TEXAS INSTRUMENTS

Minimizing Quantization Effects Using the TMS320 Digital Signal Processor Family

Application Report

1994 Digital Signal Processing Products

Texas Instruments

Printed in U.S.A., August 1994

SPRA035

Book Type
Two Lines
Volume #

Book Type
Volume #

Application
Report

Minimizing Two Lines
Subtitle Line Two

year

Title Two Lines
Subtitle

Title Two Lines

Title
Subtitle Line Two

Title
Subtitle

Book Type

Title

year

Minimizing Quantization Effects Using the TMS320 Digital Signal Processor Family

Evert Cooper, Ph.D.
Member of the Technical Staff, Texas Instruments

TEXAS INSTRUMENTS

IMPORTANT NOTICE

Texas Instruments Incorporated (TI) reserves the right to make changes to its products or to discontinue any semiconductor product or service without notice, and advises its customers to obtain the latest version of relevant information to verify, before placing orders, that the information being relied on is current.

TI warrants performance of its semiconductor products and related software to current specifications in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Please be aware that TI products are not intended for use in life-support appliances, devices, or systems. Use of TI product in such applications requires the written approval of the appropriate TI officer. Certain applications using semiconductor devices may involve potential risks of personal injury, property damage, or loss of life. In order to minimize these risks, adequate design and operating safeguards should be provided by the customer to minimize inherent or procedural hazards. Inclusion of TI products in such applications is understood to be fully at the risk of the customer using TI devices or systems.

TI assumes no liability for applications assistance, customer product design, software performance, or infringement of patents or services described herein. Nor does TI warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used.

WARNING

This equipment is intended for use in a laboratory test environment only. It generates, uses, and can radiate radio frequency energy and has not been tested for compliance with the limits of computing devices pursuant to subpart J of part 15 of FCC rules, which are designed to provide reasonable protection against radio frequency interference. Operation of this equipment in other environments may cause interference with radio communications, in which case the user at his own expense will be required to take whatever measures may be required to correct this interference.

Contents

Appendices

List of Illustrations

Figure *Title* *Page*

. . . . . .

(Optional if there are < 5 figures; require if there are 5 or more.)

List of Tables

Table *Title* *Page*

. . . . . .

(Optional if there are < 5 tables; require if there are 5 or more.)

List of Examples

Figure *Title* *Page*

. . . . . .

(Optional if there are < 5 examples; require if there are 5 or more.)

Program Listings

Figure *Title* *Page*

. . . . . .

(Optional if there are < 5 listings; require if there are 5 or more.)

1 Introduction

When a digital filter or discrete system is realized with a digital arithmetic element, as in the case with a Digital Signal Processor (DSP), additional considerations beyond time or frequency response characteristics are required to describe the performance of the filter or system. Because of its discrete nature, the DSP represents variables and performs arithmetic functions with a finite word length which produces three effects: 1) the selection of filter transfer functions is quantized in which filter poles and zeros exist only at specific locations in the z plane, 2) the input is quantized, and 3) noise is introduced from the DSP operations of multiplication and division. This paper gives the machinery to calculate finite word length effects and then shows how to minimize their significance through improved topology, differing word length, and improved DSP numerical properties.

The paper is organized as follows. Section II analyzes the effects of quantization of the multiplicand upon the discrete system transfer function using sensitivity and root locus methods. Section III considers the effects of input quantization and of noise introduced by DSP arithmetic operations. Section IV shows how to use features of the Texas Instruments TMS320 DSP family to minimize their effects. Section V examines the capabilities of two microcontrollers and compares their numerical properties to that of the TMS320 family.

The problem is considered in a state space format because any filter or discrete system may be formulated within it's general framework. Scalars are denoted by upper or lower case letters in italic type. Vectors are denoted by lower case letters in boldface type, as the vector $\mathbf{x}$ made up of components x_i. Matrices are denoted by upper case letters in boldface type, as the matrix $\mathbf{A}$ made up of elements a_{ij}, (i^{th} row, j^{th} column).

Consider, therefore, the linear, time-invariant, Single Input, Single Output (SISO) system in discrete time

$$\begin{aligned} \mathbf{x}(k+1) &= \mathbf{A}\mathbf{x}(k) + \mathbf{b}u(k) \\ y(k) &= \mathrm{c}\mathbf{x}(k) + du(k), \end{aligned} \tag{1}$$

where $\mathbf{A}$, $\mathbf{b}$, $\mathbf{c}$, d are $n\times n$, $n\times 1$, $1\times n$, and 1×1 respectively with strictly real elements a_{ij}, b_i, c_i. It is assumed that $\mathbf{A}$ is asymptotically stable with eigenvalues inside or on the unit disk, $|z| = 1$. Elements a_{ij}, b_i, c_i, d are the Arithmetic Logic Unit (ALU) multiplicands within a DSP that multiply either a state or input variable, and are quantized because they are represented within the DSP by digital words of finite length.

2 Quantization of Discrete System Transfer Functions

The transfer function of (1) is

$$\begin{aligned} G(z) &= \frac{y(z)}{u(z)} \\ &= \mathrm{c}(z\mathbf{I}-\mathbf{A})^{-1}\,\mathbf{b} + d, \end{aligned} \tag{2}$$

and has the form

$$G(z) = \frac{\beta_m z^m + \beta_{m-1} z^{m-1} + \cdots \beta_1 z + \beta_0}{\zeta_n z^n + \zeta_{n-1} z^{n-1} + \cdots \zeta_1 z + \zeta_o}, \tag{3}$$

where $m = n$ (proper transfer function) if d ≠ 0 and with m < n (strictly proper transfer function) if $d = 0$. The denominator is identically the characteristic polynomial given by

$$\det\ [z\mathbf{I}–\mathbf{A}] = \mathbf{0}, \tag{4}$$

where **I** is the $n \times n$ identity matrix. The numerator roots or zeros are given by,

$$N(z) = \beta_m z^m + \beta_{m-1} z^{m-1} + \cdots + \beta_1 z + \beta_0 = 0. \tag{5}$$

and the denominator roots or poles are given by the characteristic equation,

$$D(z) = \zeta_n z^n + \zeta_{n-1} z^{n-1} + \cdots \zeta_1 z + \zeta_0 = 0 \tag{6}$$

Since quantized elements a_{ij}, b_i, c_i, d of {**A**, **b**, c, d} prescribe the transfer function polynomial coefficients, the transfer function is quantized also, with poles and zeros existing only at discrete locations in the z-plane. Quantization of the transfer function changes numerator and denominator polynomial coefficients to slightly different values than those desired, resulting in a new, usually slightly different transfer function.

Now it remains to find the effects of finite word length upon the two polynomials. For discrete systems embodying very high Q filters these effects on polynomial coefficients may be important, and may even result in instability. There are several methods to study the effects of quantized coefficients and two are presented here.

The first method [1, 2, 3] expands the polynomial of interest, either the numerator or denominator, in a Taylor series for a coefficient sensitivity analysis. The second method uses root locus procedures to plot polynomial roots as a function of coefficient value. The first determines root movement in a precise way, but the second has a more intuitive appeal since roots are displayed graphically and since stability constraints are available for each polynomial coefficient once the root locus is calibrated. The sensitivity analysis is given first.

2.1 Sensitivity Analysis

This section gives the effect of multiplier coefficient quantization upon the transfer function with a sensitivity analysis. The discussion uses the denominator polynomial of (3) because it is significant to system stability, but the methods are equally applicable to the numerator polynomial.

If a particular coefficient, such as ζ_k, is subject to perturbation, then the polynomial (6) is a function of that coefficient as well.

$$D(z,\ \zeta_k) = \zeta_n z^n + \zeta_{n-1} z^{n-1} + \cdots + \zeta_k z^k + \cdots + \zeta_1 z + \zeta_0 = 0. \tag{7}$$

The denominator polynomial roots satisfy (7) with roots $\lambda_1, \lambda_2, \ldots, \lambda_n$, but if ζ_k is perturbed by an amount $\delta\zeta_k$ from an initial value ζ_{k_0}

$$\zeta_k = \zeta_{k_0} + \delta\zeta_k\ , \tag{8}$$

there is a change in all of the roots $\lambda_j, j = 1, \ldots n$ of (7). To find the change $\delta\lambda_j$ due to $\delta\zeta_k$ in the j^{th} root, from an initial root value λ_{j_0}, the polynomial is expanded in a Taylor series about nominal values $\lambda_{j_0}$and ζ_{k_0},

$$D(z,\ \zeta_k) = D\left(\lambda_{k_0},\ \zeta_{k_0}\right) + \left.\frac{\partial D(z,\zeta_k)}{\partial z}\right|_{\substack{z=\lambda_k \\ \zeta_k=\zeta_{k_0}}} \delta\lambda_k + \left.\frac{\partial D(z,\zeta_k)}{\partial \zeta_k}\right|_{\substack{z=\lambda_k \\ \zeta_k=\zeta_{k_0}}} \delta\zeta_k\ . \tag{9}$$

+ . . . . higher order terms

At root locations λ_j and λ_{j_0} the two polynomials $D(\lambda_j, \zeta_{k_0})$ and $D(\lambda_{j_0}, \zeta_{k_0})$are respectively zero. Since higher order terms of (9) are negligibly small because perturbation $\delta\lambda_j << 1$, equation (9) becomes

$$0 = 0 + \left.\frac{\partial D(z,\zeta_k)}{\partial z}\right|_{\substack{z=\lambda_{j_0} \\ \zeta_k=\zeta_{k_0}}} \partial\lambda_k + \left.\frac{\partial D(z,\zeta_k)}{\partial \zeta_k}\right|_{\substack{z=\lambda_k \\ \zeta_k=\zeta_{k_0}}} \partial\zeta_k\ . \tag{10}$$

Equation (10) is solved to give the change in root location $\delta\lambda_j$ due to coefficient change $\delta\zeta_k$

$$\boxed{\delta\lambda_j = -\frac{\left.\dfrac{\partial D(z,\zeta_k)}{\partial \zeta_k}\right|_{\substack{z=\lambda_{j_0} \\ \zeta_k=\zeta_{k_0}}}}{\left.\dfrac{\partial D(z,\zeta_k)}{\partial z}\right|_{\substack{z=\lambda_{j_0} \\ \zeta_k=\zeta_{k_0}}}} \delta\zeta_k\ .} \tag{11}$$

Since matrix elements ζ_{ij}, b_i, c_i, d are given within the DSP by a finite word length, each is subject to perturbation. To use result (11) the polynomial coefficient ζ_k must be related to these perturbed elements. For some dynamic systems this is an easy task, i.e., those of small order, or those expressed in a particularly simple topology. The latter category includes special canonical forms where matrix elements are identical to those of numerator and denominator polynomial coefficients.

For other systems the task is more arduous so it is useful to derive general comments that do not require the mapping from element space to coefficient space. To derive these comments the numerator and denominator of (11) are evaluated.

From (7) the numerator of (11) for the k^{th} coefficient and j^{th} root is

$$\left.\frac{\partial D(z,\zeta_k)}{\partial \zeta_k}\right|_{\substack{z=\lambda_{j_0}\\ \zeta_k=\zeta_{k_0}}} = \left. z^k \right|_{\substack{z=\lambda_{j_0}\\ \zeta_k=\zeta_{k_0}}} = \lambda_{j_0}^{k} \tag{12}$$

If the characteristic equation (7) is expressed equivalently in factored form, assuming non-repeated roots,

$$D(z,\ \zeta_k) = (z-\lambda_1)\ .\ .\ .(z-\lambda_{k-1})\ (z-\lambda_k)\ (z-\lambda_{k+1})\ .\ .\ .(z-\lambda_n), \tag{13}$$

then the denominator of (11) is

$$\begin{aligned}\left.\frac{\partial D(z,\ \zeta_k)}{\partial z}\right|_{\substack{z=\lambda_{j_0}\\ \zeta_k=\zeta_{k_0}}} &= (z-\lambda_1)\ .\ .\ .\ (z-\lambda_{k-1})\ (z-\lambda_{k+1})\ \ .\ .\ .\ (z-\lambda_n)\\ &= \prod_{\mathrm{i}=1,\ \mathrm{i}\neq k}^{n} \left(\lambda_{j_0}-\lambda_i\right).\end{aligned} \tag{14}$$

Combining these two results for the numerator and denominator of (12) gives,

$$\boxed{\delta\lambda_j = \frac{\lambda_{j_0}^{k}}{\prod_{i=1,\ i\neq k}^{n}\left(\lambda_{j_0}-\lambda_1\right)}\ \delta\zeta_k\ .} \tag{15}$$

A first comment comes from examining the numerator of (15), noting that since $\left|\lambda_{j_0}\right| < 1$ then

$$\left|\lambda_{j_0}\right|^n < \left|\lambda_{j_0}\right|^{n-1} < \cdots < \left|\lambda_{j_0}\right|^2 < \left|\lambda_{j_0}\right|, \tag{16}$$

which means that the *most sensitive coefficient will always be* ζ_0, with progressively less sensitivity for coefficient ζ_n.

A second comment comes from examining the denominator of (15), which shows that *sensitivity increases as roots become more clustered*, and furthermore that *sensitivity increases dramatically as system order* n *becomes large*. High Q or sharp cutoff filters are example systems that may have sensitivity problems because their roots are close together. The sensitivity problem diminishes for a given system order if the filter is a composite of smaller order filters in either cascade or parallel form.

Similarly, a continuous system with low frequency poles discretized with a high sample rate is subject to sensitivity problems. The motivation for a high sampling rate stems from the increase in phase margin and consequent improved performance that it gives to feedback control systems. The high sample rate,

however, detrimentally clusters system eigenvalues in the z-plane about $z = 1$ as seen from the power series expansion of the matrix exponential.

$$\mathbf{A} = e^{\mathbf{F}T_S} = \mathbf{I} + \mathbf{F}T_s + \frac{\mathbf{F}T_S^2}{2!} + \frac{\mathbf{F}T_S^3}{3!} + \cdots , \tag{17}$$

where T_S is the sampling interval and $\{\mathbf{F}, \mathbf{g}\}$ defines the continuous system to be discretized,

$$\dot{\mathbf{x}}(t) = \mathbf{F}\mathbf{x}(t) + \mathbf{g}u(t) . \tag{18}$$

For small sampling interval, i.e., as $T_S \to 0$,

$$\lim_{T_s \to 0} \mathbf{A} = \mathbf{I} , \tag{19}$$

so the eigenvalues of $\mathbf{A}$ in the s-plane cluster increasingly about $z = +1$ in the z-plane to make the discretized system increasingly sensitive to coefficient uncertainty.

Example 1.

Consider the discrete system of Figure 1 with parameter a_{11}. Suppose normally that $a_{11} = 2.5$, but quantization effects give a change δa_{11} from this nominal value, and it is desired to know the resultant change in system eigenvalues.

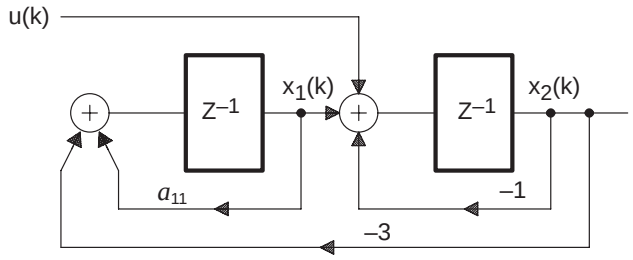


Figure 1. Discrete system of Example 1 with perturbed parameter a_{11}

From Figure 1 the state matrix equation is

$$\begin{bmatrix} x_1(k+1) \\ x_2(k+1) \end{bmatrix} = \begin{bmatrix} a_{11} & -3 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} x_1(k) \\ x_2(k) \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} \mu(k)$$
$$y(k) = [0 \;\; 1] \begin{bmatrix} x_1(k) \\ x_2(k) \end{bmatrix} \tag{20}$$

and its corresponding transfer function is

$$G(z) = \frac{y(z)}{u(z)} = \mathbf{c}(z\mathbf{I}-\mathbf{A})^{-1}\mathbf{b}$$

$$= [0\ \ 1]\left(\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix} z - \begin{bmatrix}a_{11} & -3\\ 1 & -1\end{bmatrix}\right)^{-1}\begin{bmatrix}0\\ 1\end{bmatrix} \tag{21}$$

$$= \frac{(z-a_{11})}{z^2 + (1-a_{11})\ z + 3-\zeta_{11}}\ .$$

Note that matrix element a_{11} contributes to the numerator as well as to the two denominator coefficients. Comparing corresponding coefficients between (7) and the last line of (21) gives $\zeta_0 = 3 - a_{11}$, $\zeta_1 = 1 - a_{11}$, and $\beta_0 = -a_{11}$. Inspection shows that the numerator zero has sensitivity + 1, while the sensitivity of the denominator is less obvious and is diagnosed next.

For $a_{11} = 2.5$ the denominator roots λ_i, $i = 1, 2$ are at $z = 0.5$ and $z = 1.0$. The sensitivity of the eigenvalue at $\lambda = 1.0$, for example, due to changes in ζ_{11} is found from (10). Using the chain rule for partial derivatives,

$$\delta\lambda = -\left[\frac{\left.\frac{\partial D(z,\zeta)}{\partial z}\right|_{\substack{z=\lambda_{j_0}\\ \zeta_k=\zeta_{k_0}}}\left(\frac{\partial\zeta_0}{\partial a_{11}}\right)\delta a_{11} + \left.\frac{\partial D(z,\zeta)}{\partial z_1}\right|_{\substack{z=\lambda_{j_0}\\ \zeta_k=\zeta_{k_0}}}\left(\frac{\partial\zeta_0}{\partial a_{11}}\right)\delta a_{11}}{\left.\frac{\partial D(z,\zeta_k)}{\partial z}\right|_{\substack{z=\lambda_{j_0}\\ \zeta_k=\zeta_{k_0}}}}\right] \tag{22}$$

$$= -\left[\frac{1(-1)\delta a_{11} + z(-1)\delta a_{11}}{2z + (1-a_{11})}\right]\Bigg|_{\substack{z=\lambda_{j_0}\\ \zeta_k=\zeta_{k_0}}}$$

$$= 4\delta a_{11}\ ,$$

which means that an arbitrary change in a_{11} from $a_{11} = 2.5$ to $a_{11} = 2.51$ moves the eigenvalue from $\lambda = 1$ to $\lambda = 1.04$. While the change appears modest, the system is now unstable, because an eigenvalue is outside the unit disk, $|z| = 1$.

2.2 Root Locus Methods

The sensitivity approach in Section 2.1 gives the change in polynomial root location for small perturbations in coefficient value. A second approach prescribed here extends the root locus methods that are used in the design and analysis of linear servomechanisms. The approach gives a portrait in the z-plane of root locations for values of a particular polynomial coefficient. When calibrated, the root locus gives limits for stability, displays filter instability modes, and indicates sensitivity of root locations to coefficient values.

From control literature, the closed loop transfer function $G_{\mathrm{cl}}(z)$ for a SISO control system with forward open loop transfer function $G_{\mathrm{ol}}(z)$, forward gain K, and unity feedback is

$$G_{cl}(z) = \frac{K}{1 + KG_{ol}(z)} \ . \tag{23}$$

The closed loop denominator roots satisfy

$$1 + KG_{ol}(z) = 0 \ . \tag{24}$$

Equivalently,

$$KG_{ol}(z) = -1 \ , \tag{25}$$

and root locus methods show all root locations that satisfy (25) for values of gain K.

Since $G_{ol}(z)$ is a ratio of two polynomials, any polynomial, i.e., the denominator or numerator polynomials of (3), may be manipulated to that of (25) by partitioning the polynomial into two other polynomials $R(z)$ and $S(z)$ such that

$$\gamma R(z) + S(z) = 0 \ . \tag{26}$$

where γ is the element a_{ij}, b_i, or c_i of interest. Then,

$$\gamma R(z) = -S(z) \ , \tag{27}$$

and

$$\boxed{\gamma\left[\frac{R(z)}{S(z)}\right] = -1 \ .} \tag{28}$$

Since (28) is of the same form as (25) all root locus manipulations may be applied.

Example 2.

Consider the transfer function (21) of Example 1, and suppose a desire is to plot denominator roots as a function of matrix element a_{11}. The first step is to isolate all polynomial coefficients with terms containing a_{11}.

$$z^2 + (1-a_{11})z + 3-a_{11} = (z^2 + z + 3)-a_{11}(z + 1) = 0 \ , \tag{29}$$

then

$$a_{11}(z + 1) = z^2 + z + 3 \ , \tag{30}$$

and

$$a_{11}\left[\frac{-(z + 1)}{z^2 + z + 3}\right] = 1 \quad . \tag{31}$$

The root locus of (31) is given in Figure 2 for increments in a_{11}, $\delta a_{11} = 0.1$. Because the denominator factors to $(z + 0.5 + j1.658)$ $(z + 0.5 - j1.658)$ for $a_{11} = 0$, roots originate from outside the unit circle at $z = -0.5 \pm j1.658$. The system is therefore unstable for small values of a_{11}, but becomes stable for mid-range values of a_{11}, and becomes unstable again for large values of a_{11}.

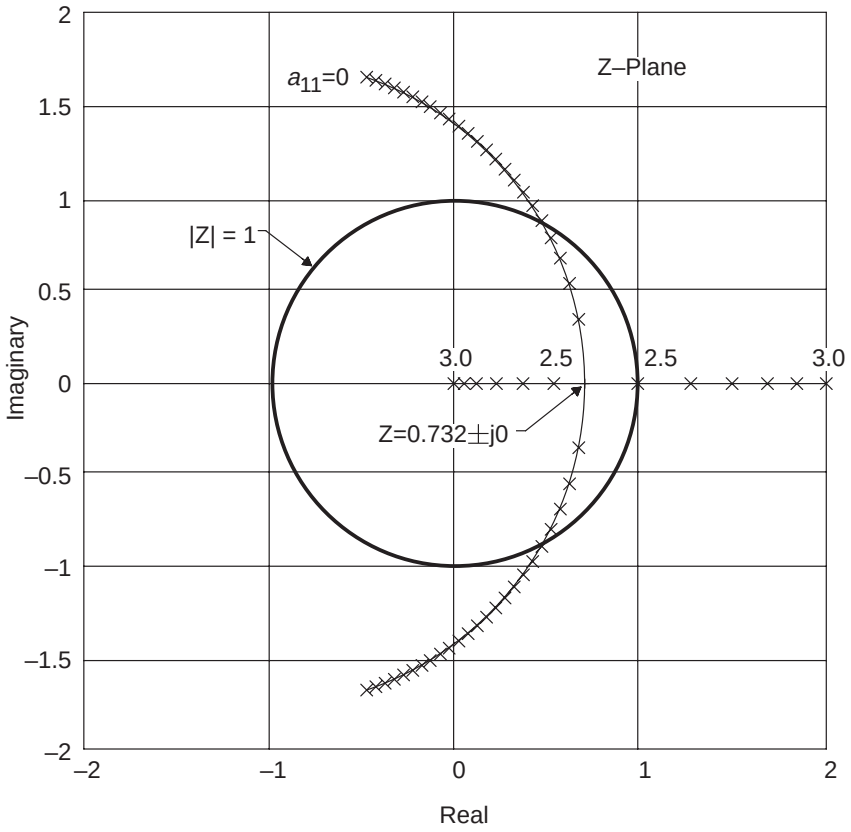


Figure 2. Root Locus of $z_2^2 + (1–a_n)\ z + 3–a_{11}$ for incremental values of $a_{11} \in [0,3]$. Roots are outside the unit disk at $z = -0.5 \pm j1.658$ for $a_{11} = 0$ and move to the real axis. Infinite sensitivity occurs when roots coincide at $z = 0.732 \pm j0$.

The stability limits for a_{11} are $2.0 < a_{11} < 2.5$. For $a_{11} < 2.0$ the instability mode is an exponentially increasing sinusoid, while for $a_{11} > 2.5$ the instability mode is an exponentially increasing ramp. Note that root sensitivity varies dramatically as a_{11} is ranged, and interestingly, this second order example has a locus point of infinite sensitivity at the real axis where roots coincide at $z = 0.732 \pm j0$. This locus point of infinite sensitivity verifies the result given by the denominator of (15) which says that sensitivity increases dramatically with root clustering, approaching infinity in the limit as roots coincide.

3 Quantization Errors Induced by A/D Conversion, Rounding, or Truncation of Computation and Output D/A Conversion

While the previous section considers the effects of coefficient quantization upon the transfer function, this section calculates the effects of quantization upon the input, the state variables within the DSP, and the output signal. The state space approach [1, 3, 4] is used for the calculation, but other methods using z-transform transfer functions [1, 3, 5, 6, 7] are available. Techniques [8, 9, 10, 11, 12] similarly exist for obtaining an upper bound on the magnitude effect of the quantization error upon the digital system.

The three sources of quantization at DSP input, within the DSP, and DSP output occur because:

- The input signal to a discrete system is inherently continuous, and analog-to-digital conversion is necessary before any DSP computations may occur.
- The digital processor discards portions of computed results, i.e., those of multiplication and division. Discarding is required because the number of bits required for exact representation of the result increases by about as many bits as in the representation of matrix coefficients a_{ij}, b_i, c_i, d. Without truncation or rounding the multiplication or division result increases without bound.
- The output of the discrete processor usually operates on a continuous analog system and typically the required digital to analog converter lacks the resolution of the digital processor. Discarding of these least significant bits gives quantization error.

All of these errors fit under the description of a noise-like quantization error. Rounded quantization error ε is the difference between the exact value of a quantity and the value of the nearest set of levels which differ by q. Alternatively, truncated quantization error ε is the difference between the exact value of a quantity and the value of the nearest lower set of levels. The rounded error is shown in Figure 3 in which the exact value is superimposed upon a grid of quantized levels and sample times.

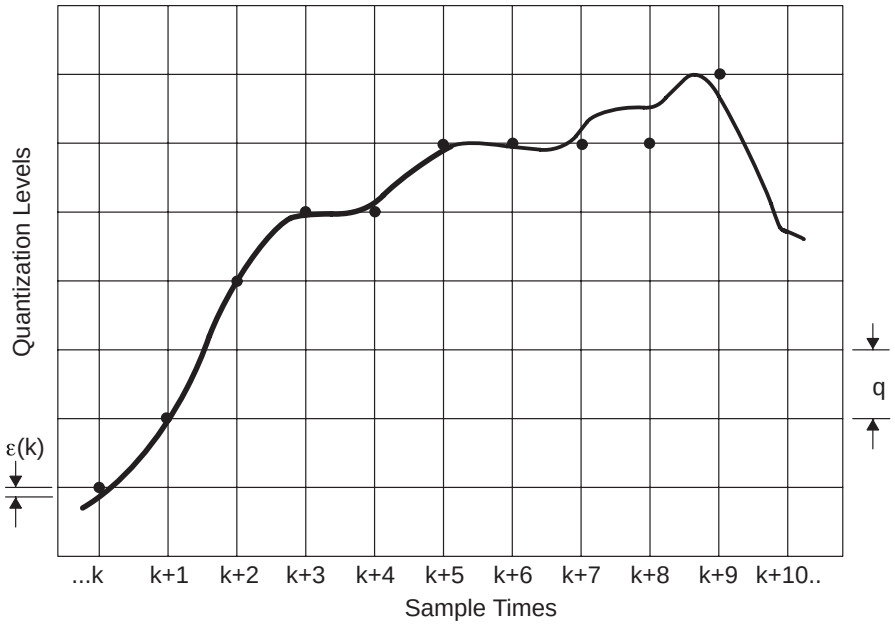
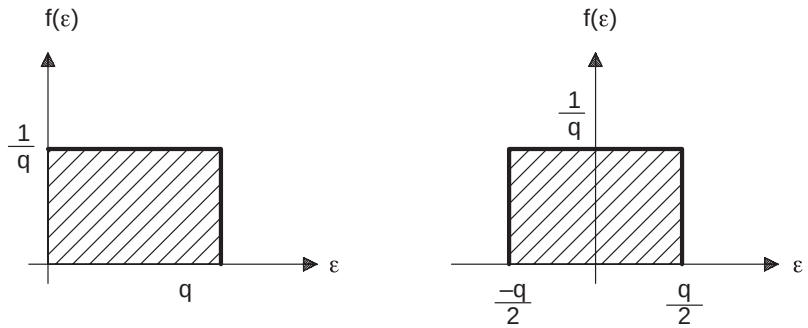


Figure 3. The continuous signal is superimposed upon a grid of sample times and quantization levels spaced by q. Rounded quantization error ε at sample time $k = 1, 2, \ldots$ is the darkened interval between the continuous signal and the nearest quantization level.

Since the difference in quantization levels is q, the rounded error ε between the continuous signal and each quantized level is at most $\pm q/2$. Under the assumption that signal fluctuation is larger than q from one sample to the next, the error ε at one sample will be statistically independent of the error ε at any other sample and will have uniform probability of being anywhere between the limits $\pm q/2$.

Parenthetically, a contrary example is a constant or slowly varying signal. For feedback control systems the statistical dependence between samples due to quantization gives rise to limit cycles. If there is predominantly only one such nonlinearity in the feedback loop, it is possible to use the method of describing functions [2] to approximately determine the limit cycle's behavior. It may be shown [1, 13, 14, 15] however, that the addition of a suitable high frequency dithering signal to the input of the system will make both the quantization error and the roundoff error uniformly distributed, white, and mutually uncorrelated.

The three sources of quantization error are equivalent to adding a noise-like term to the input, the computed results, and the output, respectively. The resultant probability density function (pdf) of the error noise may be as shown in either Figure 4a or Figure 4b, depending upon whether truncation or rounding occurs during the quantization. Truncation is a simple discarding of the least significant bits, while rounding uses the least significant bits in the quantization to minimize the absolute value of the error.



(4a) Truncation (4b) Rounding

Figure 4. Quantization error probability density functions for truncation (4a) and rounding (4b).

The average or mean μ and variance σ^2 of the noise is given by the first and second moments $\mathcal{E}\{\xi\}$, $\mathcal{E}\{\xi^2\}$ of the respective probability density functions.

Quantization due to Truncation:

$$\mu = \mathcal{E}\{\epsilon\} = \int_0^q \epsilon f(\epsilon) d\epsilon = \int_0^q \epsilon \left(\frac{1}{q}\right) d\epsilon = \frac{q}{2} , \tag{32}$$

$$\begin{aligned} \sigma^2 = \mathcal{E}\left\{(\epsilon-\mu)^2\right\} &= \mathcal{E}\{\epsilon^2\}-\mathcal{E}\{\epsilon\}^2 \\ &= \int_0^q \epsilon^2 \left(\frac{1}{q}\right) d\epsilon-\left(\frac{q}{2}\right)^2 \\ &= \frac{q^2}{12} \end{aligned} \tag{33}$$

Quantization due to Rounding:

$$\mu = \mathcal{E}\{\epsilon\} = \int_{-q/2}^{q/2} \epsilon f(\epsilon) de = \int_{-q/2}^{q/2} \epsilon \left(\frac{1}{q}\right) d\epsilon = 0 \tag{34}$$

$$\begin{aligned} \sigma^2 &= \mathcal{E}\left\{(\epsilon - \mu)^2\right\} = \mathcal{E}\{\epsilon^2\} - \mathcal{E}\{\epsilon\}^2 \\ &= \int_{-q/2}^{q/2} \epsilon^2 \left(\frac{1}{q}\right) d\epsilon - 0^2 \\ &= \frac{q^2}{12} \quad , \end{aligned} \tag{35}$$

Results (32)–(35) show that the variance for truncation is the same as that for rounding and only the means are different. The model for the noise sources at each sample instant k is then

$$\epsilon(k, \cdot) = w(k, \cdot) + \mu_\epsilon \quad , \tag{36}$$

where the notation $\varepsilon(k, \cdot)$ and $w(, \cdot)$ describe real valued stochastic processes such that $\varepsilon(k, \cdot)$ and $w(k, \cdot)$ are random variables for any fixed k, $k = 1 \ldots$.

Random variable $w(k, \cdot)$ has zero mean and variance $q^2/12$, while u_ε is either $u_\varepsilon = 0$ or $u_\varepsilon = q/2$ depending respectively upon whether rounding or truncation is used. Formulating the noise sources into their random and constant portions allows the effects of each to be considered separately with $w(k, \cdot)$ exclusively producing variance and u_ε exclusively producing a DC offset.

The error sources are shown in the state diagram of Figure 5 with input A/D conversion noise, $\varepsilon_i(k, w)$, computation noise, $\varepsilon_c(k, w)$ and D/A output noise, $\varepsilon_o(k, w)$. The corresponding state equation is

$$\begin{aligned} \boldsymbol{x}(k + 1) &= \mathbf{A}\mathbf{x}(k) + \mathbf{b}[u(k) + \epsilon_i(k)] + \epsilon_c(k) \ . \\ y(k) &= \mathbf{c}\mathbf{x}(k) + \epsilon_o(k) \end{aligned} \tag{37}$$

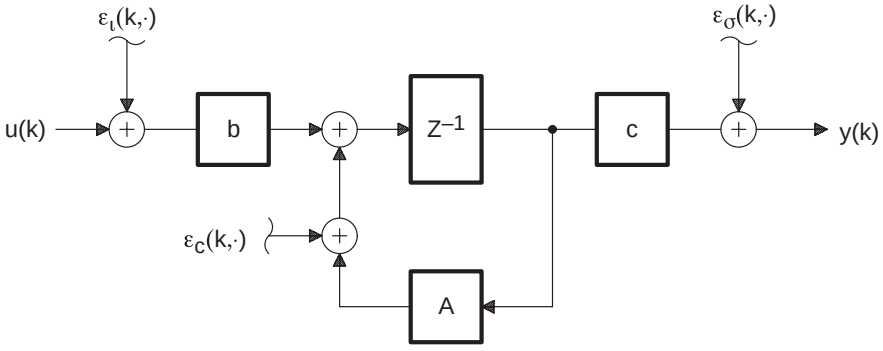


Figure 5. State diagram showing scalar input quantization error $\varepsilon_i(k, \cdot)$, vector computation noise $\varepsilon_c(k, \cdot)$, and scalar $\varepsilon_o(k, \cdot)$.

To compute output variance σ_y^2 requires the expected value of the second moment of $y(k)$.

$$\begin{aligned}
\mathcal{E}\{y(k)y^T(k)\} &= \mathcal{E}\left\{\left[\mathbf{cx}(k) + \epsilon_o(k,\cdot)\right]\left[\mathbf{cx}(k) + \epsilon_o(k,\cdot)\right]^T\right\} \\
&= \mathcal{E}\left\{\left[\mathbf{cx}(k) + \epsilon_o(k,\cdot)\right]\left[\mathbf{x}^T(k)\mathbf{c}^T + \epsilon_o^T(k,\cdot)\right]\right\} \\
&= \mathcal{E}\left\{\mathbf{cx}(k)\mathbf{x}^T(k)\mathbf{c}^T + \epsilon_o(k,\cdot)\mathbf{x}^T(k)\mathbf{c}^T + \mathbf{cx}(k)\,\epsilon_o^T(k,\cdot) + \epsilon_o(k,\cdot)\epsilon_o^T(k,\cdot)\right\} .
\end{aligned} \tag{38}$$

Since $\varepsilon_o\,(k,\cdot)$ is zero mean and uncorrelated with $\mathbf{x}(k)$

$$\begin{aligned}
\mathcal{E}\{y(k)y^T(k)\} &= \mathbf{c}\mathcal{E}\{\mathbf{x}(k)\mathbf{x}^T(k)\mathbf{c}^T\} + \mathcal{E}\left\{\epsilon_o(k,\cdot)\epsilon_o^T(k,\cdot)\right\} \\
&= \mathbf{cP}_{\mathbf{xx}}(k)\mathbf{c}^T + R_o(0) .
\end{aligned} \tag{39}$$

where $R_o\,(\tau)$, $\tau = 0$, is the lagged autocorrelation of the output noise.

Equation (39) shows that the output variance is the sum of the output noise covariance and the state covariance $\mathbf{P_{xx}}(k)$ weighted by output vector c. To compute state covariance $\mathbf{P_{xx}}(k)$ requires a recursion formula, which is found from the expected value of the second moment of $\mathbf{x}(k+1)$. From (37),

$$\begin{aligned}
\mathbf{P}_{\mathbf{xx}}(k+1) &= \mathcal{E}\{\mathbf{x}(k+1)\mathbf{x}^T(k+1)\} \\
&= \mathcal{E}\left\{\left[\mathbf{Ax}(k) + \mathbf{b}w_i(k,\cdot) + \mathbf{w}_c\,(k,\cdot)\right]\left[\mathbf{Ax}(k) + \mathbf{b}w_i\,(k,\cdot) + \mathbf{w}_c\,(k,\cdot)\right]^T\right\} \\
&= \mathcal{E}\left\{\left[\mathbf{Ax}(k) + \mathbf{b}w_i\,(k,\cdot) + \mathbf{w}_c\,(k,\cdot)\right]\left[\mathbf{x}^TA^T(k) + w_i^T\,(k,\cdot)\mathbf{b}^T + \mathbf{w}_c{}^T(k,\cdot)\right]\right\} \\
&= \mathcal{E}\{\left[\mathbf{Ax}(k)\mathbf{x}^T(k)\boldsymbol{A}^T + \mathbf{Ax}(k)w_i^T\,(k,\cdot)\mathbf{b}^T + \mathbf{Ax}(k)\mathbf{w}_c{}^T(k,\cdot)\right] \\
&\qquad + \mathbf{b}w_i(k,\cdot)\mathbf{x}^T(k)\mathbf{A}^T + \mathbf{b}w_i(k,\cdot)w_i^T\,(k,\cdot)\mathbf{b}^T + \mathbf{b}w_i\,(k,\cdot)\mathbf{w}_c{}^T(k,\cdot) \\
&\qquad + \mathbf{w}_c\,(k,\cdot)\mathbf{A}^T\mathbf{x}^T(k) + \mathbf{w}_c\,(k,\cdot)\mathbf{w}_c\,(k,\cdot)\mathbf{b}^T + \mathbf{w}_c\,(k,\cdot)\mathbf{w}_c{}^T\,(k,\cdot)\} .
\end{aligned} \tag{40}$$

Since noise sources $w_i(k,\cdot)$, $\mathbf{w}_C(k,\cdot)$ are statistically independent of each other and with x(k), the expected value of their cross products is zero. Equation (40) becomes

$$\mathbf{P}_{\mathbf{xx}}(k+1) = \mathbf{Ax}(k)\mathbf{x}^T(k)\mathbf{A}^T + \mathbf{b}w_i(k,\cdot)w_i^T\,(k,\cdot)\mathbf{b}^T + \mathbf{w}_c\,(k,\cdot)\mathbf{w}_c\,(k,\cdot)^T , \tag{41}$$

or

$$\mathbf{P}_{\mathbf{xx}}(k+1) = \mathbf{A}\mathbf{P}_{\mathbf{xx}}(k)\mathbf{A}^T + \mathbf{b}R_{\mathrm{i}}(0)\mathbf{b}^{\mathrm{T}} + \mathbf{R}_{\mathrm{c}}(0) \ , \tag{42}$$

where $R_i(\tau)$ and$R_c(\tau)$, $\tau = 0$ are the lagged autocorrelation of the input and computation noise, respectively. Equation (42) says that the state covariance at sample $k + 1$ derives from the previous sample covariance plus input and computational noise terms. If state $\mathbf{x}(k)$ were perfectly known from an initial condition, implying $\mathbf{P}_{\mathbf{xx}}(0) = 0$, then each succeeding sample $k = 1, 2, \ldots$ introduces uncertainty in $\mathbf{x}(k)$ as given by (42). Since $\mathbf{A}$ is stable with eigenvalues within the unit disk,

$$\lim_{k \to \infty} \mathbf{P}_{\mathbf{xx}}(k) \to \mathbf{P}_{\mathbf{xx}}(k+1) \ , \tag{43}$$

and computing (42) recursively is an easy task with available matrix tools[16]. Note that the resultant noise is not uniformily distributed, but rather is Gaussian as prescribed by the Central Limit Theorem [17, 18, 19].

Example 3.

Consider the dynamic system of Example 1, but with $a_{11} = 2.4$ to give a low pass filter. Suppose the input signal $u(k)$ derives from a 10 bit A/D converter with range – 1 to + 1 volts and a Least Significant Bit (LSB) of approximately 20 millivolts. Suppose also that the DSP represents the variables $x_1(k)$ and $x_2(k)$ with a 16-bit, 2's complement word with a dynamic range of –1 (8000h) to +(1–1/2^{15}) (7FFFh), where suffix "h" denotes a hexidecimal value. In making its computations the DSP truncates results, i.e., those of multiplication, to fit the 16 bit word length. If the results are sent to a – 1 to +1 volt 9 bit D/A converter with LSB of approximately 40 millivolts for output, what is the output noise?

The resultant state diagram of the problem is shown in Figure 6 with noise sources ε_i, ε_{cl}, ε_{c2}, ε_o. Note that there is no noise associated with the a_{22} coefficient of –1 since it is actually a subtraction operation.

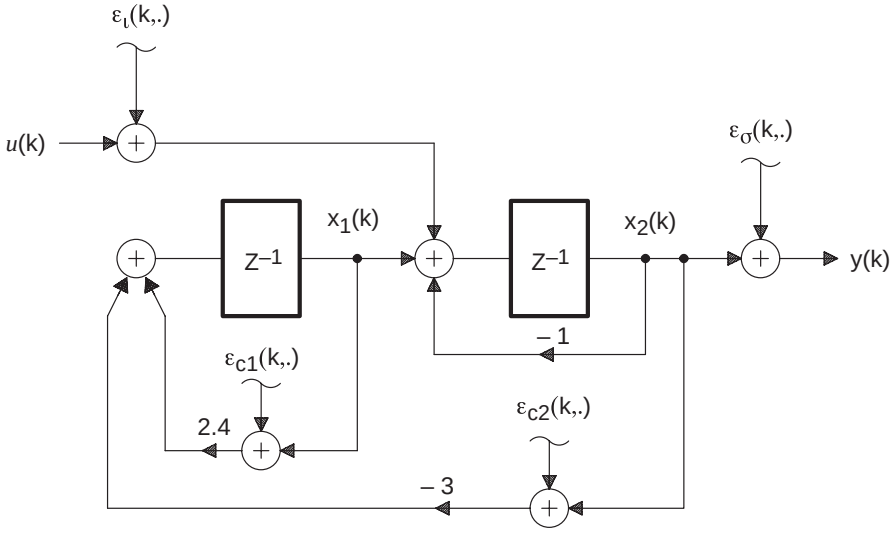


Figure 6. Example 3 state diagram showing noise sources due to quantization effects.

The means and variances of the various sources are

Input noise:

$$\mu_i = 0 \tag{44a}$$

$$\sigma_i^2 = \frac{\left(\frac{1}{2^9}\right)^2}{12} = 3.18 \times 10^{-7} \text{ volts}^2 \qquad \Rightarrow R_i = 3.18 \times 10^{-7} \text{ volts}^2 \,. \tag{44b}$$

Computation noise:

$$\mu_{c1,\,c2} = \frac{\frac{1}{2^{15}}}{2} = 1.53 \times 10^{-5} \tag{44c}$$

$$\sigma_{c1,\,c2}^2 = \frac{\left(\frac{1}{2^{15}}\right)^2}{12} = 7.76 \times 10^{-11} \,, \qquad \Rightarrow R_c = \begin{bmatrix} 7.76 \times 10^{-11} & 0 \\ 0 & 7.76 \times 10^{-11} \end{bmatrix} . \tag{44d}$$

Output noise:

$$\mu_o = \frac{\frac{1}{2^8}}{2} = 1.95 \times 10^{-3} \tag{44e}$$

$$\sigma_o^2 = \frac{\left(\frac{1}{2^8}\right)^2}{12} = 1.27 \times 10^{-7} \,, \qquad \Rightarrow R_o = 1.27 \times 10^{-7} \,. \tag{44f}$$

Using (44*a–f*) the state covariance relationship (42) is

$$\mathbf{P}_{\mathbf{xx}}(k+1) = \begin{bmatrix} 2.4 & -3 \\ 1 & -1 \end{bmatrix} \mathbf{P}_{\mathbf{xx}}(k) \begin{bmatrix} 2.4 & -3 \\ 1 & -1 \end{bmatrix}^T + \begin{bmatrix} 0 \\ 1 \end{bmatrix} 3.18 \times 10^{-7} \begin{bmatrix} 0 \\ 1 \end{bmatrix}^T + \begin{bmatrix} 7.76 \times 10^{-11} & 0 \\ 0 & 7.76 \times 10^{-11} \end{bmatrix} . \tag{45}$$

Figure 7 shows the recursive solution for $k = 0, 1, \ldots$ with the assumption that state initial conditions are perfectly known, i.e.

$$\mathbf{P}_{xx}(0) = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \quad . \tag{46}$$

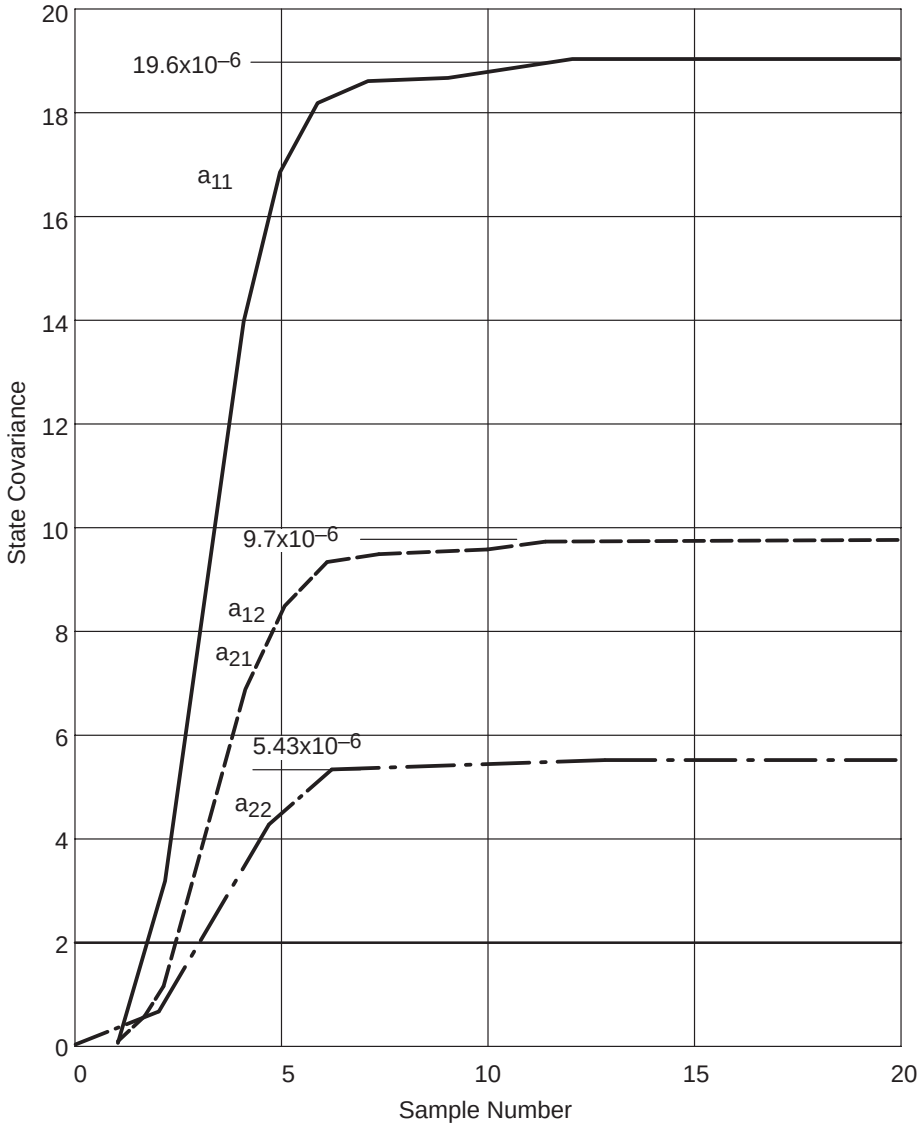


Figure 7. Time solution of covariance matrix equation. Of interest is the asymptotic solution as $k \rightarrow \infty$

It is seen that the state covariance (uncertainty) trajectory increases from zero until a terminating value is asymptotically approached. State dynamics determine the trajectory's rate of increase. The terminating value of the state covariance matrix is

$$\lim_{k \to \infty} \mathbf{P}_{\mathbf{xx}}(k) = \begin{bmatrix} 19.6 \times 10^{-6} & 9.70 \times 10^{-6} \\ 9.70 \times 10^{-6} & 5.43 \times 10^{-6} \end{bmatrix} . \tag{47}$$

The state variances are on the main diagonal of the covariance matrix (47) and are $\sigma^2_{x_1} = 19.6 \times 10^{-6}$ and $\sigma^2_{x_2} = 5.43 \times 10^{-6}$. Note that the initial assumption made as to state initial uncertainty $\mathbf{P_{xx}}(0)$ does not affect result (47). If, for example $\mathbf{P_{xx}}(0)$ is initially large, implying little knowledge of state value, state covariance $\mathbf{P_{xx}}(k)$ would exhibit a diminishing transient until result (47) is again obtained.

Once state covariance $\mathbf{P_{xx}}(k)$ is known, then output variance is found from (39)

$$\begin{aligned} \mathcal{E}\{y(k)y^T(k)\} &= \mathrm{c}\mathbf{P}_{\mathbf{xx}}(k)\mathrm{c}^T + R_o(0) \\ &= [0\ 1] \begin{bmatrix} 19.6 \times 10^{-6} & 9.70 \times 10^{-6} \\ 9.70 \times 10^{-6} & 5.43 \times 10^{-6} \end{bmatrix} [0\ 1]^T + 1.27 \times 10^{-6} \\ &= 5.43 \times 10^{-6} + 1.27 \times 10^{-6} \\ &= 6.70 \times 10^{-6} . \end{aligned} \tag{48}$$

One of the values of calculating covariance is the ability to discern prominent sources of uncertainty for overall system optimization. In this case there is negligible contribution from DSP computation, and only marginal contribution from the output D/A converter. If the overall system optimization is from the perspective of cost, then the marginal contribution of the D/A converter permits a less expensive 8-bit version.

To calculate the effects of truncation and roundoff it is seen from Figure 3 and (32) that the variances for either are the same, but that truncation gives a nonzero mean $\mu_{c1} = \mu_{c2} = q/2 = 1.53 \times 10^{-5}$ that introduces an offset to the state update from $\mathbf{x}(k)$ to $\mathbf{x}(k + 1)$. The effect is modeled from (44), but with $\mu(k) = 0$ and

$$\mu_i = 0, \ \mathbf{u}_c = \begin{bmatrix} 1.53 \times 10^{-5} \\ 1.53 \times 10^{-5} \end{bmatrix}, \ \mu_o = 1.95 \times 10^{-4} . \tag{49}$$

Using these means as inputs, the state equation (37) becomes

$$\begin{aligned} \begin{bmatrix} x_1(k+1) \\ x_2(k+1) \end{bmatrix} &= \begin{bmatrix} 2.4 & -3 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} x_1(k) \\ x_2(k) \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} 0 + \begin{bmatrix} 0 \\ 1 \end{bmatrix} 0 + \begin{bmatrix} 1.53 \times 10^{-5} \\ 1.53 \times 10^{-5} \end{bmatrix} \\ y(k) &= [0\ 1] \begin{bmatrix} x_1(k) \\ x_2(k) \end{bmatrix} + 1.95 \times 10^{-3} \end{aligned} , \tag{50}$$

which is solved recursively until

$$\lim_{k \to \infty} = \begin{bmatrix} x_1(k) \\ x_2(k) \end{bmatrix} = \begin{bmatrix} -\ 7.65 \times 10^{-5} \\ -3.06 \times 10^{-5} \end{bmatrix}, \tag{51}$$

and the output is

$$\begin{aligned} \lim_{k \to \infty} y(k) &= \mathbf{cx}(k) + R_o \\ &= [0\ \ 1] \begin{bmatrix} -\ 7.65 \times 10^{-5} \\ -\ 3.06 \times 10^{-5} \end{bmatrix} + 1.95 \times 10^{-3} \\ &= 1.92 \times 10^{-3} \text{volts} \ . \end{aligned} \tag{52}$$

4 Minimizing Quantization Effects with the TMS320 Family of Digital Signal Processors

To see how the TMS320 DSP family reduces quantization effects a brief understanding is given of how decimal numbers scale and how arithmetic performs in the DSP. The operation of the Central Arithmetic Logic Unit (CALU) [20] is described first.

The Central Arithmetic Logic Unit (CALU) of the TMS320C50 is shown in Figure 8. It incorporates an input 16-bit scaling shifter, a 16 × 16-bit parallel multiplier, a 32-bit Arithmetic Logic Unit (ALU), a 32-bit accumulator, and additional shifters at the outputs of both the accumulator and the multiplier. A typical multiplication of two numbers, $A \times B$, requires a "LT " instruction to load the "T" register with A, a "MPY" instruction whose address operand is the data memory address of the number B, a "PAC" instruction to load the accumulator with the contents of the "P" register, and a "SACH" instruction to store into data memory the 16 most significant bits of the 32-bit result that is in the accumulator. Thus, a single multiplication of two numbers requires four instructions, and if on-chip memory is used so there is no waiting for slower off-chip memory, then compute time for the TMS320C50 with an instruction time of 35 nanoseconds is the time for four instructions or 140 nanoseconds. If a multiplication combines with an addition then ALU operations of data fetching, multiplication, and accumulation may overlap in one instruction cycle with the "MACD" instruction for better performance.

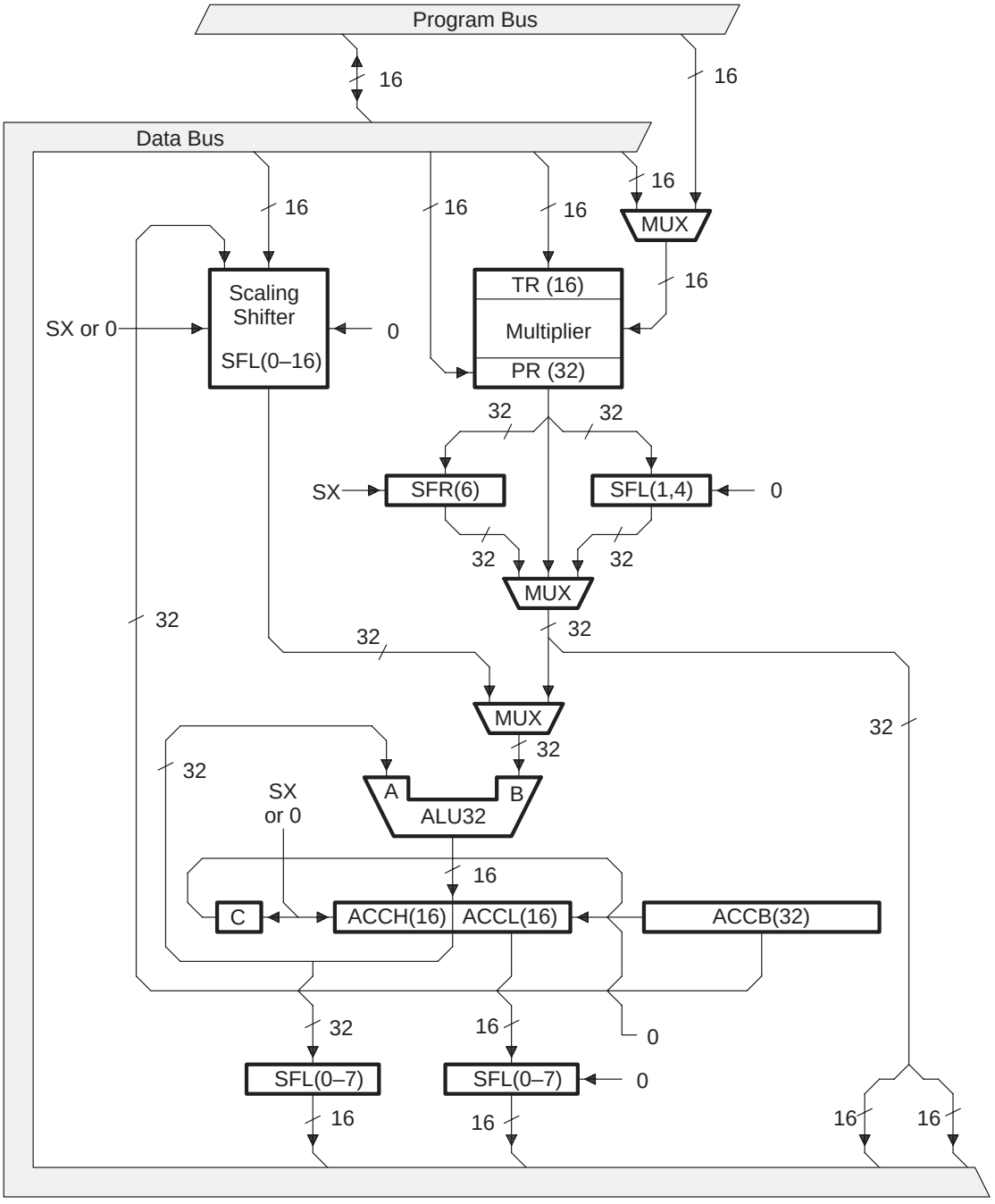


Figure 8. Central Arithmetic Logic Unit (CALU) of the Texas Instruments TMS320C50 Fixed-PointDSP. The CALU contains a 16-bit scaling shifter, a 16 × 16 bit parallel multiplier, a 32-bit arithmetic logic unit (ALU), a 32-bit accumulator and the multiplier. The following steps occur for a typical ALU instruction: 1) data fetches from the RAM on the data bus, 2) data passes through the scaling shifter and the ALU where the arithmetic is performed, and 3) the result moves to the accumulator.

Scaling of decimal numbers is accomplished by representing decimal numbers with 16-bit (single precision) scaled fractions. The digital words are of the form

	msb						lsb
Bit number	15	14	13	12	11	. . .	0
Magnitude	– S	$\frac{S}{2}$	$\frac{S}{4}$	$\frac{S}{8}$	$\frac{S}{16}$	. . .	$\frac{S}{2^{15}}$

, (53)

where bit 15 is the sign bit. An implied radix point exists between bits 15 and 14 with bits 0 through 14 giving 15 bits of magnitude. S is a positive number $S : 0 < S < \infty$ that is the digital word's scaling factor. Scaling factor S allows the digital word to represent any number with interval $[-S, +S(1 - 1/2^{15})]$ by the digital word range 8000h to 7FFFh, where suffix "h" indicates a hexidecimal value.

Common special cases are for $S = 2^{15} = 32768$, which right justifies the decimal point to make the number an integer with interval [– 32768, 32767], and for $S = 1$, which left justifies the decimal point so the decimal number represented by the digital word has interval [–1, 0.9999695]. The unique attribute of this second scaling is the avoidance of ALU overflow, since multiplication with any number of similar interval will always give another number contained within its interval, [–1, 0.9999695].

Because each digital word has an assigned scale factor, two rules are observed when performing scaled arithmetic operations:

Rule 1. Addition

If two variables x and y with respective scale factors S_x and S_y add, $x + y$, then the two scale factors must be the same, $S_x = S_y$.

Rule 2. Multiplication

If constant C multiplies a variable x with scale factor S_x to give variable $y : y = Cx$ with scale factor S_y, then C scales according to the rule

$$C_{DSP} = C \left(\frac{S_x}{S_y} \right) . \tag{54}$$

Applying this rule rescales the decimal constant C as a decimal fraction C_{DSP} with interval $[-1, (1 - 1/2^{15})]$. C_{DSP} is represented within the DSP as a digital word with interval [8000h, 7FFFh].

Rule 1 pertains to the mechanics of scaled addition and will produce quantization error if the LSBs of a shifted variable are discarded after adding to a second variable. The example additions in the next section illustrate the problem.

Rule 2 pertains to multiplication and therefore affects quantization as previously described in Section 3. Because of its significance to the quantization issue an example is now considered that makes the application of Rule 2 more clear.

Suppose the formula $y = 5x$ is to be implemented, where x has interval [1.8, + 1.6], and it is desired to scale the problem for DSP usage. Since the maximum of the absolute value of x is 1.8, scale factor S_x must be some number $\geq$ 1.8, but to minimize quantization effects, S_x is chosen as $S_x = 1.8$ exactly. With this choice of S_x the nearly symmetric interval $[-1.8, 1.8(1 - 1/2^{15})]$ with digital word interval [8000h, 7FFFh] matches the interval of x as closely as possible. Since $S_x = 1.8$, the product $y = 5x$ gives an interval of $[-5 \times 1.8, +5$

$\times 1.8(1 - 1/2^{15})]$, or $[-9, \approx 8.999725]$. To accommodate this interval S_y must be ≥ 9. Because the maximum positive value of C_{DS} is $(1 - 1/2^{15}) \approx 0.9999695$ and not 1, however, S_y is made slightly larger than 9, and is arbitrarily chosen as $S_y = 10$.

With S_x and S_y chosen, C_{DSP} is

$$C_{DSP} = 5\left(\frac{1.8}{10}\right) = 0.90000 = 7333 \text{ hex} \tag{55}$$

To apply these two rules for scaled arithmetic and to reduce quantization effects the TMS320 family of digital signal processors includes the following features for fixed point arithmetic:

1. The most obvious and valuable feature for reducing quantization effects is the width of the ALU, the accumulator, and the 16×16 bit multiplier. The 32-bit width permits accumulation of either extended precision 32-bit words or of 16 bit words with no prospect of overflow. For multiplication, the 32-bit product width matches the 32-bit width of the accumulator and either the entire signed multiplied result applies to the ALU, or extended precision 32×32 bit multiplications are possible with unsigned multiplications.

2. A second feature is an ALU input scaler shifter whose 16-bit input connects to the data bus and whose 32-bit output connects to the ALU. The scaling shifter produces a left shift of 0 to 16 bits on the input data, where the LSBs of the output fill with zeros, and the MSBs either fill with zeros or are sign-extended, depending upon the "SXM" (sign-extension mode) bit. If an addition x with scale factor S_x is made to the accumulator with contents y and scale factor S_y, and if S_x and S_y differ by powers of 2, then the scaling shifter may rescale x to S_y before adding. This second feature allows easy implementation of Rule 1, but does not relate to quantization issues because it is only applicable to addition.

3. A third feature that does relate to quantization is an ability to disable the accumulator overflow saturation mode for modulo addition and subtraction arithmetic. Modulo arithmetic is advantageous where several numbers, usually of alternating sign, add together with a result always less than the sum of their absolute values. A common example of this behavior is the accumulation of FIR filter output taps to form a FIR filter output.

   For modulo arithmetic the accumulator is scaled for the maximum numerical range of the final result instead of the maximum numerical range of the intermediate accumulations. Since scaling for the final result gives apriori knowledge that the number of positive and negative overflows will be the same, overflows are ignored in the intermediate accumulations as numbers add or subtract. In this way the bits that would otherwise prevent overflow give additional significant bits to represent the final sum in the accumulator. Modulo addition reduces quantization noise by allowing a smaller accumulator scale factor to permit accumulation of full 31-bit multiplication results with no truncation.

4. A fourth feature reduces transfer function quantization by allowing "P" register contents to be shifted before they are added to the accumulator. In the multiplication $y = Cx$ it is common for the output scale S_y to be appreciably larger than the constant C times the input scale S_x. From Rule 2, however, $S_y \rightarrow$large, makes $C_{DSP} \rightarrow$ small, and the number of significant figures defining multiplicand C_{DSP} decreases to give an increasingly quantized transfer function.

   An attractive solution to coefficient quantization is to reduce the accumulator scale factor by powers of two when calculating C_{DSP}, and to apply a countering right shift to the product in the

"P" register before it is added to the accumulator. In this way coefficient quantization is reduced by giving additional significant bits for C_{DSP} to represent C.

Available "P" register shifts are 0, +1, +4 and –6 as specified by a two bit "PM" field of status register "ST1". Left shifts zero-fill the LSBs while right shifts sign-extend the MSBs. Usually a left shift of +1 is required for proper signed multiplication because the multiplied output in the "P" register has double sign bits. If no "P" shift is made before accumulation, the effect is mathematically equivalent to a right shift for a lower scale factor. The lower scale factor $S_{\text{accum}}/2$ of the "P" register contents operate through Rule 2 and gives a left shift of C_{DSP} for an additional significant bit of resolution. Similarly, the right shift of –6 places reduces the "P" register scale factor to $S_{\text{accum}}/7$ and makes Rule 2 give a multiplier right shift of +7 places for 7 additional significant bits.

Using this feature reduces the scale factor of the accumulator either by 1/2 or $1/2^7$ when using Rule 2 to calculate C_{DSP} and gives either 1 or 7 additional significant bits to represent C_{DSP}. If the multiplied result is right shifted –6, then 5 bits of the multiplied result are lost and quantization noise is incurred. Thus, shifting of the "P" register gives a tradeoff between transfer function quantization and quantization noise. The quantization noise is very small and does not increase with the number of shifts. It is always $S_{accm}/2^{31}$, where S_{accm} is the accumulator scale factor.

5. A fifth feature is the accumulator output scaler shifter. A "SAC" instruction copies the entire accumulator into a shifter, where it shifts the entire 32-bit number anywhere from 0 to 7 bits, and then copies the upper (lower for a "SACL" instruction) 16 bits of the shifted value into data memory. The accumulator itself remains unaffected. The output scalar thus allows a large accumulator scale factor with several MSBs to prevent overflow from repeated accumulates, but gives a full extraction of 15 significant bits (single precision) from the 31 bits of the accumulator. Without this scaling feature any difference between accumulator and output scale factors would give a result that has fewer significant bits and correspondingly greater quantization.

6. A sixth feature is the easy implementation of an Automatic Gain Control (AGC) [21] function. The AGC function permits dynamic rescaling of input data whose input signal amplitude varies considerably over time. For some applications, such as those of telecommunications, the data is organized in blocks, while for other applications a moving frame of fixed length scans the incoming data. Figure 9 shows the arrangement of data and data blocks for the telecommunications case.

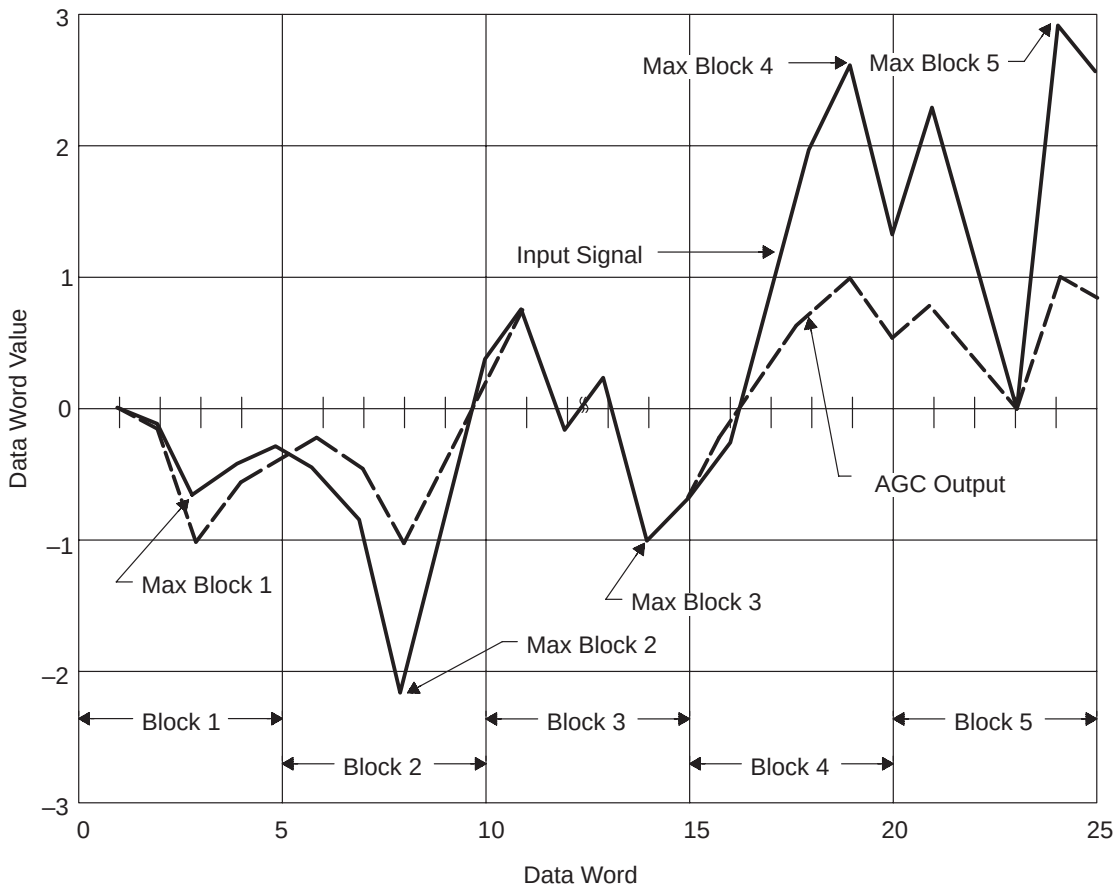


Figure 9. The input signal is organized as digital words within data blocks, but varies considerably over time with each data block having an identifiable digital word of greatest value. The greatest value within each block is respectively used to rescale each block's input signal as shown by the AGC output.

For these applications the usual DSP computational scaling to accommodate the largest amplitudes invariably subjects the smaller amplitudes to truncation and rounding error. The AGC function, on the other hand, automatically adjusts the input signal as shown in Figure 9 to an amplitude prescribed for proper scaling and minimum error.

To apply the AGC function to a block of data samples, the first word of the data block is loaded into the accumulator buffer. A "CRGT" command then tests each remaining word in the data block to find a data word of higher magnitude as each remaining word is passed through the accumulator. If a higher magnitude is found, then a "EXAR" command swaps the higher value to the accumulator buffer. After testing is complete, the data word of greatest magnitude for a particular data block resides in the accumulator buffer.

With this greatest magnitude known, its reciprocal is found either by division or by table look-up. The reciprocal is then loaded into "TREG0" and all of the words of the data block are then multiplied, i.e., rescaled, by the reciprocal. Alternatively, a table look-up may specify progressively greater left shifts (multiplication by powers of 2) for smaller block maximum values. The left shifts are mechanized by loading "TREG1" with the shift value and then using the "LACT" command to load the accumulator with a left shift specified by "TREG1".

7. Finally, a number of additional features are enumerated. While discussed last, they constitute a minimum subset of numerical capabilities that any DSP should have, but which may be absent in microcontrollers. One is the ability to store accumulator contents with a right shift, while leaving the accumulator contents unshifted and intact for no truncation error. Another is the presence of carry and borrow flags for extended precision 64-bit additions and subtractions. A third is the ability for signed and unsigned multiplications, which permits extended precision multiplications.

5 Quantization Effects with Microcontrollers

In contrast to the DSP, the microcontroller does not emphasize a powerful computational architecture. To illustrate the differences between the DSP and the microcontroller, two microcontrollers that are frequently used for disk drive control are considered in this section. They are the Intel 80196 with a clock time of 83 nanoseconds and the Motorola MC68HC16 with a clock time of 59.6 nanoseconds.

5.1 The Intel 80196

The Intel 80196 architecture [22] is shown in Figure 10 and features quick register-to-register transfers. The quick transfers are made possible by the absence of an intervening accumulator that would otherwise constrict data flow. The architecture has at least 230 bytes of on-chip RAM that may be operated upon as bytes, words, or double words. The RAM registers are interconnected through a 16-bit bus with themselves and a 16-bit ALU with the idea that each of the 232 registers is an “accumulator”.

The ALU of the 80196 has two 17-bit input ports (16 bits + sign) denoted as “A” and “B” respectively. Temporary “Upper T1” and “Lower T1” word registers give input to the A port while Temporary “T2” and “Constant” registers give input to the B port. The “Upper” and “Lower” registers have independent shift capability, but may be coupled and shifted together as a 32-bit unit. These registers are used for the repeated shift and add (shift and subtract) operations of multiplication (division). In between the “Temporary” and “Constant” registers and the ALU B port is a multiplexed inverter. The inverter gives a 1’s complement of the contents of the “Temporary” register. A further addition of 1 from the “Constant” register gives a 2’s complement.

As compared to the TMS320 architecture the 80196 numerical hardware is simple, but has functions requisite for good numerical properties. The disadvantage of this simplicity is that arithmetic operations consume an inordinate amount of execution time. To compare the effectiveness of this architecture with that of the TMS320 family a simple scaled addition example is provided next.

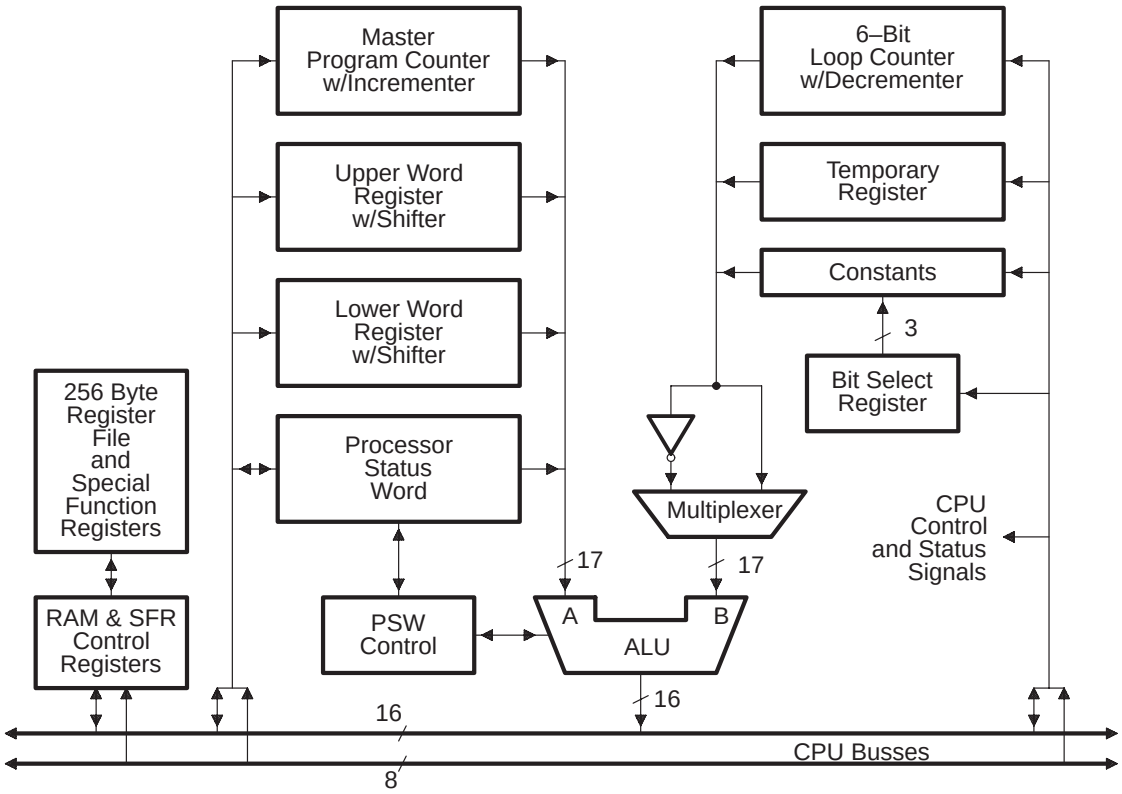


Figure 10. The Intel 80196 architecture has 232 bytes of configurable RAM registers that are connected through a 16-bit bus to an ALU. The architecture features quick register-to-register transfers.

Suppose the addition $z = x + y$ is desired, where x and y have scale factors S_x and S_y, respectively, and that these scale factors are different by powers of two, with $S_x > S_y$. Rule 1 says that the two scales are to be made equal, which is easily obtained by shifting if S_x and S_y differ by powers of 2. The instructions for the TMS320 DSP are:

	Instruction	Execution Clock Cycles
"ZALH"	Zeros the low accumulator and loads high accumulator with y.	1
"ADD"	Adds contents x of specified RAM location to accumulator with specified shift.	1
"SACH"	Stores high accumulator result with shift (if desired) to RAM z_{high} location.	1
"SACL"	Stores low accumulator result with shift (if desired) to RAM z_{low} location.	1

The equivalent Intel 80196 instruction are:

	Instruction	Execution Clock Cycles
"SHRA"	Loads the *x* register (1 of 232) contents into the upper word of register T1 and shifts concatenated upper and lower words of T1 as specified. The result is retained in the upper and lower words of T1.	$\left(6 + \frac{1}{\text{shift}}\right)$
"ADD"	Adds T1 upper word containing shifted *x* to *y* and stores result in z_{high} location.	5
"ST"	Stores T1 lower word containing shifted *x* to z_{low} location.	4

where the notation "$\frac{1}{\text{shift}}$" indicates the need for an additional clock cycle for every shift of "T1" that is required to align the *x* scale factor to that of *y*'s.

Both implementations have the same numerical properties with no quantization error since the full shifted result is saved in z_{high} and z_{low}. The execution cycles are considerably different, however, and tabulate as

$$N_{TMS320} = 1 + 1 + 1 + 1 = 4 \text{ clock cycles}$$
$$N_{Intel80196} = 6 + \frac{1}{\text{shift}} + 4 + 5 = \left(15 + \frac{1}{\text{shift}}\right) \text{ clock cycles} , \tag{56}$$

where it is seen that greater than 4 times as many clock cycles are required for the 80196 architecture. If, for example, 8 shifts are needed to align scale factors, then 15 + 8 = 23 clock cycles are required, which is almost 6 times as many clock cycles as that required for the TMS320 DSPs. For a 12 Mhz oscillator, the clock cycle time for the Intel 80196 is 83 ns and the resultant compute time for the scaled addition example is 1.909 microseconds. Its comparison to members of the TMS320 family and that of the Motorola MC68HC16 is tabulated at the end of the next section.

Note that the difference in clock cycles between the TMS320 family and the Intel 80196 becomes more acute if a second number adds to z_{high} and z_{low}, since the 16-bit number now adds to a 32-bit number and there is no formal accumulator to ease the task. Similarly, a large number of execution cycles is required if there is multiplication involved since the 80196 has no hardware multiplier.

A disadvantage of this architecture is the absence of a formal accumulator. Without a formal accumulator it is easy for the designer to inadvertently discard seemingly insignificant results and to create quantization error, as would occur in the previous example if the designer fails to save the example's shifted *x* result to z_{low}. The designer thus wrestles with tradeoffs between total execution time and quantization error. For the design of a disk drive servo control, for example, this tradeoff study is time consuming, and requires balancing the benefits of less quantization error in signal processing against the effects of a slower sample rate that are incurred by a longer execution time.

5.2 The Motorola MC68HC16

The second microcontroller examined is the Motorola MC68HC16 which is also used in disk drive control. Figure 11 shows the CPU16 register model [23].

The CPU has two 8-bit accumulators (A and B) and one 16-bit accumulator (E). The accumulators A and B may be concatenated into a second 16-bit accumulator (D). The CPU16 has a 16 ×16-bit hardware multiplier with four registers. Register H contains the 16-bit signed fractional multiplier. Register I

contains the 16-bit signed fractional multiplicand. Accumulator M is a specialized 32-bit product accumulation register. The XY mask register contains 8-bit mask values that are used in modulo addressing. There are 16 bits in a word and negative numbers are represented in 2's complement form.

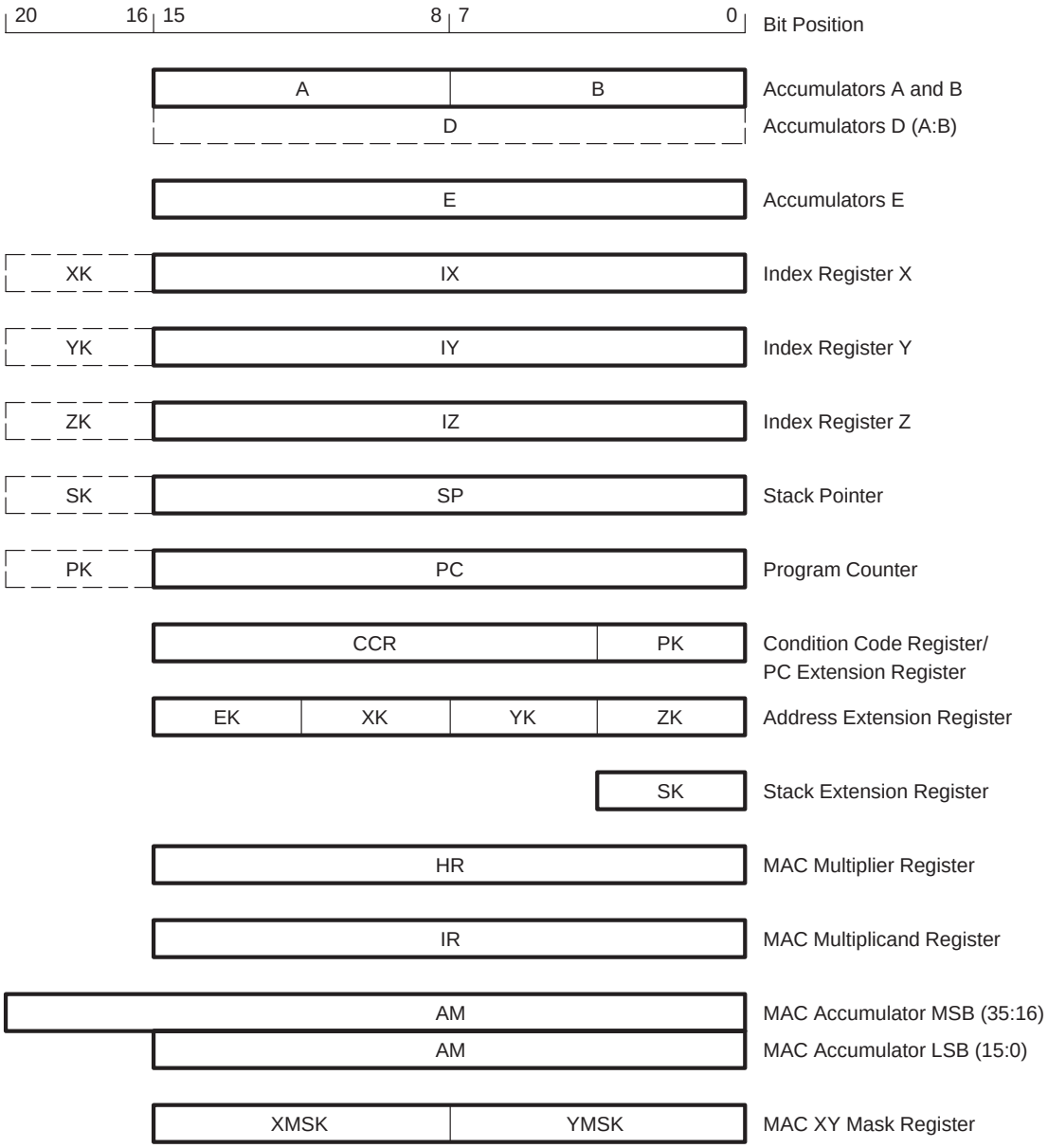


Figure 11. The Motorola MC68HC16 features three accumulators, the "A" and "B" registers, which are configurable as a 16-bit "D" register, the "E" register, and the "AM" register, which is the primary accumulator. Registers "HR" and "IR" are inputs to the 16×16-bit multiplier, and "E" is the output, which is added to the "AM" accumulator register.

A typical multiplication of two 16-bit signed numbers, say $A \times B$ requires "LDD" and "LDE" instructions to place $A \times B$ respectively in the "D" and "E" registers, an "EMUL" instruction to place the $A \times B$ result

into a now concatenated “E:D” register, an “ACED” instruction to add the concatenated “E:D” result to the “AM” accumulator register, a “TMER” instruction to transfer the rounded “AM” accumulator contents to register “E”, and a “STE” instruction to store into data memory the contents of register “E”. A single multiplication of two numbers therefore requires five instructions. If a multiplication combines with an addition then ALU operations of data fetching, multiplication, and accumulation may be overlapped for better performance with a “MAC” instruction.

Comparison of this multiplication procedure with that of the TMS320 family shows the two architectures to give similar performance, with the TMS320 having the slight advantage of requiring only four instructions versus five for the MC68HC16, while the MC68HC16 has the advantages of a larger accumulator. To illustrate some of the MC68HC16’s numerical properties the previous simple example of signed addition $z = x + y$ is now reworked.

With the M68HC16 there are two methods to perform signed addition, but the preferable approach is to use the 36-bit “AM” accumulator as follows:

	Instruction	**Execution Clock Cycles**
“LDE”	Load register “E” with contents of data memory address containing *x*.	4
“TEM”	Transfer “E” to “AM” [31:16]; sign extend “AM”; clear “AM” [15:0]	4
“ASRM”	Arithmetic shift right the “AM” accumulator an appropriate number of shifts to adjust scale factors.	4
“LDE”	Load register “E” with contents of data memory address containing *y*.	4
“ACE”	Add “E” to “AM” [31:15] to form *z*.	2
“TMXED”	Transfer “AM” to “IX”:“E”:“D” registers.	6
“STE”	Store contents of “E” to data memory address for z_{low}.	6
“STD”	Store contents of “D” to data memory address for z_{low}.	6

As in the evaluation of the TMS320 family and the Intel 80196 there is no quantization error, since the full shifted result is saved in z_{high} and z_{low}. The total number of clock cycles for the MC68HC16 is:

$$N_{\text{MC68HC16}} = 4 + 4 + 4 + 4 + 2 + 6 + 6 + 6 = 36 \text{ clock cycles} \tag{57}$$

which for a clock cycle time of 59.6 ns gives the addition execution time of 2.15 ms.

The non-preferred second approach that may be taken with the MC68HC16 is to implement signed addition with accumulators “D” and “E”. This method is inadvisable, however, because both registers are only 16 bits wide and shifting of the contents of either register to align scale factors results in loss of least significant bits.

5.3 A Comparison of Execution Times and Frequencies

A simple benchmark example of scaled addition has been shown with the TMS320 family and with Intel and Motorola microcontrollers. All three examples have the same numerical property of no quantization error; the result retains the 16-bit LSB portion of the 32-bit scaled addition result. The instruction times for the TMS320C10, TMS320C2XLP, and TMS320C50 are respectively 200, 35, and 25 nanoseconds. The

corresponding execution times and execution frequencies, which are the reciprocal of the execution times, for the benchmark scaled addition example are:

TMS320C10	TMS320C2XLP	TMS320C50
800 nanosec → 1.25 Mhz	140 nanosec → 7.1 Mhz	100 nanosec → 10.0 Mhz

The clock cycle times for the Intel and Motorola devices are 83 and 59.6 nanosecond, respectively. The corresponding execution times and execution frequencies for the Intel 90196 and Motorola MC68HC16 are:

Intel 80196	Motorola MC68HC16
$\left(1245 + \frac{83}{\text{shift}}\right)$ nanosec → .524 Mhz (for 8 shift*s*)	2145.6 nanosec → .466 Mhz

If these execution times are compared to the popular TMS320C2XLP, for example, it is seen that the ratio of execution time for the 80196 and MC68HC16 compared to the TMS320C25 is 1429/140 = 10.2X (assuming 8 shifts) and 2145/140 = 15.3X, respectively. If comparison is made with the TMS320C50, which is 1.4 times faster than the TMS320C2XLP, the equivalent ratios are 14.3X and 21.5X, respectively.

5.4 Summary

This paper has discussed the effects of finite word length in digital signal processors. Calculation of the effects of finite word length upon system transfer functions is given in Section 2, while Section 3 shows that the effect of a finite word length upon the signal is to introduce noise at various points in the signal path. Section 3 gives the tools for making trade-offs between system component quantization levels by being able to calculate the injected noise. Calculating quantization effects has limited usefulness, however, if a digital signal processor has scant ability to alleviate their causes. This paper describes, therefore, the abundant features of the TMS320 DSP family that promote a minimum quantization design. Finally, to demonstrate the performance usefulness of these features a comparison is made with two popular microcontrollers.

Acknowledgements

I wish to acknowledge the encouragements of Jaye Saha in the writing of this paper and the discussion with helpful suggestions from Russell Price and Peter Ehlig concerning the TMS320 DSP family.

References

1. Gene F. Franklin and J. David Powell, *Digital Control of Dynamic Systems*, Addison–Wesley, 1981, pp. 185–205.
2. Karl J. Astrom and Bjorn Wittenmark, *Computer Controlled Systems, Theory and Design*, Prentice–Hall, 1984, pp. 378–380.
3. Raymond G. Jacquot, *Modern Digital Control Systems*, Marcel Dekker, 1981, pp. 161–186.
4. Peter S. Maybeck, *Stochastic Models, Estimation, and Control Vol. 1*, Academic Press, New York, 1979, pp. 170–174.
5. Charles M. Rader and Bernard Gold, "Digital filter design techniques in the frequency domain," Proceedings of the IEEE, February 1967.
6. B.C. Kuo, *Discrete Data Control Systems*, Prentice–Hall, Inc., 1970, pp. 352–363.
7. B.C. Kuo, *Digital Control Systems*, Holt, Rinehart and Winston, Inc., 1980, pp. 694–718.
8. B. Widrow, "A study of rough amplitude quantization by means of Nyquist sampling theory," IRE Transactions on Circuit Theory, Dec. 1964, Vol. CT–3, pp. 266–276.
9. J.B. Slaughter, "Quantization errors in digital control systems," IEEE Transactions on Automatic Control, January 1964, Vol. AC–9, pp. 70–74.
10. R.B. Blackman, *Linear Data-Smoothing and Prediction in Theory and Practice*, Addison–Wesley, Reading, Mass., 1965.
11. George W. Johnson, "Upper bound on dynamic quantization error in digital control systems via the direct method of Liapunov," IEEE Transactions on Automatic Control, October 1965, Vol. AC–10, No. 4, pp. 439–448.
12. J.E. Bertram, "The effects of quantization in sampled-feedback systems," Transactions AIEE (Applications and Industry), September 1958, Vol. 77, pp. 177–181.
13. Pin Wah Wong, "Quantizations noise, fixed-point multiplicative roundoff noise, and dithering," IEEE Transactions on Acoustics, Speed, and Signal Processing, Vol. 38, No. 2, February 1990.
14. N.S. Jayant and L.R. Rabiner, "The application of dither to the quantization of speech signals," Bell System Technical Journal, July–Aug. 1972, Vol. 51, No. 6, pp. 1293–1304.
15. L. Schuchman, "Dither signals and their effect on quantization noise," IEEE Transactions Communication Technology, Dec. 1964, Vol. COM–12, pp. 162–165.
16. *PC-MatLab User's Guide*, The Math Works, Inc. Natick, Massachusetts, 1991.
17. Paul L. Meyer, *Introductory Probability and Statistical Applications*, 2nd Ed., Addison-Wesley, 1970.
18. Micheal O'Flynn, *Probabilities, Random Variables, and Random Processes*, Harper & Row, 1982.
19. Athanasios Papoulis, *Probability, Random Variables, and Stochastic Processes*, McGraw-Hill, 1984.
20. *TMS320C5x User's Guide*, Texas Instruments, 1993.
21. G. Troullinos, P. Ehlig, R. Chirayil, J. Bradley, D. Garcia, "Theory and implementation of a splitband modem using the TMS32010," *Digital Signal Processing Applications with the TMS320 Family, Vol. 2*, Texas Instruments, 1993.
22. *TMS320C5x User's Guide*, Texas Instruments, 1993.
23. *16-/32-Bit Embedded Processors*, Intel Corporation, 1990.
24. *CPU16 Central Processor Unit Reference Manual*, Motorola, May 1991.