpose) digital signal-processing microprocessors has been developed to enable fast execution of digital control algorithms. The Texas Instruments TMS32010 provides several beneficial features for implementing digital control system elements through its architecture, speed, and instruction set.

A prominent feature of the TMS32010 is the on-chip, 16×16-bit multiplier that performs two's-complement multiplication and produces a 32-bit product in a single 200-ns instruction cycle. The TMS32010 instruction set includes special instructions necessary for fast implementation of sum-of-products computations encountered in digital filtering/compensation and Fourier transform calculations. Most of the instructions critical to signal processing execute in one instruction cycle. References [1,2] give full details of the TMS32010 hardware and software considerations.

Many system development tools are available and may be used for digital control system design.[3] Figure 2 outlines

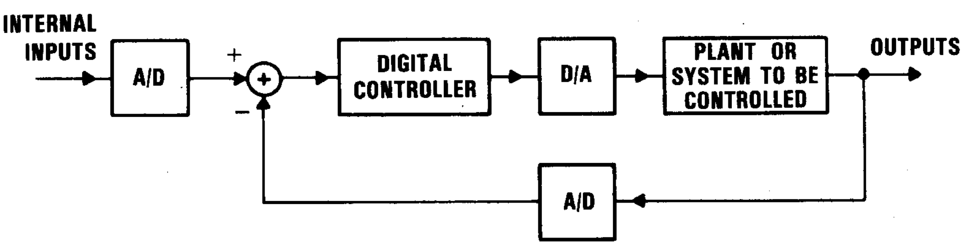

Figure 1. Digital Control System

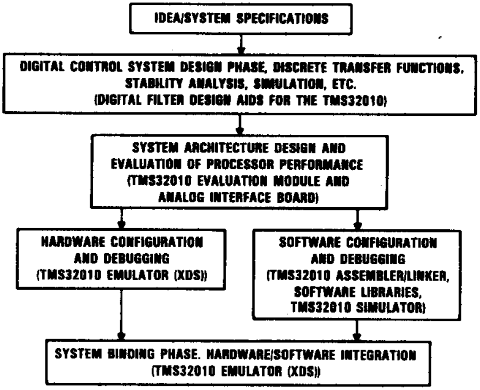

Figure 2. Digital Control System Development Phases

the system development cycle and ties the development tools to different project phases.

Among the non-TI development aids, an interactive program called the Digital Filter Design Package developed by Atlanta Signal Processors Incorporated, may be useful in digital control system design. This program designs various types of filters, compensators, and other structures. It can be used when, for example, there is a need for several notch filters to filter out unwanted frequencies in a digital feedback loop.

DIGITAL COMPENSATOR DESIGN

Alternative methods exist for designing digital compensators. This section outlines several approaches to digital controller design and points out the analytical tools useful in the design process.

Design Based on Analog Prototype

A commonly used method of designing a digital control system is to first design an equivalent analog control system using one of the well-known design procedures. The resulting analog controller (analog prototype) is then transformed to a digital controller by the use of one the transformations described below.

The design of the analog controller may be carried out in the s-plane using design methods such as root-locus techniques, Bode plots, the Routh-Hurwitz criterion, state-variable techniques, and other graphic or algebraic methods. The purpose is to devise a suitable analog compensator transfer function which is transformed to a digital transfer function. This digital transfer function is then inverse z-transformed to produce a difference equation that can be implemented as an algorithm to be executed on a digital computer. Two of the analog-to-digital transformation methods, the matched pole-zero and the bilinear transformation, are described as follows:

1. The matched pole-zero (matched Z-transform) method maps all poles and zeroes of the compensator transfer function from the s-plane to the z-plane according to the relation:

   $$z = e^{sT}$$

   where T is the sampling period.

   If more poles than zeroes exist, additional zeroes are added at $z = -1$, and the gain of the digital filter is adjusted to match the gain of the analog filter at some critical frequency (e.g., at DC for a lowpass filter). This method is somewhat heuristic and may or may not produce a suitable compensator.
2. The bilinear (Tustin) transformation method approximates the s-domain transfer function with a z-domain transfer function by use of the substitution:

   $$s = \frac{2}{T}\ \frac{z-1}{z+1}$$

   As in the matched pole-zero method, the bilinear transformation method requires substitution for s. Compensators in parallel or in cascade maintain their respective structures when transformed to their digital counterparts. This substitution maps low analog frequencies into approximately the same digital frequencies, but produces a highly nonlinear mapping for the high frequencies.

   To correct this distortion, a frequency prewarping scheme is used before the bilinear transformation. The frequency prewarping operation results in matching the single critical frequency between the analog domain and the digital domain. To achieve this result, the prewarping operation replaces each s in the analog transfer function with (ω_o/ω_p)s where ω_o is the frequency to be matched in the digital transfer function and

   $$\omega_p = \frac{2}{T}\tan\frac{\omega_o T}{2}$$

Bilinear transformation with frequency prewarping provides a close approximation to the analog compensator[5] and is the most commonly used technique. Other methods for converting a transfer function from the analog to the digital domain are: the method of mapping differentials[18], the impulse-invariance method[6], the step-invariance method[18], and the zero-order hold technique.[6]

The basic disadvantage of design based on an analog prototype is that the discrete compensator is only an approximation to the analog prototype. This analog prototype is an upper bound on the effectiveness of the closed-loop response of the digital compensator.

Direct Digital Design

The direct digital design technique is a design method where a digital control system design is carried out from the very beginning in the z-domain to produce a digital control algorithm. Various pole-placement and relocation techniques are used to position the poles and zeroes of the system's z-domain transfer function to yield the required system performance.

The root-locus method in the z-plane is similar to the root-locus design in the s-plane in that both are based on observing the position of the closed-loop poles as a function of the system gain. However, the effect of the locations of the poles and zeroes on system performance is not the same as in the s-plane. Knowledge of such correspondences in the z-plane allows those familiar with continuous system design to design digital compensators. For example, in stabilizing an unstable system, the adjustable gain is used to move the poles inside the unit circle instead of inside the left-half plane. A phase-lead controller may be employed to shift the root-locus to the left, which results in a system that responds faster. A phase-lag controller may be designed to allow for higher loop gain to produce smaller steady-state errors and improved disturbance rejection.[10]

The z-domain designer may also use pole-zero cancellation. In this technique, some of the poles and zeroes of the digital transfer function of the plant may be cancelled by zeroes and poles of the digital compensator. The compensator then introduces additional poles and zeroes at locations that enable the designer to achieve the desired performance characteristics. This method may affect system stability due to "inexact cancellation". If the poles of the plant that are to be cancelled lie close to the unit-circle, inexact cancellation may cause the system root-locus to go outside the unit circle at some point. This makes the system conditionally stable or unstable.

In a system where fast response to the control input is required, a "deadbeat" approach may be taken. The deadbeat controller cancels all the zeroes and poles of the plant and introduces a pole at $z = 1$. The result is that the system output reaches its steady-state value in one sampling period with no overshoot. In practice, an ideal deadbeat controller is difficult to implement because of inexact cancellation. Although the output does not experience overshoot, it may oscillate between sampling instants. The design is "tuned" in the sense that the system response may be acceptable for a step input but not acceptable for other inputs.

With z-plane design, conventional design techniques can be used to place the closed-loop system poles exactly where desired. The approximations associated with digitizing an analog prototype are thereby eliminated. The disadvantage of this technique is the relative difficulty of visualizing the effect of pole-zero locations in the z-plane on system performance. To overcome the disadvantage of the z-plane technique, the designer can use the w-plane or, better yet, the w′-plane design technique.[5] Both techniques transform the design to a plane similar to the s-plane by means of the same kind of substitution as in the bilinear transformation described earlier. This procedure thereby allows the use of the familiar s-plane and frequency-domain methods of continuous-system design. The designer proceeds by first transforming the continuous plant to the z-domain and thence to the w-plane or w′-plane. The appropriate compensator is then devised and transformed back to the z-plane, where it is used to specify the corresponding computational algorithm.

State-Variable Design Methods

State-variable design methods can also be used, including state-variable feedback and optimal control based on quadratic synthesis.

In the state-variable feedback technique, all of the states are measured and fed back through constant gains. This allows all of the closed-loop poles to be positioned at any desired locations in the z-plane, but does not affect the positions of the system zeroes.

In a design based on quadratic synthesis, a performance index or cost function is minimized by proper choices of the control law or feedback compensator. In the most practical design, the resulting compensator has the same form as that resulting from the application of direct digital design techniques.[7]

DIGITAL COMPENSATOR IMPLEMENTATION

Digital compensator algorithms execute on processors that use finite-precision arithmetic. The signal-quantization errors associated with finite-precision computations and the methods for the handling of these errors are presented in this section.

Fixed-Point Arithmetic and Scaling

Computation with the TMS32010 is based on the fixed-point two's-complement representation of numbers. Each 16-bit number has a sign bit, i integer bits, and 15-i fractional bits. For example, the decimal fraction +0.5 may be represented in binary as

0.100 0000 0000 0000

This is Q15 format since it has 15 fractional bits, one sign bit, and no integer bits. The decimal fraction +0.5 may equivalently be represented in Q12 format as:

0000.1000 0000 0000

This number is in the Q12 format because it has 12 fractional bits, one sign bit, and three integer bits. Note that the Q15 notation allows higher precision while the Q12 notation allows direct representation of larger numbers.

For implementing signal-processing algorithms, the Q15 representation is advantageous because the basic operation is multiply-accumulate and the product of two fractions remains a fraction with no possible overflows during multiplication. When the Q12 format is used, a software check for overflow is necessary. The subsection, OVERFLOW AND UNDERFLOW HANDLING, provides a detailed analysis of overflow handling.

In the case where two numbers in Q15 are multiplied, the resulting product has 30 fractional bits, two sign bits, and (as expected) no integer bits. To store this product as a 16-bit result in Q15, the product must be shifted left by one bit and the most-significant 16 bits stored. The TMS32010 instruction SACH allows for this one-bit shift.

In the case where a Q15 number is to be multiplied by a 13-bit fractional signed constant represented as a Q12 number, the result (to correspond with Q15) must be left-shifted four bits to maintain full precision. The TMS32010 instruction SACH allows for the appropriate shift. The Q15 and Q12 representations are used in the example in the section, DESIGN EXAMPLE: RATE-INTEGRATING GYRO STABILIZATION LOOP.

When fixed-point representations are used, the control system designer must determine the largest magnitudes that can occur for all variables involved in the computations required by the digital compensator. (Floating-point representations allow larger magnitudes, but take more time for the microprocessor to perform the required computations.) Once these largest magnitudes are known, scaling constants can be used to attenuate the compensator input as much as necessary to ensure that all variables stay within the range that can be expressed in the given representation.

Several methods are used to determine bounds on the magnitudes of the variables. One method, called upper-bound scaling, provides a useful, although sometimes too conservative, bound on the magnitude, yet it is straightforward to calculate. Consider a variable y(n) that is obtained as the output of a digital compensator H(z) when the input is the sequence x(n). The bound on y(n) is given by

$$y_{max} = |x_{max}| \sum_{n=1}^{\infty} |h(n)|$$

where x_{max} is the maximum value in x(n) and the sequence h(n) is the unit-sample response sequence for the digital compensator H(z).

Other methods for estimating the upper bound are L_p-norm scaling, unit-step scaling, and the averaging method.[9,10]

After y_{max} is determined, the scale factor can be chosen as the multiplier that is applied to x(n) prior to the compensator computations to ensure that y(n) remains within the required bounds. In addition, the control system designer may have knowledge concerning the bounds on the compensator variables based on prior experience, the characteristics of the corresponding variables of analog prototypes, and simulation results.

Finite-Wordlength Effects

All variables involved in the digital compensator — the input, the compensator coefficients, the intermediate variables, and the output — are represented as finite-wordlength numbers. This restriction gives rise to errors. Another source of errors is the truncation or rounding that takes place when the 32-bit product of two 16-bit numbers is stored as a 16-bit number. Both of these errors give rise to the finite-wordlength effects discussed in this section.

The representation of the compensator input as a finite-precision (quantized) number produces an input-quantization error. The size of this error for a rounding scheme can be anywhere from $-(2^{-B})/2$ to $(2^{-B})/2$ where B is the number of bits in a word. The input-quantization error is usefully modeled as a zero-mean random variable uniformly distributed between its positive and negative bounds. A technique[5,10] is available to calculate the variance of the corresponding error at the compensator output (its mean is zero). In this manner, the designer can determine the effect of input quantization on the compensator output.

Similar quantization errors are associated with the multiplication process. Each multiplication is assumed to produce the "true" product with an error that is a zero-mean, uniformly distributed random variable. The variance of the corresponding error at the compensator output can be calculated in the same manner as for the error due to input quantization. These individual variances are then added to measure the total effect at the compensator output for each truncation or rounding.

Another way to describe the effects of truncation or rounding is in terms of "limit cycles" which are sustained oscillations in the closed-loop system. These oscillations are caused by nonlinearities within the loop. In this case, the nonlinear quantizations are associated with the multiplications. Limit cycles persist even when the system input goes to zero, and their amplitude can be sizeable. No general theory is available to treat this nonlinear phenomenon. Bit-level simulations which model the compensator and the complete closed-loop system are used to ascertain their presence and effect on the closed-loop performance.

When a digital compensator is implemented as an algorithm to be executed on finite-precision hardware, a problem arises with implementing the coefficients present in the corresponding transfer function (see section, TMS32010 IMPLEMENTATION OF COMPENSATORS AND FILTERS). The infinite-precision compensator coefficients must be rounded and stored using a finite-length, fixed-point binary representation. Due to this coefficient-quantization effect, the performance of the implemented filter will deviate from the performance of the designed digital filter.

The deviation in performance can be estimated by computing the filter's pole and zero locations and the corresponding frequency response magnitude and phase for the compensator with the quantized coefficients. Coefficient quantization forces the filter's poles and zeroes into a finite number of possible locations in the z-plane and is of most concern for filters with stringent specifications, such as narrow transition regions.

The designer must choose the filter structure least sensitive to inaccurate coefficient representation. The choice should be of a modular rather than a direct filter structure. For example, a higher-order filter should be implemented as a cascade or parallel combination of first-order and second-order blocks. The reason for this choice is the lesser sensitivity to coefficient variations of the roots of low-degree polynomials in comparison with high-degree polynomials. Several methods for selecting the filter structures least affected by coefficient quantization are available.[9]

To quantitatively evaluate the effect of coefficient quantization on the position of the poles or zeroes of a digital transfer function, a "root sensitivity function" can be computed.[6]

Overflow and Underflow Handling

Digital control system algorithms are usually implemented using two's-complement, fixed-point arithmetic. This convention designates a certain number of integer and fractional bits. The fixed-point arithmetic computations may, at some point, produce a result that is too large to be represented in a chosen form of fixed-point notation (e.g., Q12). The resulting overflow, if untreated, may cause degraded performance such as limit cycles and large noise spikes at the filter's output which may contribute to the system's instability. The system must be able to recover from the overflow condition, i.e., return to its normal, nonoverflow state.

Consider an example of the Q12 representation. The number 7.5 multiplied by itself gives the result of 56.25, an overflow in Q12. However, no hardware overflow occurs in the accumulator; i.e.,

```
      7.5   0111.1000 0000 0000
  ×   7.5   0111.1000 0000 0000
  -------------------------------------------------
    56.25   0011 1000.0100 0000 0000 0000 0000 0000
```

For the Q12 representation, the above 32-bit product is shifted left four bits and the left-most 16 bits are retained:

1000.0100 0000 0000

The correct answer is 56.25, but the number stored in the Q12 representation is -7.75.

The TMS32010 has a built-in overflow mode of operation that, if enabled, causes the accumulator to saturate upon detection of an overflow during addition when the accumulator register overflows. During multiplication, an overflow of the fixed-point notation may also occur even though the hardware overflow of the accumulator register does not occur. This is because the 32-bit result of a multiplication of two 16-bit numbers must be stored in a 16-bit memory word in the form consistent with the chosen fixed-point notation (see above example). To adjust the location of the binary point, the storing operation requires that the number in the accumulator be shifted left and truncated on the right before storing. If the most significant bits shifted out contain magnitude information in addition to sign information, an overflow in the chosen fixed-point notation results.

To track overflows associated with the number representation, the control system software should contain an appropriate overflow-checking routine in those places where multiplications and additions occur. This routine should not rely exclusively on the TMS32010's overflow mode to intercept and correct the overflow occurrences.

Two approaches may be used to handle overflows. The first is to prevent the overflow from occurring by choosing conservative scaling factors for the numbers used in computations, as described in the subsection, FIXED-POINT ARITHMETIC AND SCALING. These scaling factors are used to limit the range of inputs to each of the basic building blocks of the compensator, namely, the first- and second-order filter sections. The scaling factor chosen reduces the input magnitude and consequently all other signal levels, thereby enabling the compensator coefficients, the expected inputs, and their products and sums, all to be represented without overflow. The scaling must also maintain the signal levels well above the quantization noise.

The second approach for handling overflow is to adjust the sum or product each time an overflow occurs. To accomplish this, an overflow checking routine must be written and executed at certain points along the computational path. The routine must check whether the number just computed and residing in the 32-bit accumulator can be stored without overflow in a 16-bit memory location in accord with the chosen fixed-point notation. Once the routine detects an overflow condition, it should replace the computed number with the maximum or minimum representable two's-complement number. This scheme simulates a saturation condition present in analog control systems. To prevent overflow limit cycles, the saturation overflow characteristic is preferred to the two's-complement, "wrap-around" characteristic.[9]

An example of the overflow checking and correcting technique for a first- and second-order filter subroutine is provided in the section, TMS32010 IMPLEMENTATION OF COMPENSATORS AND FILTERS. This Direct-Form II implementation subroutine checks for overflow occurrences upon computation of the filter's intermediate state variable and again upon computation of the filter's output.

In a digital control system, the first- and second-order building blocks are either cascaded or connected in parallel to compute a series of control algorithms. The first- and second-order filter subroutine, called to compute each of the control system's elements, uses 16-bit memory locations as storage media for its intermediate values, in which case it is appropriate to check for overflow in each block.

At the end of a computational chain — before the final, computed digital output is ready for transfer to the analog domain — it is necessary to check that the number being sent to the digital-to-analog converter is within the range based on the manner in which the converter is interfaced to the processor data bus. For example, if a 12-bit converter is wired to the 12 least significant bits (LSBs) of the 16-bit processor data bus, then the 12 LSBs must contain both magnitude and sign information, which may require that the original 16-bit number be adjusted or limited before being sent to the converter.

Underflow conditions, which can also appear during digital control algorithms, are conceptually similar to overflows in that the computed value contained in the 32-bit accumulator is too small to be accurately represented in a 16-bit memory word in the chosen fixed-point notation. One possible solution to this problem is to multiply the small result by a gain constant to raise its value to a representable level. The appropriately chosen gain constant may come as a result of gain distribution throughout the digital control system, whereby large gains from some of the building blocks get uniformly distributed over a range of the system's sections.

DIGITAL CONTROL SYSTEM SOFTWARE DESIGN PHILOSOPHY

To maximize the manageability and portability of the system software, a modular or top-down design technique should be used. This section shows how the modular software structure and the proper layout of system memory contribute to the efficient implementation of a digital control design.

Modular Software Structure

The concept of modular software design is a technique developed to make system software more manageable and portable. Top-down design is used to break up a large task into a series of smaller tasks or building blocks, which in turn are used for structuring a total system in a level-by-level form. At the end of a top-down design process, a number of modules are linked together which, under the control of a main program, perform as a complete system.

In addition to making the software-development and software-modification processes more manageable, modular design also enhances software portability. Digital control systems use a number of standard functional blocks such as compensators, notch filters, and demodulators. It is therefore likely that a designer who already has access to one digital control system will want to "borrow" some of its functional building blocks to quickly implement a new, different control unit or reconfigure the existing one. The designer who has access to these functional blocks or modules needs only modify the main program by providing a different sequence of subroutine calls. An initialization routine, a first- and second-order filter routine, a roundoff routine, and an overflow checking routine are examples of functional building blocks.

Each software module is written as a subroutine with a clear and efficient interface (for parameter passing, stack use, etc.) with the main program. In order to maintain the general-purpose function of the module, the data used in computations within a module (i.e., filter coefficients, state-variable values, etc.) should be accessed using indirect addressing rather than direct addressing. Only those variables whose values remain unchanged should be addressed directly.

Layout of TMS32010 Data Memory

The layout of the TMS32010 data memory in a digital control system implementation should be defined in accordance with the requirements of the software modules used in the implementation of the system. The procedure is illustrated by the first- and second-order filter subroutine of the section, TMS32010 IMPLEMENTATION OF COMPENSATORS AND FILTERS. This subroutine manipulates its pointer registers so that upon completion of the computations in one filter section, the registers automatically point to the set of coefficients and state variables of the next filter section.

If the software designer arranges his filter coefficient and state-variable sections in the order of execution of the control-system algorithms, a sequence of compensators and filters may be executed with a single subroutine call for each element. This scheme enables faster execution of the control algorithms since there is no need to explicitly reload the pointers in order to match the requirements of the current software module being called.

A designer must define all of the data memory locations (set up a system memory map) at the beginning of the program. An efficient way to accomplish this is to use the TMS32010 assembler's DORG (dummy origin) directive. This directive does not cause code generation. DORG defines a data structure to be used by the system; i.e., it generates values corresponding to the labels of consecutive data memory locations. Using the DORG directive, as opposed to equating labels with data memory locations through the EQU directive, provides flexibility when the data structure needs to be modified. For example, when defining a number of new data memory locations, the labels are inserted in the middle of the "dummy" block and the assembler assigns the values automatically. This function would have to be performed manually if the EQU directive were used.

The software designer must also build a table in the TMS32010 program memory that corresponds to the previously defined data memory map. The table is then loaded into the data memory by the initialization routine during system startup.

These techniques are illustrated in the next section, TMS32010 IMPLEMENTATION OF COMPENSATORS AND FILTERS. Note that in the example program in Appendix B, location ONE has to be the last location in the table. Note also that the states and the coefficients of the filters are defined in reverse order to the order in which the filters execute. This is due to the way the initialization and filter routines are written.

TMS32010 IMPLEMENTATION OF COMPENSATORS AND FILTERS

Design procedure and error handling for the standard first- and second-order compensator and filter subroutine are described in this section. Methods for implementing higher-order structures, implementation tradeoffs, and examples of typical compensators and filters are also given.

Standard First- and Second-Order Block as a Subroutine

A standard first- and second-order compensator section is a prime example of the building block philosophy discussed earlier. The routine presented here computes first- and second-order IIR filter sections using the Direct-Form II network structure[12] and performs roundoffs and overflow checking. The Direct-Form II, although it somewhat obscures the definition of the variables, is chosen over the Direct-Form I because it requires fewer "delays", i.e., data storage locations, in its computational algorithm.

Consider the second-order transfer function:

$$D(z) = \frac{N0 + N1\, z^{-1} + N2\, z^{-2}}{1 + D1\, z^{-1} + D2\, z^{-2}} = \frac{U(z)}{E(z)}$$

For Direct-Form II, the corresponding difference equations are:

$$x(n) = e(n) - D1\ x(n-1) - D2\ x(n-2)$$

$$u(n) = N0\ x(n) + N1\ x(n-1) + N2\ x(n-2)$$

The signal flowgraph for this transfer function is shown in Figure 3.

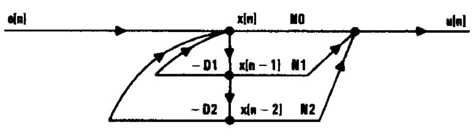

Figure 3. Direct-Form II Compensator/Filter

The filter routine accommodates a scaling scheme as defined on the main program level; i.e., the values can be scaled by 2^{15} or 2^{12}. The routine is written so that chain implementation of a number of compensators and filters is possible if the data structure, namely, the coefficient and state-variable tables, are properly arranged (see the previous section, CONTROL SYSTEM SOFTWARE DESIGN PHILOSOPHY). The routine takes its input from the accumulator and outputs the result to the accumulator so that the main program can efficiently call for the successive execution of the filter routine with a different set of parameters each time.

Two checks for overflow are made within the filter routine. One is made upon computing the value of the intermediate (state) variable and the other upon computing the filter's output. Each overflow check determines whether the 32-bit computed result can be stored in a 16-bit memory location under the adopted scaling scheme. If an overflow condition occurs, the routine "saturates" the output; i.e., it returns the maximum or minimum representable value.

A drawback of overflow checking upon computing the output is the loss of precision of the least-significant bits of the accumulator, which are truncated during the accumulator storing operation. This loss of precision is insignificant, however, in comparison with the loss of precision due to an overflow condition.

The first-order filter section is computed exactly like the second-order section. The two coefficients N2 and $-$D2 that multiply the "oldest" value of the intermediate (state) variable, i.e., the bottom branch of the filter in Figure 3, are equal to zero. This scheme reduces the second-order digital filter to the first-order filter. An example program that uses the first- and second-order filter routine to compute several elements of a digital control system is given in Appendix B.

Higher-Order Filters: Cascade Versus Parallel Tradeoffs

A higher-order filter or compensator in a digital control system can be implemented either as a single section or as a combination of first- and second-order sections. The single section or direct implementation form is easier to implement and executes faster, but it generates a larger numerical error. The larger error occurs because the long filter computation process involves a substantial accumulation of errors resulting from multiplications by quantized coefficients and because the roots of high-order polynomials are increasingly sensitive to changes in their (quantized) coefficients. For this reason, the direct realization form is not recommended except for a very low-order controller.

The suggested method of implementing a high-order transfer function is to decompose it into first-order blocks (to accommodate single real poles) and second-order blocks (to accommodate complex conjugate poles or pairs of real poles), and connect these blocks either in a cascade or a parallel configuration.

For the cascade realization (see Figure 4), the transfer function must be decomposed into a product of first-order and second-order functions of the form:

$$D(z) = K\, D_1(z)\, D_2(z) \ldots D_n(z)$$

Each second-order block has the form:

$$D_i(z) = \frac{N0 + N1\, z^{-1} + N2\, z^{-2}}{1 + D1\, z^{-1} + D2\, z^{-2}}$$

Each first-order block is obtained by equating the coefficients of (z^{-2}), i.e., N2 and D2, to zero.

Figure 4. Cascade Implementation of a High-Order Transfer Function

The designer must decide how to pair the poles and zeroes in forming the $D_i(z)$. A pole-zero pairing algorithm that minimizes the output noise is available.[10] The ordering of the $D_i(z)$ also affects output noise due to quantization and whether or not limit cycles are present.[10]

For the parallel realization (see Figure 5), the transfer function is expanded as a sum of first-order and second-order expressions of the form:

$$D(z) = K + D_1(z) + D_2(z) + \ldots + D_n(z)$$

where the first-order blocks have the form:

$$D_i(z) = \frac{N0}{1 + D1\, z^{-1}}$$

and the second-order blocks have the form:

$$D_i(z) = \frac{N0 + N1\, z^{-1}}{1 + D1\, z^{-1} + D2\, z^{-2}}$$

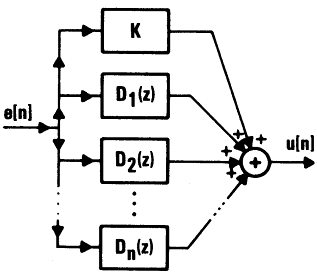

Figure 5. Parallel Implementation of a High-Order Transfer Function

The concept of ordering of $D_i(z)$ does not apply to the parallel configuration.

Pole-zero pairing is fixed by the constraints imposed by the partial-fraction expansion.

The parallel realization has an obvious advantage over the cascade form when an algorithm executes on a multiprocessor system where the filter algorithm can be split up among the processors and run concurrently.

No significant difference is apparent between the parallel and cascade realizations in performance factors such as execution speed and program/data memory use when the algorithms execute on a single processor.[13] The parallel algorithm could possibly provide more precision in computing the filter's output if the designer decided to save the double-precision (32-bit) results from each of the first- and second-order sections and perform a double-precision addition to calculate the final filter output.

Examples of Typical Compensators and Filters

Examples of structures used in digital control systems: compensators (lowpass filters), and notch filters are shown in Table 1. The analog and digital versions of the transfer functions, along with the scaled form of the transfer functions and their coefficient values, are given. A complete example of a transformation from the analog to the digital domain using bilinear transformation with frequency prewarping is presented in Appendix A.

The gain constant that appears in some transfer functions can be implemented either by integrating it into its transfer function or by distributing it over a number of filter sections in a cascade implementation scheme.

PROCESSOR INTERFACE CONSIDERATIONS

Alternatives should be considered when designing the data-acquisition portion of the digital controller hardware. This section addresses the A/D and D/A converter selection, different analog sensor interface methods, and communication with the host processor.

Table 1. Examples of Analog and Digital Versions of Common Transfer Functions

Analog Prototype Transfer Function	Digital Transfer Function (Computed by Bilinear Transformation with Frequency Prewarping; f_s = 4020 Hz)	Scaled Digital Transfer Function	Scaled Coefficients and Order of Storage in Data Memory (as required by First- and Second-Order Filter Routine)
First-Order Compensator: $G(s) = 100 \dfrac{1}{S+1}$	$D(z) = 0.012436 \dfrac{1.0 + 1.0\, z^{-1}}{1.0 - 0.99975\, z^{-1}}$	Scaling Factor = 2^{15} $D(z) = \dfrac{408 + 408\, z^{-1}}{32768 - 32760\, z^{-1}}$	N0 = 408 N1 = 408 D1 = 32760
Second-Order Compensator: $G(s) = 1000 \dfrac{S^2 + 68.25 + 3943}{S^2 + 2512S + 6.31 \times 10^6}$	$D(z) = 870.77 \dfrac{1.0 - 1.9824\, z^{-1} + 0.9826\, z^{-2}}{1.0 - 1.2548\, z^{-1} + 0.5474\, z^{-2}}$	Scaling Factor = 2^{12} $D(z) = 870.77 \dfrac{4096 - 8120\, z^{-1} + 4025\, z^{-2}}{4096 - 5140\, z^{-1} + 2242\, z^{-2}}$	N0 = 4096 N1 = −8120 N2 = 4025 D1 = 5140 D2 = −2242
100-Hz Notch Filter: $G(s) = \dfrac{S^2 + 3.9478 \times 10^5}{S^2 + 125.664S + 3.9478 \times 10^5}$	$D(z) = \dfrac{0.98467 - 1.94534\, z^{-1} + 0.98467\, z^{-2}}{1.0 - 1.94534\, z^{-1} + 0.96935\, z^{-2}}$	Scaling Factor = 2^{12} $D(z) = \dfrac{4033 - 7968\, z^{-1} + 4033\, z^{-2}}{4096 - 7968\, z^{-1} + 3970\, z^{-2}}$	N0 = 4033 N1 = −7968 N2 = 4033 D1 = 7968 D2 = −3970

A/D and D/A Conversions and Integrated Circuits

The A/D and D/A converter selection for a control system design may be based on several factors. Among the most crucial factors are the maximum conversion speed of the converter and the wordlength of the device.

The A/D conversion speed relates directly to the required sampling rate of the specific application. This rate is determined by the need to sample fast enough to prevent aliasing and excessive phase lag and to sample slow enough to avoid the unnecessary expense and accuracy of high data rates.

The A/D wordlength should be chosen based on a worst-case analysis using the following two criteria:

1. The dynamic range of the continuous input signal, and
2. The quantization noise of the A/D converter.

For dynamic range, the designer should determine the minimum and maximum values of the continuous input that need to be accurately represented and select the A/D wordlength in bits based on the resolution required within this range.

Quantization noise is due to the quantization effect of the A/D. The value of this noise during a single conversion can be represented by the difference between the exact analog value and the value allowable with the finite resolution of the A/D. This quantization noise may assume any value in the range $-q/2$ to $+q/2$ for a rounding converter or 0 to q for a truncating A/D converter where q is the quantization level. The quantization level q is equal to the full-scale voltage range divided by 2^B where B is the number of bits in the converter. The quantization noise may be modeled as uniformly distributed noise. The designer should make his choice based on the maximum acceptable quantization level.

The D/A converter wordlength should be chosen in a similar manner to choosing the A/D wordlength by considering the dynamic range of the output signal.

The effects of A/D and D/A converter wordlength on the performance of a high-speed control system are detailed in the University of Arkansas study (see the section on the design example of the TMS32010-based rate-integrating gyro positioning system). The study analyzed the time-domain performance of the system (unit impulse, step, ramp, and torque-disturbance response) as a function of A/D and D/A wordlengths. Twelve-, fourteen-, and sixteen-bit converters were used. The only significant difference found between them was the steady-state error. Twelve-bit converters were found to be adequate.

In a multi-input digital control system, the signal acquisition portion of the digital controller must provide for the multiplexing of several analog inputs into a single A/D converter. Consequently, some external devices are needed to pre-filter (antialiasing filters), sample and hold the analog signals from each channel (S/H circuits), and multiplex the signals onto the A/D converter (analog multiplexer). Multiplexing and filtering may also be necessary at the output in cases where the digital control system computes multiple outputs for the control of the plant.

Two configurations of a cost-effective, multichannel data acquisition system for a digital controller are shown in Figure 6. The first accommodates up to eight inputs; the second can accommodate up to 32 inputs. Note that in these two systems, only one S/H per eight inputs exists, and the variables are sampled in sequence with the same sampling interval between successive samples of a given signal. There will be a "skew" in time between the samples of the various

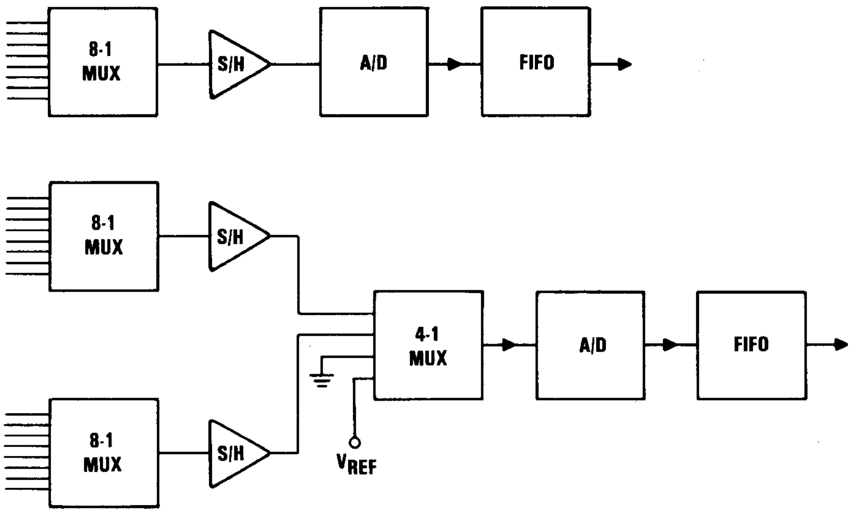

Figure 6. Cost-Effective Data Acquisition Systems

inputs with a possible unwanted effect on the system performance. In this case, the use of a fast A/D converter may be justified to minimize this effect.

If truly simultaneous sampling is required, an array of S/H circuits may be used to capture the values of all the inputs concurrently. This solution, shown in Figure 7, is more expensive due to the cost of S/Hs. In such simultaneous sampling systems, a fast conversion must be performed before the signal values present on the S/Hs start to droop. Therefore, the maximum conversion rate must be fast enough to accommodate this constraint.

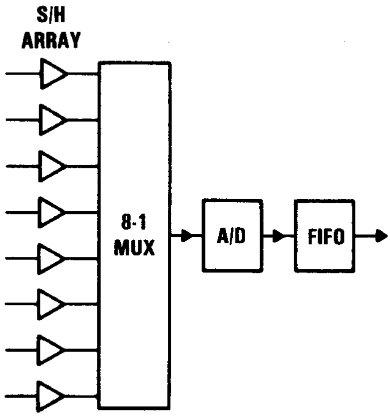

Figure 7. Data Acquisition System with Simultaneous Sampling

In some cases, high-speed A/D converters (100- to 500-kHz conversion rates) are required. Two 12-bit A/D devices that will accommodate these speed requirements are the ADC85 and the AD5240 from Analog Devices. The ADC85 allows a conversion rate of up to 100 kHz and the AD5240 up to 200 kHz. There are other converters available from several manufacturers.

D/A converters, on the other hand, are inherently faster, the selection is much broader, and the cost is less.

When high-speed converters are not needed (when only a few input channels exist or a single converter per channel is justified), devices such as the TCM2913 and TCM2914 codecs may be useful in digital control applications. Although telecommunications-oriented, these devices are low in cost and provide on-chip antialiasing and smoothing filters. The TCM2913 or TCM2914 both contain A/D and D/A converters. The 8-bit digital output of the A/D and the 8-bit digital input to the D/A are both arranged in a companded (compressed/expanded) form using μ-law or A-law companding techniques. The μ-law and A-law companding techniques allow small numbers to be represented with maximum accuracy, but require a conversion routine before the companded samples can be used in two's-complement computations. Such conversion routines are based on lookup tables and need only a few TMS32010 instruction cycles to execute.[14] The devices interface to the processor in a serial form and convert the data at a maximum rate of 8 kHz.

All of these data acquisition systems can accommodate differential inputs from analog transducers, such as pressure sensors, strain gauges, and others. To maintain accuracy in the case of a low-level input signal and to minimize noise effects, twisted-pair leads can be used to connect the transducer output to an instrumentation amplifier (differential-to-single-ended conversion circuit) that in turn is connected to the analog multiplexer. Alternatively, balanced twisted-pair leads can be connected to a differential analog multiplexer which drives an instrumentation amplifier of the same kind. The amplifier rejects the common-mode noise and presents the single-ended output to the S/H circuit and the A/D converter.[15,16] These two configurations are shown in Figure 8.

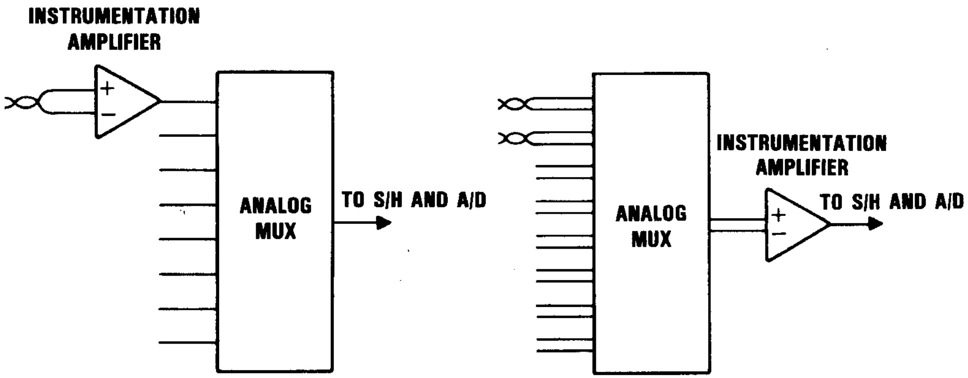

Figure 8. Differential Input Configurations

Synchronization of the Processor and External Devices

In a multichannel data acquisition system with one A/D converter, a designer must generate a sequence of timing signals to synchronize the S/H circuits, the analog multiplexer, the A/D converter, and the input/output latches with the operation of the TMS32010. The designer may decide to generate the timing signals from the TMS32010 clock by subdividing its frequency or using a timing and control circuit based on its own clock.

An alternative way to build a multi-channel data acquisition system is to designate a separate A/D for each channel. In this case, the timing-signal generation is simpler and the A/D converters used may be slower and less costly, although more of them are necessary. The designer should perform a tradeoff analysis based on board space, overall system cost, and power consumption.

Communication with Host Computer

In addition to having a fast signal-processing microprocessor, a need may exist for an executive processor to monitor the system's operation. Such an executive processor would be used for system startup/initialization (coefficient and initial-condition loading), responding to emergency conditions such as overflow and underflow, system reprogramming/reconfiguration (loading a new program or a new set of coefficients), and a thorough system test and calibration. The system should be constructed so that the executive processor can interrupt, halt, or alter the execution of the signal processor at any time in response to contingency situations.

DESIGN EXAMPLE: RATE-INTEGRATING GYRO STABILIZATION LOOP

An example of using the TMS32010 processor to implement a digital control system is presented in this section. Sampling rate selection, the system's hardware and software, and system performance are discussed.

System Description

The system used as an example of the application of the TMS32010 is a servo-control system for stabilizing a large, two-axis gimbaled platform with a DC-motor drive. Inertial rate-integrating gyroscopes mounted directly on the platform serve as angular motion sensors. Such systems are required for the precise control of line-of-sight (LOS) and line-of-sight rate for use in pointing and tracking applications for laser, video, inertial navigation, and radar systems.

At present, digital control is not normally used in systems of this type because of the fast throughput rates and computational accuracy required to perform the control computations and notch filtering. Current line-of-sight stabilization systems continue to use analog electronics to implement servo-compensation functions and error-signal conditioning. Thus, the system is representative in complexity and performance of typical systems currently in use by the aerospace industry and are candidates for microprocessor-based digital control.

The digital control system was designed as part of a research contract carried out by the University of Arkansas under the sponsorship of Texas Instruments from February 1982 to February 1984.[18,19]

System Model and Control Compensation

A single axis of the stabilization system has two primary control loops: the rate loop and the position loop. In addition, a tachometer loop exists within the position loop. The rate and position loops are identified in Figure 9, a diagram of the elevation axis of the system. In its analog version, the system employs analog electronics to implement all control compensation and signal conditioning functions.

Figure 10 identifies those filters and compensators in the rate loop that are to be incorporated into the digital control system.

This study's approach provides a digital implementation of the designated analog elements of the rate loop without sacrificing closed-loop performance. In keeping with the recommendations of the DIGITAL COMPENSATOR DESIGN section, the technique for the conversions of the analog compensators and notch filters to their digital counterparts is the bilinear transformation with frequency prewarping. Within the rate loop, the transfer functions to be implemented digitally consist of a first-order and a second-order compensator, along with six notch filters. Within the position loop, there is one first-order compensator and one notch filter. The transfer functions, shown in Table 2, list both the analog prototypes and their digital equivalents.

The sampling rate chosen is 4020 samples per second (sampling period is 249 μs). This rate is more than twice the highest frequency of consequence (1800 Hz, the highest rate-loop notch frequency) to prevent aliasing. The rate is fast enough to prevent excessive phase lag in the rate loop and is more than ten times the closed rate-loop bandwidth (approximately 80 Hz).[5] The rate was also chosen to be an integer multiple of 30 Hz, which is a commonly used update rate of the video and infrared imaging/tracker devices that provide the line-of-sight rate command to the stabilization system's rate loop. The update rate of the imaging device and the sampling rate within the rate loop are thus synchronized.

After simulating the closed rate loop, the phase margin was found to be five degrees less than it was for the all-analog system, due to the computational and other delays associated with sampling. To overcome this deterioration in phase margin, the second-order rate-loop compensator was redesigned to provide additional phase lead. The compensator was modified to provide enough additional phase lead so that the phase margin of the digital system matched that of the analog system. The modified compensator is listed in the table.

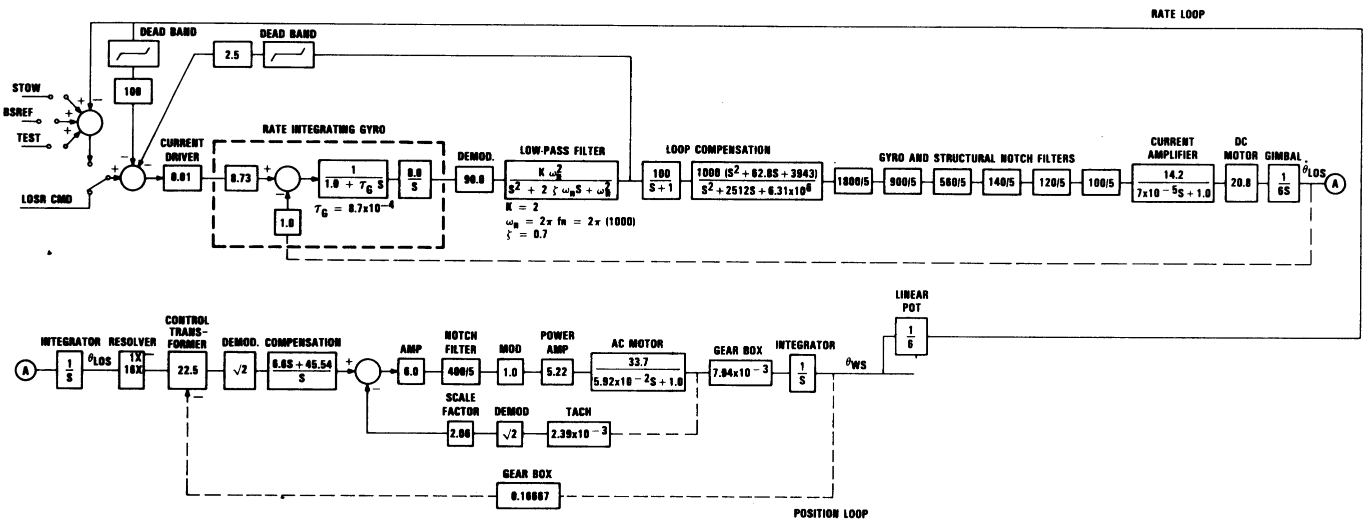

Figure 9. Line-of-Sight Stabilization/Pointing System Elevation Axis

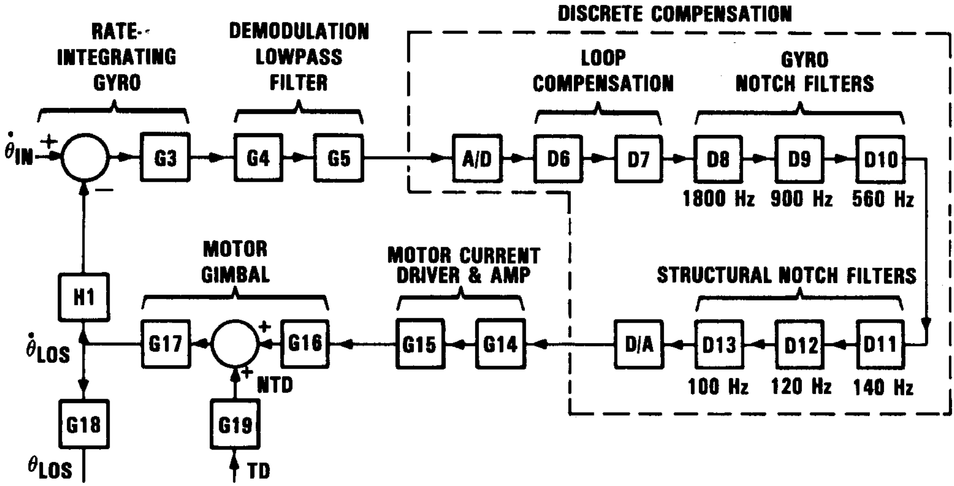

Figure 10. Line-of-Sight Stabilization/Pointing System Rate-Loop [19]

Table 2. Analog and Digital Compensators and Notch Filters

Compensator/ Filter Element	Analog Transfer Function	Digital Transfer Function (f_s = 4020 Hz)
Rate-Loop 1st-Order Compensator	$G6 = \dfrac{100}{S+1}$	$D6 = \dfrac{0.1244\ (1.0+1.0\ Z^{-1})}{1.0-0.99975\ Z^{-1}}$
Rate-Loop 2nd-Order Compensator	$G7 = \dfrac{1000\ (S^2+62.8S+3943)}{S^2+2512S+6.31\text{x}10^6}$	$D7 = \dfrac{754.7101\ (1.0-1.98426\ Z^{-1}+0.9845\ Z^{-2})}{1.0-1.255\ Z^{-1}+0.5474\ Z^{-2}}$
Rate-Loop 1800-Hz Notch Filter	$G8 = \dfrac{S^2+(2\pi\ 1800)^2}{S^2+\dfrac{2\pi\ 1800}{5}\ S+(2\pi\ 1800)^2}$	$D8 = \dfrac{0.96877+1.83411\ Z^{-1}+0.96877\ Z^{-2}}{1.0+1.83411\ Z^{-1}+0.93754\ Z^{-2}}$
Rate-Loop 900-Hz Notch Filter	$G9 = \dfrac{S^2+(2\pi\ 900)^2}{S^2+\dfrac{2\pi\ 900}{2.5}\ S+(2\pi\ 900)^2}$	$D9 = \dfrac{0.8352-0.27291\ Z^{-1}+0.8352\ Z^{-2}}{1.0-0.27291\ Z^{-1}+0.67041\ Z^{-2}}$
Rate-Loop 560-Hz Notch Filter	$G10 = \dfrac{S^2+(2\pi\ 560)^2}{S^2+\dfrac{2\pi\ 560}{5}\ S+(2\pi\ 560)^2}$	$D10 = \dfrac{0.9287-1.19021\ Z^{-1}+0.9287\ Z^{-2}}{1.0-1.19021\ Z^{-1}+0.8574\ Z^{-2}}$
Rate-Loop 140-Hz Notch Filter	$G11 = \dfrac{S^2+(2\pi\ 140)^2}{S^2+\dfrac{2\pi\ 140}{5}\ S+(2\pi\ 140)^2}$	$D11 = \dfrac{0.97875-1.91083\ Z^{-1}+0.97875\ Z^{-2}}{1.0-1.91083\ Z^{-1}+0.95751\ Z^{-2}}$
Rate-Loop 120-Hz Notch Filter	$G12 = \dfrac{S^2+(2\pi\ 120)^2}{S^2+\dfrac{2\pi\ 120}{5}\ S+(2\pi\ 120)^2}$	$D12 = \dfrac{0.9817-1.92896\ Z^{-1}+0.9817\ Z^{-2}}{1.0-1.92896\ Z^{-1}+0.96339\ Z^{-2}}$
Rate-Loop 100-Hz Notch Filter	$G13 = \dfrac{S^2+(2\pi\ 100)^2}{S^2+\dfrac{2\pi\ 100}{5}\ S+(2\pi\ 100)^2}$	$D13 = \dfrac{0.98467-1.94534\ Z^{-1}+0.98467\ Z^{-2}}{1.0-1.94534\ Z^{-1}+0.96935\ Z^{-2}}$
Position-Loop 1st-Order Compensator	$G22 = \dfrac{6.6\ S\ +\ 45.54}{S}$	$D22 = \dfrac{6.60566-6.59434\ Z^{-1}}{1.0-Z^{-1}}$
Position-Loop 400-Hz Notch Filter	$G24 = \dfrac{S^2+(2\pi\ 400)^2}{S^2+\dfrac{2\pi\ 400}{5}\ S+(2\pi\ 400)^2}$	$D24 = \dfrac{0.94471-1.53204\ Z^{-1}+0.94471\ Z^{-2}}{1.0-1.53204\ Z^{-1}+0.88942\ Z^{-2}}$

Hardware

In the digital control system, the analog compensators and the notch filters are replaced by a digital signal processor, the TMS32010, along with the additional interface hardware needed to provide the digital input signals to the controller and the analog signals to the plant. Figure 11 shows the system hardware block diagram.

The hardware was packaged onto five wirewrap boards. It was fabricated as a prototype test bed, and was constructed from commercially available components that have military-specification counterparts. Twelve-bit A/D and D/A converters were used, based on the studies of time-domain performance characteristics of the system.

Software

The TMS32010 software is composed of four modules: Initialization Routine, Main Program, Rate-Loop Subprogram, and Subroutines. Figure 12 shows the system software block diagram.

The Initialization software disables and enables interrupts, loads data memory with filter coefficients, program constants, and gain terms, and initializes the TMS32010 registers.

The Main Program software calls the Delay Subroutine at the beginning of each sample period to wait for the A/D to complete conversion of all input variables. It then does on-line compensation for the error signal sensor variation by executing the A/D Drift Subroutine. The Main Program then reads the value of the input variable, calls the Rate-Loop Subprogram to compute the control output, and, when that subprogram returns the output variable, loads it into the appropriate output register.

The Rate Loop Subprogram calls subroutines that perform each compensator and notch filter computation and checks the computed output for overflow.

The Subroutines consist of a single routine for performing any of the compensator or notch-filter computations (first- and second-order filter routine), along with routines for checking overflow, providing delay, and performing multiplication of low-precision numbers by a constant.

The A/D Drift Subroutine compensates on-line for the variations in the rate-loop error signal sensor (as a function of time and temperature). The subroutine uses an external calibration input and follows the model of the sensor variations to estimate the true value of the A/D input.

The digital control system is interrupt-driven. An interrupt occurs every 1/4020 seconds (approximately 250 μs). This starts the A/D conversion of a new set of sample inputs and restarts the TMS32010 on a new pass through its software.

A TMS32010 Evaluation Module (EVM) and Emulator (XDS) were used in the software development to permit single-step execution of the software for comparison with the corresponding computations produced by simulations written with the aid of the Continuous System modeling Program (CSMP). These simulations take into account the input/output signal quantization levels, microprocessor architecture, memory and internal register lengths.

Other software functions associated with a complete, self-contained control module include:

1. System calibration, testing, and startup
2. Error checking and contingency responses
3. Setting of gains, time constants, and other programmable or adjustable parameters
4. System shutdown.

These functions are implemented by a general-purpose executive processor (SBP9989), thus allowing the TMS32010 to handle computation-intensive tasks.

System Performance

The system performance was evaluated in the following two-step procedure:

1. A hybrid computer system was constructed

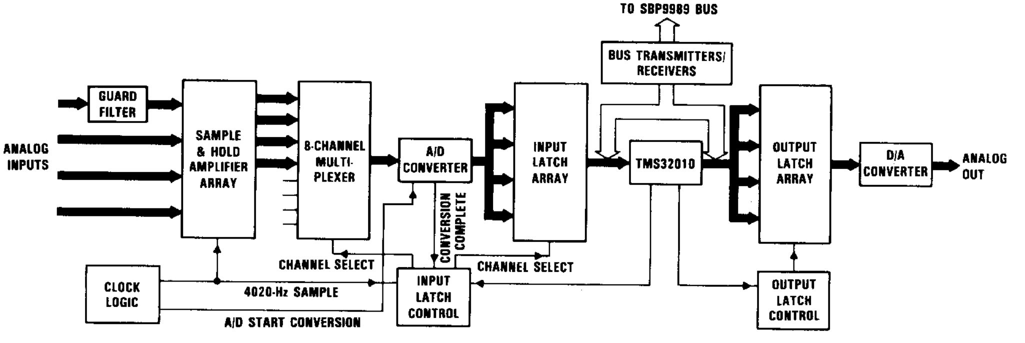

Figure 11. Digital Controller Hardware Block Diagram [19]

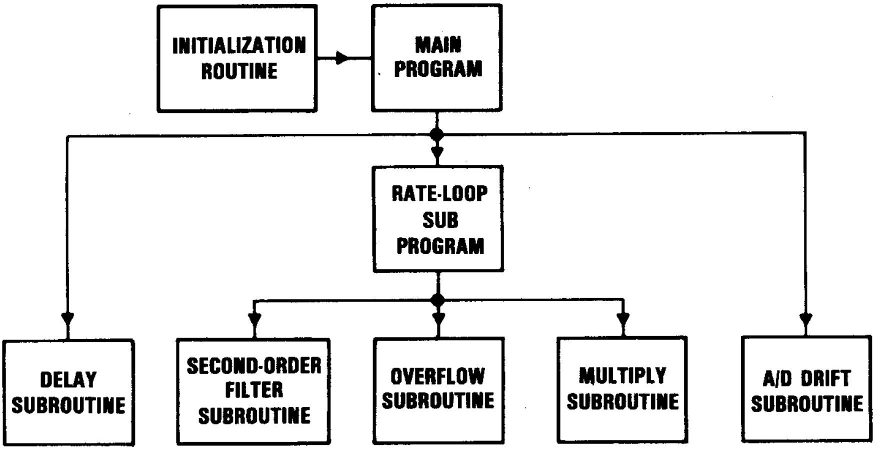

Figure 12. Digital Controller Software Block Diagram [19]

consisting of an analog-computer implementation of part of the nondigital portion of the rate loop coupled with the TMS32010-based digital controller.

2. A full-scale CSMP simulation of the entire rate loop was conducted.

The closed-loop performance of the rate loop was characterized by the following responses: rate-command step response, torque-disturbance step response, and torque-disturbance frequency response. The results are shown in Figures 13 through 15.

In the rate-command step response, the percent overshoot, peak time, and settling time are similar for both the discrete and continuous systems, but the continuous system is slightly smoother. The torque-disturbance step response shows that the discrete system is slightly slower in correcting for a torque disturbance input. In addition, the discrete system has a low-level oscillation (limit cycle). The frequency responses for a torque-disturbance input are also similar, with the continuous system having slightly better torque disturbance rejection in the low-frequency region. These results show that the analog and the digital systems are comparable even though no special efforts were made to take advantage of the capability that digital control offers.

The flexibility of the digital control system was demonstrated by programming the digital system with the capability to correct for a variation in the sensor input to the A/D converter. The system was able to correct on-line (by using a known standard, calculating the gain, and dividing it out) for a 50 percent sinusoidal variation in the sensor gain.

The conversion between two different stabilization systems serves as another flexibility example. The software of a small, two-axis stabilization system was converted to the software of the higher-precision, large, two-axis gimbaled-platform stabilization loop described earlier. The only modification required was in the Main Program and the Rate-Loop Subprogram for the latter system. The modular software design procedure made possible the use of most of the building blocks (subroutines) in the implementation of the new controller.

In general, the study demonstrated the technical feasibility of digital control for a wide-bandwidth, high-precision type of system. Due to the limited scope of the study, the full power of digital control was not utilized, in that the control algorithms were constrained by the design to emulate their analog prototypes. It is likely that significant performance improvements could be achieved by advanced control techniques.

Additional capacity in the TMS32010 remains to accommodate improved, more sophisticated compensators. Table 3 shows the TMS32010 utilization.

Table 3. TMS32010 Utilization (LOS Stabilization System Rate-Loop)

	Used	Available	% Use
Program Memory	275 words	4096 words	7%
Data Memory	76 words	144 words	53%
Execution Time	73 μs	250 μs	29% *

* Based on a 16-MHz (i.e., less than maximum) clock rate.

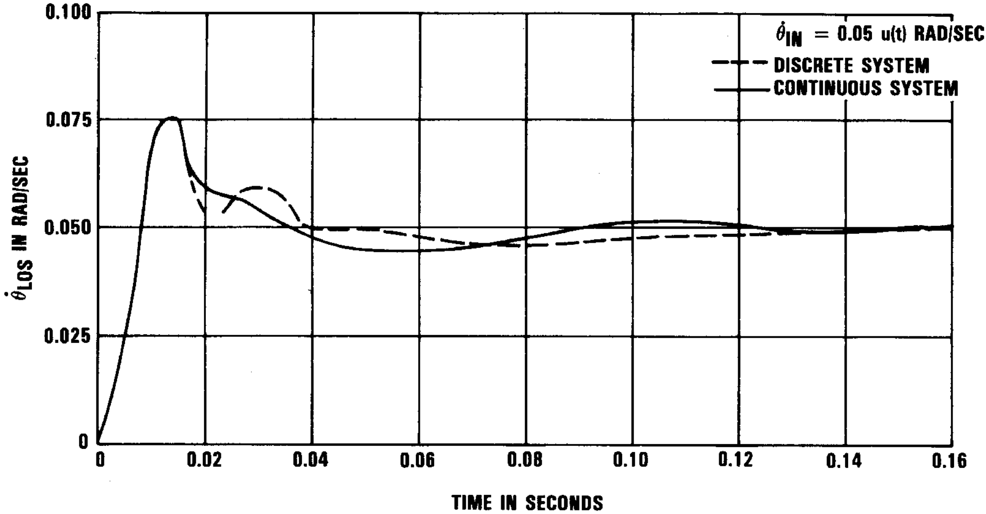

Figure 13. Rate-Loop Rate Command Step Response [19]

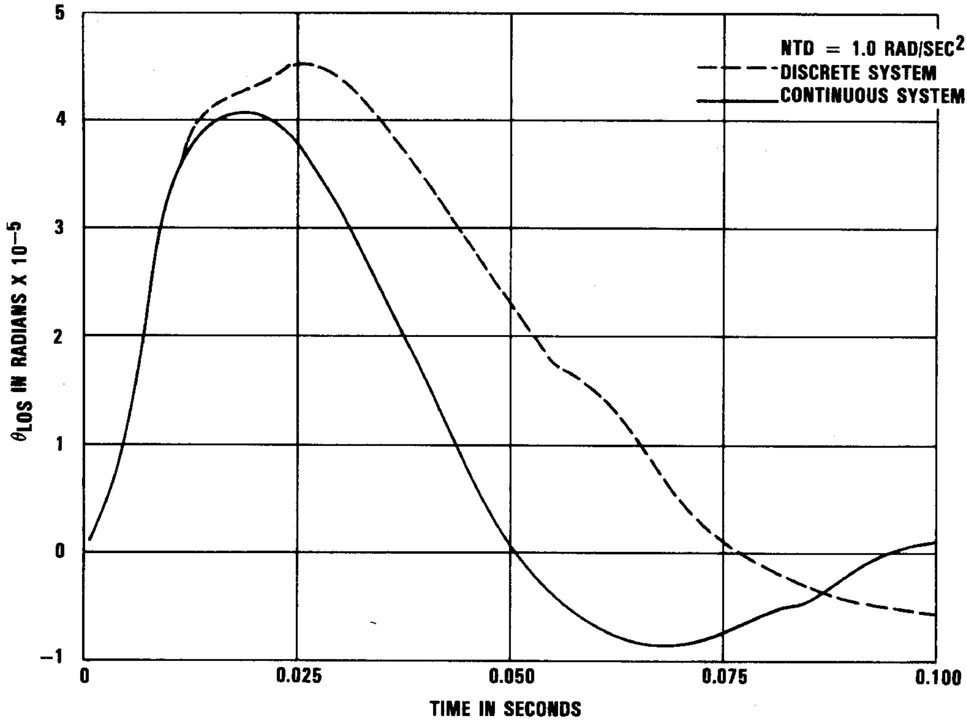

Figure 14. Rate-Loop Normalized Torque-Disturbance Step Response [19]

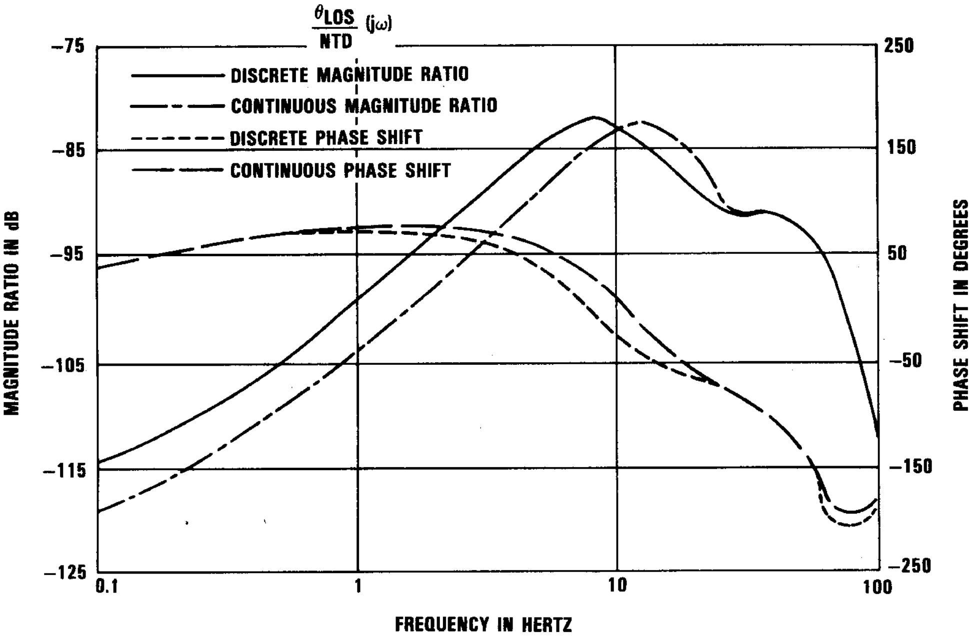

Figure 15. Rate-Loop Normalized Torque-Disturbance Frequency Response [19]

Other microprocessors that were considered for implementing this digital controller system were: Intel 8086, Zilog Z-8000, Motorola 68000, and the Fairchild 9445. These microprocessors were unable to meet the criterion that the maximum allowable time between samples for processing be 250 μs. Among the signal-processing microprocessors, the AMI 2811, while apparently fast enough, has only a 12×12 multiplier; and the Intel 2920 has only four inputs and no branching instructions.

The principal limitation of the TMS32010 was that of having eight inputs and eight outputs. Except for this restriction, the processor would have been able to carry out the processing for both axes of the two-axis gimbaled platform. This limitation could be removed by the addition of logic circuitry.

ACKNOWLEDGMENTS

The authors would like to thank Larry Chatelain of Texas Instruments Incorporated, Dallas, Texas, and Stanley Stephenson of the University of Arkansas, Fayetteville, Arkansas, for their help in the preparation of this document.

REFERENCES

TI TMS32010 Documentation

1. *TMS32010 Assembly Language Programmer's Guide*, Texas Instruments Incorporated (1983).
2. *TMS32010 User's Guide*, Texas Instruments Incorporated (1983).
3. *TMS32010 Development Support Reference Guide*, Texas Instruments Incorporated (1984).
4. *Digital Filter Design Package: Interactive Software for Digital Filter Design and Automatic Code Generation for the Texas Instruments TMS32010*, Atlanta Signal Processors Incorporated, 770 Spring Street, Suite 208, Atalanta, Georgia 30332 (1984).

References on Digital Control

5. P. Katz, *Digital Control Using Microprocessors*, Prentice-Hall (1981).
6. R. Jacquot, *Modern Digital Control Systems*, Marcel Dekker, Inc. (1981).
7. P. Moroney, *Issues in the Implementation of Digital Feedback Compensators*, MIT Press, pp. 16-17 (1983).

8. B.C. Kuo, *Digital Control Systems*, Holt, Rinehart, & Winston (1980).
9. P. Moroney, *Issues in the Implementation of Digital Feedback Compensators*, MIT Press (1983).
10. C. Phillips, H. Nagle, *Digital Control System Analysis and Design*, Prentice-Hall, Inc. (1984).
11. M. Masten, "Rate-Aiding for Line-of-Sight Stabilization/Tracking Systems," *Texas Instruments Engineering Journal*, **Vol 1,** No. 2 (September-October 1984).

References on Digital Signal Processing

12. A. Oppenheim and R. Schafer, *Digital Signal Processing*, Prentice-Hall, Inc. (1975).
13. *The Implementation of FIR/FIR Filters with the TMS32010*, Texas Instruments Incorporated (1983).
14. *Companding Routines for the TMS32010*, Texas Instruments Incorporated (1984).
15. S. Moore, S. Pietkiewicz, "Monolithic A/D Converter Interfaces Directly With Most Microprocessors", *Electronic Design* (September 6, 1984).
16. R. Jaeger, "Tutorial: Analog Data Acquisition Technology Part III," *IEEE Micro* (November 1982).
17. A. Oppenheim, A. Willsky, *Signals and Systems*, Prentice-Hall, Inc. (1983).

References for the Design Example

18. *Digital Servo Techniques, Phase One*, Department of Electrical Engineering, University of Arkansas, Fayetteville, Arkansas 72701 (1982).
19. *Digital Control Techniques, Phase Two, Feasibility Demonstration*, **Vol I and II,** Department of Electrical Engineering, University of Arkansas, Fayetteville, Arkansas 72701 (1984).

APPENDIX A
Development of a Digital Compensator Transfer Function

The development of a digital equivalent of an analog compensator transfer function using the bilinear transformation with frequency prewarping is shown in this appendix. The technique is described in the section, DIGITAL COMPENSATOR DESIGN.

Beginning with an analog prototype transfer function,

$$G(s) = 1000 \frac{S^2 + 68.2\ S + 3943}{S^2 + 2512\ S + 6.31 \times 10^6}$$

The sampling frequency to be used in converting to a digital equivalent is f = 4020 Hz (i.e., the sampling period $T_s = 1/4020\ s = 248.76 \times 10^{-6}\ s$).

The characteristic equation of this analog transfer function is:

$$S^2 + 2512\ S + 6.31 \times 10^6 = 0$$

which fits the standard, second-order form:

$$S^2 + 2\ \zeta\ \omega_n\ S + \omega_n^2 = 0$$

The natural frequency $\omega_n = \sqrt{6.31 \times 10^6} = 2511.9713$ rad/s

To compensate for nonlinear mapping of analog-to-digital frequencies by the bilinear transformation method, the natural frequency is prewarped according to the formula:

$$\omega_p = \frac{2}{T} \tan \frac{\omega_o T}{2} = \frac{2}{248.76 \times 10^{-6}} \tan \frac{2511.9713 \times 248.76 \times 10^{-6}}{2} = 2597.03 \text{ rad/s}$$

This prewarping scheme matches exactly the natural frequency in the analog and digital domains for the compensator.

To obtain the prewarped version of the analog transfer function, the complex variable s in the original transfer function is replaced with $(\omega_o/\omega_p)s$. It is therefore convenient to compute the ratio:

$$\frac{\omega_o}{\omega_p} = \frac{2511.9713}{2597.03} = 0.9672$$

The prewarped G(s), i.e., $G_p(s)$ is then computed as:

$$G_p(s) = 1000 \frac{(0.9672\ s)^2 + 68.2\ (0.9672\ s) + 3943}{(0.9672\ s)^2 + 2512\ (0.9672\ s) + 6.31 \times 10^6}$$

$$= 1000 \frac{s^2 + 70.51\ s + 4214.87}{s^2 + 2597.16\ s + 6.75 \times 10^6}$$

Bilinear transformation is next applied to G_p (s) whereby the continuous variable s is replaced by the expression that involves the discrete variable z:

$$S = \frac{2}{T} \frac{z-1}{z+1}$$

This produces the discrete transfer function D(z).

For the compensator,

$$D(z) = G_p(s)\Big|_{s = \frac{2}{T}\frac{z-1}{z+1}} =$$

$$= 1000 \left. \frac{s^2 + 70.51\, s + 4214.87}{s^2 + 2597.16\, s + 6.75 \times 10^6} \right|_{s = \frac{2}{248.76 \times 10^{-6}} \frac{z-1}{z+1}}$$

After further computations,

$$D(z) = 706.76 \frac{1.0 - 1.9824\, z^{-1} + 0.9826\, z^{-2}}{1.0 - 1.2548\, z^{-1} + 0.5474\, z^{-2}}$$

The final step is the gain adjustment in the digital transfer function. This can be accomplished by matching the analog and digital gains at some predetermined frequency, for example, DC.

For the DC case, $s = j\omega = 0$ and from the bilinear transformation:

$$z = \frac{2+sT}{2-sT} = 1$$

Therefore, at DC, $G(0) = D(1)$.

For this transfer function, $G(0) = 0.6249$, $D(1) = 0.5072$. If $G(0) = K \times D(1)$, then the constant K becomes

$$\frac{0.6249}{0.5072} = 1.2321$$

The final form of the digital equivalent transfer function is:

$$D(z) = 870.77 \frac{1.0 - 1.9824\, z^{-1} + 0.9826\, z^{-2}}{1.0 - 1.2548\, z^{-1} + 0.5474\, z^{-2}}$$

where the gain of 870.77 is the product of K $\times$ (the unadjusted digital gain), i.e., $870.77 = 1.2321 \times 706.76$.

APPENDIX B
TMS32010 Example Program

An example TMS32010 program that uses the first- and second-order filter routine to compute several elements of a digital control system is provided in this appendix. The program illustrates the concepts of modular software design, data memory layout, and cascade implementation of high-order transfer functions. The program was executed on a combination of the TMS32010 Emulator (XDS) and Analog Interface Board (AIB) using random noise as input. The input was sampled at a 4000-Hz rate.

The following transfer functions are implemented with Q12 scaling:

900-Hz Notch Filter $$D(z) = \frac{0.8352 - 0.2729\, z^{-1} + 0.8352\, z^{-2}}{1.0 - 0.2729\, z^{-1} + 0.6704\, z^{-2}}$$

1800-Hz Notch Filter $$D(z) = \frac{0.9688 + 1.8341\, z^{-1} + 0.9688\, z^{-2}}{1.0 + 1.8341\, z^{-1} + 0.9375\, z^{-2}}$$

Other transfer functions (compensators, notch filters) can be implemented in identical fashion by expanding the data structure (filter coefficients and states) and making additional filter routine calls to compute these elements.

The output, as observed on a spectrum analyzer, is shown in Figure B-1.

Figure B-1. Spectrum Analyzer output (900-Hz and 1800-Hz Notch Filters)

The first notch from the left is at 900 Hz, the second is at 1800 Hz. The attenuation of the notch frequencies is about 23 dB in reference to the passband region.

The program that produced this output is as follows:

```
                         IDT     'RIG SYS'
                         OPTION  DUNLST,TUNLST
                *
                *        The program computes a few sections of the LOS
                *        Stabilization System Rate Loop.
                *
                *        Constants
                *        The constant SCALE relates to the scaling factor
                *        through a relation:
                *        scaling factor = 2 ** SCALE (ex. 4096 = 2 ** 12).
                *        Scale constants 12 and 15 are available.
          000C  SCALE    EQU     12                  SCALING FACTOR (Q12)
                *
                *        Data memory map
     0000                DORG    0                   DATA MEM START ADRS
     0000 0000  MODE     DATA    $                   AIB MODE
     0001 0001  RATE     DATA    $                   AIB RATE
                *        Coefficient table
     0002 0002  N03      DATA    $                   FILTER #3 COEFF'S
     0003 0003  N13      DATA    $
     0004 0004  N23      DATA    $
     0005 0005  D13      DATA    $
     0006 0006  D23      DATA    $
     0007 0007  N02      DATA    $                   FILTER #2 COEFF'S
     0008 0008  N12      DATA    $
     0009 0009  N22      DATA    $
     000A 000A  D12      DATA    $
     000B 000B  D22      DATA    $
     000C 000C  N01      DATA    $                   FILTER #1 COEFF'S
     000D 000D  N11      DATA    $
     000E 000E  N21      DATA    $
     000F 000F  D11      DATA    $
     0010 0010  D21      DATA    $
          0010  COEFFS   EQU     $-1
                *        State var table
     0011 0011  X03      DATA    $                   FILTER #3 STATES
     0012 0012  X13      DATA    $
     0013 0013  X23      DATA    $
     0014 0014  X02      DATA    $                   FILTER #2 STATES
     0015 0015  X12      DATA    $
     0016 0016  X22      DATA    $
     0017 0017  X01      DATA    $                   FILTER #1 STATES
     0018 0018  X11      DATA    $
     0019 0019  X21      DATA    $
          0019  STATES   EQU     $-1
                *
     001A 001A  COMAND   DATA    $                   COMMAND INPUT
     001B 001B  OUTPUT   DATA    $                   SYSTEM OUTPUT
                *
     001C 001C  MAX16    DATA    $                   MAX 2-COMPLEMENT NUM IN 16 BITS
     001D 001D  MIN16    DATA    $                   MIN 2-COMPLEMENT NUM IN 16 BITS
          001C  MASK1    EQU     MAX16               MASK
          001D  MASK2    EQU     MIN16               MASK
     001E 001E  ONE      DATA    $                   ONE
                *
                *
     0000                AORG    0
```

```
0000 F900          B       START             RESTART VECTOR
0001 0021
               *
               *       Table in prog memory
     0002  TABLE  EQU     $
0002 000A          DATA    >A,>4DB           AIB MODE AND RATE
0004 0000          DATA    0,0,0,0,0         FILTER #3 COEFF'S
0009 0D5D          DATA    3421,-1118,3421,1118,-2746  900 HZ NOTCH FILTER
000E 0F80          DATA    3968,7513,3968,-7513,-3840  1800 HZ NOTCH FILTER
0013 0000          DATA    0,0,0             FILTER #3 INITIAL STATES
0016 0000          DATA    0,0,0             900 HZ NOTCH INITIAL STATES
0019 0000          DATA    0,0,0             1800 HZ NOTCH INITIAL STATES
001C 0000          DATA    0,0               COMMAND INPUT, SYSTEM OUTPUT
001E 7FFF          DATA    32767,-32768      MAX AND MIN 16 BIT NUMBERS
0020 0001          DATA    1                 ONE
     0020  TBLEND EQU     $-1
               *
               *
               *       Initialize the system
     0021  START  EQU     $
0021 F800          CALL    INIT              INITIALIZATION ROUTINE
0022 0039
               *
               *       Wait on sample
0023 F600  WAIT   BIOZ    GET
0024 0027
0025 F900          B       WAIT
0026 0023
               *
               *       Input sample
     0027  GET    EQU     $
0027 421A          IN      COMAND,PA2        INPUT COMMAND
0028 661A          ZALS    COMAND            GET COMMAND
0029 781C          XOR     MASK1             CORRECT A/D FORMAT
002A 501A          SACL    COMAND            UPDATE COMMAND
               *
               *       Process sample
002B 2C1A          LAC     COMAND,SCALE      LOAD SCALED COMMAND
002C 7010          LARK    AR0,COEFFS        AR0 = PTR TO COEFF TABLE
002D 7119          LARK    AR1,STATES        AR1 = PTR TP STATE VAR TABLE
002E F800          CALL    FILTR2            1800 HZ NOTCH FILTER
002F 0046
0030 F800          CALL    FILTR2            900 HZ NOTCH FILTER
0031 0046
               *
               *       Output sample
0032 5C1B          SACH    OUTPUT,16-SCALE   STORE OUTPUT
0033 661B          ZALS    OUTPUT            GET OUTPUT
0034 781D          XOR     MASK2             CORRECT FOR D/A FORMAT
0035 501B          SACL    OUTPUT            UPDATE OUTPUT
0036 4A1B          OUT     OUTPUT,PA2        AND SEND IT OUT
               *
               *       Repeat the sequence
0037 F900          B       WAIT              GO GET NEXT SAMPLE
0038 0023
               *
               *
```

```
                *      System initialization routine. The routine initializes
                *      the TMS32010 and other system components.
                *      The calling sequence is:
                *      CALL     INIT
                *
                *      Initialize the 32010
          0039  INIT   EQU      $
     0039 7F81         DINT                         DISABLE INTERRUPTS
     003A 7F8B         SOVM                         SET OVERFLOW MODE
                *
                *      Initialize Data Memory
     003B 6E00         LDPK     0                   USE PAGE 0
     003C 6880         LARP     0                   USE AR0
     003D 701E         LARK     AR0,TBLEND-TABLE          INIT PTR TO END OF DATA
     003E 7E20         LACK     TBLEND              INIT PTR TO END OF TABLE
          003F  XFER   EQU      $
     003F 6788         TBLR     *                   XFER FROM PROG TO DATA MEM
     0040 101E         SUB      ONE                 BUMP PTR DOWN
     0041 F400         BANZ     XFER                GO XFER MORE
     0042 003F
                *
                *      Inititialize AIB
     0043 4800         OUT      MODE,PA0            AIB mode
     0044 4B01         OUT      RATE,PA3            AIB RATE
                *
     0045 7F8D         RET                          RETURN
                *
                *
                *      First and second order filter routine. Computes an IIR
                *      filter  using Direct Form II algorithm and adapts to a
                *      scaling scheme defined in the calling program.
                *
                *      The routine incorporates overflow handling code upon
                *      computing the intermediate value and the output.
                *
                *      The calling sequence is:
                *      ACC = scaled filter input
                *      AR0 = ptr to coeff table
                *      AR1 = ptr to state var table
                *      CALL     FILTR2
                *      ACC = scaled filter output
                *      AR0 = ptr to next set of coeff's
                *      AR1 = ptr to next set of state var's
                *
          0046  FILTR2 EQU      $
     0046 6881         LARP     AR1                 USE AR1
                *
                *      Compute intermediate value
     0047 6A90         LT       *-,AR0              T=X2
     0048 6D91         MPY      *-,AR1              MPY X2*D2
     0049 6CA0         LTA      *+,AR0              T=X1, ACC=KU+X2*D2
     004A 6D91         MPY      *-,AR1              MPY X1*D1
     004B 6C98         LTA      *-                  T=X2, ACC=KU+X2*D2+X1*D1
     004C 6898         MAR      *-                  AR1=PTR TO X0
                *
                *      Round, store and check for intermediate overflow
     004D FA00         BLZ      LBL10               CHECK FOR +/- RESULT
```

```
004E 0058
004F 0B1E          ADD      ONE,SCALE-1      ROUND
0050 5C88          SACH     *,16-SCALE       UPDATE INTERMEDIATE VAL
0051 1C1C          SUB      MAX16,SCALE      SUBTRACT SCALED MAX POS NUMBER
0052 FB00          BLEZ     LBL20            IF ACC<=0 THEN NO OVERFLOW
0053 005F
0054 661C          ZALS     MAX16            OVERFLOW, LOAD MAX POS NUMBER
0055 5088          SACL     *                UPDATE INTERMEDIATE VALUE
0056 F900          B        LBL20            GO, COMPUTE OUTPUT
0057 005F
     0058  LBL10   EQU      $
0058 1B1E          SUB      ONE,SCALE-1      ROUND
0059 5C88          SACH     *,16-SCALE       UPDATE INTERMEDIATE VAL
005A 1C1D          SUB      MIN16,SCALE      SUBTRACT SCALED MIN NEG NUMBER
005B FD00          BGEZ     LBL20            IF ACC>=0 THEN NO OVERFLOW
005C 005F
005D 661D          ZALS     MIN16            OVERFLOW, LOAD MIN NEG NUMBER
005E 5088          SACL     *                UPDATE INTERMEDIATE VALUE
           *
           *       Compute filter output
     005F  LBL20   EQU      $
005F 68A0          MAR      *+,AR0           USE AR0
0060 6D91          MPY      *-,AR1           MPY X2*N2
0061 7F89          ZAC                       CLR ACC
0062 6B90          LTD      *-,AR0           T=X1, ACC=X2*N2, UPDATE X2
0063 6D91          MPY      *-,AR1           MPY X1*N1
0064 6B90          LTD      *-,AR0           T=X0, ACC=X2*N2+X1*N1, UPDATE X1
0065 6D91          MPY      *-,AR1           MPY X0*N0
0066 7F8F          APAC                      ACC=X2*N2+X1*N1+X0*N0
           *
           *       Check for output overflow
0067 FA00          BLZ      LBL30            CHECK FOR +/- RESULT
0068 0071
0069 0B1E          ADD      ONE,SCALE-1      ROUND
006A 5C1B          SACH     OUTPUT,16-SCALE  UPDATE OUTPUT
006B 1C1C          SUB      MAX16,SCALE      SUBTRACT SCALED MAX POS NUMBER
006C FB00          BLEZ     LBL40            IF ACC<=0 THEN NO OVERFLOW
006D 0079
006E 2C1C          LAC      MAX16,SCALE      OVERFLOW, LOAD MAX POS NUMBER
006F F900          B        LBL50            GO, RETURN
0070 007A
     0071  LBL30   EQU      $
0071 1B1E          SUB      ONE,SCALE-1      ROUND
0072 5C1B          SACH     OUTPUT,16-SCALE  UPDATE OUTPUT
0073 1C1D          SUB      MIN16,SCALE      SUBTRACT SCALED MIN NEG NUMBER
0074 FD00          BGEZ     LBL40            IF ACC>=0 THEN NO OVERFLOW
0075 0079
0076 2C1D          LAC      MIN16,SCALE      OVERFLOW, LOAD MIN NEG NUMBER
0077 F900          B        LBL50            GO, RETURN
0078 007A
           *
     0079  LBL40   EQU      $
0079 2C1B          LAC      OUTPUT,SCALE     RESTORE ACC
     007A  LBL50   EQU      $
007A 7F8D          RET                       RETURN
           *
           *
                   END
NO ERRORS, NO WARNINGS
```