August, 1981

LIDS-P-1117

ARCHITECTURAL ISSUES IN THE IMPLEMENTATION OF DIGITAL COMPENSATORS

by

Paul Moroney
Linkabit Corporation
10453 Roselle Street
San Diego, California 92121

Alan S. Willsky
Laboratory for Information and Decision Systems
Department of Electrical Engineering and Computer Sciences, MIT
Cambridge, Massachusetts 02139

Paul K. Houpt
Laboratory for Information and Decision Systems
Department of Mechanical Engineering
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

ABSTRACT

There are many techniques for designing discrete-time compensators. However, the digital implementation of such designs has not typically been addressed. The nature of digital hardware impacts the computational structure of the compensator and also can affect the original system design parameters. This paper deals with the architectural issues of serialism, parallelism and pipelining in implementing digital feedback compensators. The concepts of serialism and parallelism are shown to involve essentially the same considerations for digital compensators as for digital filters. However, the same cannot be said of pipelining, due to the feedback loop. A design technique is proposed for dealing with the problem of compensator pipelining, and several examples of pipelining LQG compensators are presented.

* This work was performed in part at the MIT Laboratory for Information and Decision Systems with support provided by NASA Ames under grant NGL-22-009-124 and in part at the Charles Stark Draper Laboratory.

I. INTRODUCTION

Control theorists have developed many methods for designing compensators for discrete-time systems. These include pole-placement concepts, optimal regulator theory, observer theory, Kalman filtering, and classical control approaches. Such designs have typically been implemented on large-scale computer systems. ~~However, the current trends are towards increased control~~ applications involving small-scale computers or dedicated digital hardware.

The implementation of control algorithms in such digital hardware has raised many new issues. These tend to fall into two categories, one involving the effects of the finite precision and fixed-point arithmetic of small-scale digital systems, and one involving architectural issues. Such questions have not generally been treated in the literature. Therefore some methodology must be established for digital feedback compensator implementation. In other words, we need some way to specify and order the critical computations that must take place in a compensator so that the resulting digital hardware performs as close to the ideal design as is consistent with the expense and speed requirements of the application.

We have addressed the issue of finite wordlength due to compensator coefficient rounding and multiplication roundoff in earlier works [1,2]. Our general approach has been to examine the concepts already developed for digital signal processing. Then, considering the compensator as a digital filter within a control loop, we can try to apply these ideas. However, the presence of the feedback path itself, and the emphasis on closed-loop performance, has frequently required us to adapt and extend the methods of digital signal processing.

In this paper, we will examine the architectural issues involved in compensator implementation, in particular, the notions of serialism, parallelism and pipelining. We will show that the serialism/parallelism concept is essentially identical to that involved in the implementation of any digital system. However, the use of pipelining in digital control systems raises several difficult questions. We will discuss these points in the context of linear-quadratic-Gaussian (LQG) control systems, although they extend easily ~~to more general cases.~~

The organization of this paper will be as follows. In Section II, we will briefly review the LQG control problem and describe the resulting ideal compensator equations. The notion of a compensator structure, which is somewhat different than a conventional filter structure, and an adapted notation for describing such structures will be reviewed from [1] and [2] in Section III. In Section IV, we will introduce and describe the notions of serialism, parallelism and pipelining for digital systems. The application of pipelining specifically to digital feedback compensators will be treated in Section VI. A typical application of pipelining for compensators will be described in Section VII, and several examples presented.

II. LQG COMPENSATOR DESIGN

In this section we will briefly review the single-input single-output steady-state LQG control problem and the optimal compensator that results. Let us assume that we wish to design a digital discrete-time compensator for a continuous-time system (plant), and that the control signal will be piecewise constant. We will also assume that the output of the plant is sampled at the rate 1/T. Given a quadratic performance index J, and a linear discrete-time model of a continuous-time plant subject to disturbances modeled as white Gaussian noise,the LQG compensator is the linear compensator that minimizes J.

Consider the following discrete-time model of a plant:

$$\begin{aligned} x(k+1) &= \Phi\, x(k) + \Gamma\, u(k) + w_1(k) \\ y(k) &= L\, x(k) + w_2(k) \end{aligned} \tag{1}$$

where x is the state n-vector, u and y are the control and output variables, Φ is the n×n state transition matrix, Γ is the n×1 input gain matrix and L is the 1×n output gain matrix. The quantities w_1 and w_2 are discrete Gaussian noises with covariance matrices Θ_1 (n×n) and Θ_2 (1×1) respectively. The performance index J is written as follows:

$$J = E\left\{ \lim_{l\to\infty} \frac{1}{2l} \sum_{k=-l}^{l} \left(x'(k)\,Q\,x(k) + 2x'(k)\,M\,u(k) + u'(k)\,R\,u(k) \right) \right\} \tag{2}$$

Thus J reflects the weighted squared deviations of the states and control. The parameters Q, M and R can be specified by the designer.

The determination of a linear compensator that minimizes J involves the solution of two Ricatti equations involving the plant and weighting parameters. However, typically the resulting control u(k) will depend on past values of the plant output up to and including y(k) [3]. Unfortunately, the resulting compensator is not directly feasible for implementation, since a certain amount of time must be allowed to compute u(k) from y(k), y(k-1), etc. Yet u(k) and y(k) refer to the control and plant output at identical times. Some delay must be accounted for and thus the design as described so far is unfeasible.

Fortunately, Kwakernaak and Sivan [4] have presented a design procedure that does account for this delay. The resulting compensator is optimal in the sense that it produces the u(k) that minimizes J, but based only on a linear function of y(k-1), y(k-2)..., and not on y(k). Such a compensator can be implemented, essentially allowing one full sample period for the computation of u(k) after the y(k-1) sample is generated. (If, however, the computation time is much shorter than the sample interval, this implies some inefficiency; the output u(k) will be available long before it is used as a control. Thus Kwakernaak and Sivan also describe a method for skewing the sample time of the plant output with respect to the rest of the compensator, which eliminates the inefficiency.)

The optimal compensator described above is of the following form:

$$\begin{aligned} \hat{x}(k+1) &= \Phi\hat{x}(k) + \Gamma u(k) + K\left(y(k) - L\hat{x}(k)\right) \\ u(k+1) &= -G\hat{x}(k+1) \end{aligned} \tag{3}$$

where $\hat{x}$ is the estimated state vector, and the $n\times 1$ Kalman filter matrix K and $1\times n$ regulator matrix G result from the solution of two discrete-time algebraic Ricatti equations [4]. Note that in equations (3), the next control u(k+1) depends only on inputs y(k), y(k-1)... . Thus the computational delay has been allowed for in this formulation.

Now if we treat this compensator as a discrete linear system and examine its transfer function, we have:[1]

$$\frac{U(z)}{Y(z)} = -G(z-\Phi+KL+\Gamma G)^{-1}K \qquad (4)$$

In a more conventional form, this can be written:

$$\frac{U(z)}{Y(z)} = \frac{a_1 z^{-1}+a_2 z^{-2}+\ldots+a_n z^{-n}}{1+b_1 z^{-1}+b_2 z^{-2}+\ldots+b_n z^{-n}} \qquad (5)$$

Note the lack of a term a_0 in the numerator. The presence of such a term would reflect a dependence of the present output on the present input. Since (5) represents a compensator that can be implemented, the a_0 term must be zero.

This delay has an important implication in the way we look at structures for implementing digital compensators, as we will show in Section III.

[1]Note that we have taken u to be the output of the digital network and y to be the input. This may be contrary to the expectations of some readers.

III. STRUCTURES FOR DIGITAL COMPENSATORS

In the nomenclature of digital signal processing [5,6] the term structure refers to the specific combination of (finite-precision) arithmetic operations by which a filter output sample is generated from intermediate values and the input. Typically, a structure can be represented by a signal-flow graph. Let us examine a simple digital filter structure to see whether it will be appropriate for representing the compensator of (5).

Specifically, let us examine a fourth-order (n=4) direct form II filter structure [5,6]: (See Figure 1, with transfer function (6)).

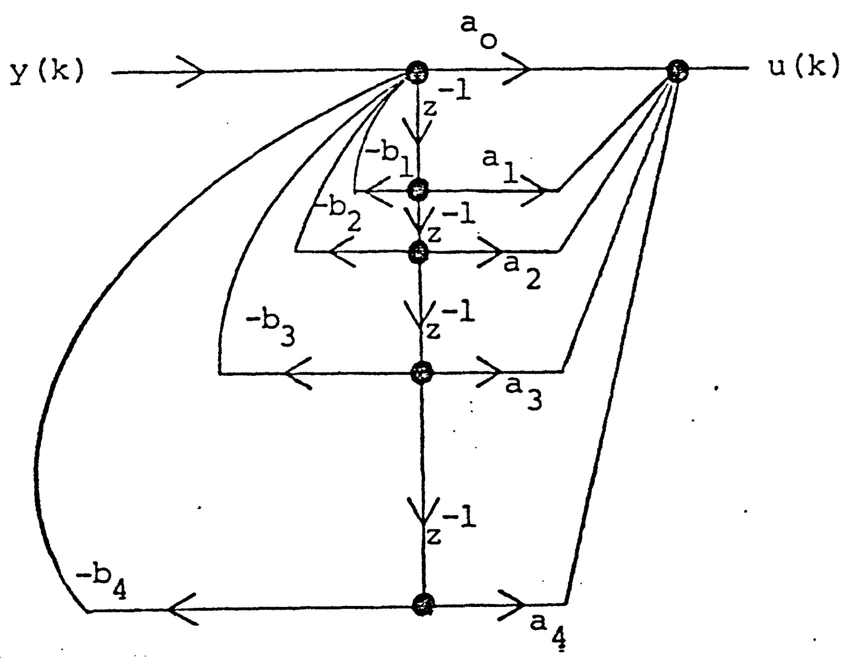


Figure 1: Direct Form II Filter Structure

$$\frac{U(z)}{Y(z)} = \frac{a_o + a_1 z^{-1} + a_2 z^{-2} + a_3 z^{-3} + a_4 z^{-4}}{1 + b_1 z^{-1} + b_2 z^{-2} + b_3 z^{-3} + b_4 z^{-4}} \tag{6}$$

Note the presence of the a_0 term. Such a structure cannot exactly represent an implementation, since computational delay (as discussed in Section II) has not been accounted for. However, such a signal-flow graph _is_ taken to represent a structure in digital signal processing; basically, the extra series delay needed for computations is assumed to be present, and is ignored. In most digital filter applications, series delay is of no consequence. However, in any control system, all delays that exist _must_ be adequately represented in the structure notation. If series delay exists in the compensator and has _not_ been accounted for, the entire control system may be unstable. Thus any treatment of compensator structures must include specification of _all_ calculation delays. This consideration basically led to the form of equation (5).

Now, let us take Figure 1 and set a_0 to zero, as in equation (5). (See Figure 2)

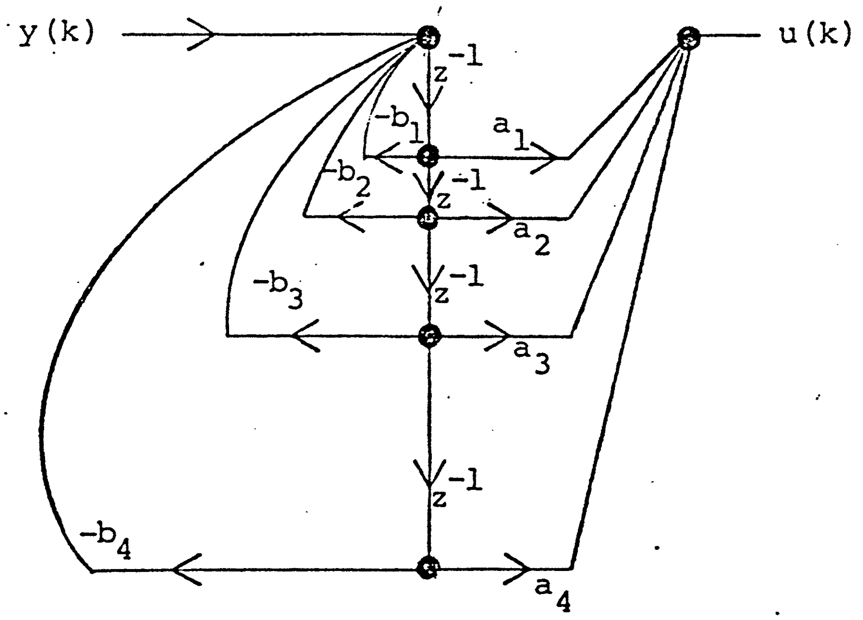


Figure 2: Direct Form II Filter Structure (a_0=0)

The signal-flow graph of Figure 2 is still not an accurate representation of an implementation of equation (5). The only time available for computation is between sample times. Yet Figure 2 shows u(k) depending on compensator state nodes (defined to be the outputs of delay elements), also at the same time k. Time must be allowed for the multiplications a_1 through a_4. Thus u(k) cannot be in existence until after the state node values are calculated.

A structure for implementing equation (5) is depicted in Figure 3. This can be derived from Figure 2 by elementary signal-flow graph manipulation.

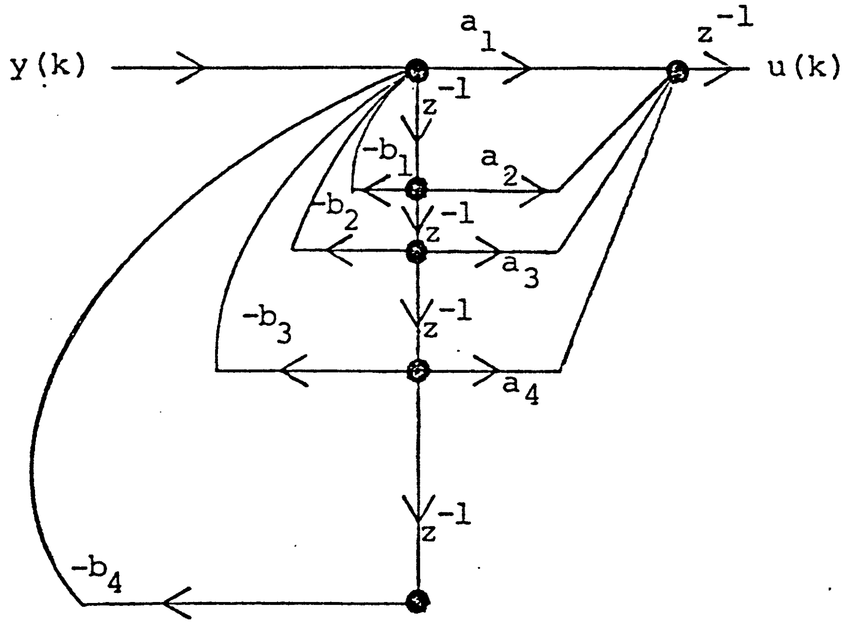


Figure 3: Direct Form II Compensator Structure

For controller implementations, this will be defined as the 'direct form II' structure. One clear result emerges; a unit delay must precede the u(k) node. Thus the u(k) node is always a compensator state node. Note that this organization of the computations was only possible due to the zero value of a_0. Thus our design procedure, allowing u(k) to depend only on past y values, results in a controller which can be implemented if we are careful to include all the actual delays inherent in the structure.

In addition to representing compensator structures with the signal-flow graph, we need a mathematical notation for describing a structure. In order to accomplish this, we will adapt the filter notation developed by Chan [5] to the case of compensator structures. Chan's notation accounts for the specific multiplier coefficients in the structure, and for the exact sequence, or precedence, to the computations and quantizations involved. Using y and u to represent a filter output and input respectively, and v the filter states (delay-element outputs), the Chan notation can be written as:

$$\begin{bmatrix} v(k+1) \\ y(k) \end{bmatrix} = \Psi_q \Psi_{q-1} \cdots \Psi_1 \begin{bmatrix} v(k) \\ u(k) \end{bmatrix} \tag{7}$$

Each (rounded) coefficient in the filter structure occurs once and only once as an entry in one of the Ψ_i matrices. The remainder of the matrix entries are ones and zeros. The precedence to the operations is shown by the ordering of the matrices. The operations involved in computing the intermediate (non-state) nodes

$$r_1(k) = \Psi_1 \begin{bmatrix} v(k) \\ u(k) \\ y(k) \end{bmatrix}$$

are completed first, then $r_2(k) = \Psi_2 r_1(k)$ next, and so forth. The parameter q specifies the number of such precedence levels.

For representing compensator structures as discussed above, several changes are necessary. First, u and y are reversed in definition: u is now the compensator output, and y the input. But more importantly, the u(k) node is now a state of the compensator. Inclusion of these changes produces the following modified state space notation:

$$\begin{bmatrix} v(k+1) \\ u(k+1) \end{bmatrix} = \Psi_q \Psi_{q-1} \cdots \Psi_1 \begin{bmatrix} v(k) \\ u(k) \\ y(k) \end{bmatrix} \tag{8}$$

Examples of the modified state space representation can be found in Section IV and in [1,2].

IV. SERIALISM, PARALLELISM, AND PIPELINING

In this and the following sections, we will examine the architectural issues involved in the implementation of digital feedback compensators. We will show that the basic concepts of serialism and parallelism as they apply to digital filter structures represented in Chan's notation extend without modification to digital compensator structures represented in the modified state space notation. However, the same cannot be said concerning the application of pipelining techniques to compensators. In fact, we will show that pipelining in control systems brings out another important issue: the interaction between the ideal design procedure described in Section II and the implementation of the resulting compensator.

Perhaps the most basic issue in any consideration of digital system architecture involves the concepts of serialism and parallelism. Essentially, this notion involves the degree to which processes, or operations, in the system run in sequence (serially) and the degree to which they execute concurrently (in parallel). At one extreme, any system can be implemented with a completely serial architecture -- executing all its processes one at a time. This procedure requires the minimum number of actual hardware modules and the maximum amount of processing time for completion of the system task. On the other hand, any system can also be implemented with a maximally-parallel architecture, having as many concurrent processes as possible. Such a design requires the maximal amount of hardware, but completes the overall system task in minimum time. Thus, the serialism/parallelism tradeoff is another example of the frequently encountered space-time tradeoff [8].

There is an important asymmetry implicit in the exploitation of serialism and parallelism. It is always possible to execute processes one at a time (totally serially). However it is not always possible to execute them all at once (in a totally parallel manner). There is a minimum amount of serialism required. Figure 4 gives a typical example, consisting of three processes

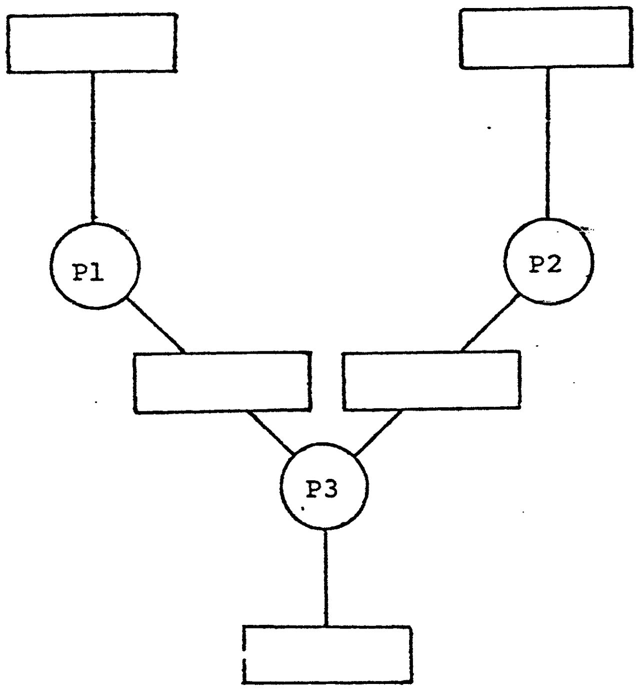


Figure 4: Three-Process System

(P1, P2, and P3), and data cells [8] for input, output, and intermediate results. Assume that each of the three processes require t seconds for completion (given specific hardware modules) and that each process executes as soon as all of its inputs are valid. Given general-purpose computing modules, then clearly a serial architecture that would use one module only and require 3t seconds to complete the overall task is possible. However, a totally parallel architecture (total time t with 3 hardware modules) is not possible; Figure 4 clearly shows that processes P1 and P2 must be finished before process P3 can begin. Consequently, only processes P1 and P2 can operate in parallel. For such a maximally-parallel architecture, two hardware modules would be required, and the total computation time would be reduced to 2t seconds.

Under certain conditions, this 'speed barrier' can be broken through the use of pipelining. If the original objective of the system is to perform a task repeatedly (as soon as the present task is completed, a new task begins), then pipelining could realize an effective throughput rate equal to (or at least closer to) that of a totally-parallel architecture. Reconsider Figure 4. Suppose that a separate hardware module is reserved for each process, the input data rate is $\frac{1}{2t}$, and the maximally-parallel 2t second architecture is used. The input and output data cells now represent registers clocked at rate $\frac{1}{2t}$, while the intermediate result registers are clocked t seconds after the input. Let us examine any 2t-second interval. During the first t seconds, module 3 (for executing process P3) will be idle, since its inputs are not yet valid. During the last t seconds, while module 3 is active, modules 1 and 2 will be idle. The total 2t second time from a task initiation until its completion cannot be reduced without faster hardware modules. However, the idle modules can be put to use by pipelining the processes. While module 3 is active and modules 1 and 2 otherwise idle, the next task may as well begin and use modules 1 and 2. The net result for this example is a doubling of the throughput rate (task completions per second) from $\frac{1}{2t}$ to $\frac{1}{t}$. It must be stressed here that any given task still takes 2t seconds from start to finish; however, successive task completions occur at t second intervals. In terms of hardware required, the pipeline would be effected by the presence of the third hardware module and by clocking all the above-mentioned registers at rate $\frac{1}{t}$.

Figure 5 shows two ways of modeling the pipelined case for this example. Basically, the pipeline splits a larger task not implementable in a totally-parallel architecture into smaller sequential sub-tasks, each of which can be implemented in a more parallel fashion (Figure 5a). An equivalent viewpoint (Figure 5b) considers pipelining to be represented by a faster-executing task coupled with some series delay.

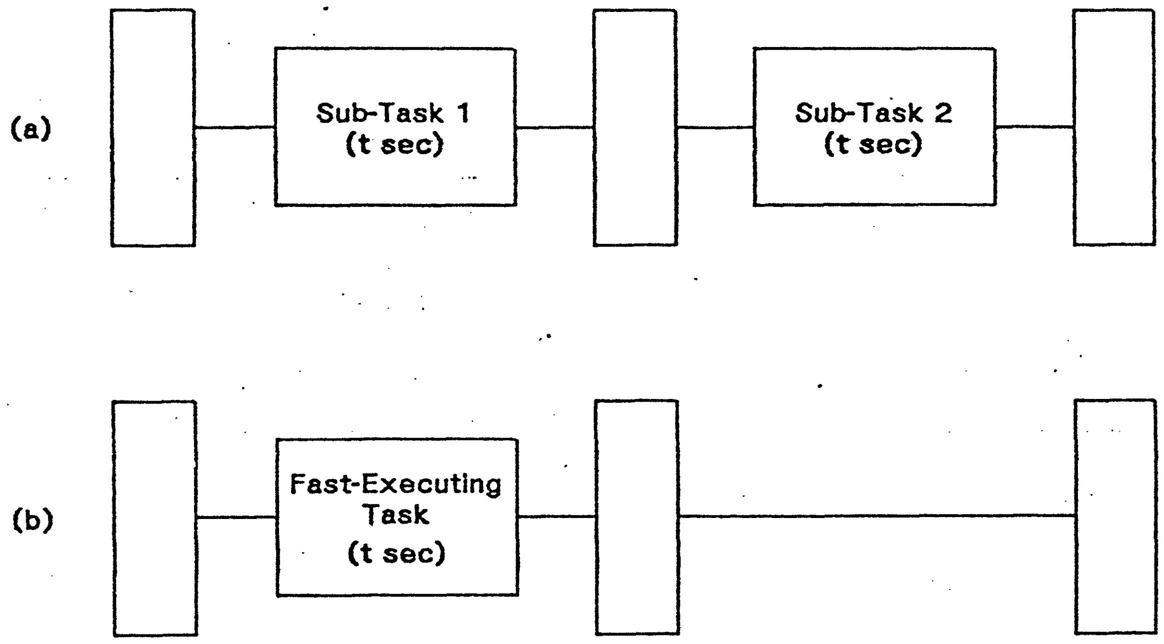


Figure 5: Models For Pipelining

An important example of pipelining is in the implementation of digital filter structures [9,10]. In such a case, the system task corresponds to the generation of a filtered output value from an input sample, and the individual processes correspond to the hardware digital multiplications and additions that exist in the particular structure (algorithm) implemented (ignore A/D and D/A operations for now). Figure 6a shows a two-pole digital filter with input y and output u. As shown, the unit (t second) delay z^{-1} can be implemented as a storage register. Thus, however they are implemented, all the arithmetic and quantization operations and the intermediate storage required must be completed in one sampling period. Computing the signal u(k+1) at node A in Figure 6a requires three multiplications and an addition. Assume that we have selected an architecture where the multiplications involving b_1 and b_2 operate in parallel, then the addition occurs, and finally the multiplication by a_1. Using three

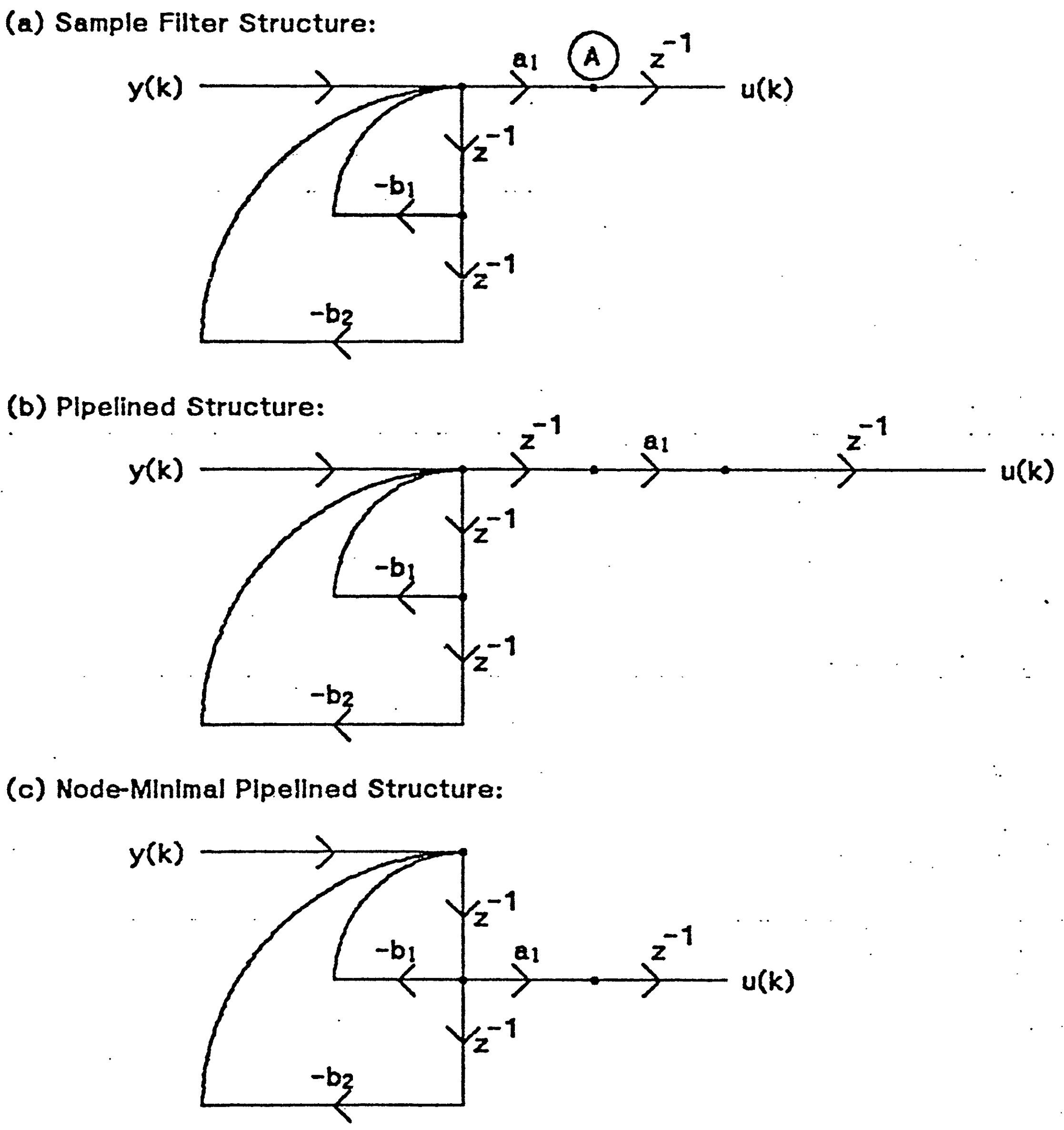


Figure 6: Pipelining a Simple Digital Filter

hardware multipliers instead of two, and assuming negligible add time, the multiply operations can be pipelined and the sample rate doubled. Since the new pipelined configuration operates at a doubled rate, each unit delay now represents half the delay time as in the original structure. Thus an additional series unit delay must be shown for the pipeline, since the total time for completing the three multiplications has not changed. The new signal-flow graph of Figure 6b can be simplified, since it contains two states that are exactly equivalent. Removal of one of these states produces the node-minimal signal-flow graph shown in Figure 6c. Thus, the pipelined structure of Figure 6c has the same number of unit delays (storage resisters) as the original structure in Figure 6a. For this particular example, pipelining did not require an overall increase in the number of unit delays. This would not be true in general. In terms of the signal-flow graph, pipelining has essentially created a new structure.

From the example of Figure 6, it is clear that pipelining ties in closely with the digital filter notion of precedence. Specifically, let us consider node precedence, that is, the precedence relations involved in the addition, multiplication, and quantization operations needed to compute the node signals. In this case, the modified state space representation (See Section III) is very convenient since it explicitly shows the number of precedence levels involved. If a structure represented in this notation has only one precedence level, then it can have a totally-parallel architecture (parallel in terms of the multiply/add computations involved in each precedence level). If more than one such level is required, no totally-parallel architecture is possible, and the number of levels q will equal the minimum degree of serialism required. Pipelining, if applicable, would actually change the structure by inserting unit delays so that a new structure (one with fewer levels and thus a faster sample clock rate) is formed. The pipelined structure would have the same transfer function as the original non-pipelined structure, except for some series delay, and would probably have more state nodes. Series delay is of little consequence in most digital filtering applications. Thus a two-level structure can be designed for a sampling period of $\frac{T}{2}$ even though the calculations require T seconds, since pipelining (given a two-level structure) will fit the calculations into a $\frac{T}{2}$ slot at the expense only of a series delay of $\frac{T}{2}$ seconds. Equations (9) through (12) show the modified state space representations and transfer functions of the non-pipelined (sampling period T) and pipelined (sampling period $\frac{T}{2}$) filters of Figure 6a and 6c respectively:

$$\Psi_2\Psi_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & a_1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 & 0 \\ -b_2 & -b_1 & 0 & 1 \end{bmatrix} \tag{9}$$

$$H_{np}(z) = \frac{a_1 z^{-1}}{1 + b_1 z^{-1} + b_2 z^{-2}} \tag{10}$$

$$\Psi_1 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -b_2 & -b_1 & 0 & 1 \\ 0 & a_1 & 0 & 0 \end{bmatrix} \tag{11}$$

$$H_p(z) = \frac{a_1 z^{-2}}{1 + b_1 z^{-1} + b_2 z^{-2}} \tag{12}$$

Note the reduction from two to one levels (see (9) and (11)) allowing the doubled sampling rate, and also the extra z^{-1} factor in the numerator of (12). The number of states in (11) remained at three due to the simplification possible in Figure 6b.

Let us now consider pipelining as it applies just to the multiply operations in a structure. Such a consideration will be valuable whenever the multiply time dominates over all the addition and quantization operation times in a structure, a situation that is not uncommon in microprocessor-based digital systems. Since we are neglecting all calculation times other than the multiply times, it is sufficient to know the precedence to the multiply operations alone in order to determine the architectures that are possible. Thus the node precedence evident from the different Ψ_i matrices of a modified state space representation will not be adequate to describe the multiplier precedence relations [9]. Such relations can be determined from the signal-flow graph or from an examination of the specific location ~~of each multiplier coefficient in the Ψ_i matrices~~. In either case, the multipliers can be grouped into precedence classes. Frequently, the number of multiplier precedence classes and node precedence levels will be the same, but the multiplier coefficients in class 1 (of highest multiplier precedence) and the multiplier coefficients in node precedence level 1 (the matrix Ψ_1) need not be identical. It will be true that all the multiplier coefficients in the matrix Ψ_1 will also be in multiplier precedence class 1. Furthermore, multiple-level structures often have fewer multiplier classes than node precedence levels.

As an example, consider the cascade structure of Figure 7 and its modified state space representation (13). For this example, assume all scaling multipliers to be simple shifts (powers of two); thus they will not

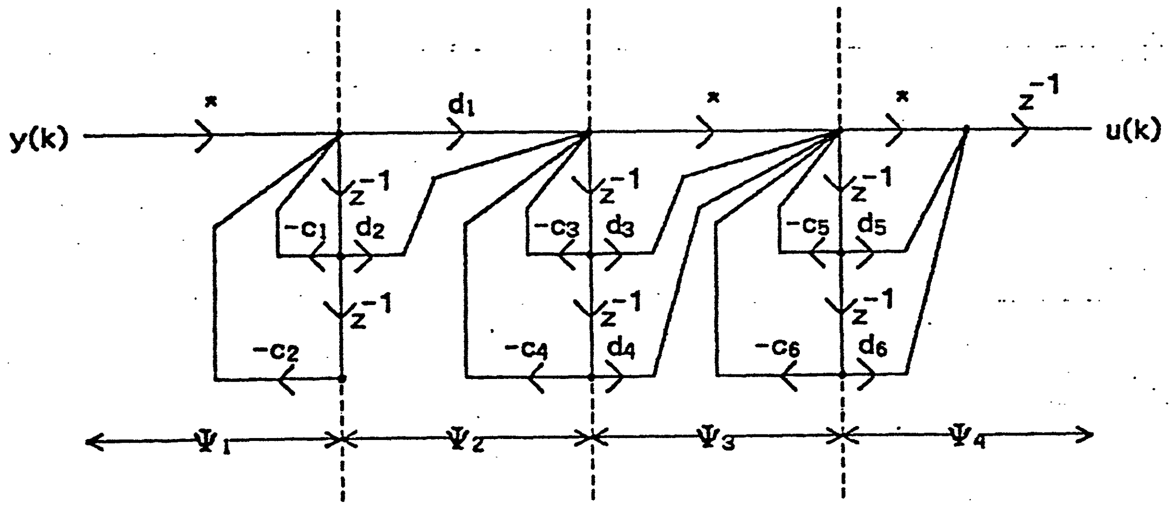


Figure 7: Cascade Structure (Direct Form II)

$$\Psi_4 = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & d_6 & d_5 & 1^* \end{bmatrix}$$

$$\Psi_3 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & d_4 & d_3 & -c_6 & -c_5 & 1^* \end{bmatrix} \quad \Psi_2 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ d_2 & -c_4 & -c_3 & 0 & 0 & d_1 \end{bmatrix} \qquad (13)$$

$$\Psi_1 = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ -c_2 & -c_1 & 0 & 0 & 0 & 0 & 0 & 1^* \end{bmatrix}$$

involve hardware multipliers. All the multiplications of coefficients by state node or input signals can occur immediately after each sampling instant and therefore fall in multiplier precedence class 1. Thus the c_1, c_2, c_3, c_4, c_5, c_6, d_2, d_3, d_4, d_5, and d_6 multiplies can operate in parallel given enough hardware multiplier modules. Only the d_1 multiplication lies in class 2; it must await the completion of the c_1 and c_2 multiplies. Of course, given the two classes and 12 multiplies, an optimal, that is maximal, use of the hardware is made with only 6 hardware multipliers (assuming no pipelining). Five of the class 1 multiplies (but not c_1 or c_2) would be computed in the second multiply cycle with the d_1 multiply. Thus the cascade of Figure 7 has two multiplier precedence classes, although it has four node precedence levels. This notion of multiplier precedence is more completely formulated in [9], but the basic conclusion is as follows: although the modified state space representation correctly describes the operations that must occur in computing the node values within a structure, the multiplier precedence relations (more easily seen directly from the signal-flow graph) are more significant for determining the possible hardware architectures when the multiply time is dominant.

Certain basic restrictions [9] must be observed when pipelining a complex structure. The first limitation in applying pipelining concerns parallel data paths within the structure. Whenever any portion of a system is pipelined to increase the sampling rate (which adds unit delays), all parts of the system that feedforward in parallel with the pipelined portion must receive equivalent delays. Otherwise, the overall transfer function of the pipelined system will be quite different from the original; the differences will be more than just a series delay. The second difficulty encountered in applying pipelining techniques involves feedback. Suppose there exists a series of operations which makes up part of a closed feedback loop within a structure. Pipelining these operations would result again in a very different transfer function. Unfortunately, there is no simple way around this problem. Consequently, pipelining within a feedback loop is typically avoided. Examples of feedforward and feedback in the application of pipelining can be found in [1].

V. PIPELINING FEEDBACK COMPENSATORS

In the context of the control problem formulated in Section II, the ideas of serialism and parallelism apply unchanged to the implementation of digital controller architectures. However, since a global feedback loop exists around the entire compensator (that is, through the plant) pipelining seems to be out of the question, as described above. Suppose that we design an LQG compensator for a system with a sampling rate of $\frac{2}{T}$, the resulting compensator has two multiplier precedence levels, and the multiply time t_m equals $\frac{T}{2}$. Pipelining would seem to be necessary unless we were willing to drop the sampling rate to $\frac{1}{T}$. Unfortunately, the series delay that would result from pipelining this compensator would introduce an unplanned-for pure time delay. The deleterious effects of pure time delay (linearly-increasing negative phase shift) on the stability and phase margin of a feedback system are well known. Even if instability does not result, the performance index J will be larger than expected and the qualitative dynamic performance will be compromised.

Fortunately, there is an approach to pipelining that will be effective for control systems. Consider the LQG system and compensator design technique described in Section II. Assume that for some original controller design, the sampling interval is not long enough to complete all the calculations involved in the compensator (which is the situation as described above). In principle, pipelining techniques could help, but unavoidable delay would be introduced. An effective use of pipelining simply means that we somehow include this unavoidable delay in the original design procedure. This aim can be realized through state augmentation [4]. Suppose that pipelining would allow a factor of two increase in the sampling rate, thus adding only a single series delay. If the plant is described at the doubled sampling rate $\frac{2}{T}$ by (14):

$$\begin{aligned} x(k+1) &= \Phi\, x(k) + \Gamma\, u(k) + w_1(k) \\ y(k) &= L\, x(k) + w_2(k) \end{aligned} \tag{14}$$

(the matrix parameters above depend on T) then, preceding u(k) with the series delay to form $\tilde{u}(k)$, the augmented plant can be modeled as follows (see Figure 8):

$$\begin{aligned} \tilde{x}(k+1) &= \begin{bmatrix} \Phi & \Gamma \\ 0 & 0 \end{bmatrix} \tilde{x}(k) + \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tilde{u}(k) + \begin{bmatrix} w_1(k) \\ 0 \end{bmatrix} \\ y(k) &= \begin{bmatrix} L & 0 \end{bmatrix} \tilde{x}(k) + w_2(k) \end{aligned} \tag{15}$$

where $\tilde{x}(k+1) = \begin{bmatrix} x(k+1) \\ u(k+1) \end{bmatrix}$. For this augmented system, the weighting matrices Q and M in the expression for the performance index (2) must also be augmented, adding an all-zero row and column to Q, and a single zero element to M. The weighting parameter R will be the same as for the system (14). Now we must treat (15) as a new system and design an LQG compensator for it. Then <u>that</u> design can be pipelined, which introduces the inherent added delay shown in Figure 8.

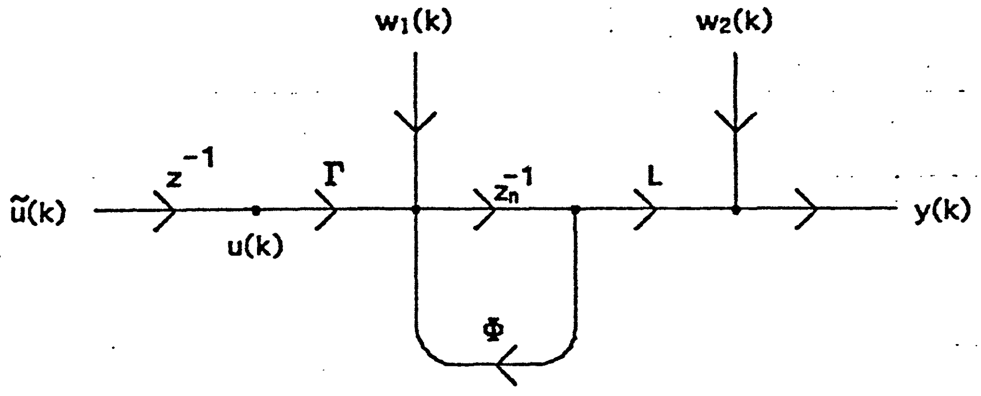


Figure 8: State Augmentation for Control System Pipelining

For this situation, two observations can be made. First, the Kalman filter portion of the LQG design for (15) will have what seems to be a difficulty due to the added delay -- the numerical routines blow up. Common sense dictates however that there is no need to estimate $\tilde{x}_{n+1}(k) = u(k)$ since it is the actual plant input, which is known. Thus we need only estimate $\tilde{x}_1(k)$ through $\tilde{x}_n(k)$, namely the vector $x(k)$. That estimation problem has already been solved as the nth-order Kalman filter for (14), with gains k_1 through k_n. Using these results, the optimal filtering gains for the augmented system (15) can be written:

$$\tilde{k} = \begin{bmatrix} k_1 \\ k_2 \\ \vdots \\ k_n \\ 0 \end{bmatrix} \tag{16}$$

The (n+1)th-order optimal regulator problem for (15) can be solved with no difficulty at all.

The second observation that we can make for this augmented-system pipelining technique involves the consistency of the design technique. A delay-canonic structure (a structure having a minimal number of unit delays) [1] for the optimal LQG compensator for (15) will be of order n+2 since (15) is of order n+1, and not of order n+1 as is the canonic compensator structure for (14). Thus this approach to controller pipelining gives rise to a compensator of higher dimension (more poles), requiring more states (delay elements) and more coefficients. Along with this increase in order comes a more important point -- the new higher dimensional compensator structure must allow the same degree of pipelining as the original structure, or the whole controller pipelining design procedure is invalid - that is, inconsistent.

This point is especially of concern when using structures whose number of precedence levels is a function of the number of compensator states (for example, the cascade forms). As an example, consider a second-order plant and a direct form II compensator structure, which requires three delays and two precedence levels. To exploit pipelining, we must augment the plant and redesign the compensator -- its direct form II structure now requires four delays (states). There would still be only two precedence levels as before, so pipelining to double the sampling rate will work as planned. However, if we decide to use a cascade of two direct form sections (assume one second-order section, one first-order section, and general scaling multipliers), then the result is three precedence levels. Pipelining to allow the $\frac{2}{T}$ sampling rate will not now result in the effect of a single added unit delay as assumed, but will involve two series unit delays, making the design procedure invalid. In other words, if we implemented the pipeline as described above, the system would not perform as expected; more delay would be present in the loop than had been accounted for in the design. Such problems can be avoided with a proper choice of structure.

There is one positive note associated with the increased dimensionality of the compensator, and it is related to the particular form of (16). Usually, an increase in dimension (number of states) by one involves at least two additional coefficient multipliers. (A fifth-order plant requires a compensator with at least ten coefficients, a sixth-order plant requires one with twelve coefficients, etcetera [1].) However, by virtue of the zero entry in (16), the general form of the compensator transfer function for the augmented system is simpler, involving only one additional coefficient [1]. This fact helps make the pipelining approach a bit more attractive, at least with certain structures(for example, any direct form and any cascade or parallel structure based on a direct form.)

One last general point should be mentioned. The application of any pipelining technique or the use of parallelism to increase the sampling rate is desirable only if it allows a decrease in the performance index J, or in whatever gauge of system performance one accepts. However, not all sytems have a performance measure that decreases (improves) monotonically with decreasing T [11]. Intuitively, any system with sharp resonances will lose controllability (implying a large J) when the sampling frequency is near a resonance. One must be aware of such cases. If such a case does not occur, then pipelining will reduce the performance index, although certainly not as much as the (non-implementable) straightforward rate-$\frac{2}{T}$ LQG compensator design which adds no delay. Whether this pipelining approach is effective enough to warrant the higher-order compensator depends on the designer's particular application.

VI. CONTROLLER I/O PIPELINING

One common application of pipelining in a feedback environment involves the often time-consuming compensator input/output (I/O) operations, namely, the sampling and the A/D and D/A conversion operations. Let us assume that a structure with one multiplier precedence level (for example, the block optimal parallel structure [1]) is choosen to implement a compensator, and that a totally-parallel architecture is used for the multipliers involved. Assuming negligible D/A time and add time, the compensator can then be modeled as a two-process task: A/D conversion and hardware multiplication. Further assume (for simplicity) that each process requires t seconds. Without pipelining, the sampling rate must be no greater than $\frac{1}{2T}$. If we now pipeline these processes, a factor of two increase in throughput and sampling rate is possible. At each sample time, sampling and A/D conversion of a new y sample would begin. Then t seconds later the structure multiplications could begin, overlapping the next sampling and A/D operation. Note that the hardware multipliers and A/D convertor will now be active 100% of the time. We can represent this pipelined system (sample rate $\frac{1}{T}$) as the designed compensator structure followed by a series unit delay resulting from the pipeline.

If we apply the design technique outlined in Section V to produce a (pipelineable) compensator for this I/O case, the order of the compensator will of course be one greater than the non-pipelined design, implying at least one additional state and coefficient. No matter what the plant dimension may be, a block optimal parallel structure will have only one precedence level [1]. Thus, I/O pipelining with a one-level compensator structure results in a valid design procedure.

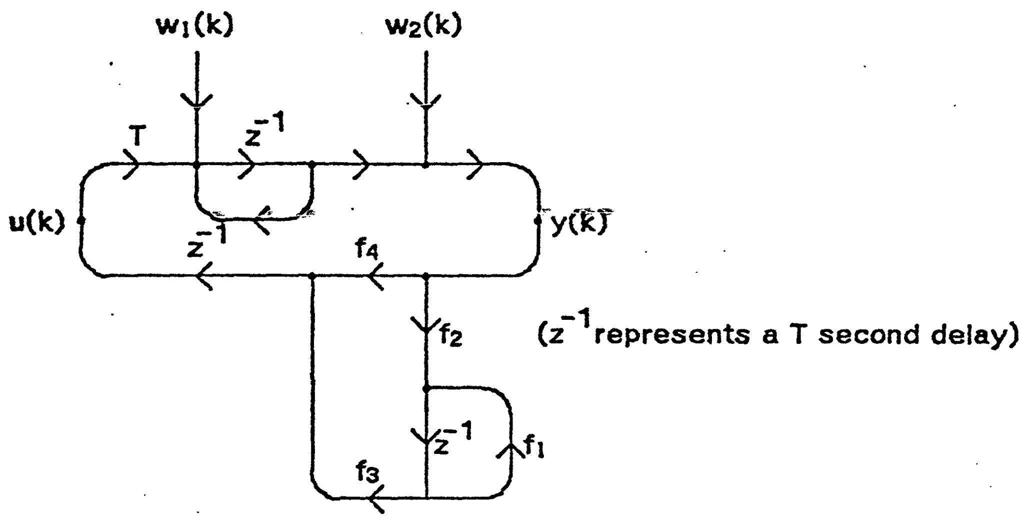
Four examples have been selected to illustrate what can occur with compensator (I/O) pipelining. Each example consists of four cases. Case 1 represents the plant discretized at a T second sampling period with its corresponding LQG compensator (no pipeline). Case 2 represents the plant discretized at a $\frac{T}{2}$ second sampling period with its corresponding LQG compensator. This case does not include any pipelining, but is not physically implementable due to the short sampling interval. The performance index for this case constitutes an unreachable lower bound to the performance of the augmented-plant approach to pipelining (case 3). Case 4 (blind pipelining) results when the compensator designed for case 2 is pipelined in order to make it physically implementable. Thus the delay due to the pipeline is ignored in the pipelined design, usually resulting in a performance level that is worse than the non-pipelined level (and perhaps even in a system that is unstable). Assuming that J is a monotonic increasing function of T, we can expect that the different cases will rank, from highest J to the lowest, as follows: case 4, case 1, case 3, case 2. (It is possible but unlikely that case 4 could have a lower J value than case 1.) Remember, however, that case 2 is not implementable.

The simplest I/O pipelining example consists of a single-input, single-output, single-integrator plant:

$$\begin{aligned} \dot{x}[t] &= u[t] + w_1[t] \\ y[t] &= x[t] + w_2[t] \end{aligned} \tag{17}$$

where T = 6 seconds. Fixed parameters were selected for the continuous-time performance index and noise intensities, and then discretized. Details can be found in [1]. Figure 9 illustrates the discretized system and the form of the compensator before pipelining (case 1) and after pipelining through state augmentation and redesign (case 3). A one-level version of direct form II structure [1] is used for the compensator. Note the extra unit delay

(a) Rate 1/T system, T=6 (case 1)



(b) Pipelined System, rate 2/T (case 3)

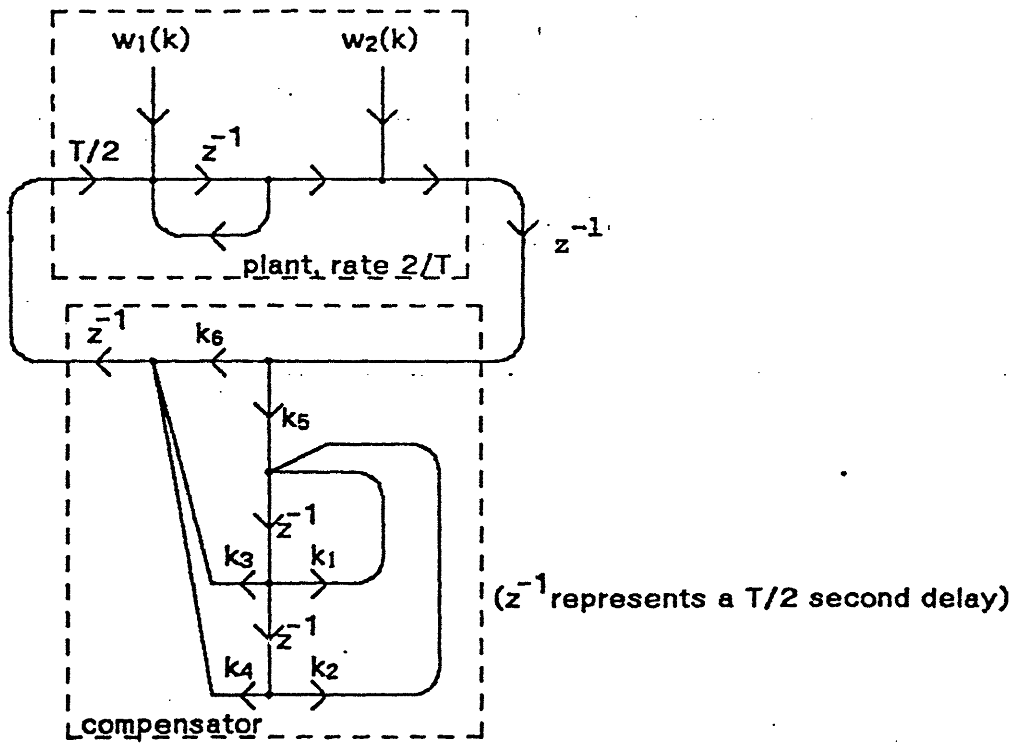


Figure 9: Compensator I/O Pipelining for the Single-Integrator Plant

in Figure 9b, as mentioned earlier in this section. The form of the system for case 2 would look the same as that in Figure 9a; however the gains of all the branches would differ. For case 4, we need only add one series delay to the signal-flow graph of case 2.

Three other examples are also considered; a double-integrator plant, a two-state harmonic oscillator plant, and a sixth-order plant derived from the longitudinal dynamics of the F8 fighter aircraft (see [1]).

The performance indices for all the various cases are shown in Figure 10.

Key:

Case 1 — rate 1/T system
Case 2 — rate 2/T system (not implementable)
Case 3 — rate 2/T pipelined system designed via state augmentation
Case 4 — blind pipelining

example plant	T	Case 4	Case 1	Case 3	Case 2
single integrator	6	(unstable)	2.42	2.05	1.34
double integrator	6	(unstable)	328	179	53.2
harmonic oscillator	6	(unstable)	32.7	12.9	9.72
6-state F8 plant	1	.0038	.00312	.00282	.00222

Figure 10: Compensator I/O Pipelining

Under case 4 we see the consequences of pipelining and ignoring the delay incurred. Three of the example systems actually became unstable, and with the fourth, the index J <u>increased</u>. As expected, all the case 2 indices were lower than case 1, with case 3 lying between the two. To judge the effectiveness of the state-augmentation pipelining method of case 3, one must examine the degree of improvement in J relative to the possible improvement (the difference between cases 1 and 2). The best improvement shown was for the harmonic oscillator, which is no surprise since the oscillator's natural frequency of $\frac{1}{2\pi}$ radians/second is close to the unpipelined sampling rate $\frac{1}{T}$. The remaining three examples also showed significant improvement. Again, whether or not the pipelineable compensator (with doubled sampling rate, one extra state and at least one extra coefficient) is to be used will depend on the particular level of performance desired, the penalty involved in complicating the hardware, and the various system sampling rate requirements.

VII. SUMMARY

In this paper we have investigated certain architectural issues associated with the implementation of digital feedback compensators. Whenever possible we have drawn on the field of digital signal processing for techniques and approaches to these issues. However the presence of a feedback loop around the digital compensator has frequently required us to modify and extend such techniques.

We have chosen the single-input, single-output, steady-state LQG control problem as a context in which to present our results, although the techniques developed usually extend to more general control systems [1].

The concept of a 'structure' for implementing digital compensators has been presented, along with a convenient and accurate notation. In Section IV we introduced the architectural notions of serialism, parallelism, and pipelining in digital systems, and explained the hardware cost/execution time tradeoff tied to these issues. The issues of serialism and parallelism were shown to involve basically the same considerations for digital compensators as for digital filters, while the issue of pipelining was more complex. Specifically, the extra delays incurred due to the use of pipelining had a deleterious effect on the performance of the feedback system. In Section V a design technique based on state-augmentation was developed for dealing with the problem of control system pipelining. Finally, the last section treated a typical application of pipelining techniques to microprocessor-based control systems. For this application, the compensator A/D input operations and multiply operations could be pipelined to realize a doubling in the system sampling rate. Four examples were presented to illustrate the technique.

REFERENCES

[1] P. Moroney, "Issues in the Digital Implementation of Control Compensators," Ph.D. Dissertation, MIT, Dept. of EE & CS, September 1979.

[2] P. Moroney, A.S. Willsky, P.K. Houpt, "The Digital Implementation of Control Compensators: The Coefficient Wordlength Issue," IEEE Trans. on Automatic Control, V.AC-25,No.4, August 1980, pp.

[3] A.P. Sage, Optimal Systems Control, Prentice-Hall, Englewood Cliffs, New Jersey, 1968.

[4] H. Kwakernaak and R. Sivan, Linear Optimal Control Systems, J. Wiley & Sons, New York, 1972.

[5] L.R. Rabiner and B. Gold, Theory and Application of Digital Signal Processing, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1975.

[6] A.V. Oppenheim and R.W. Schafer, Digital Signal Processing, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1975.

[7] D.S.K. Chan, "Theory and Implementation of Multidimensional Discrete Systems for Signal Processing," Ph.D. Dissertation, MIT, Dept. of EE & CS, May 1978.

[8] J. Allen and R.G. Gallagher, Computation Structures, MIT Course Notes for 6.032, 1977. (to be published)

[9] R.E. Crochiere and A.V. Oppenheim, "Analysis of Linear Digital Networks," Proc. IEEE, Vol. 63, No. 4, April 1975, pp.581-595.

[10] J. Allen, "Computer Architecture for Signal Processors," Proc. IEEE, Vol. 63, No. 4, April 1975, pp.624-633.

[11] G.K. Roberts, "Consideration of Computer Limitations In Implementing On-Line Controls," MIT ESL Rept. ESL-R-665, Cambridge, Mass., June 1976.