To the Graduate Council:

I am submitting herewith a dissertation written by John Allen Arnold entitled "A Microprocessor Based Simulator/Trainer for a Dry Process Control Laboratory." I have examined the final copy of this dissertation for form and content and recommend that it be accepted in partial fufillment of the requirements for the degree of Doctor of Philosophy, with a major in Chemical Engineering.

Charles F. Moore, Major Professor

We have read this dissertation
and recommend its acceptance:

Accepted for the Council:

Vice Provost
and Dean of The Graduate School

A MICROPROCESSOR BASED SIMULATOR/TRAINER FOR A DRY PROCESS CONTROL LABORATORY

A Dissertation

Presented for the

Doctor of Philosophy

Degree

The University of Tennessee, Knoxville

John Allen Arnold

June 1985

ACKNOWLEDGEMENTS

The author has been influenced and helped by many people during his years at the University of Tennessee and he would like to acknowledge his appreciation to each and every one of them; however, it is only possible to mention a few of those who have been most important.

First, the author would like to thank Mr. Raymond R. Gaddis for the eight years of friendship we have shared. Ray is a very special person who has taught the author a great deal about electronics and troubleshooting systems. Second, he would like to share his feelings of admiration and respect for Dr. Eugene E. Stansbury. Dr. Stansbury has shown that a child-like enthusiasm for one's work is essential if one is to continue growing as both a teacher and a student in life. Third, the author would like to thank Dr. Duane D. Bruns for the support and help that he has provided. Duane, through his words of encouragement and his willingness to share his "mini-courses," helped the author to discover his own abilities as a teacher. Fourth, the author would like to thank his advisor, Charlie Moore, for the support he has provided throughout the years. CFM has remained a friend throughout the ups and downs. Fifth, the author would like to acknowledge his appreciation of Ms. Jean A. Pressly for her love, support, and assistance. Without her help, the preparation of this document would have seemed impossible. Sixth, the author would like to thank Susan Dowling and Mike Foster for the many hours of conversation and

discussion that somehow were essential for maintaining a sane perspective on life. Finally, the author would like to thank the remaining members of his committee, Dr. Walter Green and Dr. Don Bogue, for their interest in his work.

ABSTRACT

This work addresses the design and operation of a small process simulator developed specifically for use in a university process control laboratory. Also presented are a number of application examples that illustrate uses of the device as a teaching/learning tool in undergraduate courses. The examples are intended to illustrate ways in which the simulator can be used to aid students in bridging the gap between control theory and practice.

State of the art microcomputer technology is utilized in the simulator which has over 10^5 unique plant configurations. The selected plant characteristics provide a reasonable representation of typical industrial processes and these characteristics (transfer functions, gain, dead time, settling time and sensor noise level) can all be independently selected. The simulator is also inexpensive, easy to operate and requires only a minimum of instructional overhead to use.

In the laboratory the simulator (a) can be configured for stand-alone operation to study process dynamics and modelling or (b) it can be connected in series or parallel with other simulators or (c) it can be connected to a wide variety of industrial control equipment. The application laboratory examples demonstate uses of the simulator in all three areas.

The laboratory examples include demonstrations of reaction curve modelling, open loop and closed controller tuning methods,

design of feedforward controllers, cascade and ratio control, multivariable processes and the relative gain array (RGA), decoupling of multivariable processes, and model reference control algorithms such as the Smith predictor.

In conclusion, the simulator discussed in this work addresses a very real need for modern laboratory equipment in process control courses. The device is shown to be versatile and quite effective as a tool for teaching the fundamentals of process control practice. Also, the device is relatively inexpensive which means that it is practical to provide multiple copies in the laboratory; this improves the quality of a laboratory designed around it because the accessibility of the laboratory is greatly increased.

TABLE OF CONTENTS

CHAPTER PAGE

III. (Continued)

CHAPTER PAGE

CHAPTER PAGE

IV. (Continued)

LIST OF TABLES

LIST OF FIGURES

FIGURE PAGE

CHAPTER I

INTRODUCTION AND BACKGROUND

The purpose in writing this document was to present a digital, portable process simulator, or PPS. The PPS, a versatile and low cost device, was designed to function as an educational tool for teaching/learning applied process control. The primary reason for developing the simulator was to meet a very real requirement for practical and up-to-date laboratory equipment in undergraduate and graduate control courses.

One of the primary design goals was to create a simulation device that provided a realistic representation of industrial process characteristics in a system that was simple in terms of operation and use. This was accomplished by providing the process characteristics through a thumbwheel switch menu; ease of use was achieved by adopting the operating system similar to that of an analog computer. The resulting system was not only quite powerful and practical, but also the machine's operation was readily learned.

The PPS offered two major operating modes which yielded a system that was easy to use and versatile in terms of the range of dynamic characteristics that could be simulated. The primary operating mode simulated a simple process with a sensor, control valve and a disturbance input. The system had two independent simulation blocks for relating the sensor response to valve and disturbance

changes; the PPS could thus simulate most common industrial process situations.

A secondary mode provided for simulations of more exotic process characteristics. In this mode, parallel transfer functions described the total relationship between the valve and the sensor. This parallel structure allowed a large number of systems with competing mechanisms to be configured and studied in a laboratory setting.

The PPS was specifically designed as a simulator for a university process control laboratory. A design criterion was that the PPS should emulate a miniature industrial control room, thus providing students a realistic setting for studying process control. The front panel of the simulator contained a process schematic and meters to indicate the current status of the control signal and the process sensor. Terminal jacks provided the link between the PPS and other laboratory equipment. The simulator could be readily connected to typical industrial process control equipment such as controllers, computers and recorders or the device could be operated in a stand-alone (manual) fashion. Selector switches were provided to allow the system to be changed from manual to closed loop operation quite easily. The PPS was, therefore, extremely flexible and allowed a wide range of dynamic situations to be studied.

The (a) versatility, (b) low cost, (c) straightforward operating system, and (d) dual operating modes were made possible by the state of the art microcomputer design of the PPS. The system was based on an inexpensive, eight-bit computer and the resulting system

was very economical in terms of construction cost. The low cost, the practical nature and the ease with which the operation of the device could be learned made it a very useful supplement to a well rounded, modern process control laboratory. The fact that the PPS was low cost made it practical to provide a number of individual stations for students to use; thus laboratory experiments could be designed for independent study.

Background

The need for modern and practical laboratory equipment results from the rapid changes that have been occuring in industrial process control. The availability of powerful microcomputer based process control (PC) systems has lead to an increasing demand in the U.S. chemical industries for chemical engineers with "a solid background in PC knowhow (1)." Unfortunately, engineering schools are ill prepared to provide this knowhow, due to a lack of appropriate laboratory equipment. The equipment available in many laboratories is obsolete (2) and possibly 80% of it is pneumatic-control based (1). Replacing outdated components with modern equipment can be quite expensive and most universities are hard-pressed to find the resources to upgrade PC laboratories (3). The result is that most conventional laboratory experiments provide only a "hint to the students of process control, but thats about all (1)."

In addition to having hardware that is outdated, experiments based on conventional unit operations equipment are expensive to operate and maintain. Also, some units such as distillation columns

take several hours to reach a steady state and, with the continued growth in undergraduate enrollment, these two factors (and others) lead to scheduling problems with laboratory space, equipment and personnel. This results in an "overburdening of obsolete equipment which" in turn "imposes futhur demands on staff and maintenance resources (3)." Such problems can be reduced somewhat by an increased utilization of low cost computers as process simulators in laboratory experiments (1,3,4).

Control experiments in the past often employed analog computers (5,6,7) as process simulators. There were several reasons for this situation and these reasons were all related to problems associated with digital computers. First, digital computers were expensive. Second, these computers were difficult to program and third, it was not generally desireable to allow students to have free access to these machines. On the other hand, analog computer simulations are by their very nature linear and important characteristics such as dead time are impractical to program. Changes in process characteristics are also difficult to implement, due to effort involved in reprogramming these devices (8). Therefore, laboratory experiments designed around analog computers are necessarily limited in scope.

However, the purpose of any process control experiment is to help students learn to integrate theory and practice. The primary concern is to learn to evaluate process dynamics, select control structure, and tune controllers (3). At the introductory level the focus is primarily on single loop control and sophisticated

experiments or simulations are unnecessary. For this reason, the use of analog simulators is in many ways adequate in spite of the limitations.

Modern digital computers on the other hand are quite flexible, readily reprogrammed and have, in recent years, become reasonable in cost (9,10,11). The nature of these machines allows them to be utilized as data aquisition devices, control systems or plant simulators and, with the introduction of low cost personal computers, there has been a rapid proliferation of microcomputer based teaching laboratories. Almost all of these laboratories use computers (a) to interface with a physical experiment (12,13,14,15,16), (b) to interface with an analog simulation (7,17,18,19) or (c) to create digital simulations of process characteristics (9,10,20), and require that students have (or develop) some basic programming skills. While most applications of computers in teaching laboratories are currently in the area of monitoring and control of "live" processes, the trend is towards a greater reliance on simulations (3). However, computer based experiments often fail to illustrate the natural separation that exists between a process and its control system. In most of these experiments the control law and the dynamic equations of the process are formulated as a single system of equations and solved using a variety of numerical integration techniques (10,11). From this experience students can learn to be comfortable with solving "the rather pat problems that can be worked out through ...simulation routes", but the real control issues of "how processes function, where to locate key control

equipment and sensors, and what setpoints to establish" (1) are not addressed.

Another approach to solving the laboratory problem is to develop a process simulator that mimics some of the operational characteristics of real processes and that also has many of the advantages of a simulator. This allows students to use conventional and modern PC equipment in performing control studies on a "pseudo-live" process with a reasonable response time. If the simulator is economical enough that multiple copies can be constructed, then groups of students or individuals can work separately and simultaneously on different control problems; this helps alleviate the scheduling problems caused by increased enrollment.

A simulation device with the characteristics described above has been in use at Texas A&M for over fifteen years: the smart pneumatic simulator (21). This device falls somewhere between the two extremes of a digital versus a "live" process and allows students to gain "some experience with actual control hardware and operation (21)." However, the range of dynamic characteristics available with the device is restricted to first, second, or third order effects. However, the simulator does allow a variety of control modes to be explored (cascade, feedforward, dead time compensation, etc.), but its use as a tool for teaching process dynamics and identification is limited. Also, the "smart simulator" cannot be readily used to explore advanced topics such as multivariable control.

However, the philosophy behind this simulator is sound, and since the original development there has been a revolution in microelectronics (22). Concurrent with this revolution, simulation techniques that are particularly suited for implementation in small microcomputers (23,24) have evolved. These factors have created the potential for simulators with the same operational philosophy as the smart simulator but with more capability. The presentation of one such device is the topic of this work.

The Research

The work presented in this work consists of (a) specification of a hardware design, (b) development of an appropriate simulation technique, (c) design of an operating system for the computer, (d) development of software, (e) selection of a reasonable spectrum of process characteristics and (f) development of example laboratory experiments. The results of this work are presented in the four chapters outlined in the paragraphs to follow.

Chapter II is concerned with the simulation technique employed in the PPS. In this chapter a powerful technique that was developed for implementation in small machines will be detailed and compared to the more traditional Z-transform approach.

Chapter III describes the basic hardware design and software structure of the PPS. This chapter begins with a discussion of the design of the interface between the computer and the external world. This section provides the basic insight into the operation of the man/machine interface and the analog interface. This discussion is

followed by discussions on (a) the microcomputer system, (b) the mathematical software design, (c) the details on the operation of the system, (d) the details of the simulation software and, finally, (e) the specific operating character of the PPS. This chapter provides the details necessary for designing process control experiments around the PPS.

Chapter IV is devoted to illustrating how the PPS may be used to teach and study process control and dynamics. The chapter covers such topics as reaction curve modelling, controller tuning, feedforward control systems, multivariable control and model heuristic control schemes such as the Smith Predictor algorithm. The purpose of each section in this chapter is to provide enough information about the basic operating character of the simulator so as to allow control experiments to be readily designed.

The final chapter is devoted to recommendations and conclusions. In this chapter some of the limitations of and ideas about future simulators are presented.

CHAPTER II

STEP TEMPLATE SIMULATION METHOD

Many techniques are available for simulating the dynamic response of process systems, but few are suitable for implementation in inexpensive and small microcomputer systems. The techniques that are particularly suited for such systems generally sacrifice representational simplicity for computational simplicity. The step template simulation method, or STM, which is discussed in this chapter, is one of these methods.

The STM, originally developed by Cutler (23) at Shell Oil Company, is a part of a digital control algorithm known as Dynamic Matrix Control (DMC). The goal of DMC is to provide optimal asymptotic regulation by using a plant model to predict future outputs of a process, given a "set of time-dependent changes in the manipulated input"; the set of input changes are selected to "minimize the set point error of the projected outputs over a specified time horizion" (23). This type of control is known as "Internal Model Control" (IMC) and DMC has been observed to be a very robust approach to IMC (25). One of the primary advantages of DMC is that the method is both a modelling and a simulation technique.

The STM process model has certain advantages that make it attractive. One of these is that it is "structure free," which means that "no assumptions have to be made about the number poles and zeros present in the system (25)." Although the structure free or non-minimal representation means that many more process

parameters need to be identified, these parameters result directly from simply normalizing and smoothing step response data. The non-minimal model also has a simplifying effect on the simulation calculations because it allows these calculations to be performed using simple additions, multiplications and shifting operations.

Garcia and Morari (25) have pointed out, however, that the simulation method can be shown to be an extension of the impulse response method and really represents no new theoretical development. The algorithm was basically derived on heuristic principles and, as presented by Cutler, was formulated more along the lines of a procedure instead of a rigorous mathematical principle. Meanwhile, B. C. Moore at the University of Toronto independently developed the STM technique, working from a state space representation (24), and claimed the STM was a "new state space theory" specifically oriented towards implementation in small microprocessors. However, the simulation aspects of both DMC and STM are equivalent; Moore's contribution, the development of a compact recursive representation (realization), is particularly easy to visualize and, hence, to implement. This formulation is also more efficient computationally than the impulse response generalization of Garcia and Morari and, therefore, is more suitable for implementation in a small machine.

Step Template Technique

Underlying assumptions. The STM is a digital technique for simulating the response of asymptotically stable processes "that can be described or approximated by linear ordinary differential

equations" (23). The basic assumptions of the method are

1. that the normalized step response data can be used directly as a process model, thus removing the need to determine ODE or transfer function parameters;
2. that process inputs are assumed (a) to be adequately modelled as piecewise constants and (b) to change slowly relative to the sampling interval; in other words, inputs are treated as a sequence of step changes;
3. that the principle of linear superposition allows a sequence of step inputs to be treated as a set of independent inputs; and
4. that the sampled step response has a finite transition or settling time or T_s.

Ideally, an asymptotically stable, linear process has an infinite settling time because such systems exponentially approach the equilibrium point (64); however, with a finite precision analog to digital converter (ADC), changes in a sampled process cannot be detected after the plant has moved to within a certain percentage (q%) of the steady state value. Thus, if an eight-bit ADC is used, no furthur changes in the process will be observed after the process has moved to within (approximately) 0.4% of the final steady state.

Therefore, a sampled (or simulated) asymptotically stable linear, process has a finite transition or settling time because in a digital system a specification on the accuracy or resolution of

the arithmetic and/or sampling device has been made. Hence, for a given process, specification of the accuracy uniquely determines T_s. This result has important implications with regard to selection of a sample time (T_{st}) and to specification of storage requirements. For example, increasing the system resolution (adding more bits of accuracy) increases the settling time and therefore, for a given number of storage locations, increases the required value of T_{st}.

The resolution of the sampled data and the maximum number (N) of memory locations available for storage are usually established by the system hardware design. T_{st} must be selected to maximize the amount of information stored in memory. If T_{st} is too large, the data in the N memory locations will consist of mostly steady state data; if T_{st} is too small, then the data will not extend to the steady state condition. Therefore, if the maximum amount of transient information is to be collected, the sample time should be specified to be essentially equal to T_s divided by N.

Simulation terminology. The following terms will be used in this and subsequent chapters in discussing details of the STM simulation technique:

SISO a single input, single output process;

MISO a multiple input, single output system; for instance, a two input, one output process or DISO;

MIMO a multiple input, multiple output system;

PISO a parallel input, single output process; for

example, a process with two parallel transfer functions having the same input;

T_{st} sample time--a time interval that elapses before the next value of a process is (a) sampled during model generation or (b) output during a simulation;

N the number of memory locations reserved for storing the step response data;

T_s settling time--time required for the process to move from one steady state to another in the absence of any further changes in process input; this time is equivalent to $N*T_{st}$;

t(0) time at which the first step input is entered into a system at an initial steady state;

u(k) the current value of a process input;

u(k-1) the value of a process input at the previous sample point;

U(k) the magnitude of the change in the input from the last sample to the current (U(k) = u(k) - u(k-1));

PV(0) initial steady state of the simulated process;

PV(k) the value of the simulated process response at the current sample time;

g	process or steady state gain; the fractional change in the process for a unity change in input;
S	step template--column vector of length N which numerically defines the process model. The individual elements of S represent the characteristic response of the normalized system to a unit step;
s_0	the value of the initial steady state of the sampled data;
s_i	the numerical value of the ith element in S;
$X(k)$	state vector--column vector of length N+1 which contains the projected response trajectory (state) of the process for the next N+1 samples, if U = 0 for all future time;
$x(k)_i$	the numerical value of the ith element in X(k) at the current sample time.

SISO model generation and simulation. The basic concept of the step template method is to characterize a process by its response to a step unit input and to use that data as a basis for recursively predicting the response of the system to any arbitrary input. The method is very general and assumes that the principle of superposition applies and that the input can be approximated by a series of steps of various magnitudes.

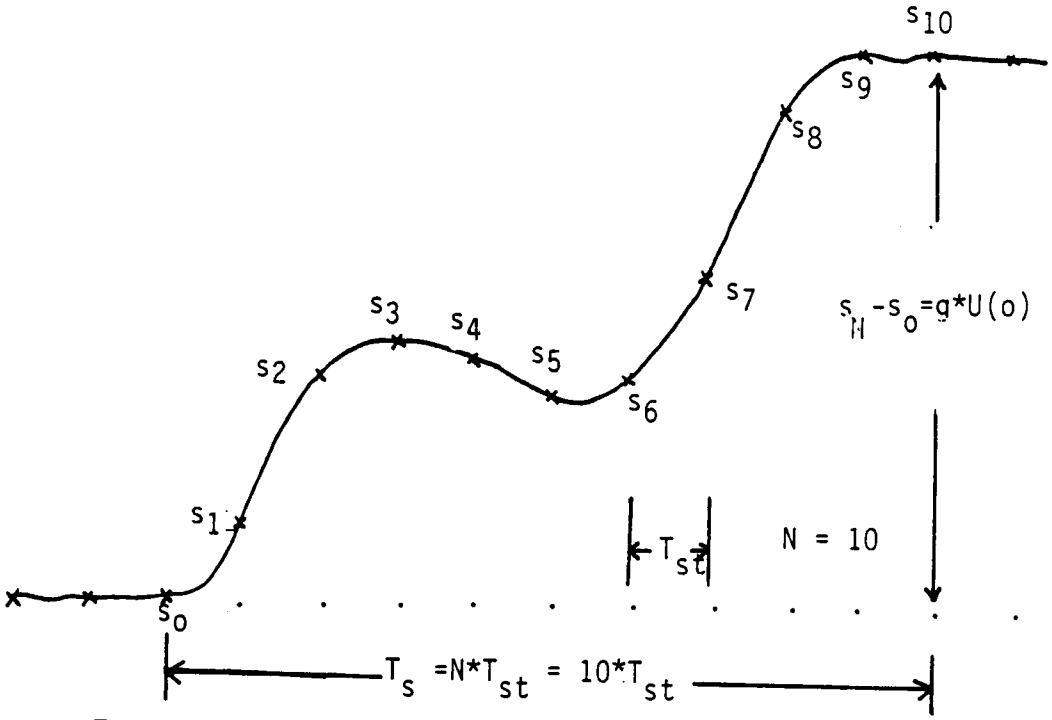
The heart of the method is the step template vector (S) in which the normalized step response data of the process is stored. The data in S can be collected directly from a real process or it can be generated by a more conventional differential equation based numerical simulation. A typical step response is shown in Figure 2.1; in the figure s_0 is the initial steady state, s_N is the final steady state and the elements in between represent the entire dynamic response of the system.

Generally the step template is normalized to represent the response of a unity gain process to a unit step input (unit step: U(k) = 1 for k=0 and U(k)=0 for k>0). Normalization involves (a) first extracting the steady state gain (**g**) from the template data, (b) subtracting s_0 from each element in S and (c) dividing the vector by the magnitude of the total change (s_N - s_0) in the sampled process. The value of **g** can be determined from S as follows:

$$\mathbf{g} = (s_N - s_0/U(k)) \qquad [2.1]$$

After normalization, S represents a map or template containing the transient response trajectory of a unity gain process to a unit step input. The vector may now be used as a template for simulating the process response to any sequence of inputs.

The basic idea behind the STM simulation approach can be best understood from a procedural or heuristic prospective. Consider an N+1 element array called **X**(k) which is loaded with the initial steady state of the simulated process (PV(0)). At t(0) a step change of magnitude U(k) is invoked in the system. Now, starting at



$$S^T = [\, s_0,\ s_1,\ s_2,\ s_3,\ s_4,\ s_5,\ s_6,\ s_7,\ s_8,\ s_9,\ s_{10},\,]$$

Raw data vector

$$S^T = \left[\, \frac{s_0 - s_0}{g*U(0)},\ \frac{s_1 - s_0}{g*U(0)},\ \ldots,\ \frac{s_N - s_0}{g*U(0)} = 1.0\ \right]$$

Normalized Step Template vector

Figure 2.1. Sample step response and corresponding step template vector.

time t=t(0) and every T_{st} time units thereafter (or t=t(0) + k*T_{st}; k=0, 1, 2,...., where k equals the number of time steps or iterations that have elapsed) both the next process output and the state trajectory (**X**(k)) are sequentially updated as follows:

$$PV(k) = x(k)_1 \qquad [2.2]$$

$$x(k)_i = x(k)_{i+1} + g*U(k)*s_i \; : \; i = 1, 2,...,N \qquad [2.3]$$

$$x(k)_N = x(k)_{N+1} \qquad [2.4]$$

Basically, Equations 2.2 to 2.4 state that (a) PV(k) is first set to the current value of the top element in **X**(k), (b) every element in **X**(k) is shifted into the next storage location and updated with the new output contribution of the current input and (c) the N+1 element in **X**(k) is loaded (filled) with the value in the Nth element. This set of actions is defined as a shift-and-fill operation. The fill operation is necessary in order to force the new steady state to move from the last to the first element in **X**(k) in N-1 iterations. Figures 2.2 and 2.3 provide a set of examples which demonstrate the operation of the equations with and without the fill operation.

So, after N-1 iterations all of the elements in **X**(k) will equal PV(0) + g*U(k), therefore, **X**(k) will be a constant vector (see Figure 2.2): that is, until a new change in U(k) is observed. Thus, the transient effect of each input change remains for N-1 iterations and then, the effect becomes just a pure bias in the state vector; the shift-and-fill operation forces the steady state effect of each change in input to reach the output after N iterations.

Iteration Number	Delta Input	Process Output	State Vector Transposed*
k	U(k)	PV(k)	X^T
-1	0	x_0	$[x_0, x_0, x_0, x_0, x_0, x_0]$
0	1	x_0	$[x_0+g^*s_1, x_0+g^*s_2, \ldots, x_5, x_5]$
1	0	$x_0+g^*s_1$	$[x_0+g^*s_2, x_0+g^*s_3, \ldots., x_5, x_5]$
2	0	$x_0+g^*s_2$	$[x_0+g^*s_3, x_0+g^*s_4, x_5, x_5, x_5, x_5]$
3	0	$x_0+g^*s_3$	$[x_0+g^*s_4, x_5, x_5, x_5, x_5, x_5]$
4	0	$x_0+g^*s_4$	$[x_5, x_5, x_5, x_5, x_5, x_5]$
5	0	x_5	$[x_5, x_5, x_5, x_5, x_5, x_5]$
6	0	x_5	$[x_5, x_5, x_5, x_5, x_5, x_5]$

* Note: $x_0 = PV(0)$; $x_5 = x_0 + g^*s_5$: $s_5 = 1.0$

Figure 2.2. Results from five iterations through Equations 2.2, 2.3 and 2.4 (shift-and-fill operation) for N = 5 and U(0) = 1 (a step input) at k = 0.

Iteration Number	Delta Input	Process Output	State Vector Transposed*
k	U(k)	PV(k)	X^T
-1	0	x_0	$[x_0, x_0, x_0, x_0, x_0, x_0]$
0	1	x_0	$[x_0+g^*s_1, x_0+g^*s_2, \ldots, x_5, x_0]$
1	0	$x_0+g^*s_1$	$[x_0+g^*s_2, x_0+g^*s_3, \ldots, x_5, x_0, x_0]$
2	0	$x_0+g^*s_2$	$[x_0+g^*s_3, x_0+g^*s_4, x_5, x_0, x_0, x_0]$
3	0	$x_0+g^*s_3$	$[x_0+g^*s_4, x_5, x_0, x_0, x_0, x_0]$
4	0	$x_0+g^*s_4$	$[x_5, x_0, x_0, x_0, x_0, x_0]$
5	0	x_5	$[x_0, x_0, x_0, x_0, x_0, x_0]$
6	0	x_0	$[x_0, x_0, x_0, x_0, x_0, x_0]$

* Note: $x_0 = PV(0)$; $x_5 = x_0 + g^*s_5$: $s_5 = 1.0$

Figure 2.3. Results from five iterations through Equations 2.2 and 2.3 (no shift-and-fill operation) for N = 5 and U(0) = 1 (a step input) at k = 0.

Now, if a new input change occurs after N-1 or more iterations, the situation is exactly the same as at t=t(0) except X(k) now has a different initial value or bias. Superposition allows the contribution of the new input to be overlayed onto the current state transition vector and at each iteration X(k) contains the sum of the residual trajectories of all previous inputs. The response due to the current change in input is simply added to the residual trajectory.

Extension to the multivariable case. The STM method may be readily extended to include multiple inputs and outputs. For the multi-input, single output case (MISO), the shift operation (Equation 2.3) of the state update equation is modified as follows:

$$x(k)_i = x(k)_{i+1} + Z(k)_i \; : \; i = 1, 2, \ldots, N \qquad [2.5]$$

where

$$Z(k)_i = g^j * U(k)^j * s^j{}_i \qquad [2.6]$$

In these two equations, P is the number of independent input signals that affect the output; g^j is the gain associated with the jth input; U^j is the magnitude of the change in the jth input at the current sample time; and $s^j{}_i$ is the ith step template element of the jth input. Therefore, each element in the state vector represents a linear combination of the effect of the independent inputs of the system. For each value of i, $Z(k)_i$ is first evaluated and the result is then added to $x(k)_i$.

The MISO simulation computations may be readily extended to include the MIMO case also. The MIMO case results from performing the MISO computations (Equations 2.2, 2.5, 2.6 and 2.4) once for each process output. Thus, a multivariable STM simulation may be considered to be a series of MISO computations.

Formulation using integer arithmetic. One disadvantage of the STM is that a large number of operations must be performed at each sample time. If the simulation technique is formulated using real or floating point arithmetic, then the computations may become unacceptably slow. The use of an integer arithmetic formulation increases the speed at which a simulation can be performed and this partially offsets the loading introduced by the large number of operations. Thus, whether the STM is implemented in a large or small computer, an integer formulation will always increase the overall speed of execution. One advantage of the STM is that it may be readily implemented using integer arithmetic--this is the feature that makes it attractive for use in a small machine. Although a microcomputer is (usually) only capable of performing binary additions, these machines may be easily programmed to efficiently perform integer additions and multiplications; thus, the STM can be easily implemented in a microcomputer.

Speed of execution and accuracy in the arithmetic are important factors in developing a real-time simulation technique for implementation in a small computer system. Consider, for instance,

a simple Z-transform representation of a first order lag with a time constant of Tau.

$$PV(k) = a*PV(k-1) + g*(1-a)*u(k-m) \qquad [2.7]$$

where

$$a = e^{(-T_{st}/Tau)} \qquad [2.8].$$

In Equation 2.7 **a** and 1-**a** are recursive coefficients that weight the effect that the previous output and the input m sample times in the past on the current output. Note that Equation 2.7 requires only four multiplications and one addition and, hence, can be evaluated in a relatively short time when compared to the STM (the STM requires N additions and multiplications at each sample time). However, if PV(k) and **a** are represented using T-bit integers then the product of PV(k-1) and **a** will be a 2T-bit number. Scaling the product back to T-bits introduces truncation error in the simulation, and the error will be compounded at each sample time. The result is that the simulated solution will show significant error with respect to the actual or analytical solution.

The STM formulation on the other hand does not require that the output trajectory (X(k)) be multiplied by any such weighting factors. So, if **g***U(k) and the step template elements are stored as T/2-bit numbers, then the product will be T-bits wide. Therefore, only two T-bit integers will be necessary to update each element in X(k). Prescaling template coefficients and **g***U(k) reduces the problem of truncation error and allows the STM to be a reasonably

accurate integer simulation technique. Thus, the STM trades off operational counts for a simpler and more accurate mathematics package.

Implementation of the recursive difference equation approach would require that a more sophisticated arithmetic package be developed and for the computer used in the PPS this was considered undesirable. For this reason the STM was chosen as the simulation technique.

CHAPTER III

PROCESS SIMULATOR--THEORY OF OPERATION

The Portable Process Simulator, or PPS, designed to be a two input, one output, linear dynamic simulator, has over 2.0 X 16^4 (131,072) different configurations. This chapter describes the microcomputer based PPS beginning with a discussion of the system hardware and the computer interface. In the first section the design philosophy of the hardware and the methods used by the computer to communicate with external world through the system interfaces are described.

The next section provides details on the data structure in the processor and the mathematical software. This section describes the requirements and limitations on data representation and discusses the subroutines required to perform mathematical functions such as addition, subtraction, division and multiplication.

The third section describes the operating system. In this section details that are necessary to understand the basic operation system and the simulation software are supplied. The final section presents the operating characteristics of the PPS. This section supplies the detailed explanations of the various system configurations and parameters.

3.1 SIMULATOR HARDWARE

A photograph and an equipment schematic showing the major system components and inteconnections of the PPS are shown in

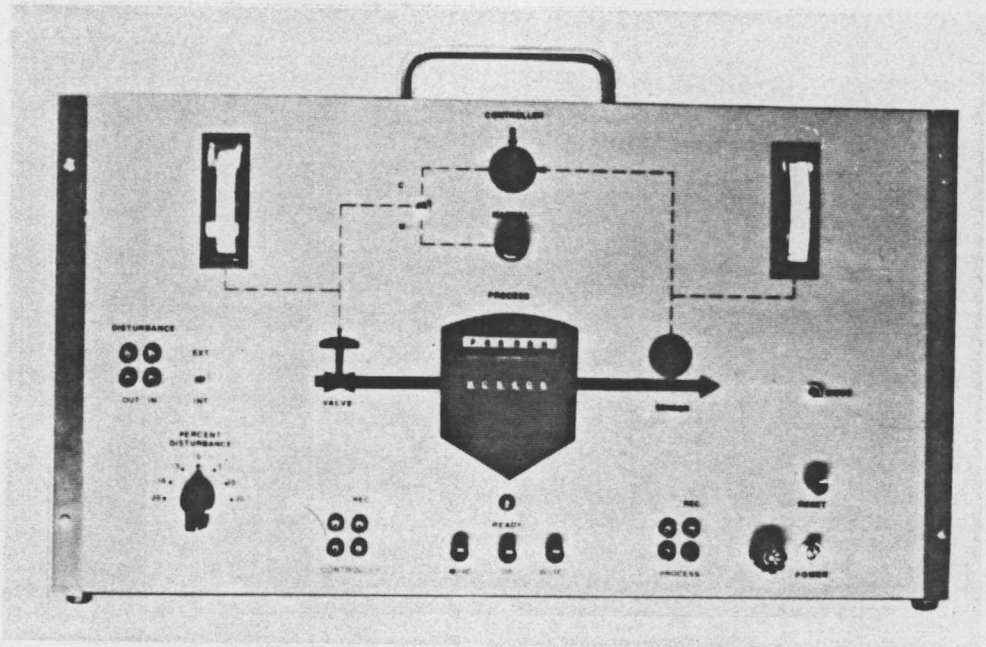
Figure 3.1. The primary system components are the microcomputer, expansion memory, expansion input/output (I/O), man/machine interface (MMI), and analog interface. The discussions on the system hardware begin with the analog interface.

Analog Interface

The analog interface provides a convenient link between the microprocessor and the various pieces of electronic instrumentation which might be used with the simulator (manual valve station, industrial controllers, recorders, function generators, et cetera). The interface includes an eight-bit, two channel analog to digital converter (ADC) and a single channel eight-bit digital to analog converter (DAC), as well as some additional circuitry which provides for scaling and signal conditioning. The range of analog inputs which can be handled by the interface is either one to five volts or four to twenty milliamps.

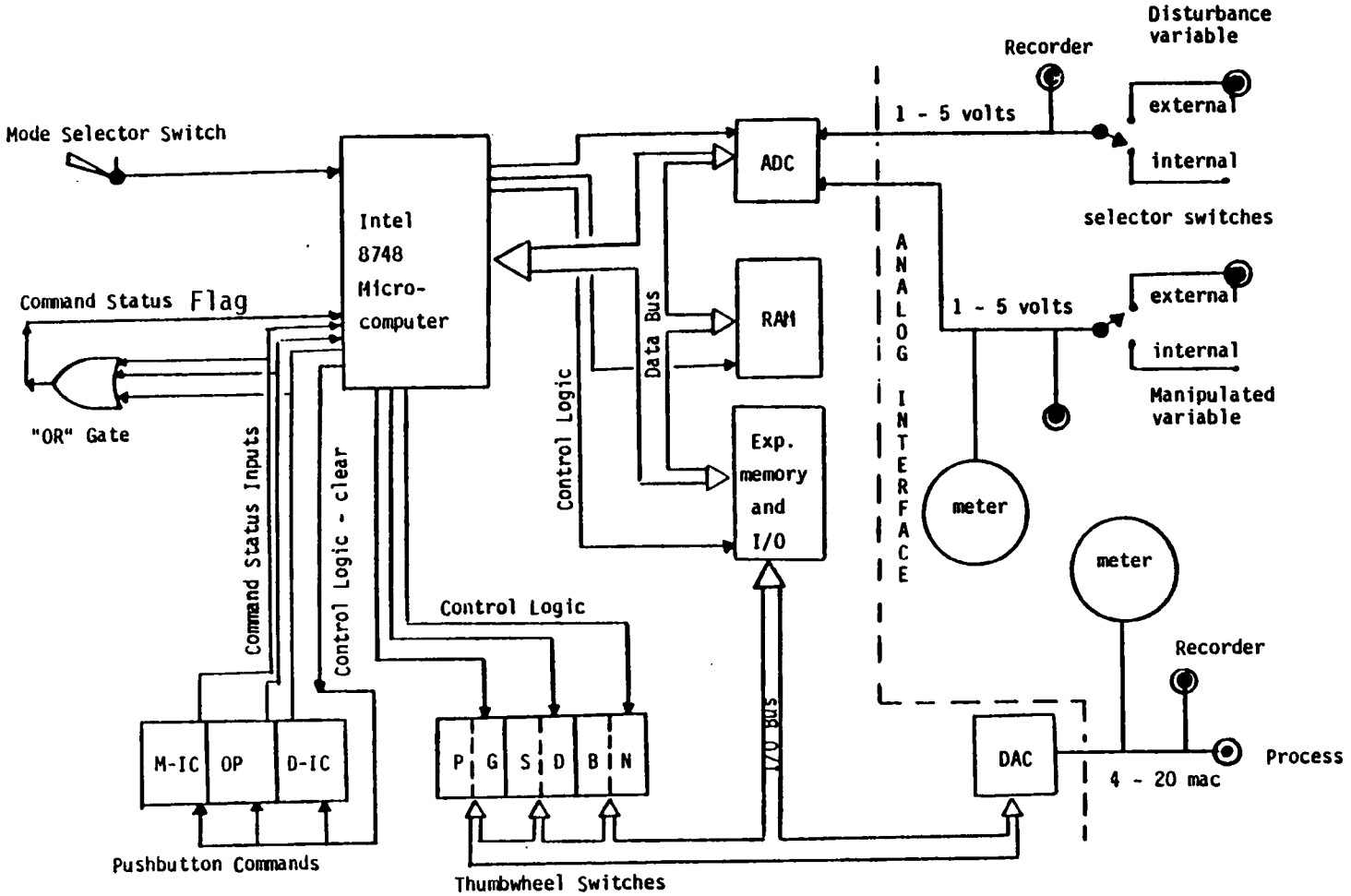
The two analog inputs are the manipulated variable (MV) and the disturbance variable (DV). Each can be provided by an external source or by an internal potentiometer and voltage reference. This feature allows the simulator to be operated in conjunction with other laboratory equipment or to be operated in an entirely stand-alone fashion.

There is a single output, the process variable (PV), which is also available as a one to five volt or four to twenty milliamp signal. Both inputs and the output are supplied as voltages on



(a)

Figure 3.1. Photograph and hardware schematic of the PPS, (a) photograph and (b) hardware schematic.



MAN MACHINE INTERFACE (b)

Figure 3.1. (continued)

terminal jacks and MV and PV are supplied to meters to provide visual feedback to the user.

Man/Machine Interface

The MMI consists of the components which allow a user to configure the system and to issue commands to the operating system software. Configuration programming is menu driven through a set of six thumbwheel switches (TWS) and a mode selector switch (MSS) allows the user to select one of two operating modes. Commands are entered via three pushbuttons labeled M-IC (Manipulated variable Initial Condition), D-IC (Disturbance variable Initial Condition), and OP (OPerate). A RESET pushbutton is also provided for resetting the system circuitry and simulation software.

The procedure for configuring and implementing a simulation using the MMI is quite simple because the PPS was designed to be a dedicated process simulator. The user selects a configuration using the thumbwheel switches and the MSS and then issues a command (M-IC or D-IC) to implement the configuration. A simulation is initiated by the OP command; once a configuration has been selected, a simulation can be repetitively initiated with the same parameter settings by simply pressing the OP command again and again.

Microprocessor and System Components

An eight-bit microcomputer manufactured by Intel Corporation is the processor around which the PPS was designed. The processor (Intel 8748) has 1K (1024) bytes of static on-board program memory, 64 bytes of volatile data memory, a 12-bit address/data bus, 12-bits

of individually configurable I/O, one level of hardware interrupt, an eight-bit timer/event counter, and two hardware settable/software testable flags (T0 and T1). The computer has approximately 255 instruction codes that are provided as understandable mnemonics.

The processor has an architecture in which all data transfers must be processed through a central location called the accumulator. Data memory transfers are made through two sets of eight byte register banks and each bank has two registers designated as index pointers. The primary limitations of this structure are that (a) the register banks quickly fill up with temporary data and (b) if a number of variables need to be accessed the process of moving addresses into the index pointers becomes somewhat tedious. Also, data memory and program memory are physically distinct and the computer's access to program memory is limited. Therefore, if static program memory is used for permanent storage of data, the data has to be transferred into data memory before being used, otherwise, reading the data requires considerable processing.

Logical tests on data and software loops are easily programmed. All of the bits in the accumulator and two software flags (F0 and F1) can be readily cleared, set, or tested by software. Loops can be generated using the DJNZ instruction or "decrement and jump if the result is not zero" which is an instruction that is relatively unique to this microcomputer.

From the above description it should be apparent that the processor is relatively powerful and easy to program. Even so, because the STM turned out to be such a memory intensive technique,

additional program memory (for template and parameter storage) and data memory (for the state vectors associated with each input and other temporary data) were required by the system. Additional program memory and two banks of eight-bit I/O were added using an Intel 8755 chip and data memory was added using two 4-bit by 1K random access memory (RAM) chips. Thus, the system had 3K of program memory, 28 bits of I/O, and 1088 bytes of data memory.

Hardware Operation

The processor communicates with system components through the address/data bus and simple I/O. Control logic is used to select devices (RAM, ADC, 8755 chip, TWS chips) and clear flipflops. Control logic and simple I/O signals are unidirectional while the data bus is bidirectional (in Figure 3.1b, page 27, control logic is a thin line with a terminating arrow at the device being controlled; simple I/O is represented by a thin line originating at the signalling device and ending with a terminating arrow at the computer; the data bus is shown as a double-wide line with a terminating arrow at both ends to indicate that information may flow in both directions).

Simple I/O is used to connect the command pushbuttons (M-IC, D-IC, and OP) and the MSS to the processor while the RESET pushbutton is connected directly to the reset pins on the 8748 and 8755 chips. Pressing the RESET command pulses both reset pins which results in (a) the internal circuitry of the two chips being reset and (b) the processor's program counter being set to zero. Pushing one of the command pushbuttons forces a two state flipflop

associated with the pushbutton to be set to the "on" or "true" state; the state remains true until a "CLEAR" command is issued by the processor.

This CLEAR command also resets the flipflop connected to the MSS; this flipflop has to be cleared whenever the MSS changes state; otherwise, the simulation software will not function correctly. The user forces the MSS flipflop to be cleared by simply issuing any of the three pushbutton commands.

The output signal from each of the command flipflops enters the microprocessor at two locations; the first location is a simple I/O pin on the processor and the second location is the interrupt pin of the processor. The state of the interrupt pin is set directly by the output of an "OR" gate which has three inputs--one for each flipflop. Whenever any flipflop becomes true, the OR gate output becomes true as well. The microprocessor monitors the state of the interrupt pin (OR gate) to determine that a command has been entered. The status of the interrupt is monitored by a small light on the front panel of the simulator as well; whenever this light is off (interrupt true), the user can tell that the computer has not yet serviced a command.

Upon detecting an interrupt, the computer reads the I/O pins on the microprocessor chip, stores the result and issues a CLEAR command (using control logic) to set the state of all flipflops to the "off" or "false" condition. The computer next sequentially tests the bits of the saved I/O data, identifies which of the three commands has been pressed and implements the requested function.

The processor communicates with the TWS and the DAC through the expansion I/O chip (8755) over the data bus. To access either device requires that (a) the 8755 chip be exclusively selected (RAM chips off) using control logic and then (b) an indexed memory move be made to the TWS address (location zero) or the DAC address (location one). The switches are read in sets of two and the desired set is chosen using control logic. Examples of the subroutines that were written (a) for reading an ADC channel, (b) for writing to the DAC or (c) for reading the TWS are shown in Figures 3.2, 3.3, and 3.4 respectively. (Note that all numbers in the figures are in octal).

RAM is also read using the data bus. To read RAM requires that a page addressing scheme be followed and that the 8755 be switched off. The 1K of RAM is treated as four pages with 256 bytes (0 to 255) in a page. To read a particular data memory location requires that the processor

1. switch on the RAM;
2. select a page (0 to 3);
3. load an address (0 to 255) into an index pointer;
4. perform a memory read to get the data.

An example of the programming required to read RAM memory is shown in Figure 3.5. Control of the hardware is straightforward to accomplish and all of the necessary programming information is included in Figures 3.2 through 3.5.

Digital to Analog Converter Subroutine

```
DACOUT:
     CALL RAMOFF     ;  TURN OFF THE RAM CHIPS
     ORL P2,ON8755   ;  SWITCH ON THE 8755 CHIP
     MOV R1,001      ;  MOVE THE DAC ADDRESS INTO REGISTER ONE
     MOVX @R1,A      ;  WRITE THE CONTENTS OF THE ACCUMULATOR ONTO
                     ;  ONTO THE DAC
     ANL P2,0F8755   ;  SWITCH THE 8755 OFF
     RET             ;  RETURN TO THE CALLING ROUTINE
RAMOFF:
     ANL P2,334      ;  TURN OFF ALL RAM CHIPS
     RET             ;  RETURN TO CALLING ROUTINE
```

Figure 3.2. A subroutine for performing a digital to analog conversion.

Subroutine to Read the Thumbwheel Switches on the PPS

```
RDTWS:
      ANL P1,370      ;  SET THE TWS CONTROL LOGIC TO OFF
      ORL P1,001      ;  SELECT THE FIRST BANK OF SWITCHES
      ORL P2,020      ;  TURN ON THE 8755 CHIP
      MOV R0,000      ;  THUMBWHEEL SWITCH ADDRESS
      MOVX A, @R0     ;  LOAD THE ACCUMULATOR WITH THE SETTINGS ON
                      ;  THE FIRST BANK OF TWS
      MOV @R1,A       ;  STORE THE NUMBER IN THE RAM ADDRESS POINTED
                      ;  TO BY REGISTER ONE
      ANL P1,370
      ORL P1,002      ;  TURN ON THE NEXT BANK
      MOVX A,R0
      DEC R1          ;  DECREASE THE COUNT IN REGISTER ONE BY ONE
      MOV @R1,A
      ORL P1,001      ;  SELECT THE NEXT BANK
      DEC R1
      MOVX A, @R0
      MOV @R1,A
      ORL P2,357      ;  SWITCH OFF THE 8755 CHIP
      RET             ;  EXIT FROM THE ROUTINE
```

Figure 3.3. A subroutine to read the thumbwheel switches on the PPS.

Subroutine to Perform an Analog to Digital Conversion

```
ADCV:
     ORL P2,ADCON    ;  SWITCH ON THE ADC
     MOV RO,OOO      ;  LOAD A ZERO INTO REGISTER ZERO
     MOVX @RO,A      ;  DUMMY WRITE TO START A CONVERSION
     MOV R2,036      ;  LOAD A COUNT OF 30 (DECIMAL) INTO REGISTER
                     ;  TWO
EOC:
     DJNZ R2,EOC     ;  WAIT HERE UNTIL 30 X 5 MICROSECONDS
                     ;  PASS - GIVE THE ADC A BIT OF TIME TO
                     ;  GET STARTED
EOC1:
     JNT1 EOC 1      ;  JUMP TO EOC1 UNTIL THE HARDWARE FLAG
                     ;  CALLED T1 BECOMES TRUE.  THIS IS THE
                     ;  CONVERSION COMPLETE SIGNAL.
ADCRD:
     MOVX A,@RO      ;  READ THE CONVERTED SIGNAL INTO THE
                     ;  ACCUMULATOR
     ANL P2,ADCOFF   ;  SWITCH THE ADC OFF
     RET             ;  RETURN TO THE CALLING ROUTINE
```

Figure 3.4. A subroutine to read the analog to digital converter on the PPS.

Read RAM Memory Subroutines

```
RDRAMO:             ;ROUTINE TO READ RAM LOCATION IN PAGE ONE
                    ;USER LOAD ADDRESS IN REGISTER ONE
                    ;ROUTINE RETURNS RESULT IN ACCUMULATOR

     CALL RAMOO     ;  SWITCH ON RAM PAGE ZERO

     MOVX A,@R1     ;  READ THE CONTENTS OF THE ACCUMULATOR INTO

                    ;  THE ACCUMULATOR

     RET            ;  RETURN TO THE CALLING ROUTINE

RAMOO:

     CALL RAMOF     ;  TURN OFF ALL RAM CHIPS

     ANL P2,040     ;  FOR PAGE 1 LOAD 041, PAGE 2 042, PAGE 3 043

     RET            ;  RETURN TO CALLING ROUTINE
```

Figure 3.5. A subroutine for performing a RAM memory read

This completes the description of the system hardware. In the next section the mathematical software of the PPS will be described in detail.

3.2 SIMULATION SOFTWARE

Numerical Considerations

The 8748 microprocessor has the inherent capability of performing only one arithmetic operation--an eight-bit unsigned integer addition. All other mathematical functions were formulated in the integer mathematics package that was developed. The package included subroutines to perform sixteen-bit signed additions and subtractions, and unsigned divisions (with truncation) by powers of two. The multiplication subroutine was designed to take two eight-bit unsigned numbers and return a sixteen-bit product. A nine-bit by eight-bit multiplication routine was also developed; however, this special case software used the basic 8x8 multiplication subroutine with some additional processing to handle the extra bit. The final result of all mathematical operations was always a sixteen-bit product.

The basic word size in the mathematics package was limited to sixteen-bits because of the architecture of the microcomputer. The processor works most effectively with eight-bit numbers (bytes), but it can be readily programmed to perform multibyte operations. However, operations with results that occupy more than two bytes (a word) are difficult to program because of the number of data moves

and temporary storage locations required. Limiting the results from the basic multiplication routine to sixteen-bits and using sixteen-bit addition, subtraction and division kept software manageable, but it did affect the way data was represented in the system.

Data representation. The arithmetic operations in which a number was used controlled how it was represented in the processor. Numbers used in additions, subtractions or divisions were all treated as signed sixteen-bit numbers, but any number used in the basic multiplication routine had to be a positive eight-bit integer; if the magnitude required more bits, then special computations were necessitated. The variables affected by this restriction were step template elements (s_i), steady state gains (g), input changes (U(k)) and intermediate products (g*U(k)).

Since numbers used in the multiplications were to be positive, some method had to be employed to indicate the signs of products; a sign and magnitude convention (27) was adopted. However, the sign was handled differently for each variable type, but the basic convention remained the same.

Storage of step template elements. A state template was stored as a normalized vector with the last element in the vector corresponding to one. Generation of a representation of the template data consisted of selecting a sign and magnitude coding notation scheme, selecting an array length and determining how the number one would be represented.

In order to maximize the transient information stored in the template it was determined that a full eight-bits was needed for storing the magnitude. This meant that the sign of each element had to be stored separately and thus, an additional bit was required for each element. This dictated reserving an additional byte of storage for every set of eight bytes included in the template. Therefore, if the basic length of the template data array was 100 bytes then 113 bytes of storage had to be reserved for the complete array.

The basic length of the step template data array was set to 113 bytes for three reasons. First, choosing 113 as the overall step template length allowed two vectors (226 bytes) to be stored per page (256 bytes) and since the hardware was designed to accommodate eight pages of storage, sixteen different templates were possible. Second, programming considerations dictated that approximately 25 bytes be reserved in each page of storage for a subroutine that loaded the templates on that page into RAM. Third, each element in the state vector (X) was stored as a signed sixteen-bit word and limiting the template transient data to 100 bytes meant that 202 bytes (less than a page of memory) were required to store the state vector. Memory manipulation in the processor is page oriented and limiting the size of the state vector to less than a page made the programming task easier.

Each of the system's sixteen step template vectors (S) were stored as a data array and a sign array in program memory. There was a one to one correspondence between the bits in the sign array and the step template elements and if a bit was "on" or true then

the number associated with it was negative. The most significant bit (MSB) in the first byte of sign array indicated the sign of the first template byte and the least significant bit (LSB) set the sign of the eighth byte and so forth.

The final data representational consideration was the selection of an integer number to correspond to one. The last entry in the template data array was by definition unity; this number had to be large enough to provide sufficient transient information and yet small enough to allow values greater than one. Choosing a particular eight-bit integer to represent unity meant that after a set of multiplication computations were performed the resulting product had to be divided by that number; this, in effect, represented a scaling operation. The scaling coefficient was selected to be 128 or, 2^7, for the following reasons:

1. it was representable as a power of two and so was compatible with the division subroutine;
2. it was the largest eight-bit number that could be represented directly as a power of two and still allow template values greater than one to be stored. This enabled underdamped ODEs to be represented in a template.

The scaling equation shown in Equation 3.1 was used whenever a multiplication by a step template element was required.

$$Z(k)_i = (P(k) * s_i + 64)/128 \qquad [3.1]$$

In Equation 3.1, $Z(k)_i$ is a sixteen-bit product, P(k) is an

eight-bit multiplier and s_i is an element from the step template data array. There was a need to add 64 to the product because the division routine always truncated the last seven bits (rounding to the next smallest integer) and the addition forced rounding to the nearest integer. The determination of the sign of $Z(k)_i$ will be discussed in another section.

Representation of the steady state gain. The steady state gains were stored as eight-bit signed numbers in program memory. The MSB was the sign identifier and the other seven bits represented the positive magnitude. As with the step template if the sign bit was true, then the number was negative. To use the gain in a multiplication required that the sign bit be saved in temporary storage and that the bit be cleared in the gain byte. Gains were stored as integer numbers ranging from +16 to -16 which represented gain values of +2 to -2. To convert the stored number of counts to the actual gain value required a scaling operation.

Since all the elements in the template were multiplied by the product of the gain times U(k) (g*U(k)) this product was computed, scaled and stored as a single positive sixteen-bit number in memory. The scaling operation consisted of multiplying U(k) by the number of counts in the gain and then dividing by eight.

$$P(k) = (\text{Counts} * U(k) + 4)/8 \qquad [3.2]$$

The result was a number (P(k)) that was used to represent the product of g*U(k). As an example of the output from Equation 3.2

consider a situation where a gain of 1.5 is desired. To achieve a gain of 1.5 requires that the number of counts be 12. Multiplication of U(k) by twelve and then dividing by eight yields a result that is 1.5 times the original value of U(k); therefore, P(k) in E quation 3.2 is equivalent to **g***U(k). As with $Z(k)_{ij}$ in Equation 3.1, the discussion of the determination of the sign of P(k) will be deferred.

Intermediate products. Intermediate products such as P(k) or computed variables such as U(k) had to conform to the eight-bit restriction of the multiplication routine. Now, U(k) was never allowed to exceed eight-bits (maximum value of 255,) because if it did the software took certain actions which restored it to 255 or less; this will be discussed in the next and later sections. It has already been stated that one characteristic of P(k) or **g***U(k) was that it was stored as a sixteen-bit number but since the magnitude of U(k) never exceeded 255 and the maximum gain was two (16 counts represented a gain of two), P(k) could, in actuality, never exceed 510, which is a nine-bit number. Thus, P(k) never occupied more than nine of the sixteen-bits in the word.

Now recall that the step template update equation (Equation 2.3, page 17) included the term on the right hand side of Equation 3.3.

$$Z(k)_{ij} = \mathbf{g}^*U(k) * s_i \qquad [3.3]$$

P(k) or **g***U(k) has already been shown to be a sixteen-bit number, so

P(k) could not be used directly in the basic multiplication routine. However, the fact that P(k) was known to never exceed nine-bits allowed a special algorithm to be constructed that would detect when P(k) exceeded eight-bits and invoke the special nine-bit multiplication algorithm. If P(k) occupied eight-bits or less, then it could be treated as an eight-bit number and no special processing was required. This will be discussed in the section on multiplication subroutines.

Sign bit of computed numbers. The sign bit of computed variables (U(k), P(k) or $Z(k)_i$) had to be stored dynamically. Since the processor has two flags that can easily be set and tested in software the decision was made to store the signs of these variables using one of these flags (F0). Dynamically tracking the changing signs of the products involving these variables required several processing steps.

The first step was to compute the magnitude of U(k) and store the sign. U(k) was typically computed (except when certain special configurations were in effect) by subtracting the current eight-bit ADC reading (u(k)) from the previous one (u(k-1)). Since the value of u(k) was always a positive eight-bit number, U(k) had to range in value from +255 to -255; therefore, including the sign while retaining the full eight-bits of magnitude required an extra or ninth bit; F0 was used for this purpose. The magnitude was stored in RAM and the intitial state of F0 was set so that "on" (true) meant +U(k) and "off" (false) meant -U(k). (Note that this

notation is opposite of that used for indicating the sign of the template elements or the gains (true meant a negative number). The notation scheme was adopted because of the limitations on the way the processor can test F0.)

After setting F0 to indicate the sign of U(k), the processor determined the magnitude and sign of P(k). The sign bit of **g** was first saved and then stripped (bit eight set to zero) from **g** and then **g** was multiplied by U(k). The sign of the product was determined using F0 and the gain's sign bit. If both were true (+U(k), -**g**) or both were false (-U(k), +**g**) then the product's sign was negative, so F0 was set to false (-P(k)). If one was true and the other false (+U(k), +**g** or -U(k), +**g**), then the product was positive (same sign multiplication) and F0 was set to true (+P(k)). The logic required to implement this set of tests is shown in Figure 3.6. The algorithm represents a subroutine that expected the calling routine to establish the state of F0 (sign of first byte) and load the most significant bit (MSB) of the accumulator with the sign bit of the second byte. The routine returned with F0 set (+) or cleared (-) to reflect the sign of the product.

Once P(k) was evaluated then the magnitude (Equation 3.3) and sign (algorithm of Figure 3.6) of $Z(k)_i$ needed to be determined and used to update an element in the state vector. The sign of the step template elements were stored using the same convention as the gain (true = $-s_i$). The sign bit of each template element was read from the sign array using two indexed pointers. The first one pointed to the byte that contained the sign of the desired template

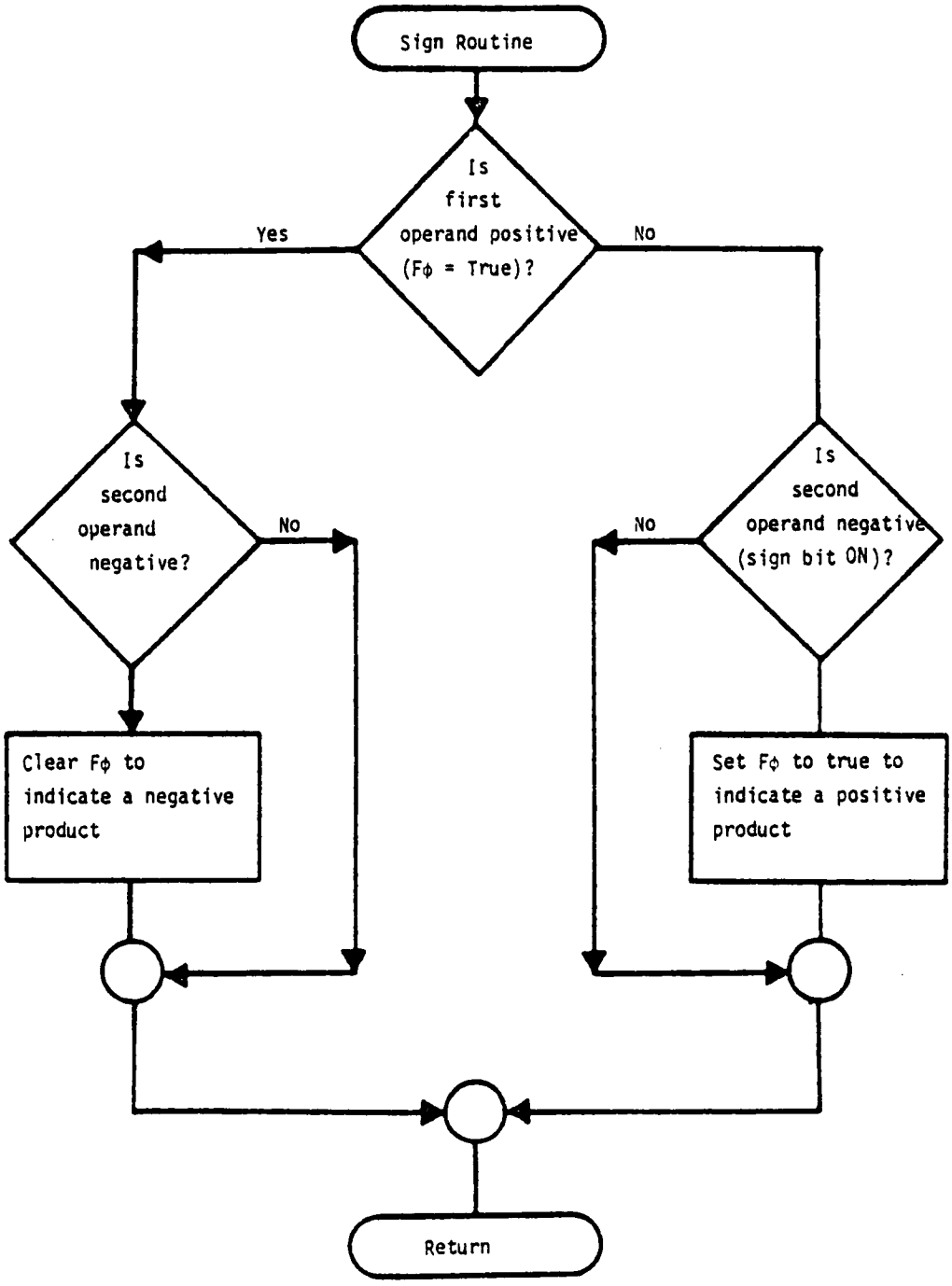


Figure 3.6. Flowchart of algorithm to determine the sign of the product of a multiplication when both operands are stored using sign and magnitude coding.

element and the second pointed to the specific bit. The bit was then loaded into the accumulator's MSB and the sign of the product was determined using the algorithm in Figure 3.6.

Once the element in the state vector was updated then the original sign and magnitude of P(k) had to be restored. The original magnitude of P(k) and the state of F0 were always stored in RAM memory before the update operation was undertaken. So, before executing Equation 3.3 with a new template element both P(k) and F0 were restored the original values.

Mathematical Considerations

Multiplication. The multiplication subroutine takes two unsigned eight-bit numbers and returns a sixteen-bit product. The two methods (26) most commonly used for performing binary multiplication are

1. repetitive addition;
2. register shifting.

In the software package of the PPS, the shifting algorithm is implemented because of its faster execution speed.

The shifting algorithm (see Figure 3.7) involves sequentially testing each bit of the multiplier word starting with the least significant bit (LSB). The procedure requires a sixteen-bit addition followed by a division by two. If a particular bit in the multiplier is one, then the multiplicand is shifted to the left eight places (multipled by 2^8); the result is added to the partial product; and finally, the partial product is shifted to the

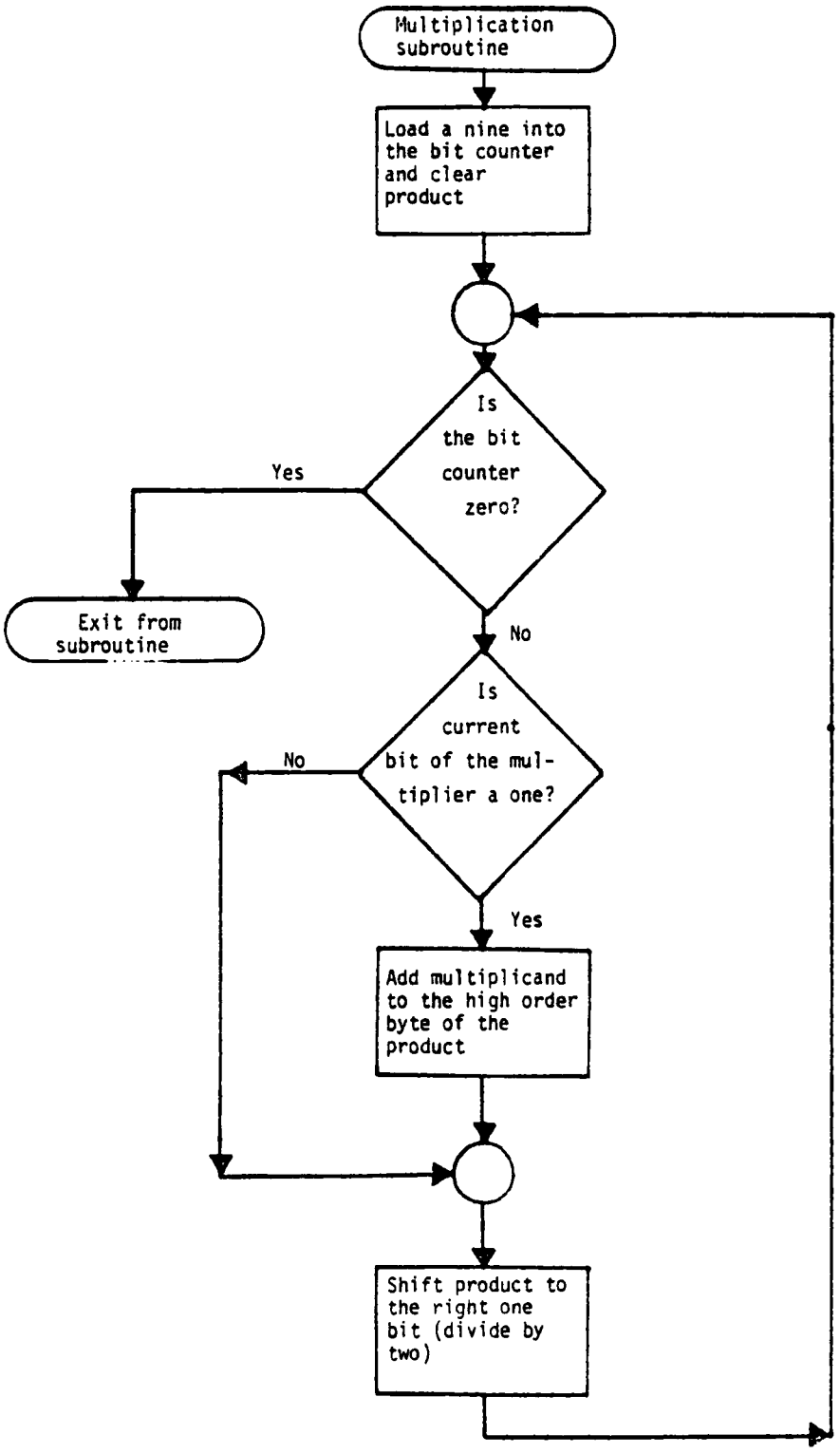


Figure 3.7. Eight-bit multiplication algorithm.

right by one place (division by 2). If the multiplier bit is clear, then only the division of the partial product needs to be performed. The add-then-shift procedure is equivalent to the following summation:

$$P = bit_i * multiplicand * 2^{(i-1)} \qquad [3.4]$$

The shifting algorithm requires at most eight additions while repetitive addition can need up to 255, thus the shifting algorithm is preferred because the execution rate is consistently faster.

Division. Binary division by a power of two was relatively easy to program because it requires only a series of shifting operations. Since a division of a sixteen-bit word by 2^x requires that the word be shifted x positions to the right, the divide routine was programmed to add $2^{x/2}$ before performing the shift to insure that the result was rounded to the nearest integer.

Nine-bit multiplication/scaling algorithm. In evaluating equation 3.3 a series of multiplications and divisions are performed. Equation 3.2 is first evaluated to obtain P(k) and the product of this operation is stored as a sixteen-bit number: recall that all bits above the ninth are always clear. Next, each element in the step template is multiplied by this product (equation 3.1) then divided by 128.

The nine-bit by eight-bit product is formed by first multiplying the eight-bit step template element by the lower order byte of the multiplier. If the ninth bit of the multiplier is set, one

extra addition is needed but a right shift is not (see the basic multiplication routine description). The resulting product is potentially seventeen bits long (overflow into the carry bit of the accumulator), but, the division was not structured to deal with seventeen-bit numbers; special processing is required when this result occurs. This is handled in the routine by

1. saving the state of the seventeenth bit (carry bit);
2. performing the standard division by 128 on the resulting sixteen-bit number;
3. then setting the state of the tenth bit in the quotient according to the value of the saved seventeenth bit.

The tenth bit corresponds to the location that the seventeenth bit should occupy after a division by 128. In Figure 3.8 is a flowchart of the logic required to implement this extended multiplication/scaling algorithm.

Two's complement arithmetic. Signed binary addition and subtraction are essentially the same operation--addition. The difference between the two lies in the data representation. The most common way of representing signed binary integers is called "twos-complement" or 2s-complement (27). In this scheme a number consists of (n+1) bits with the MSB being used to indicate the sign, so in sixteen-bit (n=15) 2s-complement the allowable range of numbers is from -32768 to +32767. A binary number is converted to its 2s-complement by first changing all bits to the opposite state (1s to 0s and 0s to 1s), then adding one. The result is known as

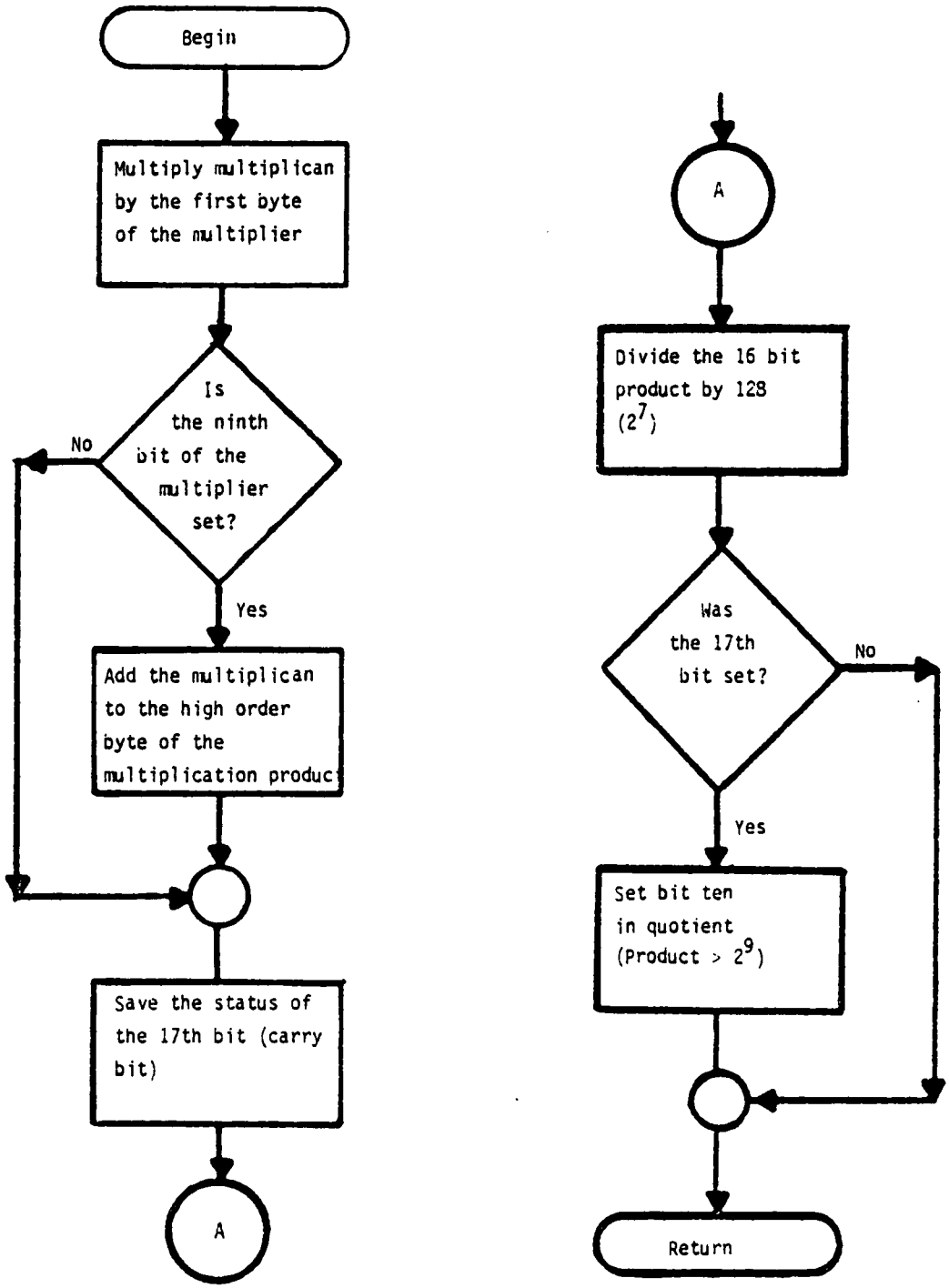


Figure 3.8. Flowchart for a nine-bit by eight-bit multiplication followed by a division by 2^7.

the additive inverse, because when the original number and its 2s-complement are added together the result is zero. A binary subtraction of two numbers consists of 2s-complementing the subtrahend and adding it to the minuend. In binary addition the 2s-complement step is bypassed.

Algorithms that are to perform n-bit signed additions and subtractions must perform some bookkeeping to avoid erroneous results. In an addition it is possible for positive addends to yield negative results; this is called additive overflow which means that the result exceeds the representable range of positive numbers. Subtractive overflow means that the addition of two negative numbers yielded a positive number, which again means that the result lies outside the range of n-bit arithmetic. Both the ADD and the SUBT subroutines in the PPS detect overflow conditions and return the maximum representable signed value.

The logic required to perform addition and subtraction is shown in Figure 3.9. Before invoking the algorithm the calling routine loads two sets of register pairs with the numbers to be added or subtracted. One of the register pairs is always destroyed in computing the result, but the other pair is left unchanged. The undisturbed register pair is used to help determine whether an overflow has occured. In the subtraction subroutine the number in the subtrahend is always destroyed.

Subtraction. The subtraction subroutine's first operation was to convert the subtrahend to its additive inverse. If the

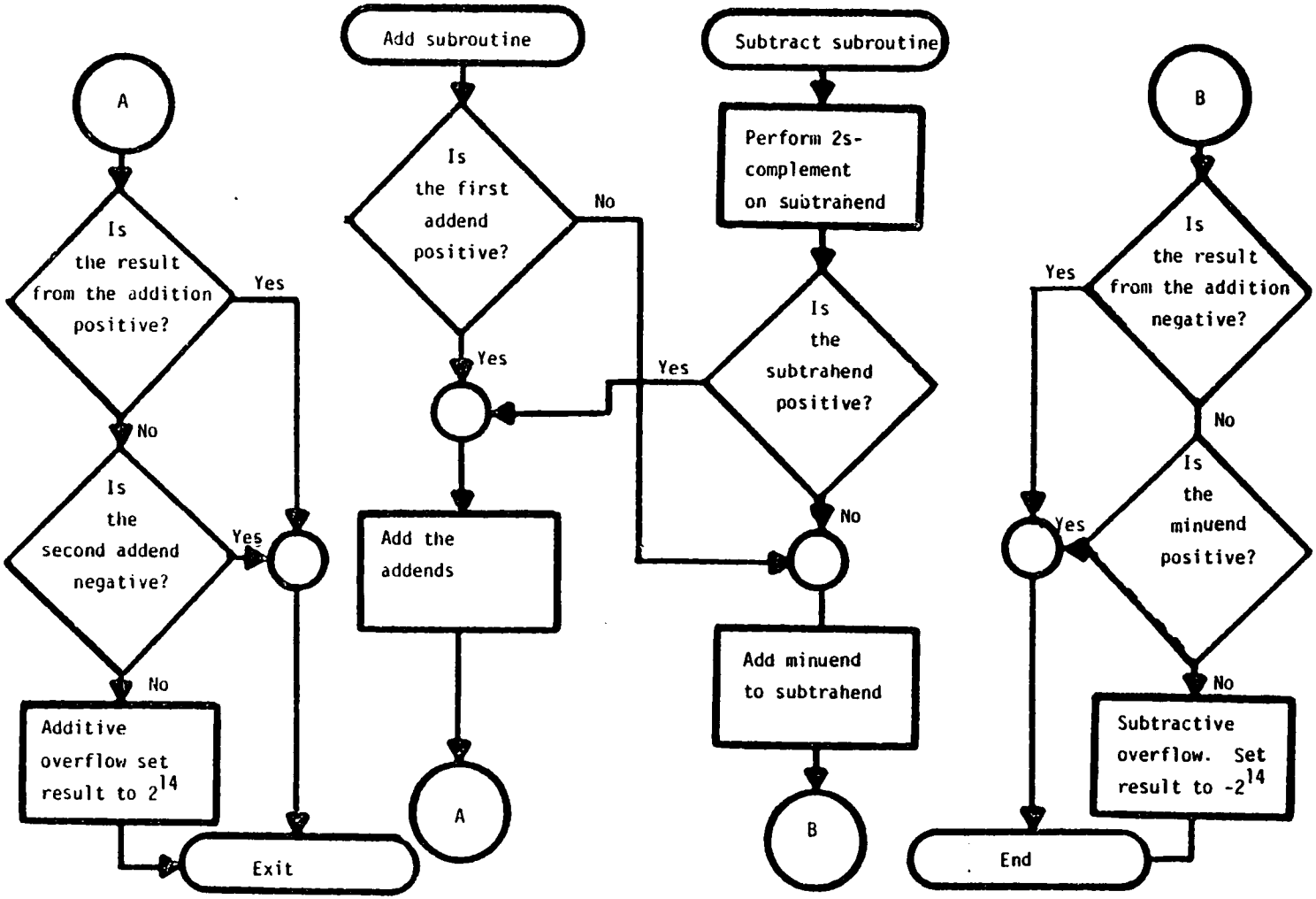


Figure 3.9. Logic diagram for the signed binary addition and subtraction routines.

inverse was positive then, in effect, the routine was being asked to perform an addition of two numbers in which at least one number was positive. Therefore, only additive overflow could result, so control was passed to the addition subroutine.

If the inverse was negative (a true subtraction), then the minuend was added to the subtrahend, and the answer stored in the subtrahend register pair. If the result from the subtraction was negative, then control was returned to the calling routine; this was because a negative answer was the expected result. If the answer was positive, then the minuend had to be positive also, otherwise, a subtractive overflow condition existed because the addition of two negative numbers would have yielded a positive result. Upon detection of an overflow condition, the result returned was -32768.

Addition. As with SUBT, the ADD routine began by testing the sign of one of the numbers. Control was passed to the subtraction subroutine if this number was negative; if both numbers were positive, then the two addends were summed together. A positive answer indicated that no overflow had occured; a negative result indicated that an overflow condition might exist; with a negative result one of the addends had to be negative as well. The initial test upon one of the addend insured that at least one of the numbers was positive. If the other addend was positive too, then two positive numbers had yielded a negative result: an additive overflow. This condition resulted in 32768 being returned as the computed sum.

3.3 OPERATING SYSTEM

The operating system of the PPS was designed to connect the man/machine interface (MMI) to the simulation algorithms. The system consisted of the subroutines and algorithms that allowed the user to configure (program) the device and to issue commands to initialize and initiate dynamic simulations. The command entry station of the system was modelled on the pushbutton structure of analog computers; configuration programming was menu driven. The basic structure of the operating system was chosen because it was easy to learn and simple to operate.

Organization of the Operating System

A menu of six thumbwheel switches was provided for configuration programming; each switch had sixteen (0 to 15) different settings and the function of a switch was identified by a one character mnemonic located above it. The first four switches (P, G, S, D) were used to characterize the dynamic relationship between an input and the output while the last two (B and N) specified the magnitude of a bias term and the process noise level. The following list identifies the mnemonic and provides a functional definition for each switch.

- P was used to identify the particular shape of the transient step response such as first order, second order overdamped, second order underdamped and et cetera.
- G established the entry point into a table of steady state gains.

S established the entry point into a table of sample times.

D established the entry point into a table of dead times. This parameter was used to delay an input a specific number of sample times.

B established the entry point into a bias table. The bias term was added to the ADC reading during an initialization operation and the result loaded into the state vector and written to the DAC.

N established the level of a noise term that was added to the output every 0.09 seconds.

Whenever one of the three "read" command pushbuttons (RESET, M-IC or D-IC) was pressed, the complete set of switches was loaded into the system and stored in one of two sets of configuration arrays (CA). Each switch provided four bits of information to the computer and data from two switches were stored in a single byte of memory known as a configuration element. Both inputs (MV and DV) had three configuration elements reserved for storing switch settings; these sets of storage were the configuration arrays.

A power up or RESET operation resulted in the same switch settings being stored in both CAs. This effectively configured the simulator as shown in Figure 3.10. Whenever the arrays were different the dual input, single output (DISO) simulation network shown in Figure 3.11 resulted. A particular CA (MV's or DV's) was

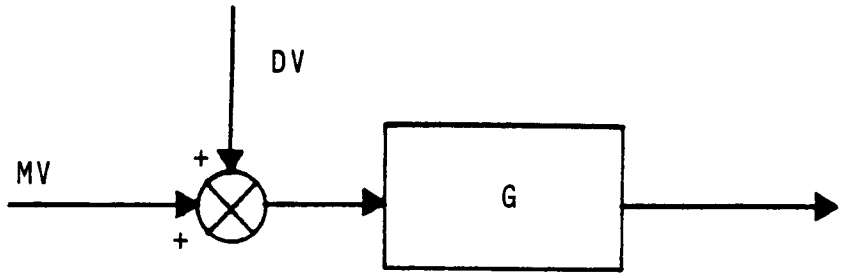


Figure 3.10. Simulator configuration after a power up or a reset operation.

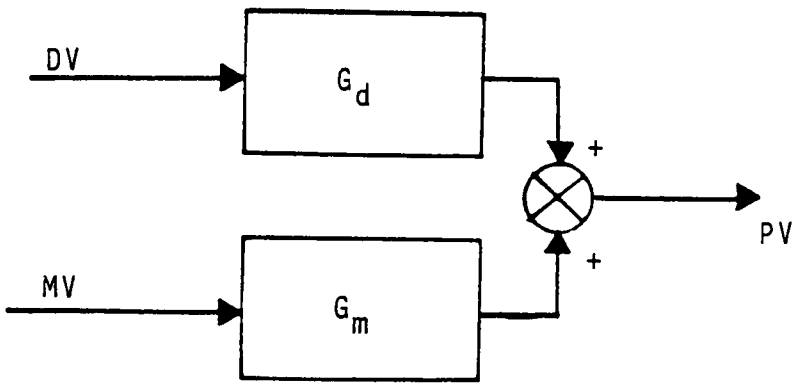


Figure 3.11. Simulator configuration when the configuration arrays are different.

changed using the command pushbutton (M-IC or D-IC) associated with that array.

Pressing any of the four command pushbuttons (or turning on the power switch) forced the processor to enter the command string interpreter (CSI) algorithm shown in Figure 3.12. If the algorithm was entered via a "read" command, the first action was to read the thumbwheel switches and store the values in the appropriate CA; next the DAC output was initialized using the current MV reading. The system then issued a "Clear" command to restore the command flip-flops, and entered a waiting state until the detection of an OP command.

Entry into the CSI via the OP command did not initiate a read operation; however, it did force a complete simulation initialization using the numbers already in the CAs. Initialization consisted of several processing steps. The first step was to read each ADC channel and store the number in the appropriate memory location. The next step was to compute PV using MV and its B thumbwheel setting. This operation is summarized in Equation 3.5

$$PV = MV + t(B) \qquad [3.5]$$

where t(B) is element number B in the bias table (see Table 3.1) and PV is a sixteen bit signed integer. PV was then used to load the state vector associated with MV (first 202 locations in page one of RAM) and the DAC output (if PV range from 0 to 255). The fourth step was to clear the state vector associated with DV (third page of RAM).

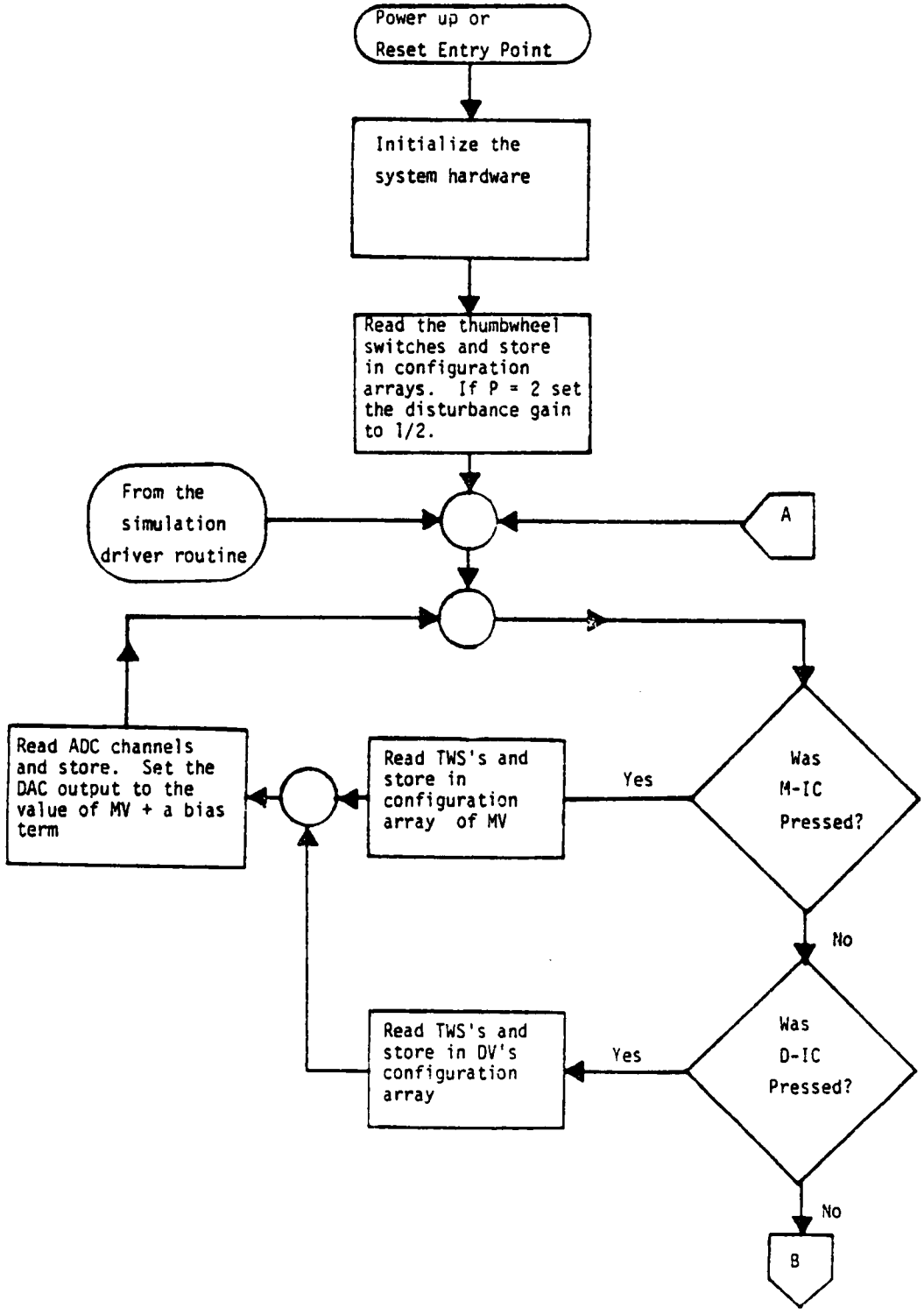


Figure 3.12. Command string interpreter and power up/reset algorithm.

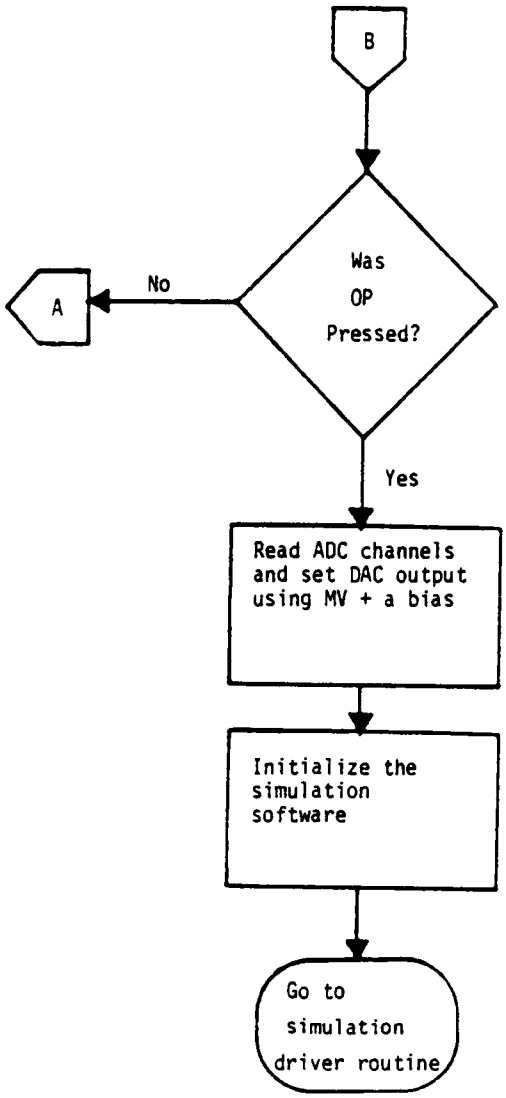


Figure 3.12. (continued).

Table 3.1

Bias Values Versus the "B" Thumbwheel Switch Settings

Thumbwheel Switch #	Magnitude of the Bias Term %
0	00
1	05
2	10
3	15
4	20
5	25
6	30
7	40
8	50
9	- 45
10	- 40
11	- 35
12	- 30
13	- 25
14	- 20
15	- 10

After clearing page three, the processor initialized two sets of ring pointers and three delay tables. There was a table and a set of pointers for MV and two tables and a set of pointers for DV (three tables were necessary because of the way the system operated in mode two).

A set of ring pointers consisted of an input ring pointer (IRP) and an output ring pointer (ORP). The IRPs were used to store the current ADC values in the delay tables and the ORPs were used to read previous values from the tables. The IRPs were initially loaded using the D thumbwheel settings in the two CAs (as with B, the number stored in the CA acted as index into a table) and both ORPs were initially set to one.

The delay tables were initialized once the ring pointers were loaded. The upper 128 bytes of RAM page two served as a delay table for storing MV. DV was stored in the lower 128 bytes of RAM page two and in the upper 128 bytes of page zero (the template array was stored in the lower 113 bytes). These tables, along with the IRPs and ORPs, were used by the simulation subprograms to simulate dead time.

The final step of the initialization routine was to issue the Clear command which reset the flipflops connected to the command pushbuttons and switched on the "simulator ready" light. The Clear command involved first setting then clearing the most significant bit of the control logic on the microcomputers's output port. Once the Clear command was issued the light on the front panel would be illuminated to indicated that the program was running.

The OP command initiated two other tasks besides the initialization routine. The first task was to read the current value of the eight bit system clock and store it; this number was used in a routine for generating random noise. The second task was to initiate the simulation software.

3.4 SIMULATION SOFTWARE ORGANIZATION

Simulation Software Overview

The simulation software consisted of a central coordinating program and three subprograms. The main routine (OPERATE) was responsible for co-ordinating the execution of two independent simulation subprograms (MSIM and DSIM) and a DAC output subroutine (OUTPUT). The simulation subprograms were used to generate a DISO simulation network (see Figure 3.11 page 56) by invoking the step template simulation subprogram (SIM). The DAC output subroutine was responsible for combining the outputs from the two simulations and adding noise to the result. Figure 3.13 was provided to show the organization of the main routine and the calling sequences.

Main driver routine. The first task of OPERATE was to test the command status input (interrupt line). A true result indicated that control of the computer was to be returned to the command string interpreter (CSI). A false condition meant that the routine had to decide whether

1. to call MSIM;

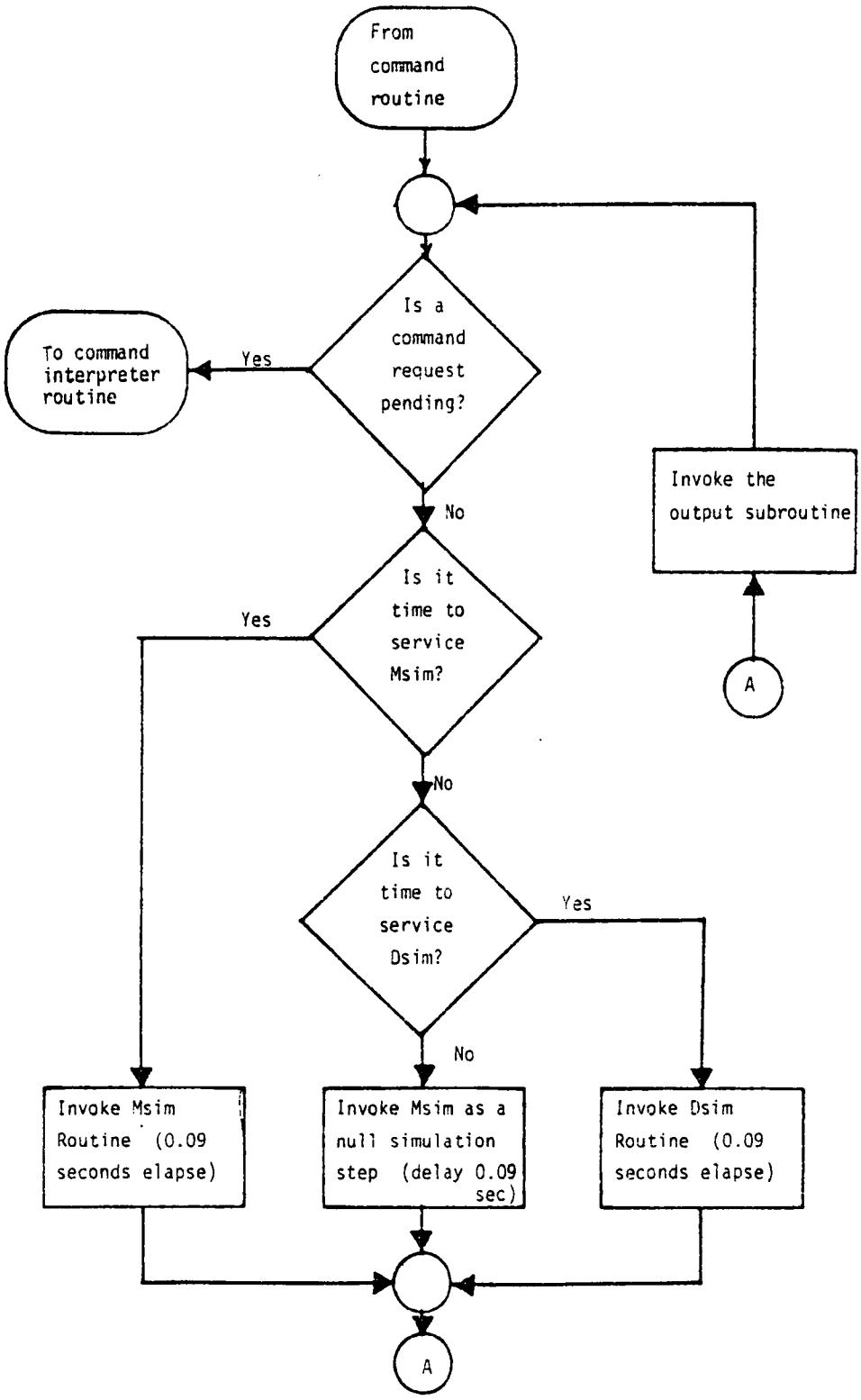


Figure 3.13. Main driver routine (Operate) for simulation software.

2. to call DSIM; or

3. to mark time by setting the null status byte to zero and calling MSIM.

The execution of any one of these routines required 0.09 seconds and the process of calling the subprograms was used to establish the basic execution rate of the software.

The maximum rate at which either of the two simulation routines could be executed was limited by the computation time of SIM which was 0.09 seconds; SIM was invoked by both MSIM and DSIM. However, for correct execution the maximum rate at which the output from either one of the routines could be updated was 0.18 seconds. (This will be clearer after the timing operation is explained.) The timing operation was handled in the following way:

> Every time SIM was called it incremented two counters (MCOUNT AND DCOUNT) to indicate that a time step had been taken. OPERATE's second task was to compare MCOUNT to MSPEED (MSPEED and DSPEED were set using the S thumbwheel switch; Table 3.2 provides the thumbwheel switch settings, the corresponding counts and resulting sample times). If MCOUNT and MSPEED were equal, then OPERATE set MCOUNT to zero and invoked MSIM (which called SIM); this resulted in both counters being incremented by one.
>
> If the first test failed, DCOUNT was compared to DSPEED and if the two matched, then DCOUNT was cleared and DSIM was executed. Otherwise, the null or

Table 3.2

Thumbwheel Switch Settings, Counts, and Actual Sample Times Associated with the "S" Thumbwheel Switch

Thumbwheel Switch #	# of Counts	Sample Time (seconds)
0	2	0.18
1	6	0.36
2	10	0.54
3	14	0.72
4	18	1.08
5	22	1.26
6	26	1.44
7	30	2.16
8	38	2.88
9	42	3.60
10	46	3.78
11	50	3.96
12	54	4.32
13	58	5.04
14	62	5.76
15	66	6.12

dummy algorithm was executed; this marked time or delayed the execution rate by 0.09 seconds.

In Figure 3.14 is an example which shows the subroutine calling sequence and illustrates how values of the counters changed during several passes through OPERATE. In the example MSPEED was four and DSPEED was two; this yielded sample times of 0.36 and 0.18 seconds respectively.

Simulation sample times were restricted to even multiples of 0.18; this was because if either subroutine's sample time was an odd, at some point in time both routines would require servicing simultaneously. The conflict could not be resolved and one routine would get out of synch (because one counter overflowed its queuing value). This would result in the subroutine having a time varying sample time and the simulation would exhibit nonlinear behavior. The initial values of MCOUNT and DCOUNT also had to be staggered in order to avoid the possibility of having both routines pending simultaneously.

The process variable was written to the DAC once the appropriate simulation subroutine had been executed. This job was the responsibility of the OUTPUT subroutine. After invoking this subroutine the main driver routine cycled back to retest the command status input and start the simulation computations again.

Output subprogram. The output subprogram (OUTPUT) was executed on every pass through OPERATE. The responsibility of this routine was

Time Elapsed Since "op" Command	M count	D count	Routine Called
0	0	1	Null
0.09	1	2	Dsim
0.18	2	1	Null
0.27	3	2	Dsim
0.36	4	1	Msim
0.45	1	2	Dsim
0.54	2	1	Null
0.63	3	2	Dsim
0.72	4	1	Null
0.81	1	2	Dsim

Figure 3.14. Example of simulation subroutine calling sequence when Mspeed is four and Dspeed is two.

1. to add the top elements in the state vectors together to form PV;
2. to determine the magnitude of the random noise term and add this to PV;
3. to write the value of PV out to the DAC.

Noise was generated using a table lookup procedure. Whenever OP was pressed the current value on the timer was read and loaded into a one byte storage location called RAND; this was used as the seed for generating noise. RAND pointed to a location in a table of 254 randomly generated numbers that ranged from -5 to +5; these numbers were from a normal or Gaussian distribution having a mean value of zero and a standard deviation of one bit. (The method for generating the set of gaussian random numbers was suggested by Forsythe, Malcolm and Moler (65)).

Every time OUTPUT was called, RAND pointed to the next random number to be read. The magnitude of the additive noise was determined by the N value in MV's CA; the actual value ranged from +5N to -5N counts. Once generated, the noise signal was added to PV and the resulting number was tested for a DAC overflow or underflow. If neither error condition occurred, PV was written out to the DAC; otherwise, the output had to be limited to within the eight bit range. Figure 3.15 shows a block diagram for the logic of OUTPUT. A typical set of process step responses for three different levels of noise is shown in Figure 3.16.

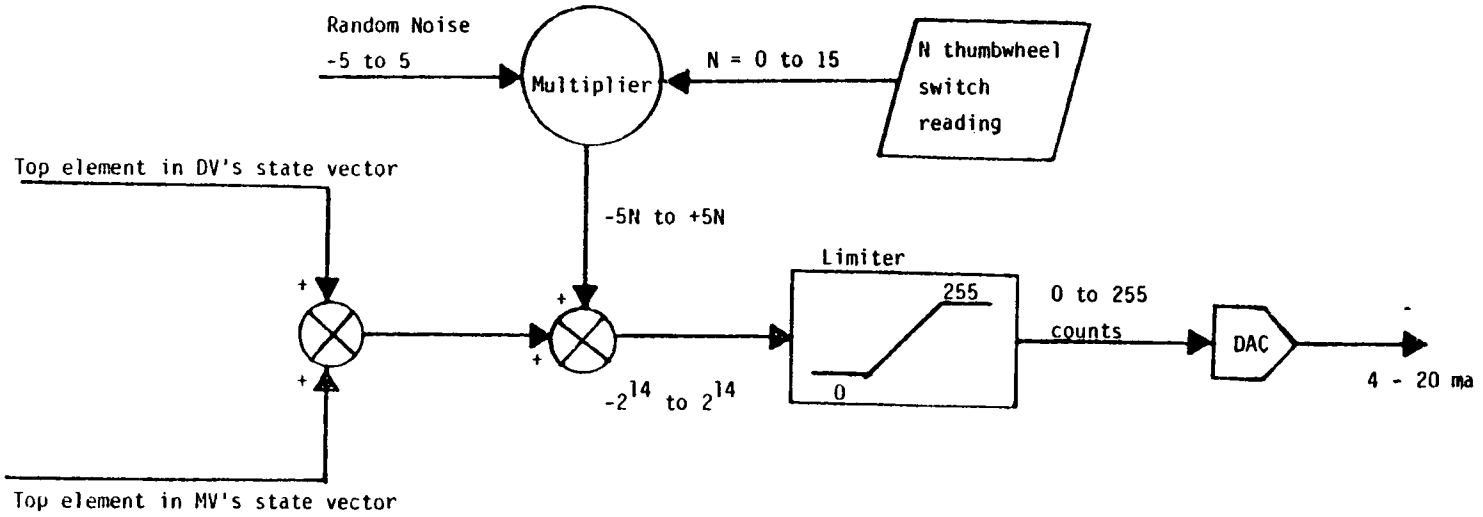
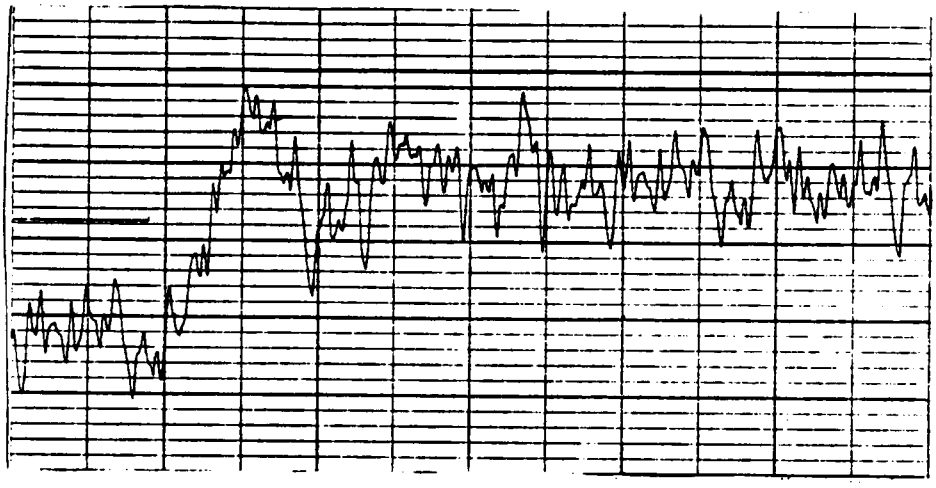
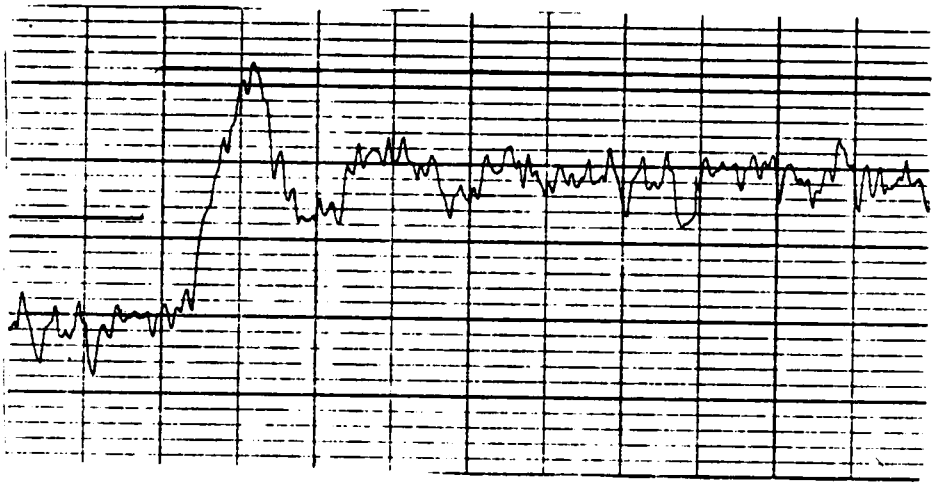


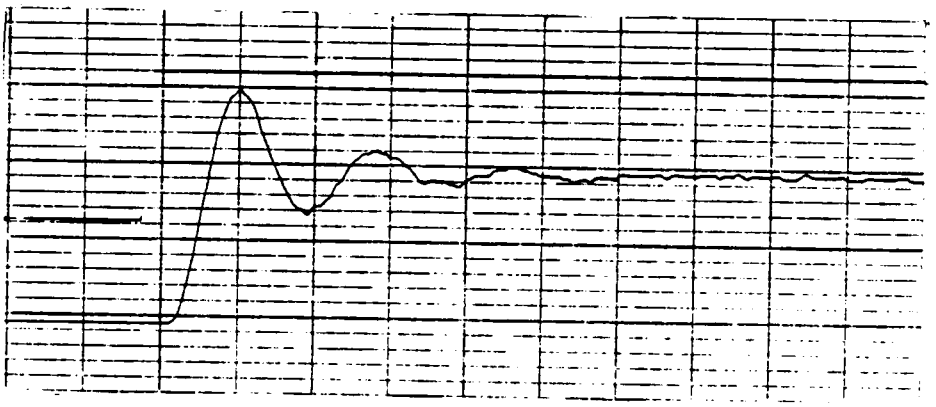
Figure 3.15. Block diagram of the output subroutine.



(c)



(b)



(a)

Figure 3.16. Step response of process six for N thumbwheel switch settings of (a) zero, (b) five and (c) ten.

Simulation subprograms. The basic simulation software consisted of a subroutine (SIM) that performed one complete iteration of a step template simulation and two subprograms (MSIM or DSIM) that used SIM. The calling routine was identified by the state of a bit in the simulation status byte; this information was used to determine the appropriate state vector to use in the computations.

SIM was also used by the main driver routine (through MSIM) to control the execution rate or sample time of the simulation blocks. If the main routine (OPERATE) needed to mark time because neither subprogram was queued, then it simply invoked MSIM with the null status byte set to zero. This informed SIM that all of the RAM chips were to be switched off, which, in effect, created a null simulation step. SIM took approximately 0.09 seconds to execute and set the overall execution rate because the subroutine was invoked on every iteration through the main routine.

Simulation Subroutines--Operating Details

Every time one of the simulation subroutines (MSIM or DSIM) was invoked, the first two tasks were (a) to load the proper step template into RAM and (b) to set up the simulation status byte. Because the step template was stored in program memory it could not be readily accessed and therefore, had to be transferred into RAM memory before the processor could manipulate it effectively. Both subroutines performed essentially the same functions; however, there were some differences. Some of these differences will be pointed out in the discussions on the details of the specific simulations

subprograms. These will be discussed in the next two sections.

Manipulated variable simulation subprogram--MSIM. MSIM was responsible for simulating the response of PV to MV; however, if the null status byte was cleared by OPERATE before calling MSIM, then a null step was taken. Recognition of the null command in the status byte informed the subroutine that the ring pointers were not to be updated, that RAM was to be left switched off and that u(k-1) should not be loaded with u(k). This negated all of the computations initiated by MSIM and simply resulted in a delay of 0.09 seconds.

After loading the template and setting the status byte, MSIM (a) read the current values of MV and DV, (b) stored the values in the delay tables and (c) loaded the delayed values of MV and DV from the tables. (It was necessary to read DV because in mode two the variable was needed by SIM.)

Delay tables and ring pointers (IRPs and ORPs) were used to simulate dead time. The IRP pointed to the location in the table where the current ADC readings were to be stored; the ORP pointed to a location where an input D samples in the past had been stored. The update policy of the IRP was increment-then-write and the policy for the ORP was read-then-increment; the policy allowed dead times from 0 to 255 sample times to be programmed. For example, if the initial settings of the IRP and ORP were one and two respectively, the result would be zero samples of dead time. However, if the IRP was loaded with a number equal to or larger than the number in the ORP then the input would be delayed by a number of samples;

the delay would be equal to the difference in the two numbers plus one.

The delay tables were 128 bytes long and resided in the upper half of RAM pages zero and two. The ring pointers cycled through the these bytes because the MSB was always set to one by the computer. If the pointer "rolled over" or went from 255 to 0 (256 cannot be represented with an eight bit number) setting the MSB restored the value to 128. The current MV reading was always written into page two and DV was written into page zero (both using the same IRP). The table used to store DV was denoted as the MDV storage array. The next values of MV and MDV that were to be used by SIM were read from the delay tables using the ORP.

Once the new input values were read from the buffers, MSIM would load the u(k) address pointer (index) register with the address to be used in computing U(k). Control of the processor would then be passed to the simulation subroutine (SIM).

Disturbance variable simulation subprogram--DSIM. The operation of DSIM and MSIM differed in two ways. First, DSIM read only one ADC channel, not two, and second, when the MSS was set to mode two or if M-IC was initially pressed with a P value of four, then channel zero (MV) was read instead of channel one (DV). This forced the two independent simulation blocks to have the same input variable (MV) and was defined as the parallel input, single output (PISO) mode of operation. The determination of which channel to read was always made by the ADC read subroutine. When DSIM called

MIN1 (channel one read) this subroutine would test the status of the MSS and the P setting in MV's CA and if either one was set, the routine passed control over to MINO (channel zero read). Thus, the operating mode was transparent to the basic software of DSIM. A flowchart showing the logic of the ADC subroutines is shown in Figure 3.17.

As with MSIM, DSIM loaded the address where the last value of DV was stored so that SIM could compute U(k). SIM was entered with the MSB of the simulation status byte clear; this designated DSIM as the calling routine.

Step template simulation subroutine--SIM. SIM performed a step template simulation using the step template update equation.

$$x(k)_i = x(k)_{i+1} + g*U(k)*s_i: \; i=1,2,\ldots,100 \qquad [3.6]$$

The appropriate state vector was determined by the one bit flag in the simulation status byte. If the MSB was set (on), the state vector resided on page zero of RAM (MV's state vector) and if the bit was clear (off) then page three contained the correct vector.

SIM normally (a) computed the value of U(k) and (b) determined the product and sign of the gain times U(k) and (c) stored u(k) in u(k-1). However, under some conditions the update operation on u(k-1) was not implemented due to the different ways in which U(k) could be computed. Also, U(k) was not always computed as simply u(k)-u(k-1). If the MSS was set to mode two (parallel mode)

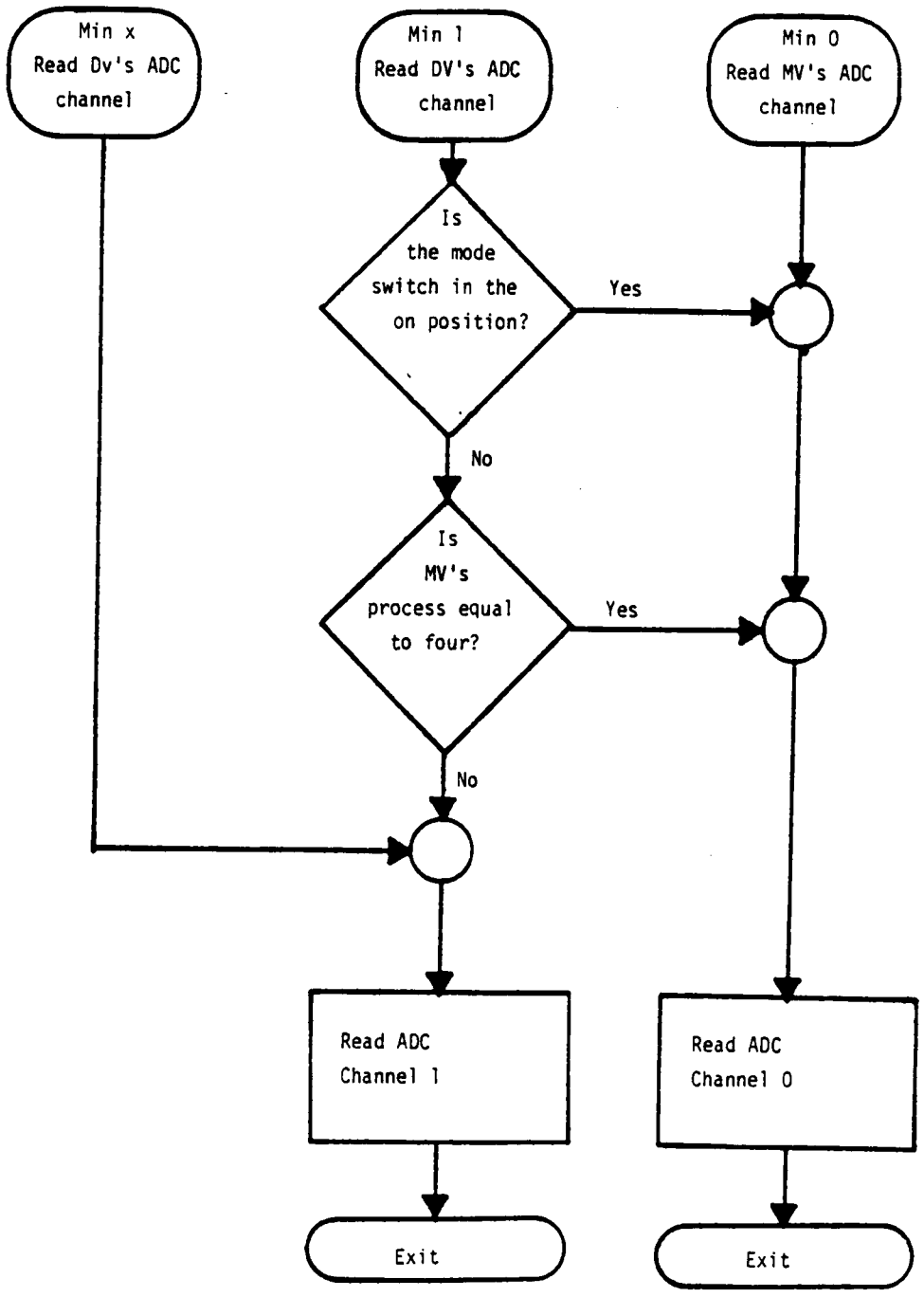


Figure 3.17. Logic for reading the analog to digital converter channels.

and MSIM was the calling routine, then Equation 3.7 was used to compute U(k).

$$U(k) = MV(k) - MV(k-1) + MDV(k) - MDV(k-1) \qquad [3.7]$$

If DSIM was the calling routine and the value of the P thumbwheel switch P was one or two, then U(k) was computed using Equation 3.8.

$$U(k) = DV(k) - DV(k-1) + PV(k) - PV(k-1) \qquad [3.8]$$

Equation 3.8 indicates that PV was used as a positive feedback signal to the simulation computations; this created potential for unstable responses.

When either equation was evaluated, there was a possibility that the result could exceed eight bits. This meant that P(k) could exceed nine bits and thus, would not be compatible with the basic or special multiplication subroutines. Whenever this situation was detected some set of actions had to be taken and the particular set depended upon the equation being evaluated.

If the overflow resulted from Equation 3.7, the actions were (a) to compute U(k) = MV(k) - MV(k-1), (b) to set delta MDV to zero and (c) to not update MDV(k-1) with the current value of MDV; this essentially delayed the effect of the disturbance by one sample time. If the overflow resulted from Equation 3.8, then the current delta PV was simply discarded (but PV(k-1) was updated) and U(k) was computed using MV(k) and MV(k-1) exclusively. Since delta PV was normally a small number (usually one or two counts) this resulted in

very little loss of accuracy. Also overflow conditions did not occur often because large swings in the inputs were unusual.

An overflow was not the only condition that would result in a particular u(k-1), (MDV(k-1) or PV(k-1)) being left unchanged. If process zero or three was selected, then U(k) was evaluated using Equation 3.9. In this case u(k-1) or u(ss) always retained the initial value read during an initialization operation.

$$U(k) = u(k) - u(ss) \qquad [3.9]$$

When Equation 3.9 was substituted into Equation 3.6 with a unity step template vector (all ones) the result was a process which had the characteristics of an "integrator" process.

Once U(k) was evaluated, the product and sign of g*U(k) had to be determined. This was followed by an update of the state vector using Equation 3.6 and a fill operation. The update routine was programmed to begin by loading two index pointers, ST and SV. ST was used to access the 100 bytes of the state template and SV pointed to the state vector. ST was initialized to zero (starting point of the template), while SV was loaded with a two; this was the storage location of the lower order byte of the second word of the state vector.

Each template element was read from RAM page zero and multiplied by g*U(k) using the nine bit multiplication and scaling algorithm. The result was a sixteen bit positive product. The sixteen bit state vector element was then loaded, and the ADD routine called if F0 was true (positive g*U(k)) and if F0 was false the SUBT

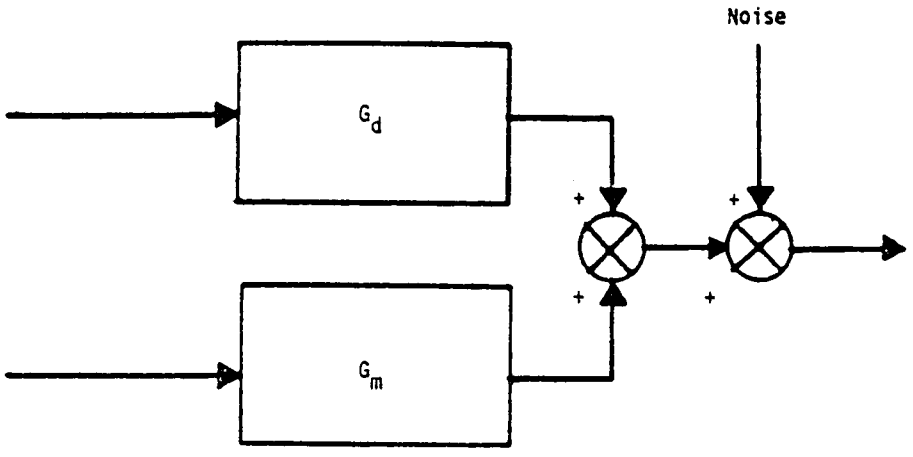
subroutine was called; the resulting sixteen bit number was then stored in RAM using SV. The processor cycled through every element in the template and when the operations were completed, the state vector held a new process output trajectory; SIM was then almost ready to return control to OPERATE. Before exiting, the last responsibility of SIM was to update MCOUNT and DCOUNT to indicate that a time step had been taken.

3.5 OPERATING CHARACTERISTICS OF THE PPS

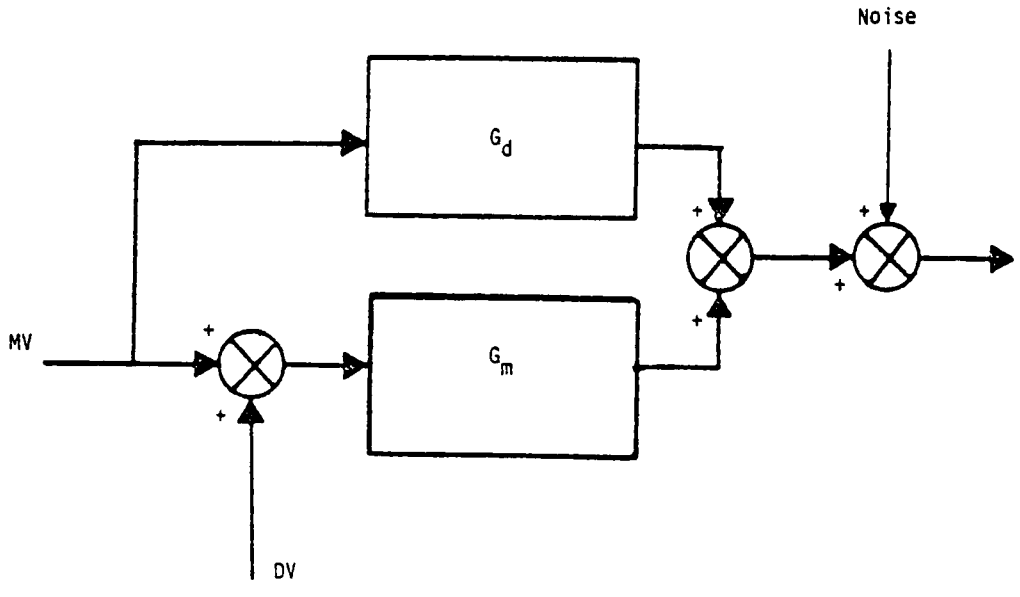
The PPS was designed to have two major operating configurations (modes) and four submodes. Modes were selected using the mode selector switch (MSS). The standard, or DISO, configuration resulted whenever the MSS was set to the off or down position. The up position specified the parallel input (PISO) mode which was discussed in conjunction with Equation 3.7 and in this configuration the ADC software supplied MV to both MSIM and DSIM. Block diagrams which illustrate these two different operating configurations are shown in Figure 3.18.

Submodes were provided to allow the P thumbwheel switch to perform a dual function. However, in order to create submodes using P, limitations had to be imposed on the system; selection of a submode forced at least one of the two simulation blocks to a default process template. This allowed P to function as a submode specifier and a template selector.

In each submode the software was programmed to deal with the process inputs in special ways. Two of the submodes (process one or



(A) Dual Input, single output mode (DISO)



(B) Parallel Input, single output mode (PISO)

Figure 3.18. The two operating modes of the simulator--set using the mode selector switch.

two) were structured to be positive feedback in nature (see Equation 3.8). The block diagrams of the DISO and PISO versions of the positive feedback processes are shown in Figure 3.19.

Two submodes (processes three and four) were structured to be inherently PISO in nature and were provided to allow parallel processes in the basic menu of characteristics. In process three Equations 3.7 and 3.9 were combined to compute U(k) while in process four only Equation 3.7 was used to compute U(k)

In the remaining set of twelve process templates there were eleven asymptotically stable and one marginally unstable process characteristics. The process characteristic associated with each of the P thumbwheel switches will be described in the following paragraphs.

Process Step Templates

Process zero--integrating process. Process zero was programmed to have a pole at the origin; this resulted in the process being marginally unstable. The response to a step input was a ramp output which is typically defined as an integrating characteristic.

Equation 3.9 was used to evaluate U(k) as u(k)-u(ss). The value of u(ss) was loaded during an initialization as u(0), which was always stored in the u(k-1) storage location. If, during the execution of the program, the software determined that process zero had been selected, then the value in this location was never updated (u(k-1)=u(ss)). The result was that U(k) was not zero unless u(k)

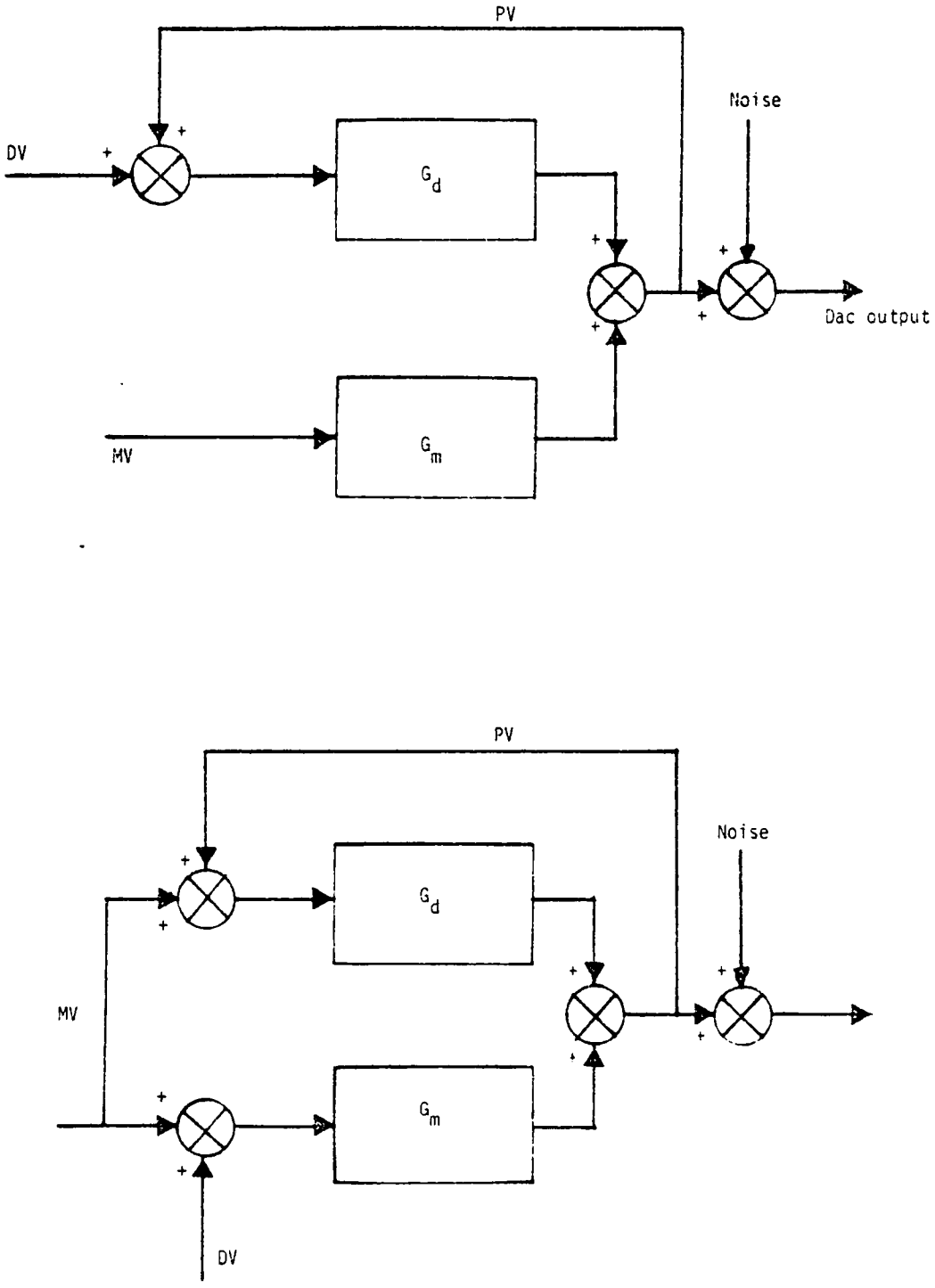


Figure 3.19. Positive feedback submodes for DISO and PISO modes of operation (processes two and three).

equaled u(ss); this meant that the input was always driving the output unless the two were equal. If the template was loaded with all ones then the STM update equations simplified to Equation 3.10

$$PV(k) = PV(k-1) + g*U(k) \qquad [3.10]$$

which is just the recursive equation of a pure integrator. Thus the rate of change of the output depended on the magnitude of the gain, the magnitude of U(k) and the sample time. A block diagram of the process and an example pulse response are shown in Figure 3.20.

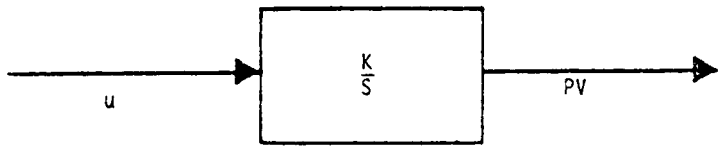
Process one--positive feedback process. Selection of process one specified to DSIM that delta PV was to be summed with U(k) (Equation 3.8); however, this submode could be invoked using either the M-IC or the D-IC command. Figure 3.21 illustrates the difference between the two ways of selecting the submode. If the process was selected using M-IC, then G_m became fixed, but G_d could be chosen freely. If the process was selected using D-IC, then G_d became fixed. The one limitation on G_x was that processes three and four (the other two submodes) could never be used as G_m or G_d.

The default transfer function for process one was a first order lag and is given in Equation 3.11.

$$G_m = g/(20s + 1) \qquad [3.11]$$

The s-domain transfer function that related PV to MV and DV is given in Equation 3.12.

$$PV = (G_m*MV + G_d*DV)/(1-G_d) \qquad [3.12]$$



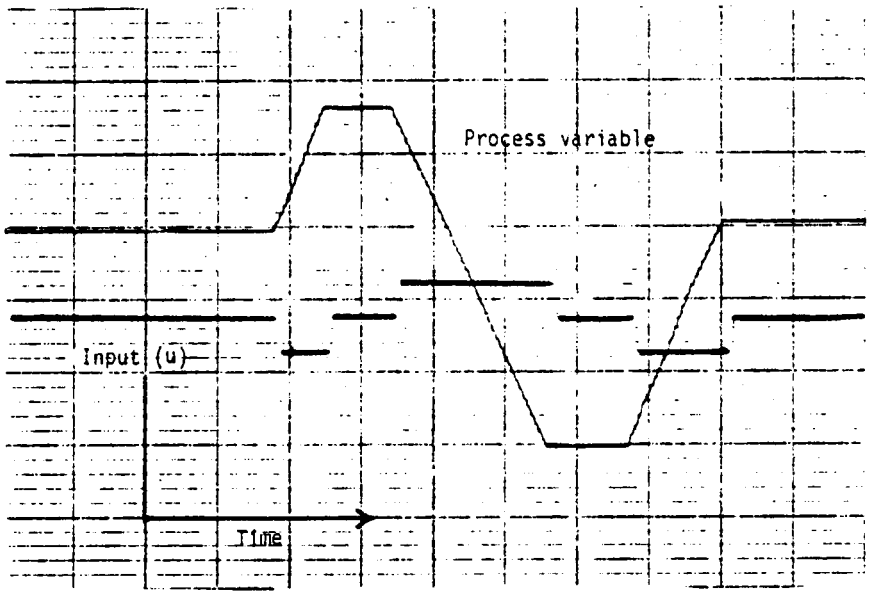
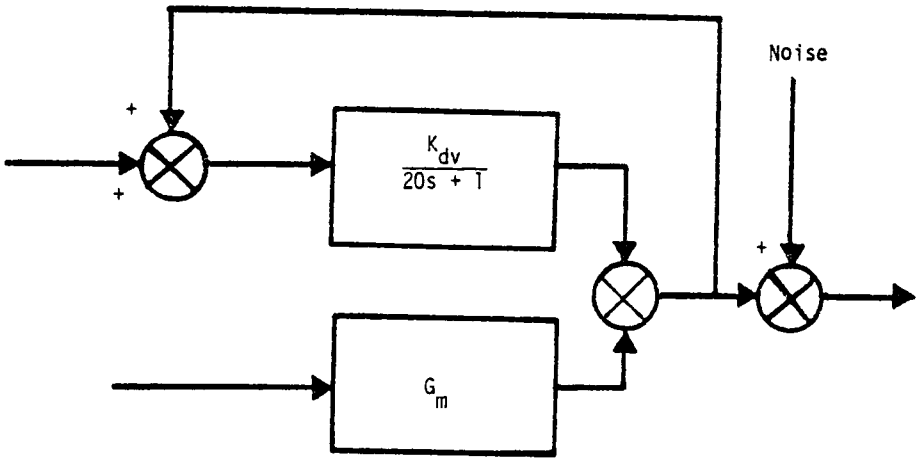
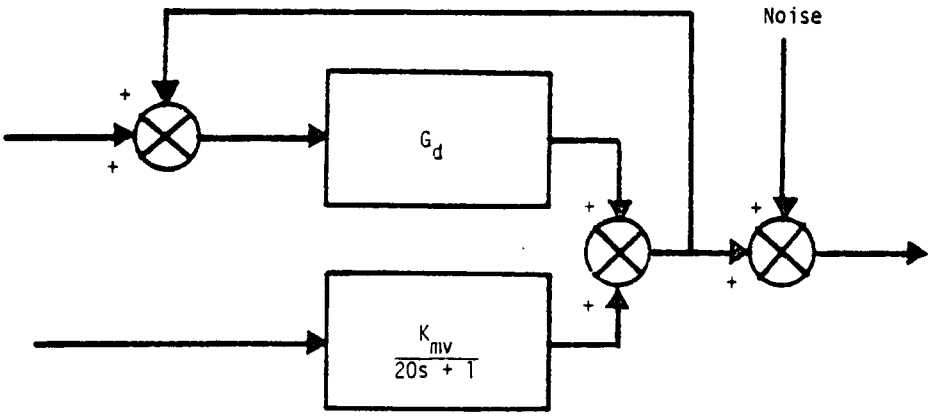


Figure 3.20. Process zero's (integrating process) block diagram and pulse response.



(A) Default configuration when D-IC is used to select process one



(B) Default configuration when M-IC is used to select process one

Figure 3.21. Default simulation configurations of process one.

Forcing the process variable to be fed back as an input signal had the effect of creating stable, marginally unstable or unstable process characteristics. To see this, consider the case where both transfer functions are given by 3.11. If delta MV is zero and 3.11 is substituted into 3.12, the resulting transfer function is

$$G = g*DV/(20s+1-g) \ . \qquad [3.13]$$

The characteristic equation (CE) of 3.13 is

$$CE = 20s + 1-g \ . \qquad [3.14]$$

Setting the CE equal to zero and solving for s yields

$$s = (g-1)/20.0 \ . \qquad [3.15]$$

When the value of **g** is one, the pole or root of the CE (Equation 3.15) is located at the origin and a marginally unstable or integrating process characteristic should result. If the gain is greater than one, the pole is located in the right half plane and the process becomes exponentially unstable.

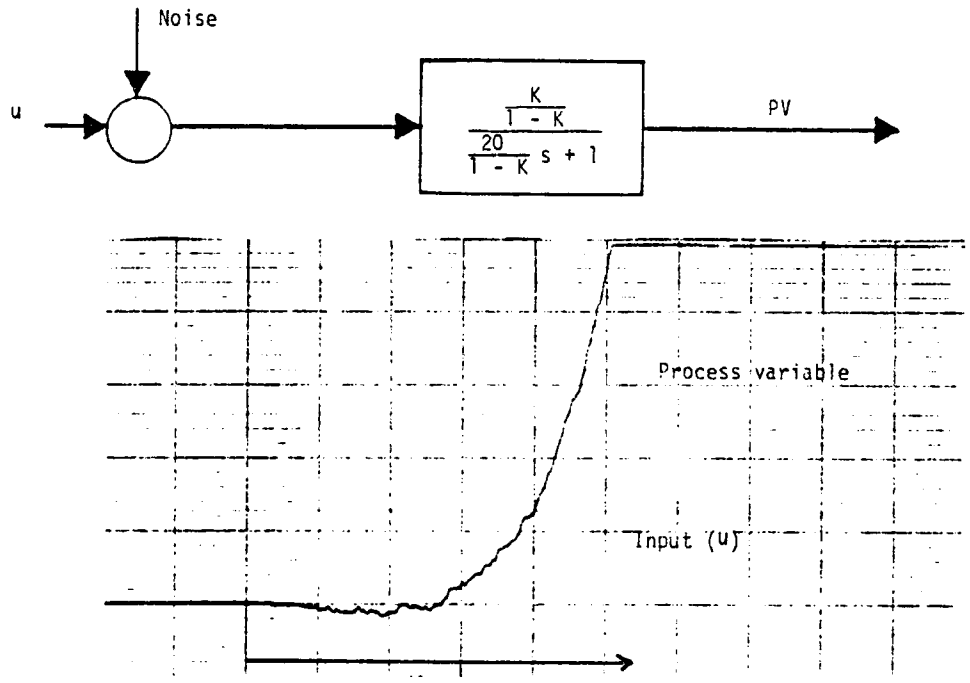
For gains less than one, stable process responses should have resulted. However, the integer arithmetic resulted in a nonlinearity in the low level gain which yielded unstable behaviors even for gains of less than one. The explanation lies in the round off operation (Equation 3.1). Suppose that the number of counts used to compute the gain is four (gain of 0.5). If U(k) (delta DV + delta PV) in Equation 3.8 is two counts, then the product of gain

times U(k) (in Equation 3.2) will be one; this corresponds to a gain of 0.5. However, if U(k) is one count, the round off equation also returns a count of one, so in this case the gain is one. Thus, the gain can oscillate between 0.5 and 1.0. This situation creates a small nonlinearity in the gain and results in unstable behaviors even when the gain is chosen to be 0.75 or 0.5.

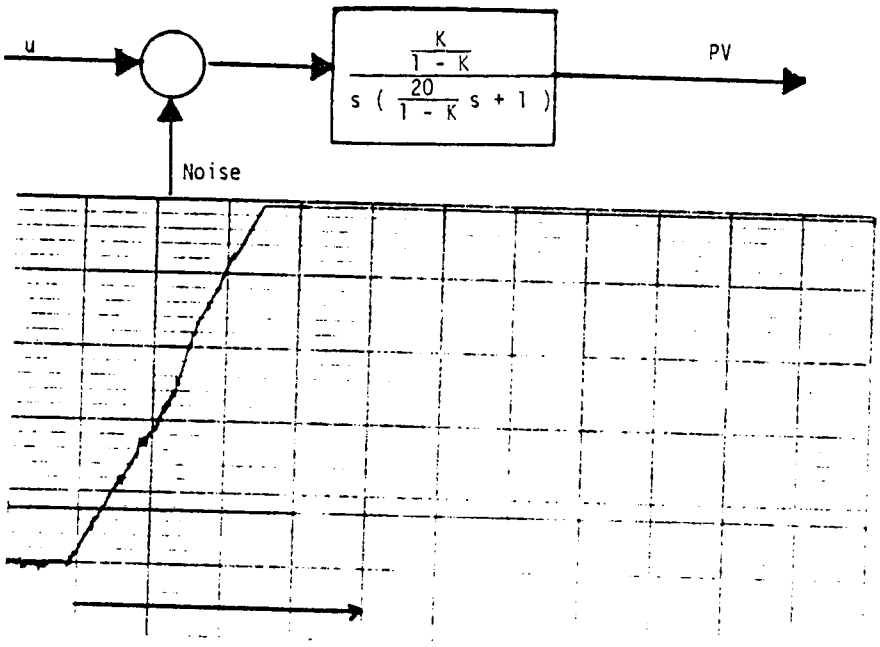
Changes in the PV were usually on the order of one to two bits and the effect of the nonlinearity was to make the plant respond as an integrating process driven by biased noise. The bias resulted from the fact that the current process output affected the next one and thus, the process drifted even when the process input was constant. Even a one bit change in the input was enough to start the process integrating (this is because the gain around the one bit level is one). Examples of the response of process one for gains of 0.5 and 1.5 are shown in Figure 3.22.

In Figure 3.22, both plant responses exhibited unstable behavior. However, if 0.25 had been chosen as the process gain value the plant would have exhibited stable behavior. This was because, for a gain of 0.25 the number of counts required was two and two times one plus four divided by eight ((2*1+4)/8) is zero in integer arithmetic. So underflow in the arithmetic resulted in the low level gain being below the value necessary for instablity.

The problem with the gain at the one bit level could have been corrected by (a) not updating PV(k-1) with PV(k) unless U(k) was nonzero (U(k) of at least two counts) and (b) by removing the round off operation from Equation 3.1. This would have built a dead



(A) Process Response when K > 1



(B) Process Response when K < 1

Figure 3.22. Process one's (positive feedback process) block diagrams and open loop responses.

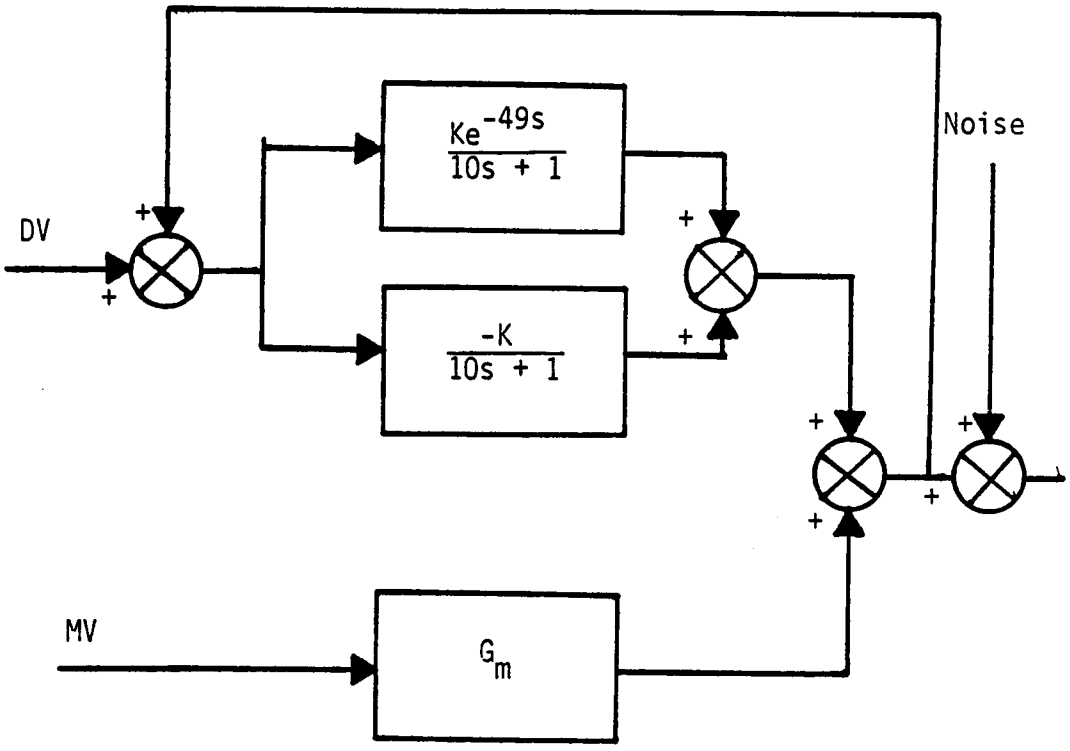
zone around the one bit level which would correct for the gain problem; however, this solution was not implemented because with process two it was beneficial to have the small nonlinearity in the gain.

Process two--an oscillating process. Process two was also designed to have a positive feedback loop. This process configuration differed from process one in that the disturbance process transfer function characteristic was fixed, and only the process parameters could be changed. The block diagram and open loop response for the submode of the plant are shown in Figure 3.23. The transfer function that was used to generate the step template for G_d had two first order lags in parallel with equal but opposite signs; one block had 49 sample times of delay while the other had no dead time. If feedback were not applied the process would show a dynamic response to changes in DV; however, the steady state gain would be zero.

The closed loop chacteristic equation relating the output to the input for the submode is given in Equation 3.16.

$$C.E. = 10s+(1-g) + g*e^{-49s} \qquad [3.16]$$

In the equation the process time constant is ten sample times and the process dead time is 49 sample times. The Laplace variable (s) is a complex variable and can be rewritten as $s = a+j*b$, where j is defined as the square root of negative one. Substituting the complex variable notation into equation 3.16 and using Euler's identity



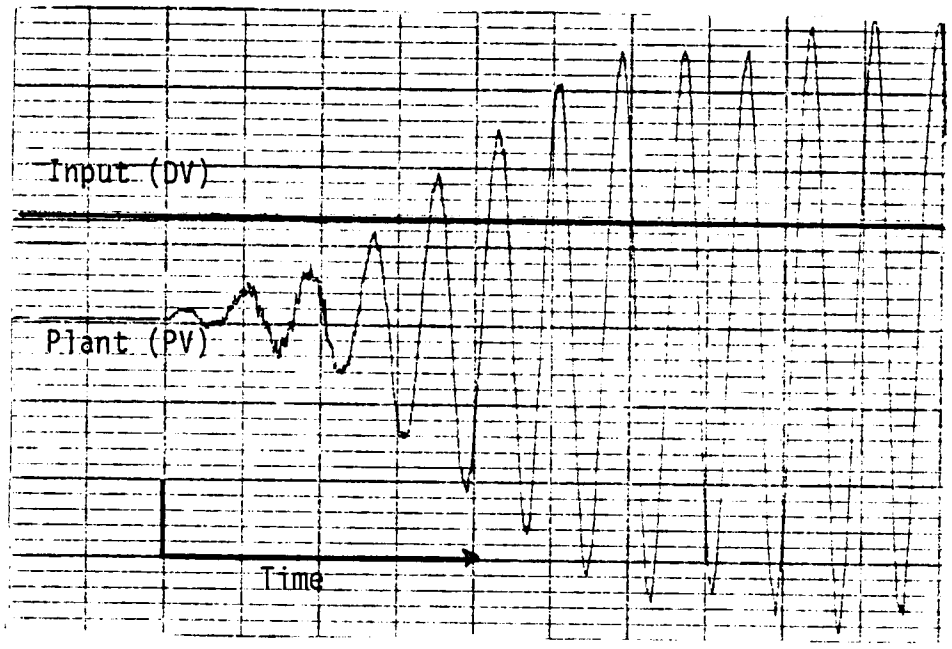


Figure 3.23. Process two's (oscillating plant) block diagram and an example of the plant behavior when the value of gain is 0.5.

($e^{-jx} = \cos(x)-j\ \sin(x)$) allows the equation to be written in terms of sines and cosines with a real and an imaginary component.

$$RE = 10a + (1-g) + g*e^{-49a}(\cos(49b) \qquad [3.17]$$

$$IM = 10b-g*e^{-49a}(\sin(49b) \qquad [3.18]$$

The determination of whether a particular value of gain makes the open loop response stable, marginally unstable or unstable can be ascertained by selecting a gain and solving for the values of a and b that satisfy both equations.

The solutions of the equations for selected values of gain are included in Table 3.3. From the table it is readily seen that as the gain is moved from -2 to +2 the roots of the characteristic equation move from the left half to the right half of the s-plane. The transition point (sustained oscillations) occurs at the cross over point (around a gain of 0.6); it was observed that when the simulator was operated with a gain of 0.5, the open loop plant response exhibited oscillatory behavior.

The reason for the oscillation occuring below the expected value of gain can again be traced to the nonlinear nature of the gain around the one bit level and the fact that the system uses sampled data. The act of sampling adds additional dead time to the system which would lower the cross over gain somewhat. The nonlinear process gain also would cause the loop gain to oscillate, thus causing the location of the roots of Equation 3.16 to oscillate

Table 3.3

Roots of Equations 3.17 and 3.18 as the Gain Varies From -2 to 2

Gain	Real Root	Imaginary Root (±)
-2.00	-0.0092	0.1202
-1.75	-0.1000	0.1202
-1.50	-0.0116	0.1187
-1.25	-0.0134	0.1178
-1.00	-0.0159	0.1166
-0.25	-0.0354	0.1101
0.25	-0.0216	0.0489
0.50	-0.0005	0.0474
0.75	0.0006	0.0446
1.00	0.0169	0.0419
1.25	0.0269	0.0332
1.50	0.0370	0.0266

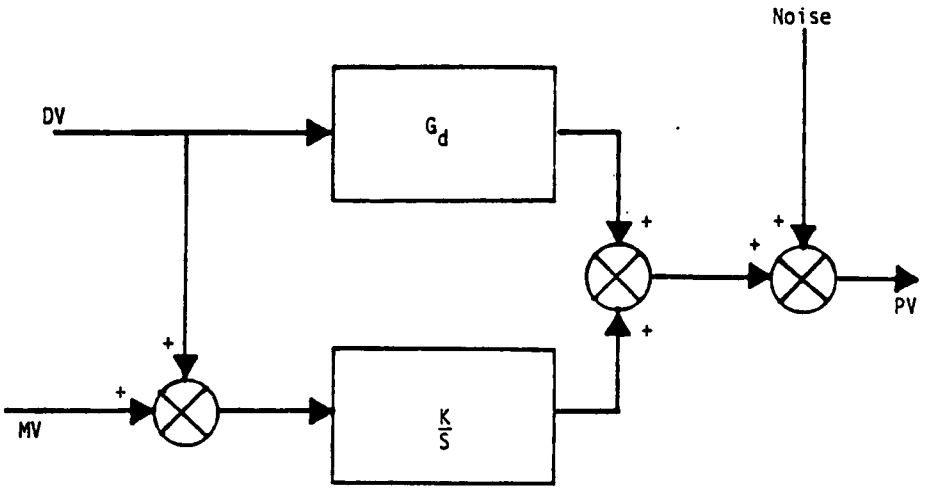
and change the crossover point. Both of these factors combine to yield an oscillation at a gain below that predicted by the equation.

The default configuration of process two is most readily selected using the reset switch. When the processor goes through the reset routine, if the P thumbwheel switch is set to two, then the disturbance transfer function's gain is set to 0.5 and G_m is set to that given in Equation 3.11.

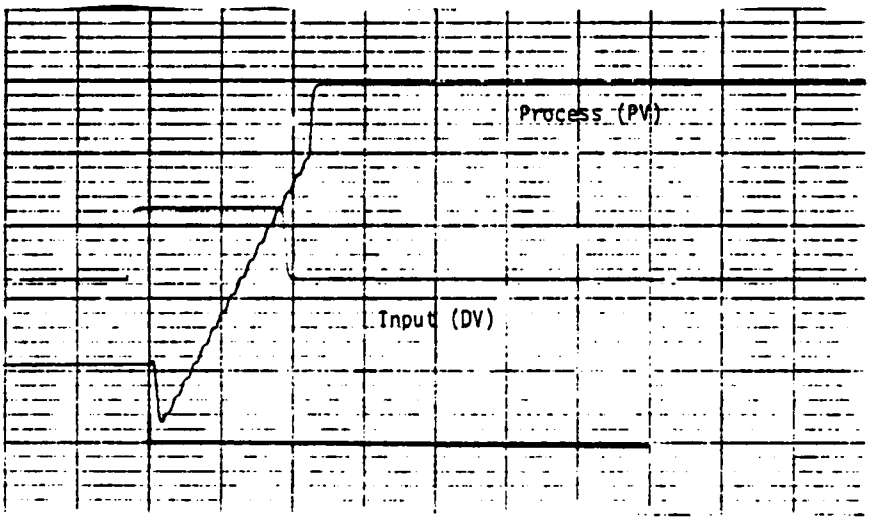
The D-IC command can be used to change the sample time and the other parameters of the DV process, but G_d itself is fixed. Compare this to process one, on the other hand, where G_d could be changed. (Note that process two is actually a subset of process one because it can be created using processes two and nine.) The reason for having this fixed default is to directly create an oscillating process as one of the menu process characteristics; this eleviates the requirement of a series of programming steps to generate the oscillating process characteristic.

The dynamic response of this process is shown in Figure 3.23. Note that while the sample time or speed thumbwheel (S) switch actually controls the period of oscillation (by changing the rate at which the state vector is shifted), it does not change the relative position of the roots of the characteristic equation because the template is fixed in character.

Process three--integrating process. The block diagram of process three is given in Figure 3.24. This process configuration is similar to the parallel mode of operation. The difference lies



(A) Block diagram



(B) Pulse Response when G_d = -1.0, G_m = 0.25/S

Figure 3.24. Process three's block diagram and pulse response.

in the way the parallel data is supplied to the processor. In this submode the disturbance input is also fed into both transfer function blocks and the default plant chacteristic for G_m is the same as that for process zero (integrating plant).

For this process to perform properly, the M-IC command has to be used for selecting it because the software that performs the summation of MV and DV is controlled by MSIM. If three is selected using D-IC, then the parallel software will not recognize that the submode has been selected. A typical response of this plant to a disturbance pulse is shown in Figure 3.24. In this example response the process characteristic chosen for G_d was that of a pure gain (plant 11, which will be discussed in the appropriate section).

Process four--parallel input submode. Process four is another PISO submode configuration. This configuration mimics the parallel mode of operation shown in Figure 3.18B, page 79 with one exception: selecting process four limits G_m to the first order transfer function of Equation 3.11. G_d is still free to be any of the sixteen processes. As with process three, this submode will only function properly when process four is selected using M-IC. If the mode is selected using D-IC, then the parallel input will work, but the software will not recognize the need to sum MV and DV.

Process five--fourth order process. Process five is the first of the eleven asymptotically stable process characteristics. The process template was generated using a fourth order plant with the transfer function given in Equation 3.19.

$$G = 1/(13s + 1)(5s + 1)^3 \qquad [3.19]$$

A typical step response of this plant is provided in Figure 3.25.

Process six--second order underdamped process. The template used in process six was generated using a second order, underdamped transfer function with a damping coefficient of 0.21 (51% overshoot), a period of oscillation (T_p) of 33 sample times and a natural, undamped frequency of $0.19/T_{st}$ radians per second. A sample step response is shown in Figure 3.26.

Process seven--second order underdamped process. Process seven's template was also generated using a second order, underdamped transfer function. The damping coefficient used was 0.40 (25% overshoot); T_p was 54 samples and the natural, undamped frequency was $0.12/T_{st}$ radians per second. The step response of process seven is given in Figure 3.27.

Process eight--second order overdamped process. The step response of process eight is shown in Figure 3.28. The step template used in process eight was generated using a second order, overdamped transfer function. The transfer function had two real poles and is given in Equation 3.20.

$$G = (5s+1)(10s+1) \qquad [3.20]$$

Process nine--zero gain process. Process nine was first discussed in conjunction with the oscillating process or process

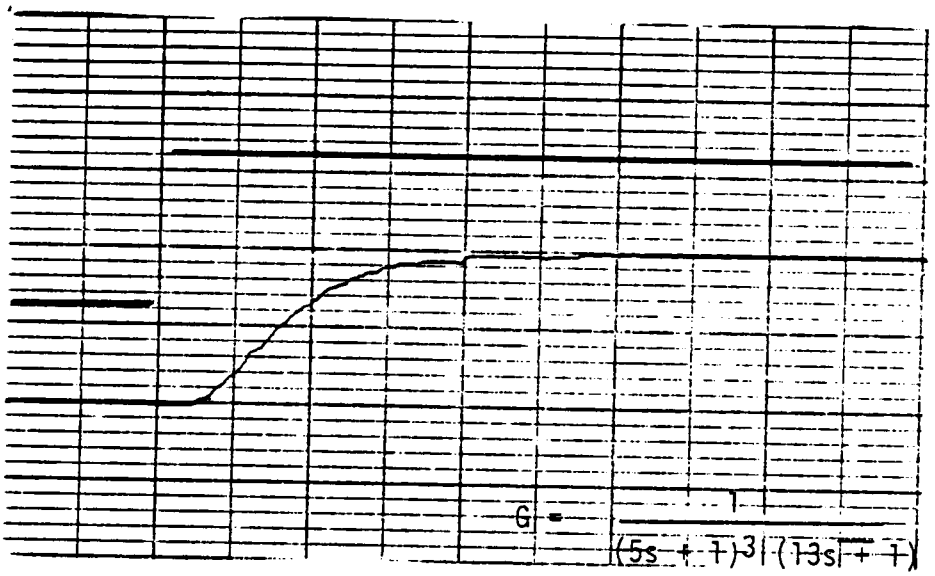


Figure 3.25. Step response of process five--a fourth order overdamped plant.

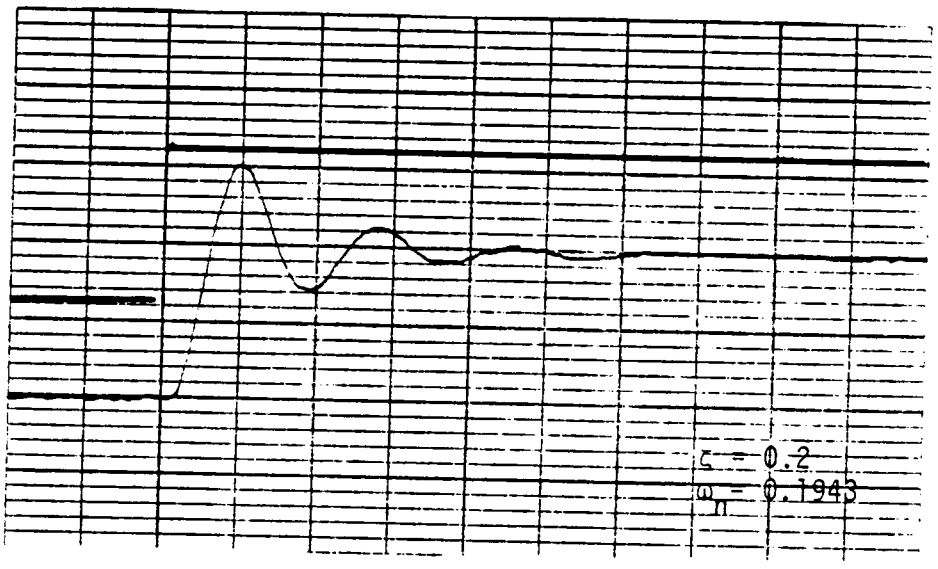


Figure 3.26. Reaction curve of process six--a second order underdamped plant.

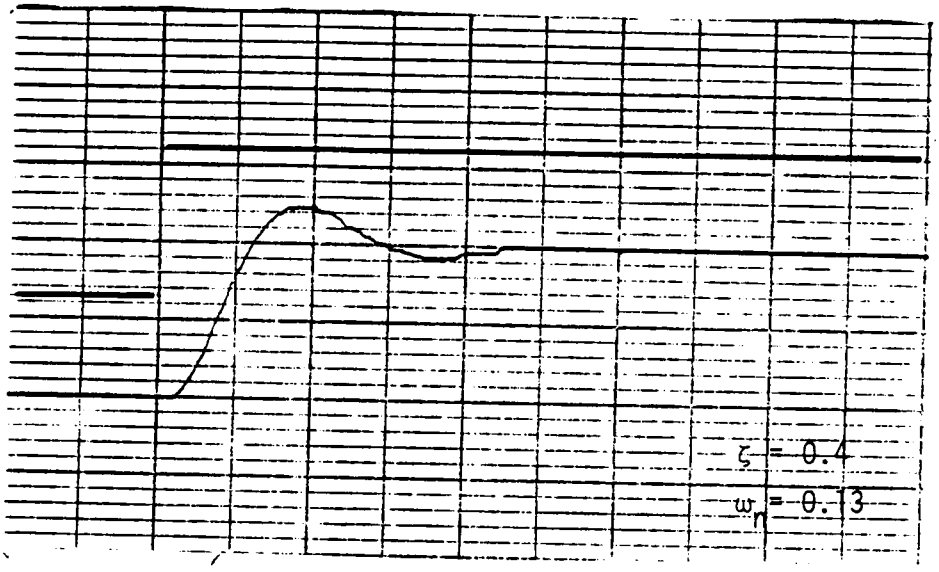


Figure 3.27. Process seven's step response (second order underdamped process).

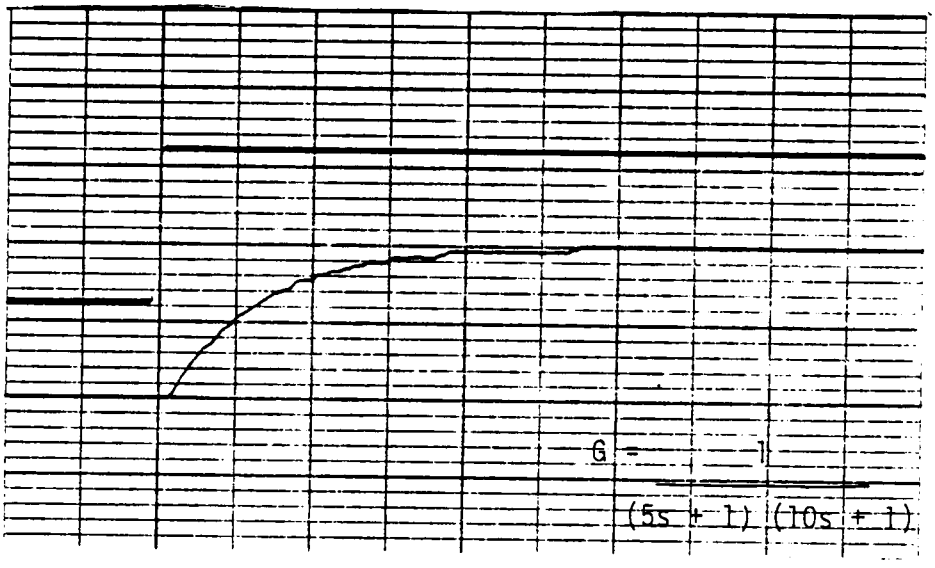


Figure 3.28. Process eight's step response (overdamped second order plant).

two. The dynamic response of this template was generated by a step input into the process whose block diagram is shown in Figure 3.23, page 89. The zero gain process's step response is provided in Figure 3.29.

Process ten--first order lag process. The template used in process ten was generated using a first order transfer function with a time constant of twenty sample times. The process transfer function was first given in equation 3.11. The step response is provided in Figure 3.30.

Process eleven--pure gain process. The step template used for process eleven was generated by loading the vector with all ones; this yielded a pure gain process. The process step reponse is shown in Figure 3.31.

Process twelve--lag/lead process. The process step response of process twelve is shown in Figure 3.32. The template used in process twelve was generated using a lead/lag transfer function. A lead/lag process with the lead time constant smaller than the lag time constant will respond to a step input with an initial jump in output, and then exponentially grow to the final steady state value. The transfer function used in generating the step template is given in Equation 3.21.

$$G = (10s+1)/(20s+1) \qquad [3.21]$$

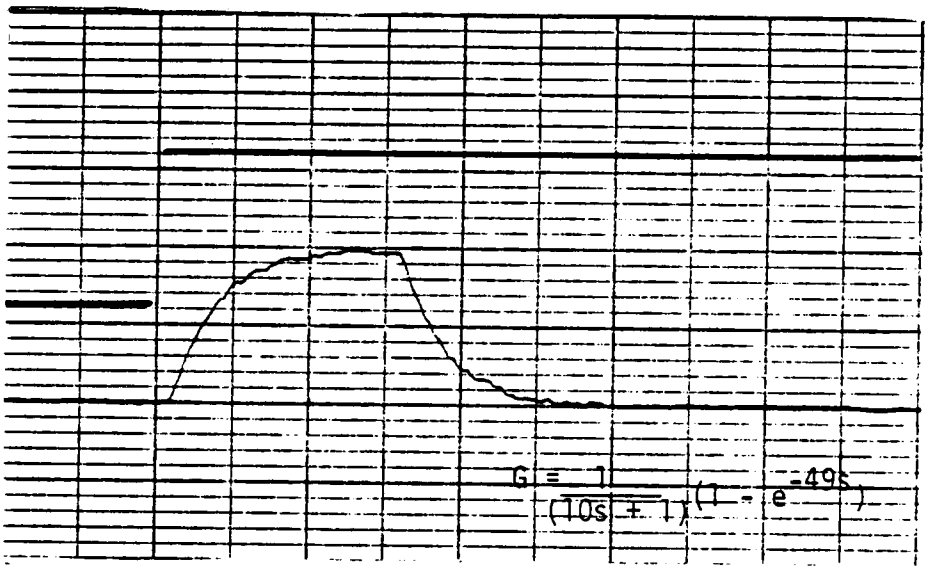


Figure 3.29. Process nine's step response (two conflicting first order lags in parallel).

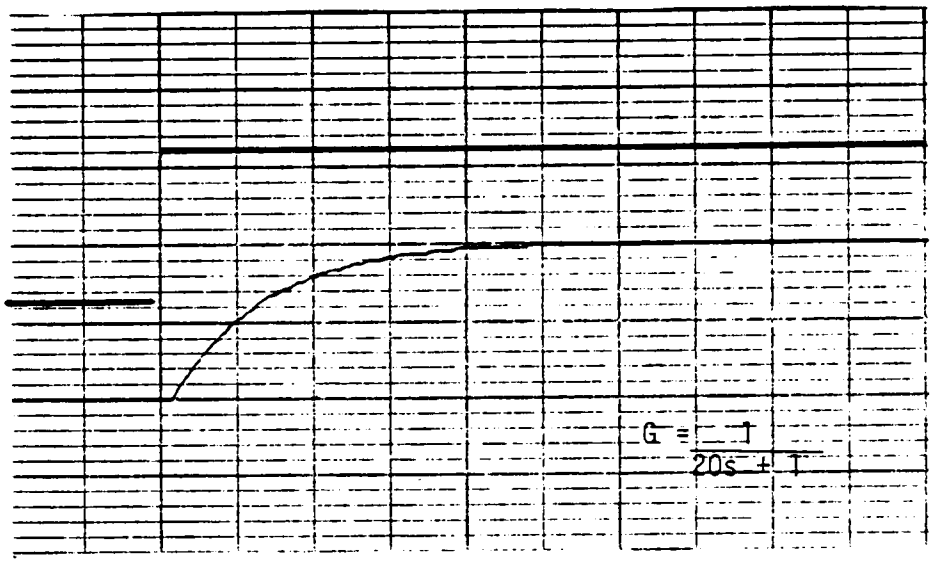


Figure 3.30. Step response of process ten--a first order lag.

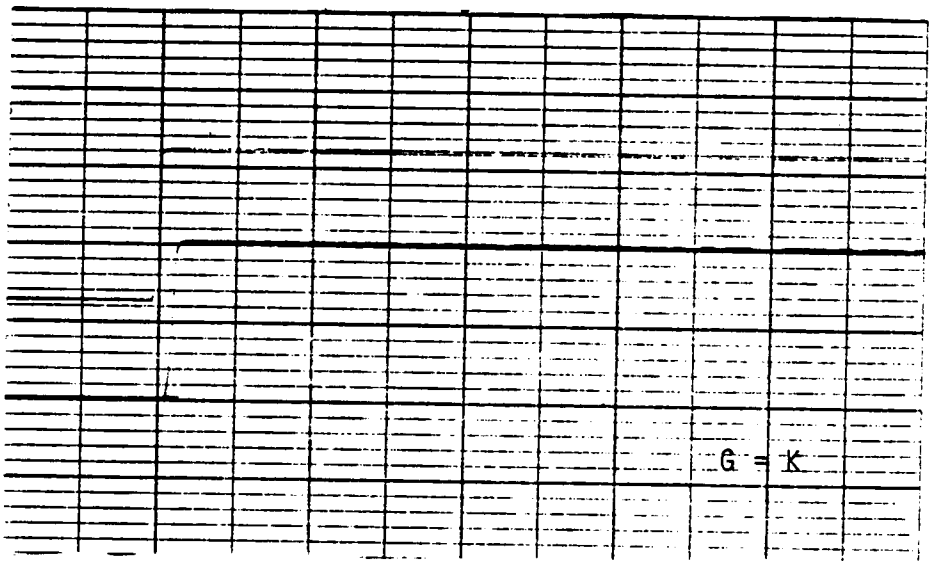


Figure 3.31. Process eleven: step response of a pure gain process.

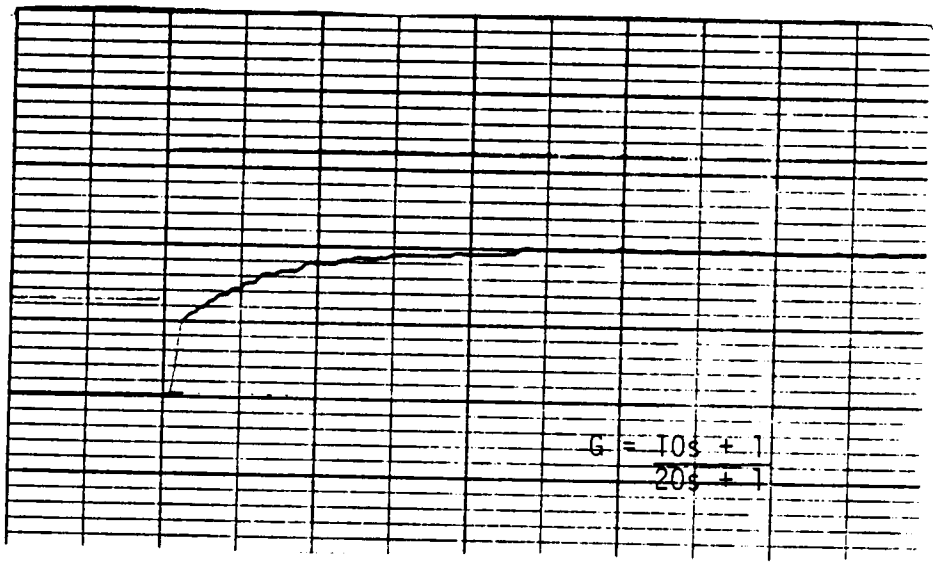


Figure 3.32. Process twelve: step response of a lag/lead process element.

Process thirteen--lead/lag process. The template used in process thirteen was also generated using a lead/lag transfer function. For a lead/lag process if the lead time is greater than the lag time in a lead/lag process, then the initial step response overshoots the final steady state and exponentially decays to the final steady state. The transfer function of this process and the step response are provided in Equation 3.22 and Figure 3.33 respectively.

$$G = (30s+1)/(20s+1) \qquad [3.22]$$

Process fourteen--second order underdamped process. The step response of this plant is shown in Figure 3.34. Process fourteen's step template was generated using a second order, underdamped transfer function with a damping coefficient of 0.717, which corresponds to an overshoot of 4% and the natural, undamped frequency is $0.0704/T_{st}$ radians per second. A complete summary of all of the second order processes and the corresponding dynamic parameters is provided in Table 3.4.

Process fifteen--inverse acting or sagging process. The final process characteristic that was provided in the step template array was generated using the "inverse acting", or sagging, plant. The plant is called a sagging process because the initial response to a step input is in the opposite direction from the final steady state. An example of the step response in provided in Figure 3.35.

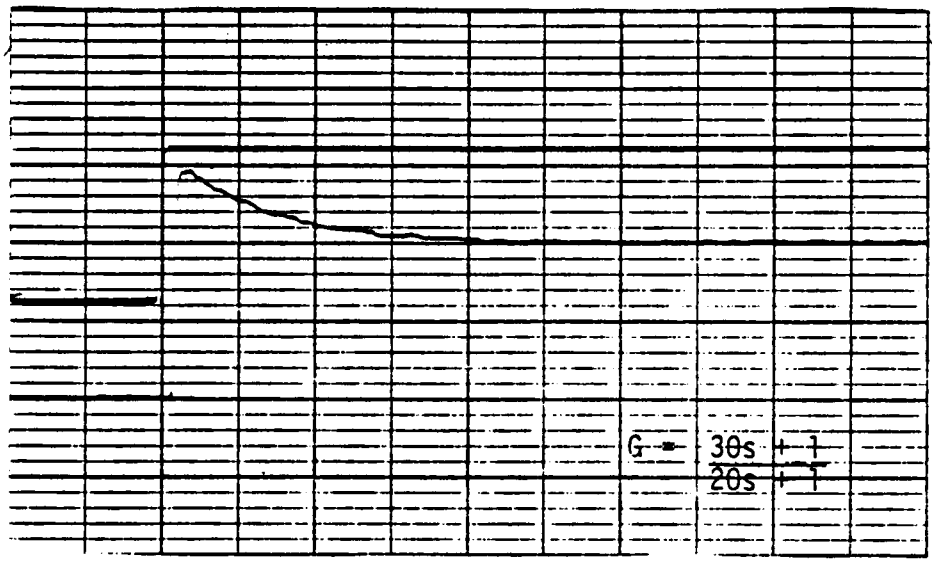


Figure 3.33. Process thirteen: step response of a lead/lag process.

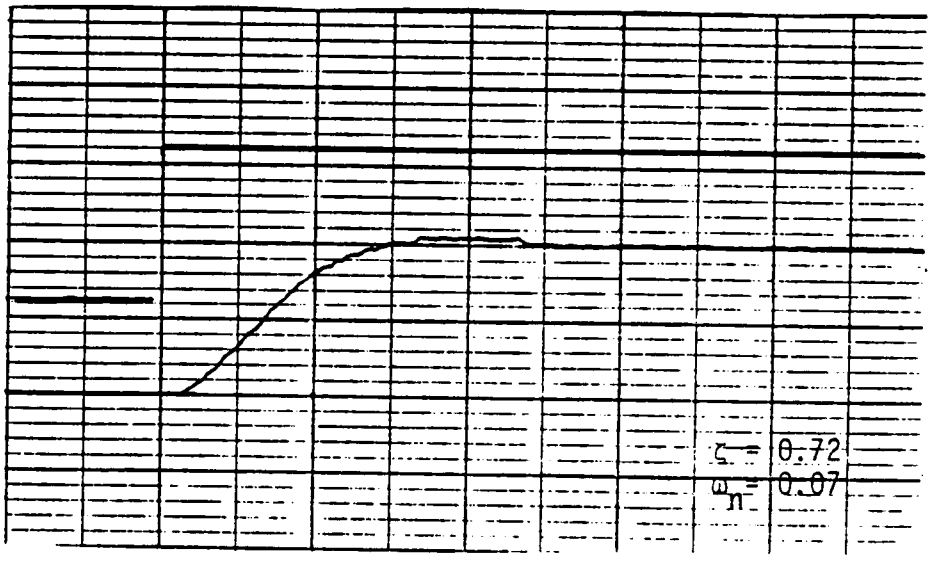


Figure 3.34. Process fourteen: step response of a second order underdamped process.

Table 3.4

Second and Higher Order Process Characteristics

Process #	Order	Classification	τ_1	τ_2	Damping Coefficient (ζ)	Normalized Natural, Undamped Frequency ($\omega_n * T_{st}$)
5	4	overdamped	5	13	-	-
6	2	underdamped	-	-	0.20	0.194
7	2	underdamped	-	-	0.40	0.127
8	2	overdamped	5	10	-	-
14	2	underdamped	-	-	0.717	0.070

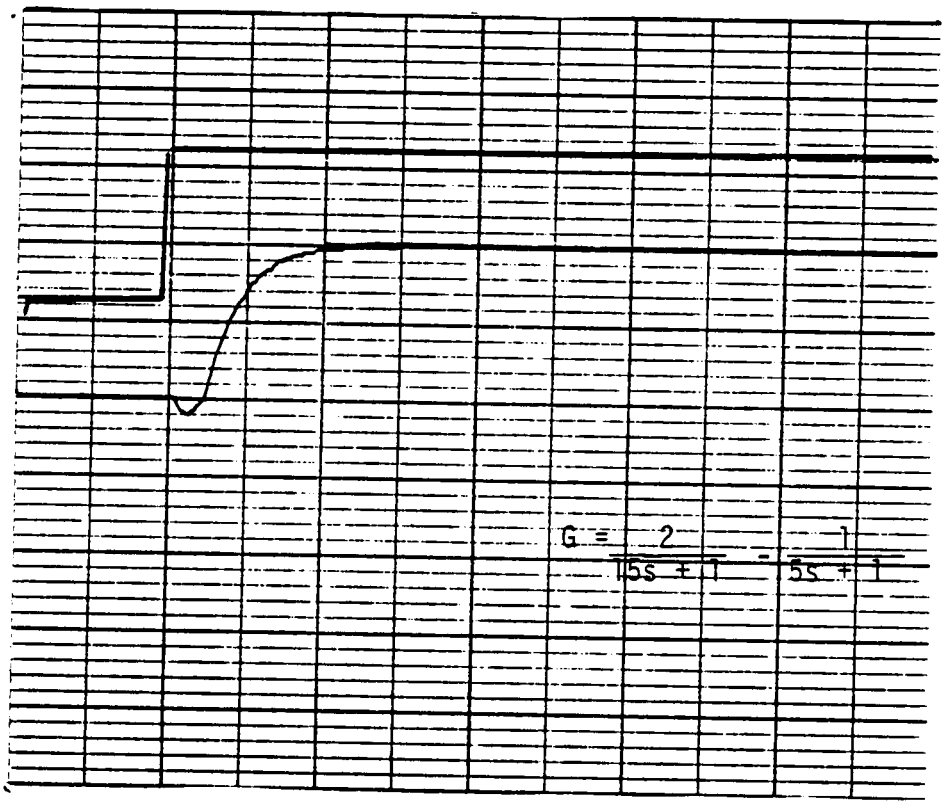


Figure 3.35. Process fifteen: step response of an inverse acting or sagging process.

Process Parameters

The three plant parameters are gain, speed and dead time. There are sixteen values of each that may be independently selected. This means that for each process characteristic (template) 16^3, or 4096, different combinations of process parameters may be dialed in through the thumbwheel switches. The gain and speed are independent of one another, but the dead time is related to the sample time; this is because of the way in which a dead time is simulated in the system. In Table 3.5 is presented a complete list of the process characteristics and parameters that are associated with each thumbwheel switch setting. In the following paragraphs each of the three parameters will be described in some detail.

Steady state gain. The steady state gain relates the change in output to the change in input. If the gain is chosen as one, then a one percent change in the input will result in an observed steady state change in the output of one percent. This is the value that the change in output will reach after 100 sample times (if no further changes in input are observed). The state vector has 100 elements and thus, it takes 100 samples for the last entry in the vector to move from the bottom to the top of the vector and be written to the DAC.

Speed. The S thumbwheel switch on the PPS is used to set the execution rate or sample time for each simulation block (MSIM or DSIM). As previously discussed the sample time controls the rate at which the state vector is shifted upward because of the

Table 3.5

Thumbwheel Switches and Associated Parameter Settings

Thumbwheel Switch #	Process Template P	Steady State Gain G	Sample Time (sec) S	Dead Time (samples) D
0	Integrator	0.25	0.18	00
1	Unstable	0.50	0.36	02
2	Oscillator	0.75	0.54	04
3	Integrator	1.00	0.72	06
4	Parallel	1.25	1.08	08
5	Fourth Order	1.50	1.26	10
6	Underdamped	1.75	1.44	12
7	Underdamped	2.00	2.16	20
8	Overdamped	0.00	2.88	20
9	Zero Gain	-1.75	3.60	30
10	First Order	-1.50	3.78	40
11	Pure Gain	-1.25	3.96	44
12	Lag/Lead	-1.00	4.32	50
13	Lead/Lag	-0.75	5.04	54
14	Underdamped	-0.50	5.76	70
15	Sagging	-0.25	6.12	85

shift-then-fill policy of the STM. Since the fastest sample rate on the PPS is 0.18 seconds the fastest transient time available is 18 seconds. The slowest sample time is 5.94 seconds, yielding a transient speed of 196 seconds; thus, the ratio of slowest to fastest response speed is 33.

Dead time. The dead time parameter specifies the number of sample times that an input is delayed before it is presented to the simulation algorithm. The sample time setting therefore, has a direct affect upon the magnitude of the dead time. The fact that the sample time setting has a direct affect upon the dead time does not change the ratio of dead time to transition time because sample time shows up in both the numerator and denominator of the ratio, and thus, cancels out. Table 3.6 presents the range of ratios of dead time to time constant available for the first order lag process (process ten).

3.6 CONCLUDING REMARKS

In this chapter a versatile simulation computer was presented--the Portable Process Simulator. The PPS was shown to provide many of the typical process characteristics that are commonly discussed in a process control course. The PPS was designed to essentially be a special purpose real time computer that had the more desireable characteristics of both the analog computer and digital computer; this goal was accomplished through a streamlined operating system and a powerful simulation technique.

Table 3.6

Possible Ratios of Dead Time to Time Constant for Process Ten

Thumbwheel Switch #	Ratio (θ/τ)
0	0.00
1	0.20
2	0.35
3	0.70
4	0.95
5	1.45
6	1.70
7	1.95
8	2.45
9	2.70
10	2.95
11	3.25
12	3.45
13	3.65
14	4.45
15	4.95

The programming was structured to allow a wide range of process characteristics to be combined to create the potential for an almost infinite range dynamic responses.

CHAPTER IV

APPLICATION OF THE PPS TO THE STUDY OF PROCESS CONTROL

The purpose of this chapter is to illustrate uses of the PPS as a tool for teaching, demonstrating and studying process control concepts. The chapter consists of a series of discussions on application examples that were developed in various undergraduate laboratories at the University of Tennessee in the Chemical Engineering Department. The underlying objective in this chapter is to provide guidance and suggestions that aid in the development of laboratory experiments utilizing the PPS.

The chapter begins with a discussion of a simple laboratory experiment designed (a) to introduce students to the operational characteristics of the simulator and (b) to provide an introduction to process control concepts. This is followed by a discussion of a two part experiment on process dynamics and identification. Part A of the experiment addresses the characteristics of first order plus dead time processes and part B focuses on second and higher order plant dynamics. In part B, three open loop methods for modelling higher order processes are discussed.

In the second section, the topic of feedback control is addressed. The objective of this section is to provide sufficient information on the operational characteristics of the PPS so as to allow meaningful controller tuning experiments to be developed. The section begins with a discussion on the classical continuous cycling tuning method. This discussion is followed by a look at closed loop

modelling. The section closes with a comparative study of empirical tuning techniques.

Discussions in the next section focus on applications of multi-simulator networks to the study of advanced process control. The discussions begin with a look at feedforward, ratio and cascade control. This is followed by a discussion on the application of two and three simulator networks to the study of multivariable systems.

The chapter concludes with a brief outline of Model Algorithmic Control (MAC) using three simulators. The focus of this section is primarily on dead time compensation using the Smith predictor type of algorithm, but the application of MAC to inverse acting or sagging processes is also included.

4.1 LABORATORY APPLICATIONS OF THE PPS TO THE STUDY OF PROCESS DYNAMICS AND MODELLING

Introductory Experiment

One of the primary reasons for developing the PPS was to provide a tool for supplementing the material being taught in introductory process dynamics and control courses. The simulator was designed to appear as a miniature control room with meters to provide visual feedback and with adjustable knobs for setting the valve and disturbance levels. The objective was to provide a black box which contained process characteristics typical of those found in industrial systems. Also, enough individual stations were provided so experiments could (a) be essentially self-study and (b) be

structured to provide a "hands-on" or intuitive understanding of the dynamical systems discussed in the classroom. The following paragraphs present an outline of a first level process control experiment designed to introduce fundamental concepts. The objectives were:

1. to introduce the operation of the PPS;
2. to develop the notion of manual control;
3. to illustrate start-up;
4. to define deviation variables in terms of the steady state;
5. to present step inputs and step responses;
6. to show how the steady state gain determines the valve position required to achieve a particular operating level for linear processes;
7. to indicate the importance of understanding the dynamical nature of systems in designing effective control strategies.

These objectives were achieved in a simple introductory laboratory experiment by having students simply manipulate the control valve and observe the response of the sensor. The experiment was introductory in nature thus did not need to resort to complex concepts, such as Laplace Transforms or computer computations. Nevertheless, the student gained significant insight into the nature of process control through direct observation and experience as well as an intuitional foundation on which to build a more formal set of mathematical techniques.

Simulator operation. To perform the laboratory experiment required an understanding of the basic operation of the PPS. Experience in teaching a first level course in process dynamics indicated that a single one hour lecture was sufficient to provide the necessary instruction. During the presentation, the PPS became a typical process system and, concomitant with a demonstration of its operating procedures, a set of definitions necessary for discussing process control was established. The instructor was able to develop the concepts of independent, manipulated, and dependent variables while demonstrating the corresponding handles available on the simulator (disturbance rotary switch, manual knob or valve, and sensor). Fundamental notions such as steady state, step input, and deviation variables were also defined in the context of the basic operation of the simulator.

The author used the PPS several times in teaching a course in process dynamics and found that the response to the device was generally very favorable. In fact, students were usually excited about the prospect of using the simulator and asked to begin the laboratories immediately. Bringing the simulator into the classroom significantly increased the level of student participation, as students who were generally not inclined to enter discussions often asked significant and useful questions regarding the simulator operation. The net effect was a less formal, higher quality classroom experience for all concerned.

At the University of Tennessee a video tape presentation was also provided which included instructions for operating as well as

instructions for interfacing the PPS to other devices (strip charts and controllers). By providing such a tape in the laboratory, self study experiments could be designed which required minimal supervision of the students.

Laboratory outline. The experiment developed was simple and effective in presenting the basics of process dynamics without having to resort to differential equations or Laplace transforms. The simplicity allowed concepts to be explored before and concurrent with the classroom presentation of the underlying mathematics.

The only piece of equipment required to perform the experiment was the PPS. Each student was given several sets of thumbwheel switch settings and was given the following set of instructions:

1. adjust the manual knob until the valve is closed;
2. set the thumbwheel switches to one of the assigned sets;
3. press the reset pushbutton, and then the OP pushbutton;
4. bring the sensor reading (right hand meter) to 50% by adjusting the valve.

This experiment provided an initial introduction into the basics of feedback control (at this stage in the course little had been discussed about control). The student, through trial and error (feedback control), determined the valve position needed to meet the operating specification (set point of 50%). Accomplishing this goal ranged from easy (process 10 without dead time) to extremely difficult (process six without dead time and with a high gain) depending

upon the particular plant characteristic and parameters selected. The students were assured that all of the assigned plants were asymptotically stable and controllable (though not in these terms) and told that in spite of any difficulties, they were to continue adjusting the valve until the desired steady state was reached. The goal was to illustrate to students that some (stable) plants are inherently easy and others extremely difficult to control.

After finding the valve position necessary for a sensor reading of 50%, the next task was to compute the steady state gain. Since both the plant and valve started at zero (the bias thumbwheel switch was zero in all plant assignments), process gain was readily computed by dividing the set point by the valve position (it was duly noted that with real processes this generally would not be true). The students were then requested (a) to determine how much the valve needed to change if the set point was increased to 70%, (b) to implement the change and (c) to verify the result. The specific instructions were to reposition the valve quickly (an approximate step change) and to note the general character of the sensor's dynamic response to the change. From this observation, the student was asked to make a qualitative judgement as to why a particular characteristic might be easy or difficult to control.

The students were then asked to repeat the start-up experiment with a new set point. Students observed (or should have) that the knowledge gained from the first experiment improved their ability to control the unit considerably. From this, the axiom that

"the more that is known about the dynamic character of the process, the easier it is to control" was directly demonstrated.

In the next lecture session, the following explanations regarding the observations made in the laboratory were provided. First, depending upon the gain selected, a certain amount of bias was observed between the final valve position and sensor reading. This was pointed out as a typical result for real processes because of factors such as sensor spans, changes in process parameters, and valve sizing. Second, an attempt was made to relate the degree of difficulty encountered in start up directly to the dynamic character of the plant. Was a highly overdamped plant easier or more difficult to bring to steady state than was a highly underdamped one; or, did a significant lag or dead time increase or decrease the degree of difficulty? Third, deviation variables were presented in the context of the steady state and the gain. How much did the process deviate from the original steady state when the valve was changed by a certain amount? Fourth, the notion of a step change in input and the corresponding response of the plant was defined in terms of the experiment where the valve position was changed to a new level and the process response was observed. Finally, the point was emphasized that the second start up of the plant should have been much easier, due to the increased process knowledge gained through the observations of the plant's behavior.

The experiment outlined above illustrates how fundamental process control concepts can be demonstrated through an organized exploration using the simulator. Most of the knowledge gained by

students was through qualitative observation and only a single set of simple computations was necessary. The experiment was simple to explain and easy to perform. Student reaction was found to be positive and overall comprehension level of the basic concepts was high.

In the next section an experiment on process identification using the PPS will be presented. This laboratory experiment focused on defining the basic parameters of first and second order processes. The purpose of this experiment was to begin providing a more concrete and mathematical methodology for explaining the observations made in the first experiment.

Process Identification Experiment--Part I

The first section of the second laboratory experiment dealt with the characteristics of first order lag plus dead time processes. This process characteristic has been prominent in the textbooks (5,6,28,29) and literature (30,31) of process control primarily because of the following factors:

1. only three parameters are required to completely discribe the behavior of such processes;
2. most dynamically complex processes may be faithfully approximated by this combination of characteristics (29);
3. most controller tuning techniques assume that the controlled process is approximated by this form of model (30,31).

Because of the importance of first order dynamics in the literature,

it was deemed appropriate to begin process identification experiments with a study of the behavior of this process. The primary objectives of this experiment were to

1. teach students to recognize the pattern associated with the step response of the process;
2. define the three parameters (gain, time constant, and dead time) necessary to characterize the steady state and transient behavior;
3. show how to identify and calculate the process parameters.

A secondary objective of the experiment was to show the importance of paying careful attention to details in performing any experiment. This was accomplished by requiring that a strip chart recorder, used to monitor the process, be calibrated before the experiment was performed. If the monitoring equipment was not properly calibrated, calculations using the experimental results were invalid and the experiment had to be repeated. Thus, poor experimental procedure resulted in extra labor for the student.

Strip chart calibration. The PPS was connected to a two pen recorder. The first requirement was to connect and calibrate the recorder. The strip chart was a flexible laboratory instrument and as such could monitor a wide range of voltage levels. The chart also had a zero and an attenuation adjustment for each pen. This flexibility and adjustability insured that students could never be certain from one session to the next whether the chart's adjustments

or calibration settings had been modified. In order to guarantee the validity of the experimental results, the device had to be calibrated during each session.

Calibration consisted of determining the span or total travel of each pen as the process variable being monitored travelled from the minimum to the maximum level. The procedure for performing this task was (a) to close the valve, (b) to press the reset pushbutton, (c) to open the valve to 100%, (d) to press reset again, and (e) to repeat the procedure several times. The repetitive nature of the procedure resulted in both the valve and the sensor traversing between limits (provided the value of the bias thumbwheel switch was zero) and two distinct lines were drawn on the chart paper. These lines allowed spans and the pen offset to be calculated.

The pen offset resulted from the fact that the pens could not physically lie on top of one another and so one pen had to lead the other. Students were instructed to note the value of the pen offset because it would play a role in the dynamic computations that followed; if offset were ignored, the results were either that the dead time was too large or that the process appeared to respond before the valve moved (an impossibility for causal (physical) systems). In the former case, the numerical results were incorrect and in the latter, there was confusion. Students learned from this, that in order to perform a valid experiment, the process and sensing equipment had to be understood adequately.

The purpose of the procedure described above was to instill an understanding of the total system. This point was further

emphasized by requiring that results be correct before being accepted by the instructor. This allowed any conceptual problems to be cleared up quickly, and insured that set-up would be part of the standard operating prcedure in later sessions.

First order plus dead time identification experiment. The actual experiment consisted of identifying whether a process was a pure dead time or a first order lag plus dead time. Each student was given a set of switch settings and asked to

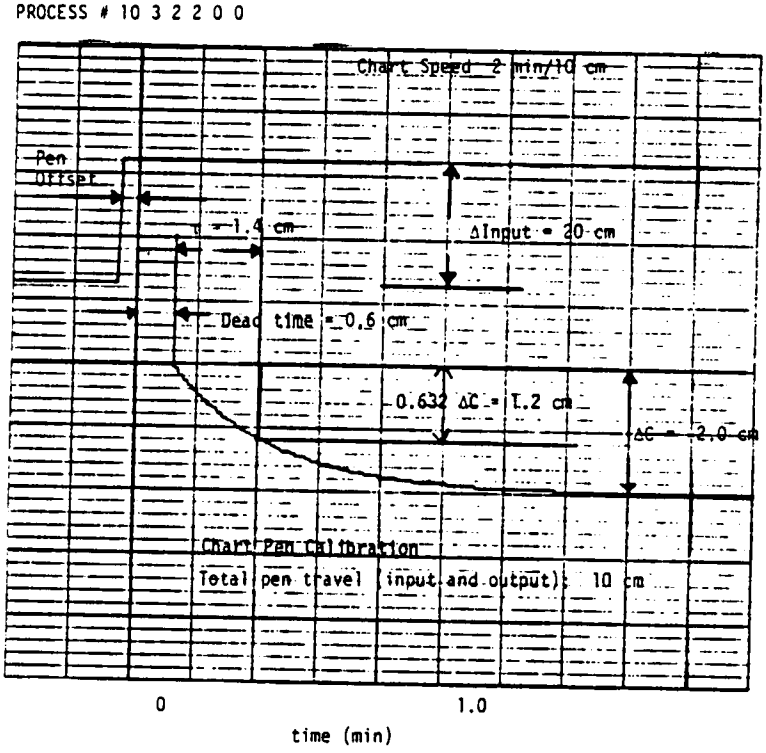
1. initialize the simulator;
2. instigate a step change in the valve position;
3. record the valve and process response on the strip chart;
4. attach the strip chart record to a single sheet of paper;
5. identify the process characteristic and compute the appropriate parameters;
6. compile a table summarizing the results.

A typical example of a summarizing table is shown in Table 4.1. Each student had to include a complete set of strip chart recordings and a set of computations for verification of results. Examples were provided to the class; two such examples, generated using the simulator, are shown in Figures 4.1 (first order plant) and 4.2 (pure dead time process). Several sets of switches were assigned so that enough repetition of computations were performed to allow

Table 4.1

Experimentally Determined Parameters for
First Order Plus Dead Time Processes

Plant #	Change in Input U (%)	Steady State Change in Output C (%)	Gain C/ U	Time Constant τ (sec)	Dead Time θ	Ratio θ / τ
1030000	20	20	1.0	3.6	0.2	0.05
1032000	20	20	1.0	16.8	0.3	0.0179
1032200	20	20	1.0	16.8	7.2	0.4286
10120000	20	-20	-1.0	3.6	0.2	0.05
10120400	20	-20	-1.0	3.6	3.5	0.97
10122200	20	-20	-1.0	16.8	7.2	0.43
1080000	20	0	0	3.6	0.18	0.05
1032400	20	20	1.0	16.8	17.4	1.03



CALCULATIONS

Process characteristic: first order

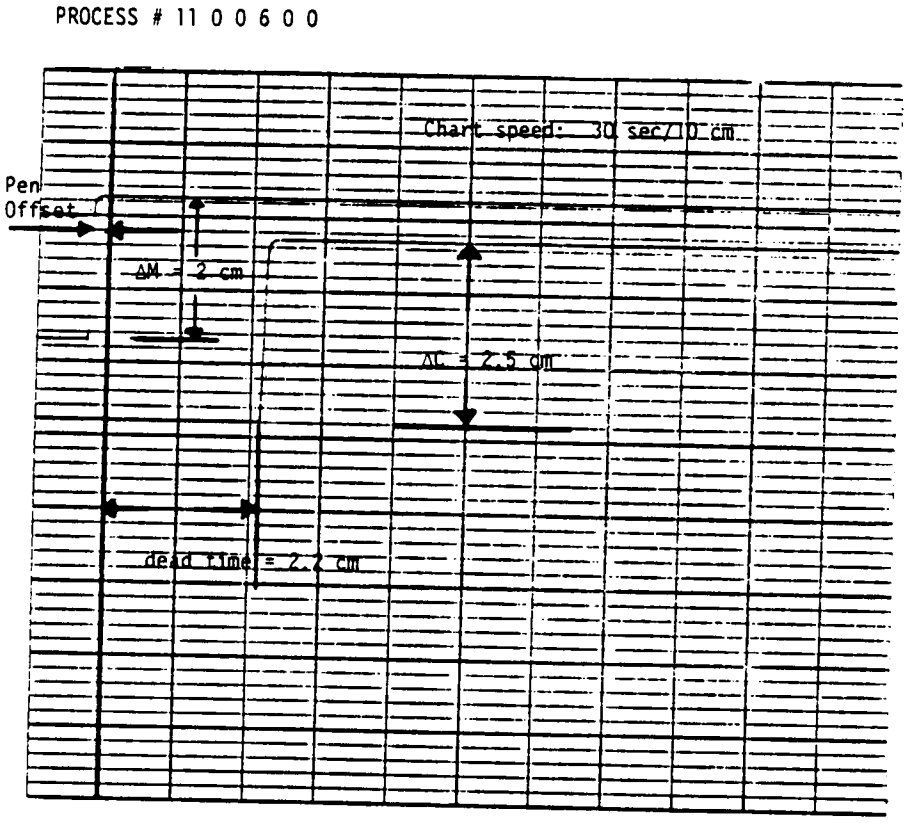
Process gain (ΔC/ΔInput): (-2/10)/(2/10) = -1.0 %/%

Process Time constant: (1.4 cm x 2 min)/10 cm = 16.8 sec

Process Dead time (0.6 cm x 2 min)/10 cm = 7.2 sec

Plant Model: $$G = \frac{-1.0\ e^{-7.2s}}{16.8s + 1}$$

Figure 4.1. Example strip chart of process response and sample calculations for a first order lag plus dead time identification experiment.



CALCULATIONS:

Process Characteristic: Pure dead time

Process Gain: 2.5/10 / 2.0/10 = 2.5/2.0 = 1.5 %/%

Process dead time 2.2 cm x 30 sec/10 cm = 6.6 sec

Process Model: $G(s) = 1.5\ e^{-6.6s}$

Figure 4.2. Sample calculations and strip chart recording for a pure dead time process.

students to become comfortable with the notions and definitions of process gain, time constant, and dead time.

After performing the experiment, students were able to identify the three parameters associated with a first order plus dead time process, and to discuss how such parameters could be identified experimentally. This prepared students for the next phase of the experiment which consisted of identifying models for other types of process characteristics. This experiment will be detailed in the next section.

Process Identification Experiment--Part II

One of the most popular methods used in process control for identification (ID) is the reaction curve technique developed by Cohen and Coon (30). Most textbooks present this method in conjunction with a discussion on controller tuning (28,29,32), but in this work tuning and process ID will be treated separately.

In this section a short discussion of three methods for generating process models using the reaction curve method will be presented. This is followed by an outline of a process identification experiment which deals with the characteristics of second and higher order plants.

The reaction curve method is primarily a graphical technique used to generate a first order lag plus dead time approximation of systems with higher order dynamics. The classical technique requires that a tangent line be drawn through the inflection point of a sigmoidal shaped step response of a second (or higher) order

process, and that the line be extended to intersect both the original and the new steady state of the plant. The dead time is approximated by measuring the distance, along the time line, from the point where the step change was initiated to the first point of intersection. The time constant is determined by measuring the distance from the dead time to the time where the tangent line intersects the new steady state.

The technique for computing the time constant was modified slightly by Miller (31). In this method the time constant is computed as the time required for the process to travel from the intersection of the tangent with the original steady state to the time where the plant has reached 63.2% of its final value.

Smith (32) noted that drawing the tangent line is not always simple in practice and suggested the following modification which removes the need for drawing the tangent line. In this approach the analytical solution is used to derive the following simultaneous equations.

$$t_{28.4} = \text{theta} + \text{tau}/3 \qquad [4.1]$$

$$t_{63.2} = \text{theta} + \text{tau} \qquad [4.2]$$

From the step response of the plant, the times required to reach 28.4% and 63.2% of the total change are recorded and used in Equations 4.1 and 4.2 to solve for the dead time and time constant. This is basically the reaction curve method, but it removes the requirement for finding the inflection point.

A comparision of the three methods by Smith shows that Cohen and Coon's approach yields the smallest dead time and largest time constant while the simultaneous equations method has the opposite result. Since the original intent of all three methods was to develop useful approximations for determining tuning constants, some question arises as to which approach yields the best result. Smith does not address this question in his textbook though he points out that Miller's technique appears to give the best approximation. The three approaches will be compared in this context in the section on controller tuning.

The reason for using any of these simple techniques is to generate enough information about a process so that controller tuning constants can be estimated from empirically determined relationships (33). The constants are typically chosen to yield an oscillatory (underdamped) response to a change in set point or to a load disturbance. It has been generally accepted that a quarter amplitude decay ratio represents a desireable degree of oscillation (29,34).

In general, for process systems, closed loop responses are tuned to be underdamped and open loop process responses are overdamped. Overdamped are often approximated using first order models and underdamped are described via second order models (35). The open loop behavior makes the study of first order processes important (part A of this laboratory); the closed loop response makes the study of second order processes important (part B). Part B of the

identification experiment focused upon second order process characteristics--both overdamped and underdamped.

Modelling of higher order processes. Part B of the second experiment taught students how to recognize the difference between first and higher order plant dynamics. A set of response curves, generated using the PPS, were segregated into two classes--first and higher order. This helped illustrate that first order processes initially have a maximum slope and higher order plants initially have a minimum slope. The higher order plants were classified further into either overdamped or underdamped response types. If a process was overdamped, then the three reaction curve modelling methods were applied and tables similar to Table 4.2 were generated. A separate table was required for each method developed. The information in the tables was used in a later experiment on controller tuning.

Underdamped processes were characterized with respect to overshoot, damping coefficient, decay ratio, and natural frequency of oscillation. This information was compiled in a tabular format as shown in Table 4.3. A set of example calculations was also provided for the student; an example of a typical set of response curves and corresponding computations are shown in Figures 4.3, 4.4, and 4.5.

Table 4.4 was provided to the instructor in order to ease the burden of verifing the students' results. The table contains a set of equations that allows the dynamic parameters of six processes

Table 4.2

Experimentally Determined Parameters for
First Order Lag Plus Dead Time
Approximation of a Higher
Order Process

Plant #	Change in Input U (%)	Steady State Change in Output C (%)	Gain C/ U	Time Constant τ (sec)	Dead Time θ	Ratio θ/τ
830100*	20	20	1.0	6.0	0.90	.13
830100**	20	20	1.0	4.2	0.9	
830100***	20	20	1.0	3.15	1.95	

*Cohen and Coon's Reaction Curve Method

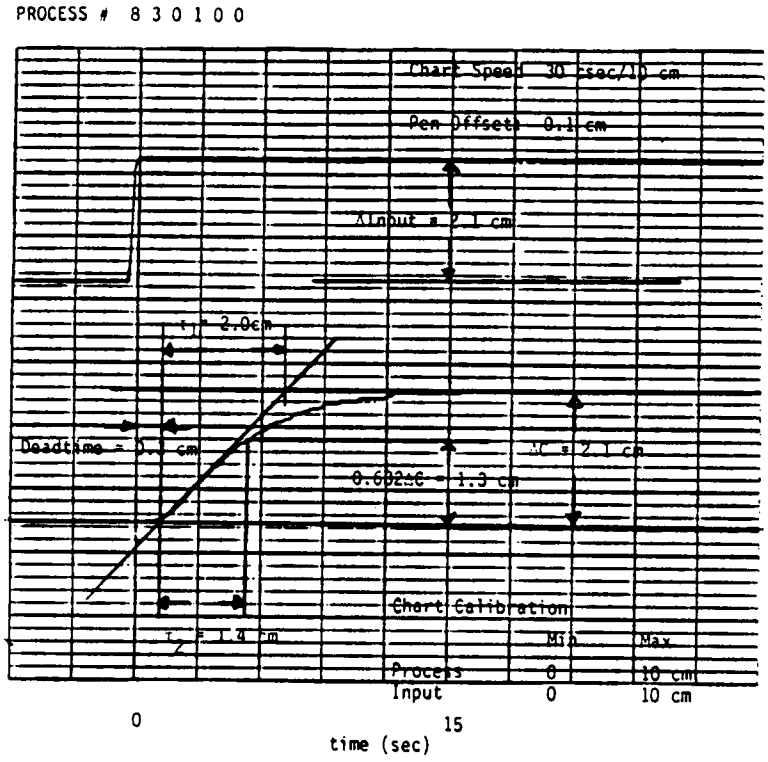
**Miller's Reaction Curve Method

***Smith's Simultaneous Equation Method

Table 4.3

Experimentally Determined Parameters for Several Second Order Underdamped Process Settings

Process #	Gain %/%	Dead Time (sec)	Overshoot	Damping Coefficient ζ	Decay Ratio	Period T (sec)	Time to first peak (sec)	Natural Undamped Frequency (ω_n) (radians/sec)
631000	1.0	0.8	0.52	.2	0.2704	17.31	8.65	0.3704
631200	1.0	2.16	0.52	.2	0.2204	17.31	8.65	0.3704
68100	0	-	-	-	-	-	-	-
7122200	-1.0	3.6	0.25	.4	0.0625	48.58	24.29	0.1411
733000	1.0	0.45	0.25	.4	0.0625	48.58	24.29	0.1008
14140000	-1/2	0.18	0.04	.717	0.0016	23.04	11.52	0.3911



CALCULATIONS

Process characteristic : overdamped

Gain ($\Delta C/\Delta Input$) : $(2.1/10)/(2.1/10) = 1.0$

Time constant (Cohen) : $(2.0 \times 30 \text{ sec})/10 \text{ cm} = 6.0 \text{ sec}$

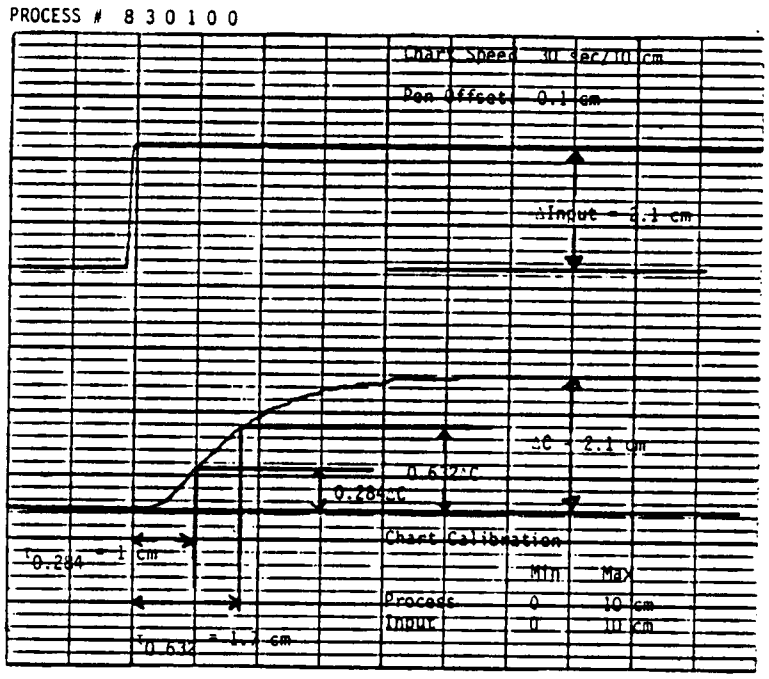
Time constant (Miller) : $(1.4 \times 30)/10 = 4.2 \text{ sec}$

Dead time : $(0.3 \times 30)/10 = 0.9 \text{ sec}$

Model (Cohen-Coon) $$G = \frac{1.0\, e^{-0.9s}}{6.0s + 1}$$

Model (Miller) $$G = \frac{1.0\, e^{-0.9s}}{4.2s + 1}$$

Figure 4.3. Illustration demonstrating two different (Cohen-Coon and Miller) methods for performing reaction curve calculations.



CALCULATIONS

Process characteristic : overdamped

Gain ($\Delta C/\Delta Input$) : $(2.1/10)/(2.1/10) = 1.0$

Time constant and dead time

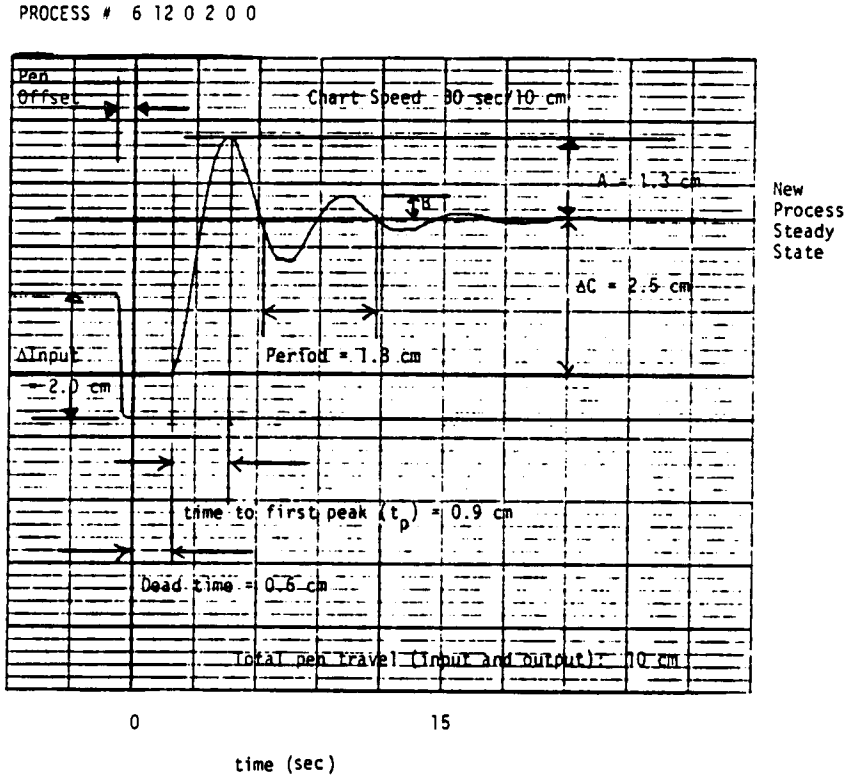
$t_{0.284} = (1 \text{ cm} \times 30 \text{ sec})/10 \text{ cm} = \theta + \tau/3$

$t_{0.632} = (1.7 \times 30)/10 = \theta + \tau$ solve

$\tau = 3.15$ sec $\theta = 1.95$ sec

Model $G = \dfrac{1.0\, e^{-1.95s}}{3.15s + 1}$

Figure 4.4. Computations illustrating Smith's process reaction curve technique for estimating time constant and dead time.



CALCULATIONS

Process characteristic: underdamped

Process gain ($\Delta C/\Delta Input$): $(2.5/10)/(-2.0/10) = -1.5$ %/%

Overshoot (o.s. = $A/\Delta C$): 1.3/2.5 - 0.52

Damping Coefficient (o.s. = $\exp(-\pi\xi/(1 - \xi^2)^{1/2})$: 0.20

Period (T): (1.8 cm x 30 sec)/10 cm = 5.4 sec

1. Natural frequency - ω_n: $T = (2\pi/(\omega_n(1 - \zeta^2)^{1/2})$

   1.18 radians/sec

2. Natural frequency - ω_n - second method:

   $\omega_n = (\pi/t_{peak}(1 - \zeta^2)^{1/2})$: 1.18 radians/sec

   Damping ratio (o.s.^{2} = B/A): 0.27

Figure 4.5. Sample calculations showing how to estimate the parameters of second order underdamped plants.

Table 4.4

Summary of Equations for Verification of Students' Results in Laboratory One

Process 10--First Order Plant

Dead Time = (D(TWS)+0.5)*S(TWS)		[1]
Time Constant = 18*S(TWS)		[2]

Process Five--Fourth Order Overdamped

	$(\frac{\theta}{\tau})$	$(\frac{\tau}{\theta})$	
Cohen-Coon	31*S(TWS)	(7+D(TWS)*S(TWS)	[3]
Miller	20*S(TWS)	(7+D(TWS)*S(TWS)	[4]
Smith	16.25*S(TWS)	(11+D(TWS)*S(TWS)	[5]

Process Eight--Second Order Overdamped

	$(\frac{\theta}{\tau})$	$(\frac{\tau}{\theta})$	
Cohen-Coon	33*S(TWS)	(3+D(TWS)*S(TWS)	[6]
Miller	23*S(TWS)	(3+D(TWS)*S(TWS)	[7]
Smith	18*S(TWS)	(7+D(TWS)*S(TWS)	[8]
Yuwana-Seborg	33*(S(TWS)	(10+D(TWS)*S(TWS)	[9]

Table 4.4 (continued)

Process Six--Second Order Underdamped	
Overshoot = 0.52	[10]
Damping coefficient = 0.20	[11]
Period = 33*S(TWS)	[12]
Natural, undamped frequency = 0.1943/S(TWS)	[13]
Process Seven--Second Order Underdamped	
Overshoot = 0.25	[14]
Damping coefficient = 0.40	[15]
Period = 54*S(TWS)	[16]
Natural, undamped frequency = 0.1270/S(TWS)	[17]
Process Fourteen--Second Order Underdamped	
Overshoot = 0.04	[18]
Damping coefficient = 0.717	[19]
Time to first peak = 64*S(TWS)	[20]
Natural undamped frequency = 0.0704/S(TWS)	[21]

*S(TWS) = T_{st}

**D(TWS) = Number of samples of dead time

[5, 6, 7, 8, 10, 14] to be evaluated as functions of the dead time and sample time thumbwheel switch settings (D(TWS) and S(TWS), repectively in Table 4.4). This enabled the wide range of dynamic responses available on the PPS to be used effectively in a laboratoy experiment, because results could be verified without the necessity of performing experiments. Only the numerical values for dead time and sample time were needed to evaluate the process parameters; this information is available in Table 3.5, page 106.

4.2 FEEDBACK CONTROL STUDIES

Controller Tuning Studies

Feedback is the dominant mode of control in process systems and the three mode PID (proportional-integral-derivative) controller is a standard. Therefore, considerable attention was directed to studies illustrating uses of the PPS in feedback configurations with conventional controllers.

Recent advances in microcomputer technology have had a pronounced effect on process control hardware (36) and adaptive self-adjusting controllers have become off-the-shelf items (37). This has led to an increased emphasis on overall system and strategy design (38), and a decreasing emphasis on controller tuning (29). Nevertheless, the need to study the role of the controller and its effect on system performance will continue. Tuning studies provide a vehicle for exploring this topic and should remain an integral part of control education.

Controller tuning studies are readily performed using the PPS because the simulator was designed to connect to standard control and computer hardware. The following discussions provide information and examples intended to aid in the design of studies, but will not outline specific laboratory experiments.

Continuous cycling tuning. The Ziegler-Nichols (39) continuous cycling technique was one of the first tuning methods developed. Tuning methods have traditionally been classified as either open or closed loop. Open loop techniques, such as the reaction curve method, attempt to develop approximate models in the absence of a controller. Closed loop methods do not always directly produce a model. The continuous cycling technique is one such method.

The continuous cycling method is a technique that identifies two characteristic parameters of a process: the resonant frequency (w_c) and the ultimate gain (K_u). The continuous cycling technique is a trial and error procedure for determining the ultimate gain (K_u) of a process. The ultimate gain is the proportional controller gain (K_c) at which a process develops a sustained oscillation. K_u represents a unique operating point for a process because increasing or decreasing K_c results in growing or diminishing oscillations. This uniqueness has been utilized by Ziegler and Nichols (39) to develop emperical relationships for setting PID tuning constants.

The continuous cycling procedure is quite time consuming because of the trial and error nature of the technique; this is one

of the disadvantages of this approach. One advantage is that the method accounts for the phase shift introduced by sampling in a digital control system (33). (This means that the Ziegler-Nicols relationships do not have to be modified to account for sample time.) Another advantage is that behavior in the neighborhood of the desired operating point determines the controller settings.

Another disadvantage is that the process must be taken to a potentially dangerous limit in order to identify process parameters; this is not always acceptable. In spite of these drawbacks, the continuous cycling method is a likely candidate for a process control experiment because of its historical significance and several advantages. The technique also serves to demonstrate stability versus instability in a control loop--an important concept that is not always intuitively obvious to students.

Tables 4.5 and 4.6 contain a collection of K_u values for several of the processes on the PPS. Table 4.5 contains the ultimate gain and period of oscillation (P_u) for processes five, eight, ten and fifteen. Prcesses five (fourth order), eight (second order) and ten (first order) have already been shown to be important industrial characteristics. Process fifteen, an inverse acting or sagging process, has been observed in distillation columns (5), boiler feed water drums (34) and has also been widely discussed in the process control literature (40,41). Table 4.6 contains similar data for three underdamped processes [6, 7, and 14]. This table shows the effect of damping on closed loop stability.

Table 4.5

Ultimate Gain and Normalized Period of Oscillation
for Several Overdamped Processes

Dead Time Switch	Process: 5		8		10		15	
	K_u	P_u/T_{st}	K_u	P_u/T_{st}	K_u	P_u/T_{st}	K_u	P_u/T_{st}
0	4.6	38.9	-	-	-	-	3.16	-
1	3.5	47.2	8.2	29.6	10.0	13.4	2.8	27.7
2	2.8	52.8	5.2	36.1	6.8	19.4	2.34	38.9
3	2.46	58.3	4.1	44.4	4.8	27.8	2.0	44.4
4	2.2	63.9	3.1	51.4	4.0	33.3	1.88	50.0
5	2.04	69.4	2.7	55.0	3.2	36.7	1.78	52.8
6	1.76	77.8	2.4	66.7	2.64	56.0	1.63	72.2
7	1.5	94.4	1.86	86.1	2.0	66.7	1.42	88.9
8	1.25	113.8	1.46	113.8	.	88.9	1.22	105.5
9	1.2	122.2	1.36	116.6	.	100.0	1.18	116.7
10	1.1	138.9	1.26	128.8	1.54	111.0	1.13	130.5

Table 4.6

Ultimate Gain and Normalized Period of Oscillation for Several Underdamped Process Characteristics

Dead Time Switch	Process 6 (ζ=0.2)		Process 7 (ζ=0.4)		Process 8 (ζ=.72)	
	K_u	P_u/T_{st}	K_u	P_u/T_{st}	K_u	P_u/T_{st}
0	0.86	19.4	6.0	19.4	-	-
1	0.60	25.0	2.24	33.3	5.0	41.7
2	0.44	27.7	1.34	38.8	3.8	44.4

The results in Tables 4.5 and 4.6 were generated experimentally but the transfer function models given in Chapter III could have been used directly. Using the nominal parameters values for process ten, given in Table 4.4, page 133 to calculate K_u and P_u, yielded results that were within 10% of the experimental data. This was considered acceptable. Comparision of the predicted and observed stability limits showed that the simulator did approximate a linear system, but a curious phenomenon was observed.

At gains lower than K_u, oscillations with "beats" were observed. This behavior made identification of the ultimate gain more difficult. In physical systems, "beats" are observed when a harmonic oscillator interacts with another which has a slightly different frequency (42). For the PPS, an explanation of this behavior lies in the sampling algorithm. As discussed in Chapter III, the simulation software potentially followed a different path on each iteration. Each path had a slightly different execution time which meant that the simulator had a time varying sample time and, hence, was a nonlinear process (in Chapter III, a nonlinearity in the gain was also discussed). In a closed loop configuration, the variation in sampling rate results in the observed "beating" behavior because the nonlinearity leads to time varying ultimate gains and resonant frequencies.

Closed loop modelling. The continuous cycling method identifies closed loop parameters, but does not produce a process model. In this section a closed loop modelling technique developed by

Yuwana and Seborg (43) that yields a first order plus dead time model will be discussed.

The performance of a closed loop control system has traditionally been characterized in terms of the transient response; the decay ratio is the standard performance index (35) and is defined as the ratio of successive peaks in the transient response. The method of Yuwana and Seborg uses the decay ratio and period of oscillation (time between peaks) to produce a model. The model is derived by using a pade approximation for the dead time in the closed loop characteristic equation of a first order plus dead time process. The result is a second order characteristic equation which is then used to derive expressions for time constant and dead time as functions of damping coefficient and loop gain. The method is limited to processes where the dead time is small relative to the dominant lag.

The process is assumed to be under proportional control and tuned to have an underdamped response. The magnitudes of the first peak (c_{p_1}), the first minimum (c_{m_1}), the second peak (c_{p_2}), and the time between peaks (T_p) are measured. The final steady state (c_{ss}) can be computed using this transient data.

$$c_{ss} = (c_{p_1} * c_{p_2} - c_{m_1}^2)/(c_{p_2} + c_{p_1} - 2 * c_{m_1}) \qquad [4.5]$$

The decay ratio (DR) may be calculated via equation 4.6.

$$DR = (c_{p_2} - c_{ss})/(c_{p_1} - c_{ss}) \qquad [4.6]$$

The experimental decay ratio is used to compute the damping coefficient (z).

$$z = -\ln(DR)/[(4*pi^2 + (\ln(DR))^2)]^{0.5} \qquad [4.7]$$

The proportional gain and magnitude of the set point (A) change allows an estimate of the process gain (g_p) to be determined.

$$g_p = c_{ss}/(K_c*(A - c_{ss}) \qquad [4.8]$$

The model gain or **g** may then be determined by dividing g_p by the proportional gain (K_c). The process characteristics calculated in Equations 4.7 and 4.8 are then applied in Equations 4.9a thru 4.10 to generate the first order time constant (tau_m) and the dead time ($theta_m$) of the model.

$$R_1 = (g+1)^{0.5} \qquad [4.9a]$$

$$R_2 = [z^2*R_1 + g]^{0.5} \qquad [4.9b]$$

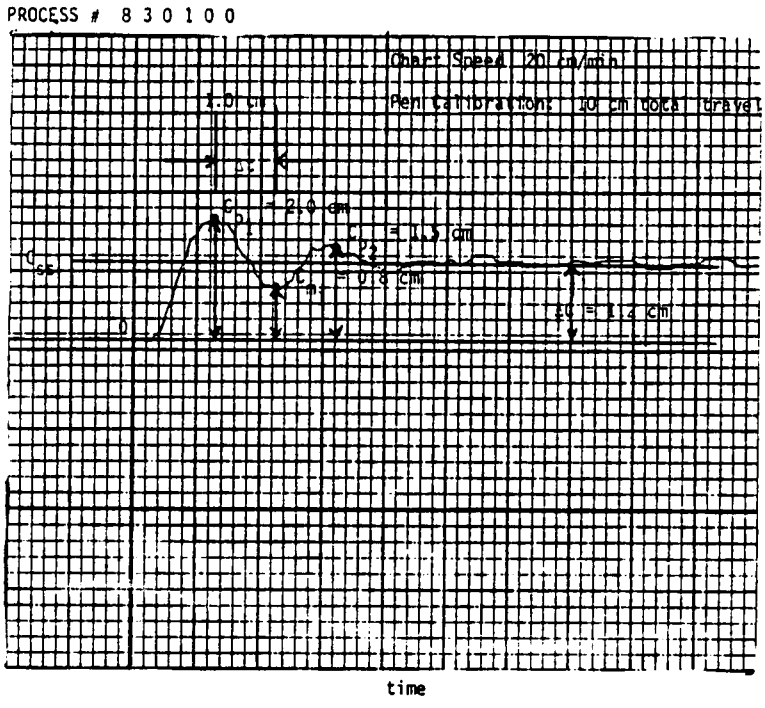
$$R_3 = [(1-z^2)*R_1^2)]^{0.5} \qquad [4.9c]$$

$$R_4 = T_p/pi \qquad [4.9d]$$

$$tau_m = (R_4/2)*[(z*R_1 + R_2)*R_3] \qquad [4.10a]$$

$$theta_m = (R_4*R_3)/(z*R_1 + R_2) \qquad [4.10b]$$

A sample problem which demonstrates the method is shown in Figure 4.6. The process used was the same one modelled in the reaction curve examples. Table 4.7 summarizes the results from this and the open loop experiments for process 8. Table 4.8 shows similar



Proportional Gain : 4.5

Set point change : 0.15

Example calculations using Yuwana's Method

$$C_{ss} = \frac{C_{p_2} \cdot C_{p_1} - C_{m_1}^2}{C_{p_2} + C_{p_1} - 2 \cdot C_{m_1}} = 1.24 \quad \frac{C_{ss}}{(\text{pen calibration})} = \frac{C_{ss}}{10} = 0.124$$

$$K_m = K_c \frac{C_{ss}}{(\Delta \text{Set point} - C_{ss})} = \frac{0.124}{(0.15 - 0.124)4.5} = 1.06$$

$$\text{Decay ratio (dR)} = \frac{C_{p_2} - C_{ss}}{C_{p_1} - C_{ss}} = \frac{(1.5 - 1.24)}{(2.0 - 1.24)} = 0.34$$

Figure 4.6. Example illustrating Yuwana and Seborg's closed loop identification technique.

Damping coefficient (ξ)

$$\xi = \frac{-\ln(dR)}{(4\pi^2 + (\ln(dR))^2)^{1/2}} = \frac{1.079}{6.375} = 0.169$$

Time constant (τ_m)

$$\tau_m = \left(\frac{\Delta t}{\pi}\right) \left(\xi\sqrt{K_cK_m + 1} + \sqrt{\xi^2(K_cK_m + 1) + K_cK_m}\right) * \left(\sqrt{(1 - \xi^2)(K_cK_m + 1)}\right)$$

$K_cK_m = 1.06 \cdot 4.5 = 4.76$ $\quad \Delta t = 1.0\text{cm}/(20\text{cm/min}) = 0.05$ min

$$\tau_m = \frac{0.05}{\pi} (0.405 + 2.214)(2.378) = 0.0991 \text{ min}$$

$= 5.94$ sec

Dead time (θ_m)

$$\theta_m = \left(\frac{2\Delta t}{\pi}\right) \frac{\sqrt{(1 - \xi^2)(K_cK_m + 1)}}{\xi\sqrt{K_cK_m + 1} + \sqrt{\xi^2(K_cK_m + 1 + K_cK_m)}}$$

$$= \frac{2(0.05)}{\pi} \frac{2.378}{0.405 + 2.214} = 0.0289 \text{ min}$$

$= 1.73$ sec

First order plus dead time model (G_m)

$$G_m = \frac{1.06\, e^{-1.73s}}{5.94s + 1}$$

Figure 4.6. (continued).

Table 4.7

Comparison of First Order Plus Dead Time Modelling Parameters
Generated Using Four Approximation Techniques
on Process Eight (process number 830100)

Modelling Method	Gain	Time Constant τ	Dead Time θ	Ratio θ/τ
Cohen & Coon*	1.0	6.0	0.9	0.15
Miller*	1.0	4.2	0.9	0.21
Smith*	1.0	3.15	1.95	0.62
Yuwana & Seborg**	1.06	5.94	1.73	0.29

*Open loop method

**Closed loop method

Table 4.8

Comparison of First Order Plus Dead Time Modelling Parameters
Generated Using Four Approximation Techniques
on Process Five (process number 530000)

Modelling Method	Gain	Time Constant τ	Dead Time θ	Ratio θ/τ
Cohen & Coon*	1.0	5.7	1.2	0.21
Miller*	1.0	3.6	1.2	0.33
Smith*	1.0	2.92	2.03	0.69
Yuwana & Seborg**	1.05	4.62	2.33	0.504

*Open loop methods

**Closed loop method

results from studies performed with process 5. Each method yielded significantly different parameters and the effectiveness of the models for generating tuning constants will be compared after a discussion on tuning methods.

Controller tuning--emperical techniques. Many empirical relationships are available for selecting PID tuning constants for a first order plus dead time process. In all of the methods proportional gain is determined from the dead time to time constant ratio (R). In some methods, Cohen-Coon and Lopez (44), the integral (T_i) and derivative (T_d) times are related to R. In others, Ziegler-Nichols and Shinsky, the magnitude of the dead time is used to compute the constants.

Studies using the PPS were performed to compare the relationships of Lopez and Shinsky to the optimal controller constants (OCC) for process ten. These methods were chosen because

1. these were the most recently developed of the techniques mentioned above;
2. the relationships optimize the response to a load disturbance;
3. the same criterion was used to determine the "best" set of constants for each method.

The optimal PI tuning constants (OCC) for processes 5, 8, 10, 15, 6, 7, and 14 are compiled in Tables 4.9 and 4.10. These constants were determined by a direct search of the cost surface for a load disturbance. The cost was evaluated according to the

Table 4.9

Optimal IAE Tuning Constants for Several Overdamped Processes
Tuned for a 20% Load Disturbance

Dead Time Switch	Process 5		8		10		15	
	KK_c	T_i/T_{st}	KK_c	T_i/T_{st}	KK_c	T_i/T_{st}	KK_c	T_i/T_{st}
0	2.7	32.2	5.0	22.2	-	-	1.42	18.0
1	2.1	38.9	4.0	27.7	4.0	11.1	1.26	19.4
2	1.68	44	3.20	36.1	3.0	13.9	1.04	22.2
3	1.5	48.6	2.6	38.8	2.2	15.2	0.9	23.6
4	1.2	52.7	2.1	40.8	1.8	19.4	0.84	26.3
5	1.0	44.4	1.9	45.8	1.44	24.1	0.8	29.1
6	0.88	38.8	0.9	30.5	1.32	27.7	0.74	32.6
7	0.76	40.0	0.66	33.3	1.0	38.8	0.64	33.3
8	0.62	41.6	0.62	40.2	0.82	41.6	0.59	36.6
9	0.6	45.8	0.56	38.8	0.8	43	0.54	40.3
10	0.56	47.2	0.5	43.0	0.72	45.8	0.50	41.6

Table 4.10

Optimal Tuning Constants for Underdamped Processes at Several Dead Time Settings

Dead Time Switch	Process 6		Process 7		Process 14	
	KK_c	T_i/T_{st}	KK_c	T_i/T_{st}	KK_c	T_i/T_{st}
0	0	30.5	3.0	27.7	4.0	33.3
1	0	30.5	1.2	25.0	2.2	41.6
2	0	30.5	0.66	13.9	1.6	38.8

integral of the absolute error (IAE):

$$IAE = \int_0^{t'} \text{error}\ dt \qquad [4.11]$$

where the error is the difference between the desired set point and the process sensor reading. This is the same criterion used by Lopez and by Shinsky. The optimal tuning constants for process ten in Table 4.9 were used to determine empirical relationships for PI tuning constants. These are compiled in Table 4.11 along with those of Lopez, Shinsky, and the continuous cycling method.

Integral criterion are commonly used in controller tuning studies because results may be compared quantitatively. Other criterion have been proposed (35) but the IAE is mostly used in process control (44) mainly because processes tuned according to the IAE generally exhibit an approximate decay ratio of 0.25 (quarter amplitude decay) which represents a trade-off between initial speed of response and settling time. This is known as a "tight" response (28).

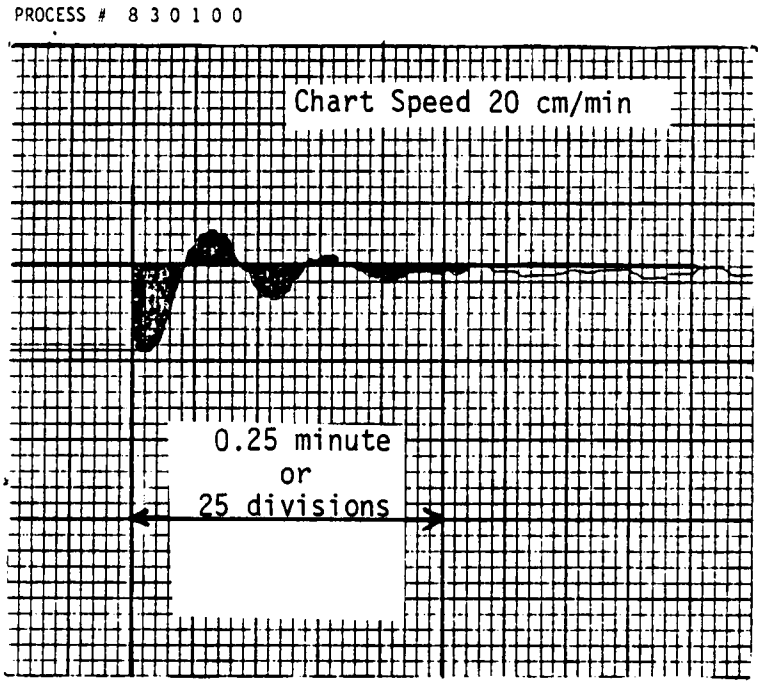
In Figure 4.7 is shown the closed loop response of process 5 to a 15% change in set point. The fiqure illustrates a crude method of calculating the IAE and was provided to students to demonstrate computing an IAE from strip chart data. In a laboratory experiment, students were expected to generate the set point or disturbance response of a closed loop system with different controller settings and compare the performance via an IAE criterion. Such an experiment was simple to structure and provided a quantitative and qualitative demonstration of what is meant by a "good" response.

Table 4.11

Equations for Four Emperical PI Tuning Relationships

Continuous Cycling	Lopez	Shinsky	OCC
$KK_c = K_u/2.2$	$KK_c = 0.984*(R^*)^{-0.986}$	$KK_c = 0.48*R$	$KK_c = 1.14*R^{-0.674}$
$T_i = P_u/1.2$	$\tau/T_i = 0.608*R^{-0.707}$	$T_i = 1.74*R$	$T_i/\tau = 1.32*R + 0.33$

$^*R = \theta/\tau$



$K_c = 4.5$

ΔSet point = 0.15

1 Time division (T) = 0.2 cm x (min/20 cm) = 0.01 min

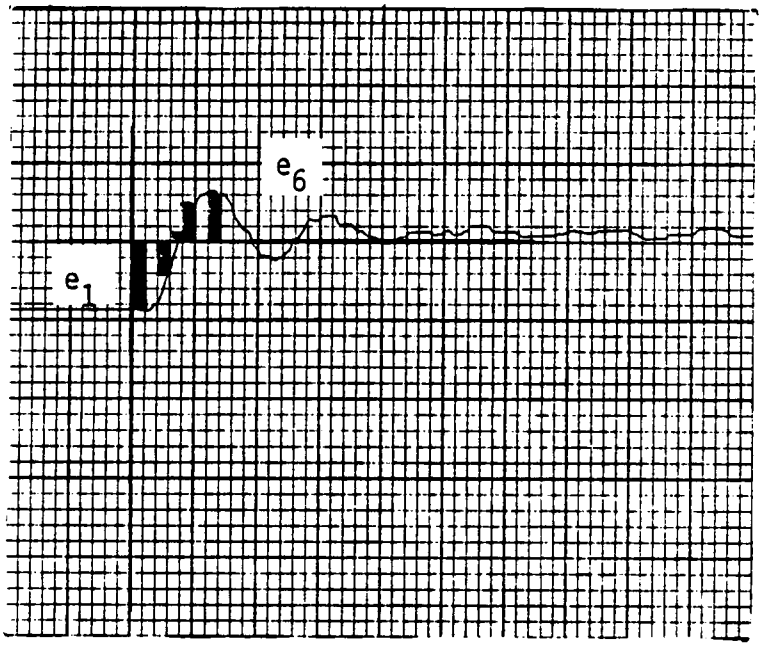
Error = shaded area

$$IAE = \Sigma_{i=o}^{N} \; T \cdot Error_i \quad \text{(Rectangular integration)}$$

$$= T \; \Sigma_{i=1}^{N} \; Error_i$$

$$= 0.01 \; \Sigma_{i=1}^{25} \; Error_i = 0.01 \; \Sigma_{i=1}^{25} \; e_i$$

Figure 4.7. Example calculation showing how to compute the integral of the absolute error for a set point change.



e_1 = 0.9 cm e_3 = 0.4 e_5 = 0.5 e_7 = 0.7 (samples)

Σe_i = 6.95 cm

IAE = 0.0695 cm-min

$$\text{Normalize IAE} = \frac{\text{IAE}}{\Delta S_p} = \frac{0.0695}{0.15} = 0.463 \text{ cm-min}$$

Figure 4.7. (continued)

In Table 4.12 are compiled the results from such an experiment.

Table 4.12 provides a comparison of three tuning relationships (Lopez, Shinsky, and the optimal settings) for seven ratios of dead time to time constant (D thumbwheel switch settings of 1 to 7). Comparison of the data shows that Shinsky's method has the poorest overall performance but it was comparable to Lopez's for ratios up of 0.5. Lopez's method yielded acceptable results for all seven cases with a maximum deviation from the optimal IAE of 31%. All three methods gave comparable integral time values, but proportional gains differed considerable. Shinky's method sets the gain in proportion to the R, while Lopez's and the optimal method indicate that an exponential relationship is more appropriate. Thus, Shinsky's method is applicable only to ratios of less than 0.5, while the method of Lopez is acceptable up to ratios of at least 1.2. The fact that all three methods yield comparable IAE's for ratios up to 0.5, even though the controller constants are significantly different, suggests that the cost is relatively insensitive to variations in these parameters. This was ,in fact, observed to be true during the studies used to generate the optimal controller settings.

The study presented above provides a demonstration of the usefulness of the simulator in studying process control problems. The discussion also supplies results and expressions that should be useful in developing laboratory experiments using the PPS. In the next section, a comparison of the modelling methods as means for generating controller constants will be presented.

Table 4.12

Comparison of the Closed Loop Performance of Process Ten When the Four Emperical Tuning Relationships are Applied

Dead Time* Switch	θ/τ	Lopez			Shinsky			OCC		
		KK_c	T_i	IAE**	KK_c	T_i	IAE**	KK_c	T_i	IAE**
1	0.15	6.47	1.55	6.81	3.31	0.94	7.29	4.0	2.0	5.9
2	0.25	3.81	2.26	8.7	1.94	1.6	8.01	3.0	2.5	8.3
3	0.37	2.60	2.98	10.1	1.39	2.4	10.1	2.2	2.75	8.9
4	0.46	2.12	3.45	12.0	1.06	2.9	12.2	1.8	3.5	9.1
5	0.62	1.58	4.26	12.8	0.79	3.9	18.3	1.4	4.25	11.7
6	0.80	1.23	5.1	15.0	0.61	5.0	26.8	1.3	5.0	14.3

*Process 1070X00 (X = 1,2,3,4,5,6)

**Integral of absolute error for a 20% load disturbance ($G_{DV} = G_{MV}$)

Comparison of tuning and modelling methods. In this section the modelling methods of Cohen-Coon, Miller, Smith, and Yuwana-Seborg will be compared. The process models for two plants (processes 5 and 8) were used to generate tuning constants via the method of Lopez. The models compiled in Tables 4.7, page 145 and 4.8, page 146 were used in the studies. Tuning constants and the performance criterion (IAE) are shown in Tables 4.13 and 4.14.

The conclusions that may be drawn from the data in the tables are:

1. the Cohen-Coon models and the tuning relationship of Lopez yield constants that produce unstable closed loop behavior;
2. Miller's models produce stable behavior for the second order plant but unstable behavior for the higher order plant;
3. the Yuwana-Seborg models yield controller constants that are comparable to the optimal ones;
4. Smith's models produce closed loop performance that is reasonably close to the optimal.

Comparison of the models of each method suggests that following explanations:

1. the Cohen-Coon model produces an acceptable time constant but underestimates the dead time. This results in R being too small, yielding a gain that is too high and an integral time that is too small; the result is that the loop becomes unstable;
2. Miller's model provides a smaller estimate of the time

Table 4.13

Comparison of the Closed Loop Performance When the Various Modelling Methods are Used to Tune the Controller on Process Five

Modelling Method	KK_c*	T_i**	IAE***
Cohen-Coon	4.58	3.11	unstable
Miller	2.93	2.70	unstable
Smith	1.42	3.69	12.75
Yuwana & Seborg	1.93	4.68	11.0
Optimal	2.7	5.8	10.5

* $KK_c = 0.984(\theta/\tau)^{-0.986}$

** $\tau/T_i = 0.608(\theta/\tau)^{-0.707}$

*** Process number 570000 with a 20% disturbance

Table 4.14

Comparison of the Closed Loop Performance When the Various Modelling Methods are Used to Tune the Controller on Process Eight

Modelling Method	KK_c *	T_i **	IAE ***
Cohen-Coon	6.38	2.58	unstable
Miller	4.58	2.29	24.97
Smith	1.57	3.69	10.72
Yuwana & Seborg	3.32	4.07	9.00
Optimal	4.00	5.0	8.50

* $KK_c = 0.984(\theta/\tau)^{-0.986}$

** $\tau/T_i = 0.608(\theta/\tau)^{-0.707}$

*** Process number 570000 with a 20% disturbance

constant which increases R. This brings the proportional gain down closer to the optimal setting but the decreased value of time constant yields an integral time that is too fast. The result is that the closed loop response is either unstable or unacceptable;

3. Smith's method yields a large value of R and a small time constant. This results in a lower proportional gain and a slower integral time (relative to Cohen-Coon and Miller). It has already been observed that the cost surface was relatively flat with respect to K_c and the integral time and Smith's method benefits from this fact;
4. Yuwana-Seborg's method develops a much better approximation because it utilizes the actual closed loop behavior of the process in generating the model.

The results presented in this section illustrate the ways in which the PPS may be used to study feedback control. The situations and studies were selected from the set of typical methods and results presented in process control textbooks. The ease in which the simulator may be set up and operated allowed each of the experiments to be generated easily. The next section will address another important area of single loop control, feedforward control.

4.3 FEEDFORWARD CONTROL STUDIES

This section presents two applications of the PPS in the area of feedforward control. In the first, the simulator is placed in the PISO mode and used to study the dynamic characteristics of a common feedforward compensator, the lead-lag controller. In the second, the basic principles of feedforward control will be illustrated, using multiple simulators cascaded together.

Feedforward Controller Dynamics

Lead-lag compensator. The most common industrial feedforward controller is the lead-lag compensator (34). The transfer function of the controller is

$$G = g*(a*T*s +1)/(T*s+1) \qquad [4.13]$$

where g is a pure gain, T is the lag time, and a, which typically ranges from 0.1 to 10 (35), is the ratio of time constants.

Equation 4.13 can be rearranged to the form shown in Equation 4.14 if the term a-a is added to the numerator.

$$G = g*(a +(1-a)/(Ts+1)) \qquad [4.14]$$

The compensator now consists of a pure gain (g*a) and a first order lag (with a gain of g*(1-a)) in parallel; changes in a affect the initial response but leave the steady state (s=0) unchanged. Initial responses that undershoot (a<1) or overshoot (a>1) the final steady state can be obtained by adjusting a accordingly. If a is

one, the dynamic portion of G disappears and the compensator behaves as a pure gain with a magnitude of **g**. Thus, three characteristic response types are possible with a lead-lag compensator and all three are represented in the basic menu of processes in the PPS.

The inherently lead-lag dynamics are in processes 11, 12 and 13. The a values are 1/2 (undershoot), 1 (pure gain), 3/2 (overshoot) respectively. The set can be expanded considerably, using the PISO mode in conjunction with processes 10 (first order) and 11 (pure gain). The only restriction is that the process gains must sum to one if the dynamics are to be governed by Equation 4.14 (g=1). In Table 4.15 is compiled a representative sample of the a values, ranging from -1.0 to 2.0, that are possible using this configuration. Relaxing the restriction that the sum of the gains must equal one allows an even larger number of ratios.

The large number of a values that are possible with lead-lag dynamics has counterparts in other configurations; the lead-lag element is only one example of the complex process dynamics made possible through the PISO mode. For instance, the inverse response may be modelled by two competing parallel processes as can pulses, terminated ramps and PI controllers [processes 11 and 0 combined]. Thus, the flexibility of the simulator facilitates the design of experiments focused on the study of complex dynamics.

Feedforward Control Using Multi-Simulator Networks

Two, three or four simulators may be cascaded together to form complex dynamic networks that are useful for studying a variety

Table 4.15

Disturbance and Gain Thumbwheel Switch Settings and the Resulting Lead/Lag Time Constant Ratios in the PPS

Disturbance Thumbwheel Switch	Process Thumbwheel Switch	Resulting Ratios of Time Constants
9	7	-1
10	6	-0.75
11	5	-0.5
12	4	-0.25
8	3	0
0	1	0.25
1	1	0.5
3	8	0
4	15	1.25
5	14	1.5
6	13	1.75
7	12	2

Note: The simulator must be in the PISO mode of operation and Processes 10 and 11 selected for the plant characteristics. The given combinations of gains will always sum to one.

of process control topics. In this section, feedforward and ratio control will be discussed.

Feedforward control is based on the concept that the controller should be a model of the process (34); a good model is essential for feedforward control. Since two simulators are exactly alike, it would be expected that a reasonable demonstration of feedforward control could be achieved using one simulator as the process and one as a model of the process. This was, in fact, found to be true.

Mathematical background. The basic notion of single loop feedforward control is that a disturbance variable (DV_1) is used to adjust a manipulated variable (MV_1) in order to hold a process variable (PV_1) constant. However, feedforward control is seldom used alone because modelling errors always exist and unmeasured disturbances do affect the process. Usually the overall control loop consists of feedforward and feedback components with the manipulated variable being adjusted by both modes. The equation describing this relationship is

$$MV_1 = M_{ff} + M_{fb} \qquad [4.15]$$

where M_{fb} is the contribution of the feedback controller and M_{ff} is that of the feedforward controller.

Changes in PV_1 are related to MV_1 and DV_1 via the following expression:

$$PV_1 = G_{dv_1} * DV_1 + G_{mv_1} * MV_1 \qquad [4.16]$$

If changes in DV_1 are to have no effect on the output ($PV_1 = 0$) and the feedback controller is in manual ($M_{fb}=0$), then the feedforward control law may be determined by rearranging Equation 4.16. Substituting 4.15 into 4.16, setting M_{fb} to zero and solving for M_{ff} yields:

$$M_{ff} = (-G_{dv_1}/G_{mv_1}) * DV_1 \qquad [4.17]$$

Thus, the feedforward control law is the negative ratio of transfer functions (disturbance to manipulated). The ratio is usually approximated by a lead-lag transfer function. The validity of the approximation depends on the appropriateness of first order plus dead time models. (This was discussed in the section on process modelling.) If process ten is used for G_{mv} and G_{dv}, the ideal lead-lag control law results. In the discussion to follow the ideal lead-lag controller will be used to develop a feedforward control demonstration.

The steps required to design a feedforward control demonstration will be outlined in the following paragraphs. The demonstration utilizes two simulators cascaded together with one functioning as the process and the other playing the role of the feedforward controller.

Two simulator feedforward network. The physical connections required to cascade two simulators together are shown in Figure 4.8. The configuration allowed Sim-2 to act as the feedforward controller

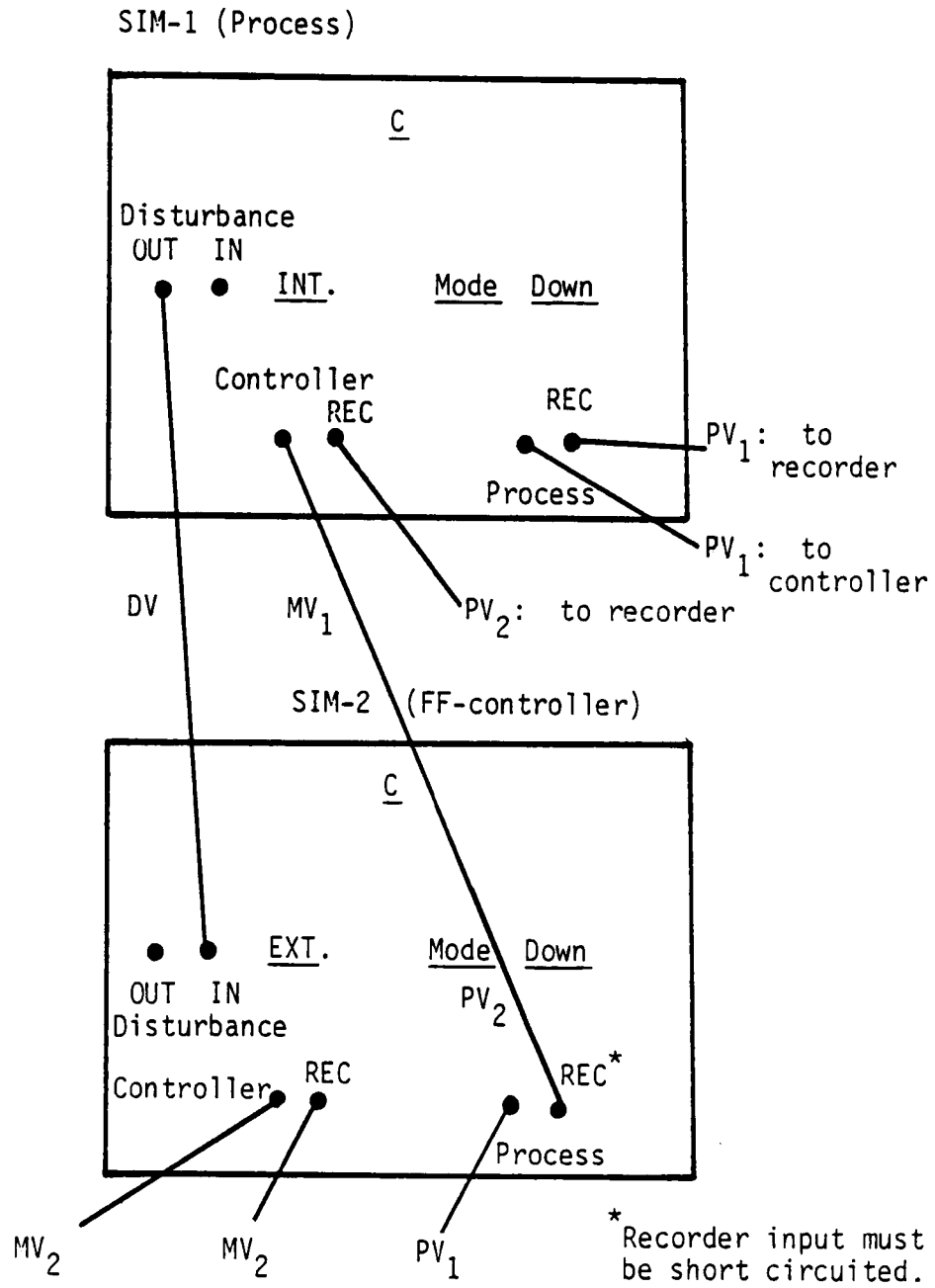


Figure 4.8. Interconnections and variable definitions for using one simulator as a process (SIM-1) and one as a feedforward controller (SIM-2).

and Sim-1 as the process. Sim-2 also served as a summing junction (through G_{mv_2}) for combining the outputs from the feedforward algorithm and the feedback controller. The block diagram of the network is shown in Figure 4.9 (the network included a feedback controller).

If G_{dv_1} and G_{mv_1} in Figure 4.9 are first order processes (number 10) and the time constant ratios are 1/2, 1 or 3/2, then the feedforward control law (G_{dv_2}) will be described by processes 12, 11 or 13 respectively. Time constant ratios and the corresponding S thumbwheel switch settings for Sim-1 are shown in Table 4.16. (Selecting the same S values gives a ratio of 1.0) Table 4.17 summarizes the process (P) thumbwheel switch values for Sim-2. Sample gain settings are compiled in Table 4.18, and Tables 4.19, 4.20 and 4.21 show the allowable dead times for both simulators.
(A table is needed for each ratio because the S thumbwheel switches are different and the dead time is a function of sample time (see Chapter III).) A procedure for designing and implementing a feedforward demonstration using Tables 4.16 through 4.21 follows:

1. The simulators are connected according to Figure 4.8. If desired, an external feedback controller may be connected with the valve signal going to Sim-2 and the sensor reading coming from Sim-1.
2. The desired time constant ratio is selected and used to determine the S thumbwheel switch settings for Sim-1 (Table 4.16). The G_{dv} (P and S) settings for Sim-2 are chosen from Table 4.17. Note that G_{mv} for Sim-2 should be 11 3 0 0 0 0; this sets up the system so the signal

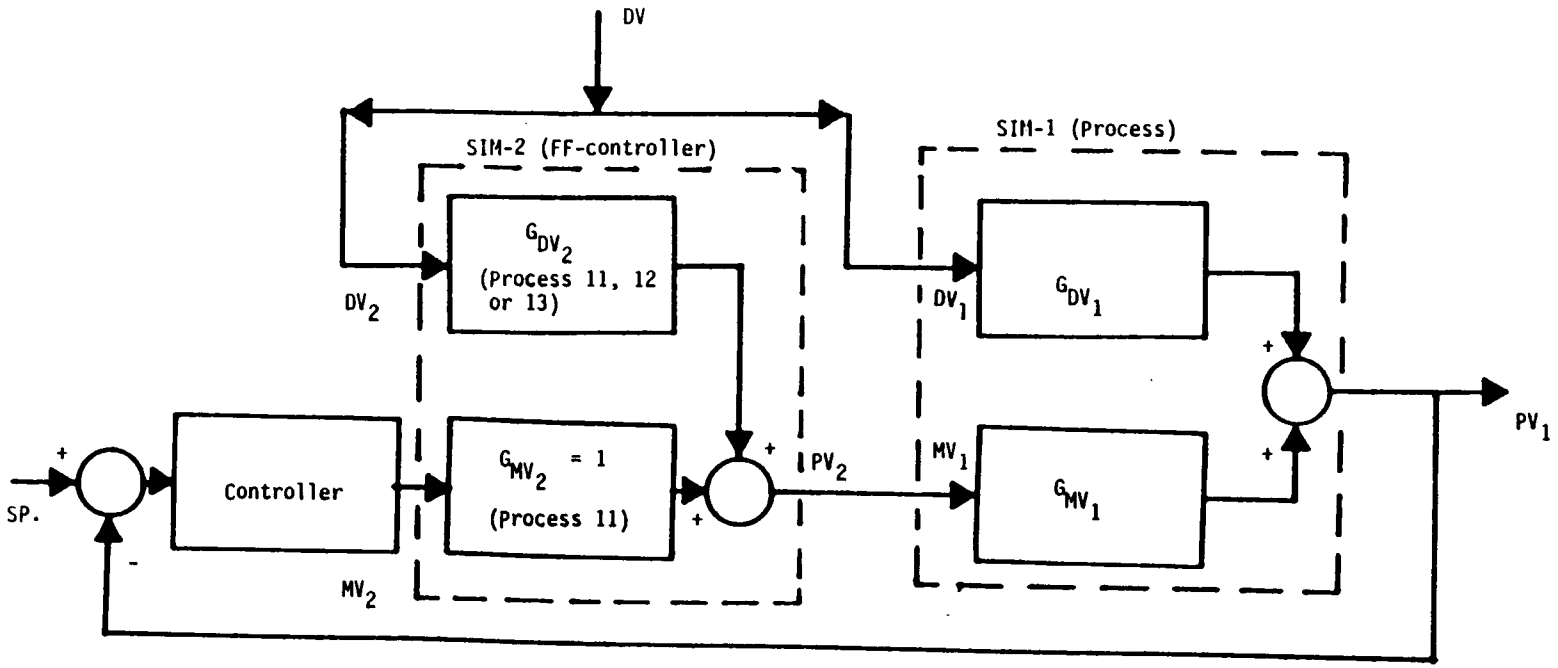


Figure 4.9. Feedforward plus feedback control system using two simulators and a conventional 3-mode controller.

Table 4.16

S-Thumbwheel Switch Settings for Obtaining Time Constant Ratios Needed in Feedforward Control Studies

Time Constant Ratio	Thumbwheel Switch Settings $MV\text{-}_{TWS}/DV\text{-}_{TWS}$						
.25	3/8	1/6	2/7	4/12	6/14	5/3	
.33	6/12	1/4	3/7	0/3	5/10		
.50	0/1	1/3	2/4	3/6	4/7	6/8	7/12
1.50	2/1	4/3	7/6	9/8			
.75	7/8	4/6	2/3				

Table 4.17

Time Constant Ratios in SIM-1 and Corresponding P-Thumbwheel Switch Settings in SIM-2

Time Constant Ratios in SIM-1 (τ_{DV}/τ_{MV})	P-Thumbwheel Switch Setting In Disturbance Process For SIM-2
0.5	12
1.5	13
1	11

Note: Speed setting in SIM-2's disturbance block is the same as the disturbance block in SIM-1. P setting for the manipulated variable process should be 11 and the gain setting should be three.

Table 4.18

Feedforward Gain Setting When SIM-1
is a Feedforward Controller for SIM-2

Gain in SIM-1's Disturbance Block		Gain in SIM-1's Process Block		Feedforward Controller's Disturbance Block Gain	
Switch Setting	Gain	Switch Setting	Gain	Switch Setting	Gain
2	3/4	1	1/2	10	-3/2
2	3/4	14	-1/2	5	+3/2
13	-3/4	3	1	2	+3/4
2	3/4	3	1	13	-3/4
12	-1	3	+1	3	+1
3	1	3	1	12	-1
14	1/2	0	1/4	7	2

Table 4.19

Sample Dead Time Settings for SIM-1 (the Process) and Corresponding Values of the Disturbance Dead Time Setting on SIM-2 (the FF Controller) When the Time Constant Ratio is 0.5

SIM-1		SIM-2
Process Disturbance Switch Setting	Disturbance Switch Setting	Dead Time Switch Setting
2	3	2
2	4	3
3	4	3
4	5	4
2	6	5
4	7	5
2	8	7
4	8	6
5	9	6

Table 4.20

Sample Dead Time Settings for SIM-1 (the Process) and Corresponding Values of the Disturbance Dead Time Setting on SIM-2 (the FF Controller) When the Time Constant Ratio is 1.0

SIM-1		SIM-2
Process Disturbance Switch Setting	Disturbance Switch Setting	Dead Time Switch Setting
1	6	5
2	3	2
2	4	2
2	5	4
2	7	5
3	6	4
4	5	3
4	7	4
5	6	2

Table 4.21

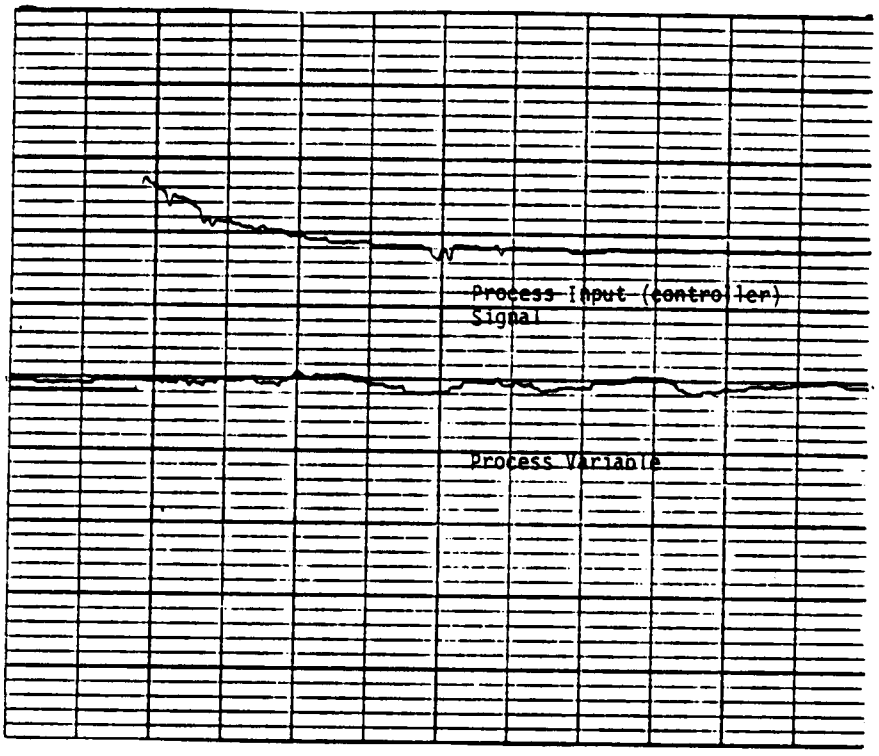
Sample Dead Time Settings for SIM-1 (the Process) and Corresponding Values of the Disturbance Dead Time Setting on SIM-2 (the FF Controller) When the Time Constant Ratio is 1.5

SIM-1		SIM-2
Process Disturbance Switch Setting	Disturbance Switch Setting	Dead Time Switch Setting
1	2	1
1	4	3
2	5	3
3	5	2
4	6	1
3	7	4
2	8	5
4	8	3
5	9	1

from the controller is summed directly with the feedforward output.

3. Gains are chosen from Table 4.18 for both simulators.
4. Dead time switch settings are selected from either Table 4.19, 4.20, or 4.21. These switch settings yield physically realizable controllers, i.e. positive dead times are not needed.
5. The valve on Sim-2 is initialized and the thumbwheel switch settings for both simulators are implemented via M-IC and D-IC commands.
6. The OP buttons on both simulators are pressed--Sim-2 first; this insures that the proper initial conditions are established in both devices.
7. Finally, a disturbance is entered using the disturbance rotary switch on Sim-1, and the system's response recorded.

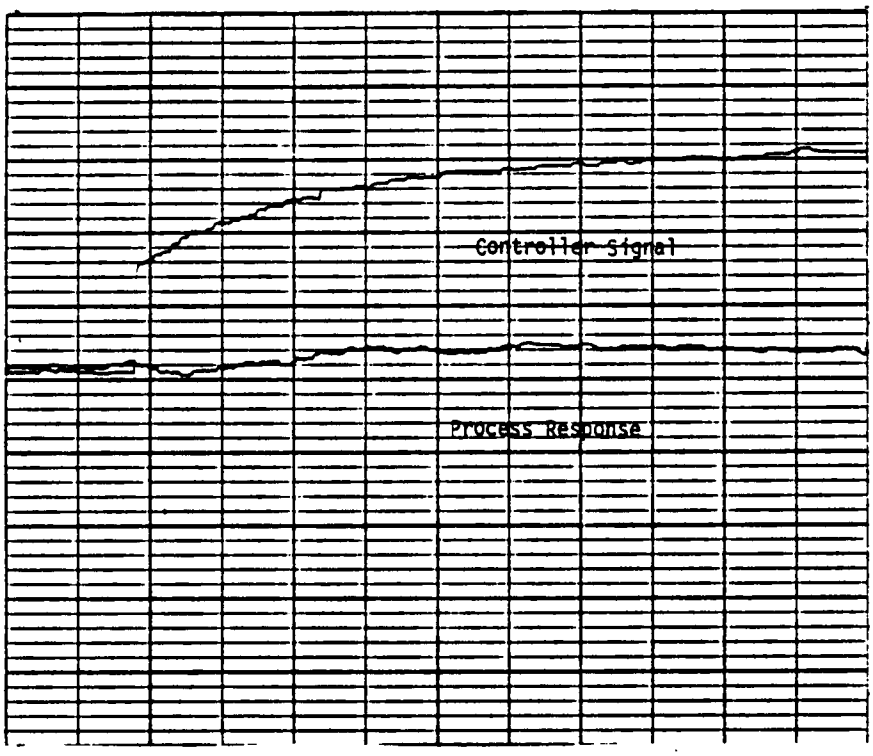
In Figures 4.10 and 4.11 are two examples of feedforward control loops designed using the procedure described above. The figures include strip chart recordings of the system's response to a 20% load disturbance as well as a summary of the switch settings. In both examples, the feedback controller was in manual. For comparison Figure 4.12 shows the reponses of a feedforward/feedback and a feedback only network to 40% load disturbances. The block diagram for the system in Figure 4.12 is shown in Figure 4.13. The procedure described previously was used to select the thumbwheel switch settings of the processes shown in the figure. The feedforward



Thumbwheel Switch Settings

		P	G	S	D	B	N
Process (SIM-1)	Disturbance Block	10	3	2	0	0	0
	Process Block	10	12	1	0	0	0
Feedforward Controller (SIM-2)	Disturbance Block	13	12	1	0	0	0
	Process Block	11	3	0	0	0	0

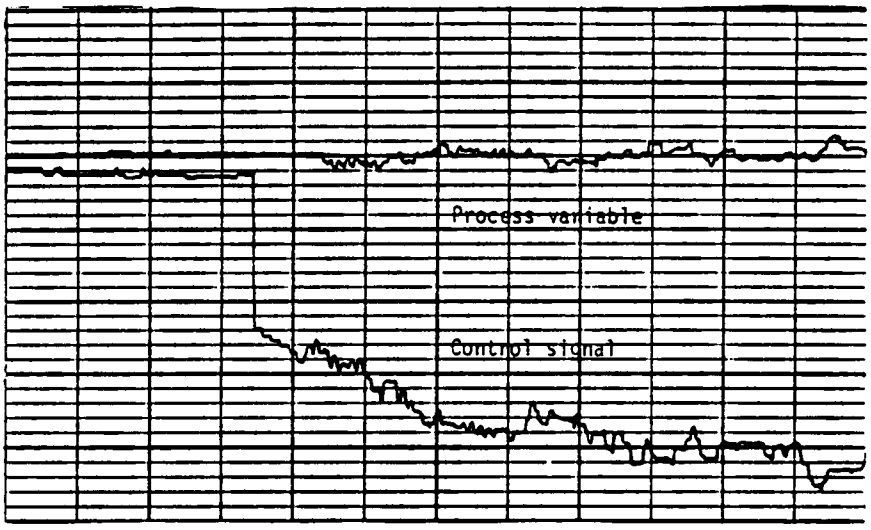
Figure 4.10. Input and process response to a 20% disturbance when two simulators are connected in a feedforward fashion (Process 13 is used as the lead/lag dynamic element).



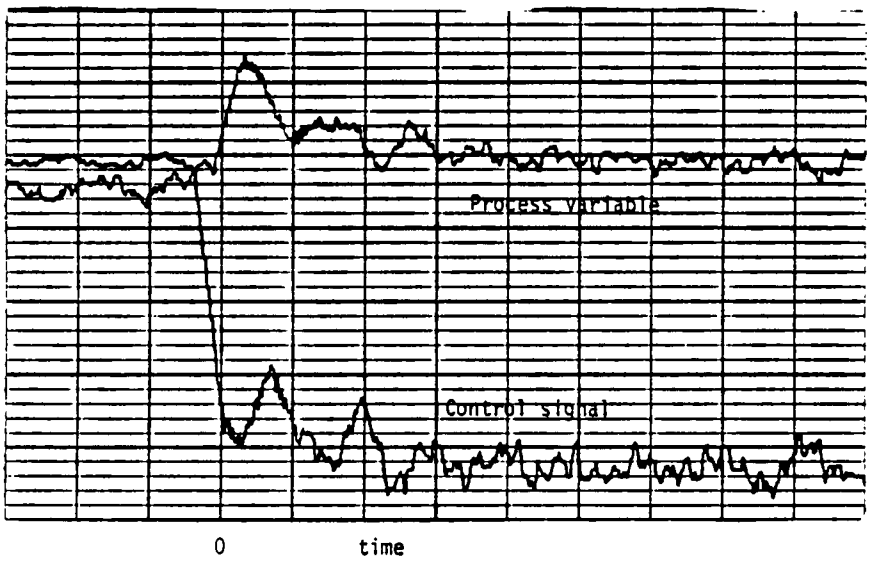
Thumbwheel Switch Settings

		P	G	S	D	B	N
Process (SIM-1)	Disturbance Block	10	2	3	0	0	0
	Process Block	10	1	1	0	0	0
Feedforward Controller (SIM-2)	Disturbance Block	12	10	3	0	0	0
	Process Block	11	3	0	0	0	0

Figure 4.11. Input and process response to a 20% disturbance when two simulators are connected in a feedforward fashion (Process 12 is used as the lead/lag dynamic element).



B) Process under PI and feedforward compensation



A) Process under PI control only

Figure 4.12. Response of process and controller to a 40% disturbance: with and without a feedforward compensator.

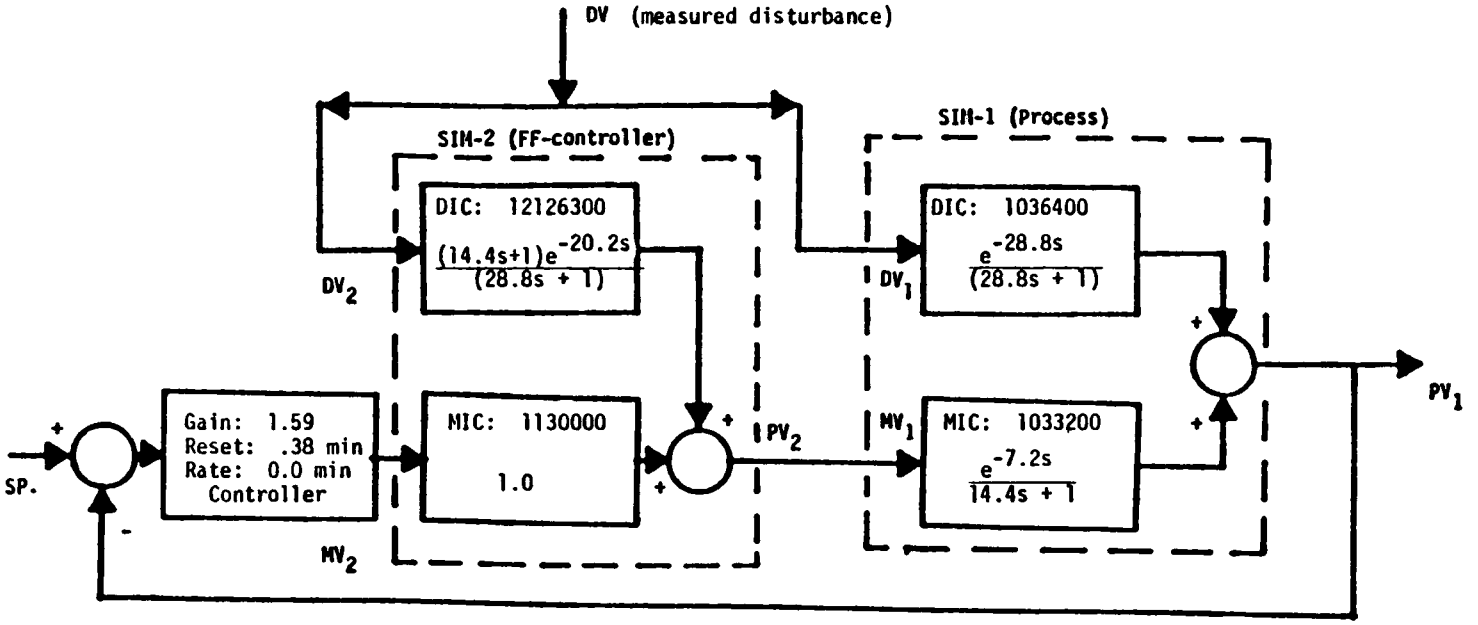


Figure 4.13. Feedforward plus feedback control system demonstrating the use of two simulators and a conventional 3-mode controller.

and/or feedback loop could be disconnected by setting the disturbance input to internal and/or setting the manual selector switch to the "M" position. In each example the feedforward control loop gives nearly perfect control, as would be expected. So the two simulator network does provide a valid demonstration of feedforward control. The primary benefit of using one simulator as a feedforward controller is the ease in which a network may be setup for an in-class demonstration of lead-lag feedforward control.

Ratio and cascade control using a three simulator network. Ratio control is another feedforward control strategy that may be demonstrated using a multiple simulator network. The objective in ratio control is to maintain the ratio of two process variables constant in the face of changes in the free or "wild" variable. To adequately demonstrate this control mode requires the addition of a third simulator to the network. This allows the system to have measured and unmeasured disturbances as well as dynamics in the variables being ratioed. (Demonstrations using one (or two) simulator networks would be limited to pure gain controllers and unmeasured disturbances would be impossible.) In Figure 4.14 are shown the connections required to form a three simulator network; Figure 4.15 shows the resulting block diagram. A block diagram of a multi-loop control system that employs the configuration is shown in Figure 4.16. The block diagram provides an example of cascade as well as ratio control.

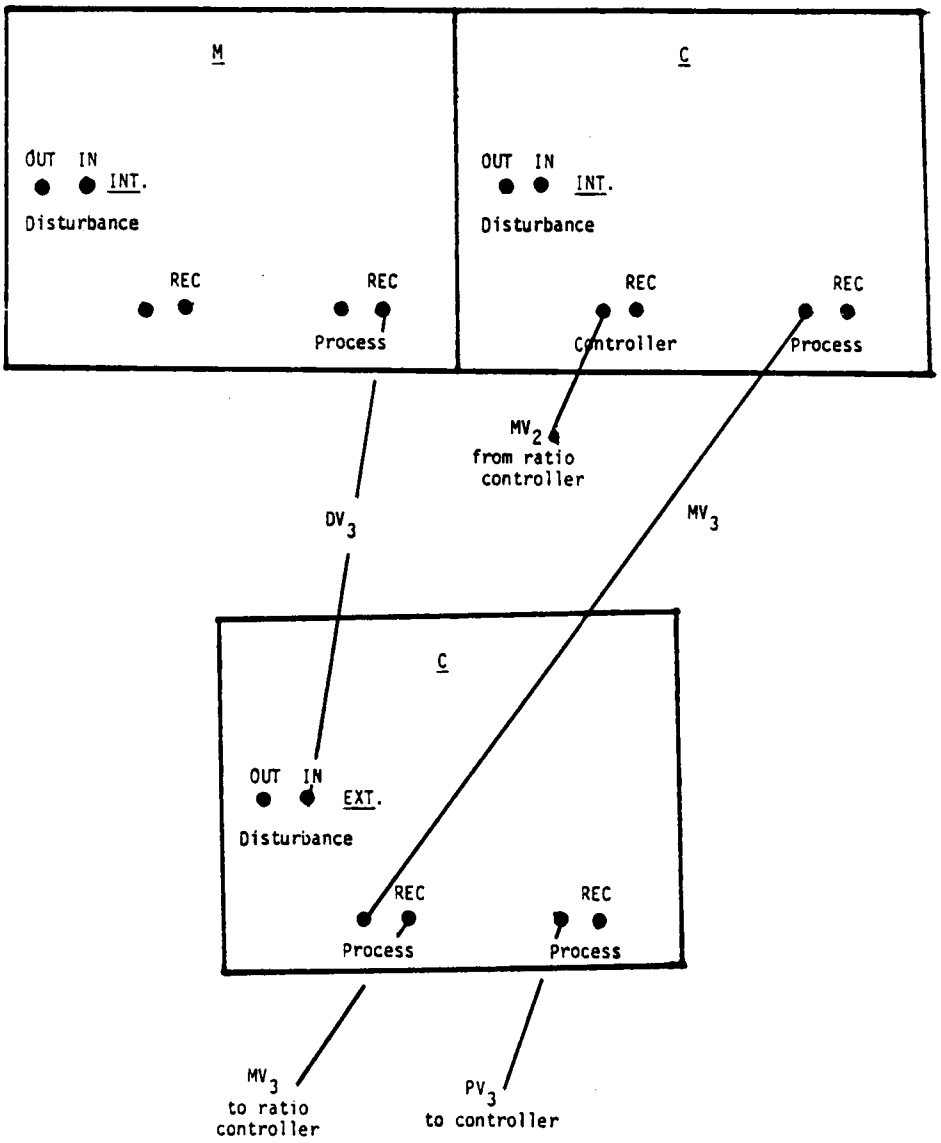


Figure 4.14. Three simulator cascade network connections and variable definitions corresponding to the block diagram of Figure 4.15.

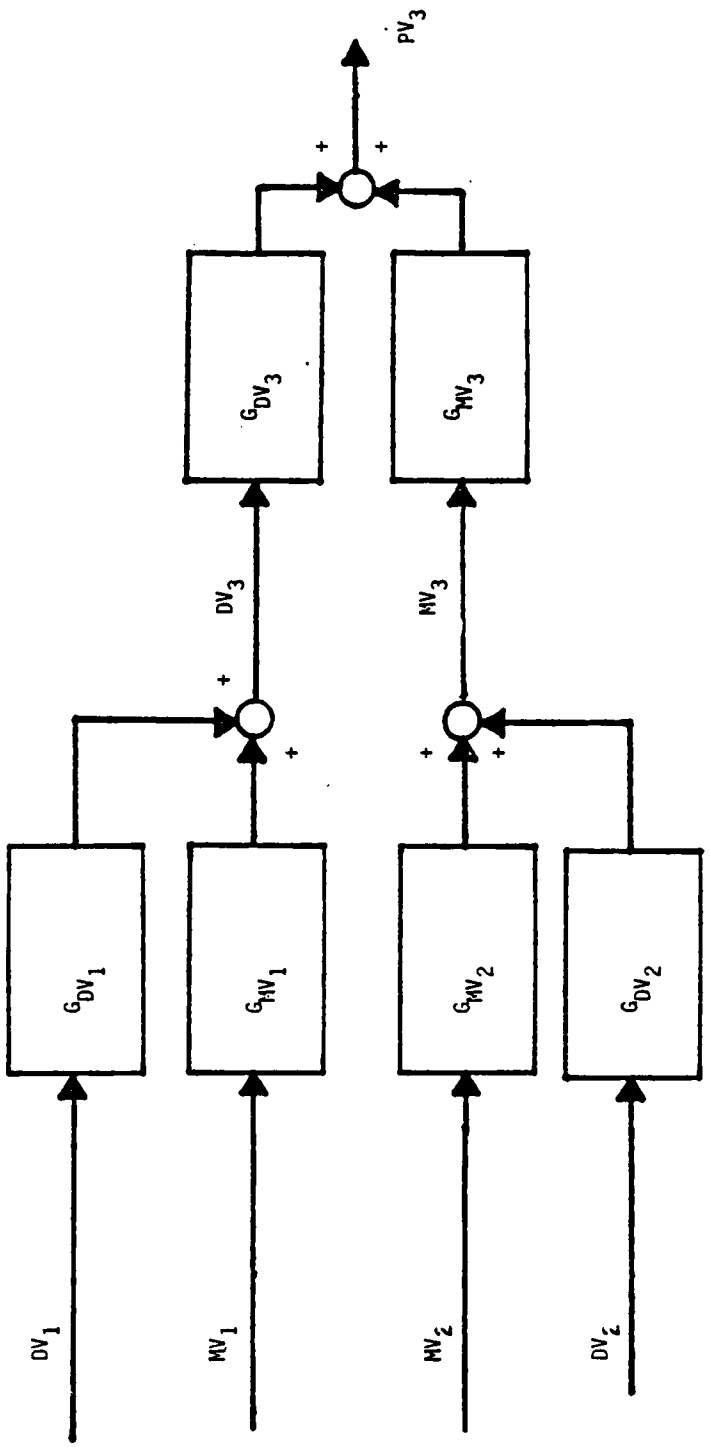


Figure 4.15. Block diagram for a three simulator cascade network.

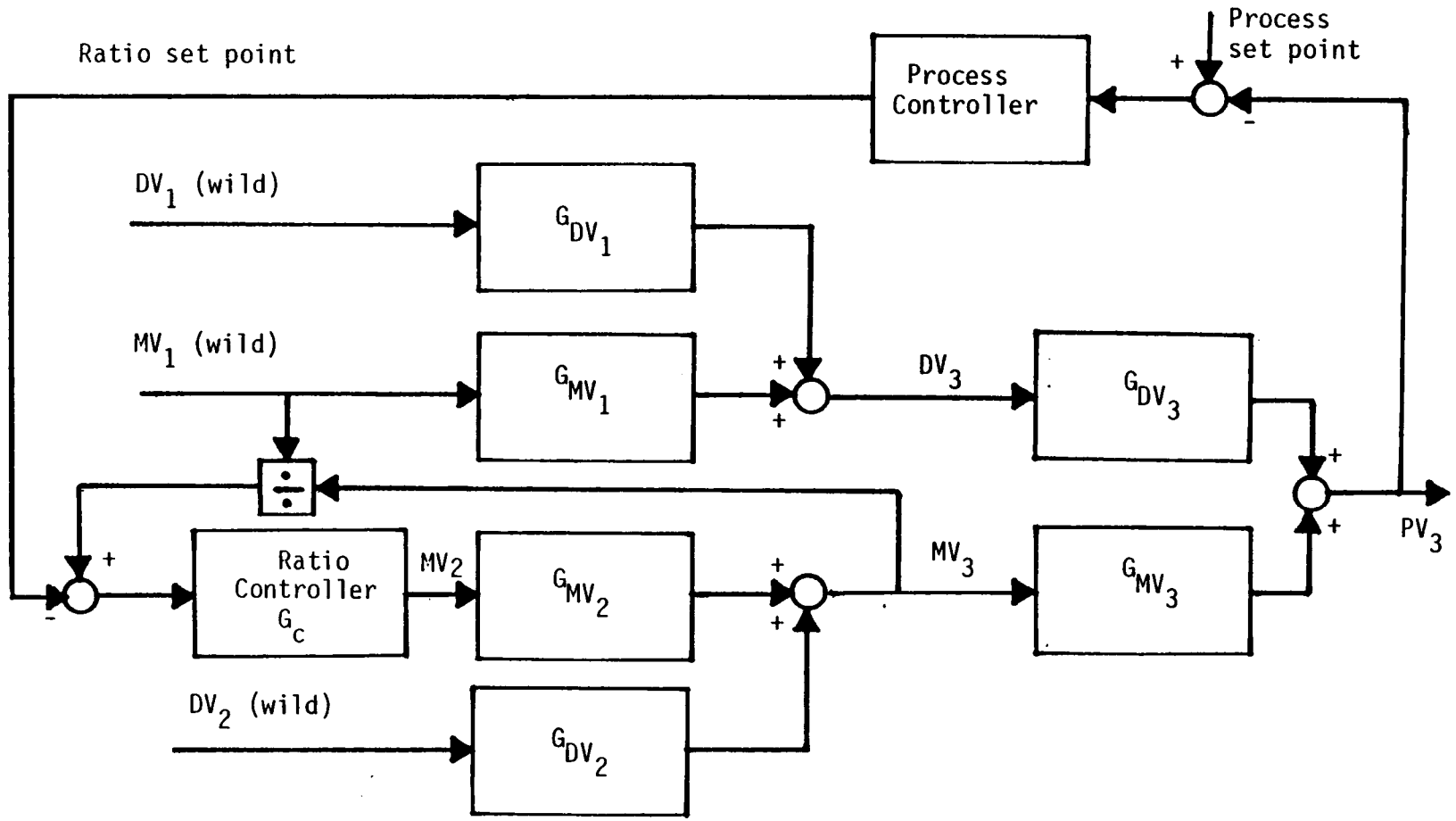


Figure 4.16. Example of a ratio/cascade control system using the three simulator cascade network.

In the example, MV_1 is a measured disturbance variable and maintaining a constant ratio of MV_3 to MV_1 is the feedforward control objective. Keeping PV_3 constant is the overall objective; the feedback controller must adjust the ratio set point (cascade control) to achieve this goal. Unmeasured disturbances enter the system either through DV_1 or DV_2. If the disturbance enters through DV_1, then the change is not detected by the ratio controller; the feedback or master controller in the cascade loop detects the disturbance through changes in PV_3 and adjusts the set point to the ratio controller to restore PV_3 to the desired level. This serves to demonstrate the role of the master controller. If the disturbance enters through DV_2, the change is seen by the ratio controller and action is taken to counteract the change before the process variable is affected. This demonstrates the role of the inner or slave controller in a cascade system. The feedforward controller senses changes in MV_1 and responds by changing the feedback signal to the ratio controller. Thus, the configuration sketched in Figure 4.16 illustrates ratio and cascade control strategies. This is a somewhat more complex control strategy than feedforward or ratio applied alone, but the extension is logical and demonstrates a common control solution.

In this section, two examples of using multi-simulator configurations have been shown. In the next section, uses of the PPS to study multivariable systems will be described.

4.4 MULTIVARIABLE PROCESSES AND MULTISIMULATOR NETWORKS

Multi-input, multi-output systems may be studied using networks of two or more simulators. However, the two input, one output structure of the PPS limits the number and type of multivariable configurations possible. The two input, two output (2X2) network is the easiest to form; this is also the one most commonly discussed in process control textbooks and literature (5,29,34). In the following sections, the PPS will be shown to be applicable to the study of multivariable systems. The discussions begin with a brief description of standard canonical forms and the corresponding hardware connections. This will be followed by a discussion of interaction analysis using Bristol's relative gain array (46). Finally, the topic of dynamic decoupling using a four simulator network will be addressed.

2X2 Multivariable Simulation Networks

Multivariable systems are catagorized according to the canonical form or inherent input, output structure; specification of a canonical form serves to describe exactly how the inputs and outputs interconnect. The P-canonical and V-canonical formulations (47) are the ones most often encountered in process control (48,49) and the P-canonical structure, by far the most popular formulation (49), is discussed below.

P-canonical networks. A P-canonical process has a non-interactive output structure. Each input effects each output

through independent transfer functions, but the outputs are not directly interconnected; a change in one output (due to a disturbance for instance) does not have a direct effect on any other. In Figure 4.17 are shown the connections required to configure two simulators as a P-canonical process. The expressions relating inputs and outputs may be written directly from the accompanying block diagram. These expressions are given in Equations 4.18a and 4.18b, where K_{ij} represents the steady state gain and G_{ij} represents the dynamic portion of the transfer function of the process.

$$PV_1 = K_{11}\, G_{11}\, MV_1 + K_{12}\, G_{12}\, MV_2 \qquad [4.18a]$$

$$PV_2 = K_{21}\, G_{21}\, MV_1 + K_{22}\, G_{22}\, MV_2 \qquad [4.18b]$$

The P-canonical structure shown in Figure 4.17 is a popular process description because of the non-interacting output structure; this allows Equations 4.18a and 4.18b to be rewritten using matrix notation and the techniques of linear algebra applied for analysis and design. This structure also has other features that enhance its utility. Two such features are: that the transfer function matrix is directly determined from the process step responses and that open loop stability is assured if the individual dynamic elements (G_{11}, G_{22}, etc.) are stable. The V-canonical form does not have either of these characteristics and is thus a more complex representation.

V-canonical networks. The wiring diagram and corresponding block diagram for a V-canonical network are shown in Figure 4.18. The mathematics describing the process is given in 4.19a and 4.19b.

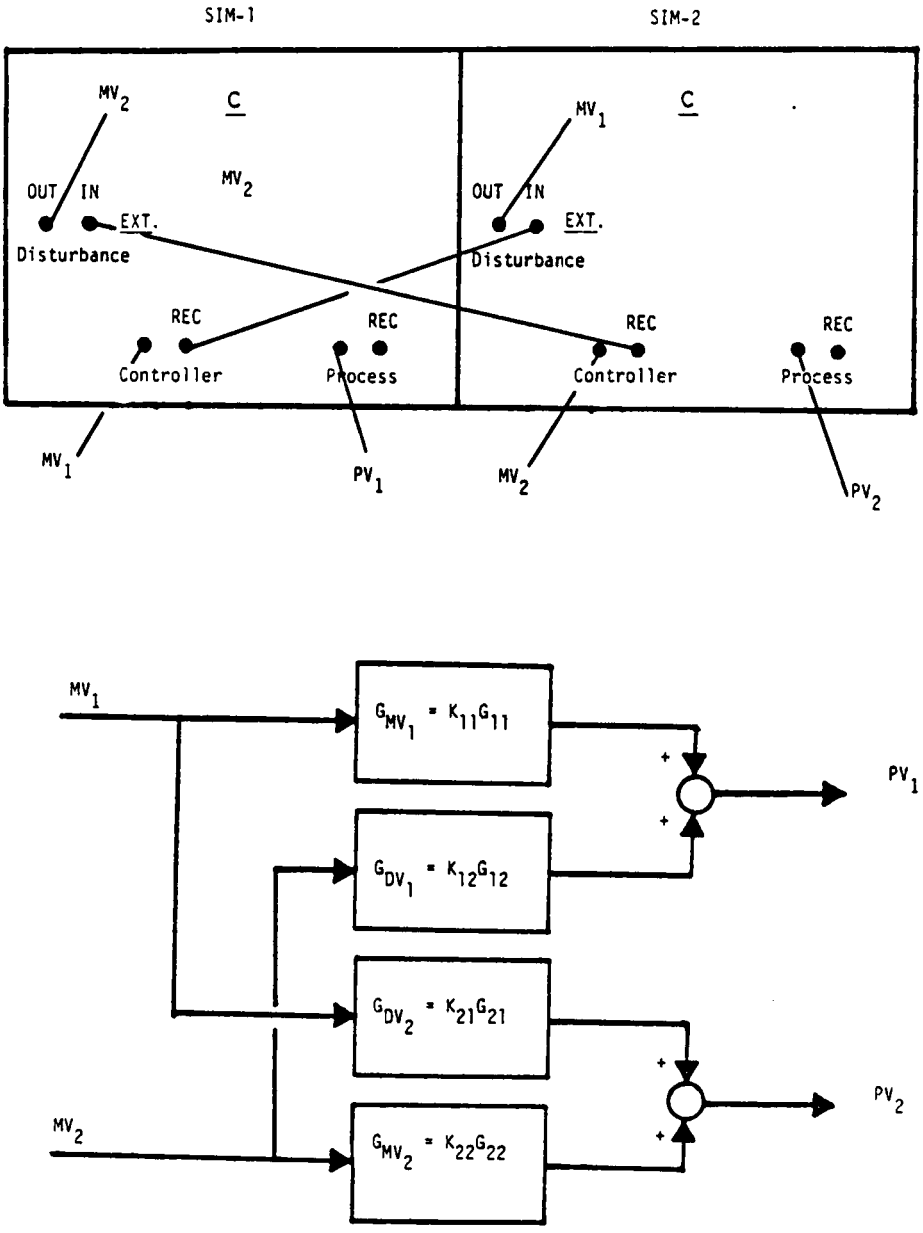


Figure 4.17. Connections required to form a P-Canonical multivariable network using two simulators.

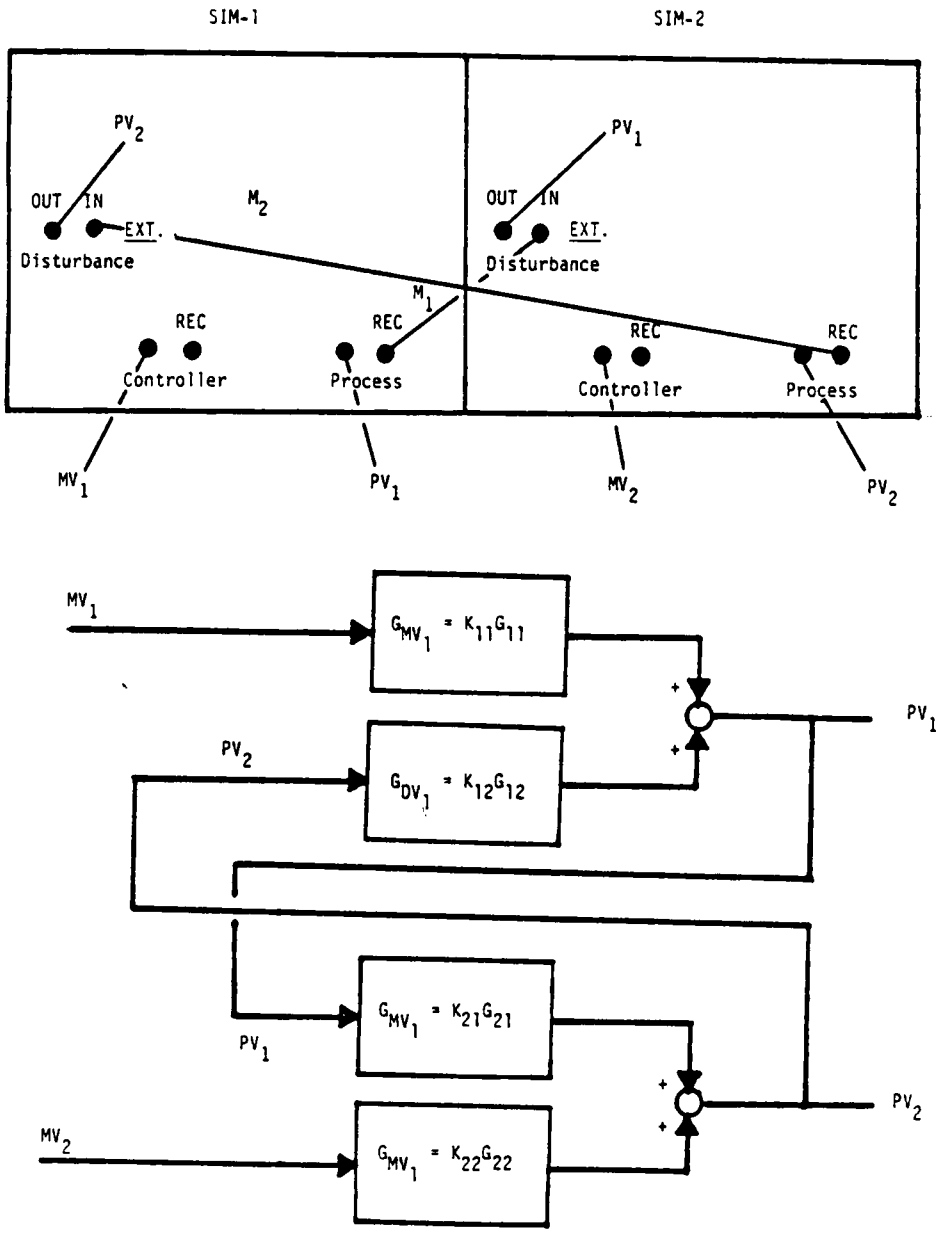


Figure 4.18. Connections required to form a V-Canonical multivariable network using two simulators.

$$PV_1 = K_{11}\, G_{11}\, MV_1 + K_{12}\, G_{12}\, PV_2 \qquad [4.19a]$$

$$PV_2 = K_{22}\, G_{22}\, MV_2 + K_{21}\, G_{21}\, PV_1 \qquad [4.19b]$$

The V-canonical is said to have an interacting output structure because each output appears in both equations; the result is that the V-canonical structure has an inherent feedback loop built into it. The feedback creates the potential for complex dynamics through these interactions. This can be more readily seen if Equations 4.19a and 4.19b are rearranged to the P-canonical form given by Equations 4.20a and 4.20b.

$$PV_1 = (1/G)\,(G_{mv_1}\, MV_1 + G_{dv_1}\, G_{mv_2}\, MV_2) \quad . \qquad [4.20a]$$

$$PV_2 = (1/G)\,(G_{mv_2}\, MV_2 + G_{dv_2}\, G_{mv_1}\, MV_1) \qquad [4.20b]$$

$$G = 1.0 - G_{dv_2}\, G_{dv_1} \qquad [4.21]$$

If all four transfer functions in Equations 4.20 and 4.21 are first order lags with equal time constants but with different gains, these equations become

$$\mathbf{PV} = \begin{matrix} K_{11}*(Tau*s+1)/CE & , & K_{12}*K_{21}/CE \\ K_{12}*K_{21}/CE & , & K_{22}*(Tau*s+1)/CE \end{matrix} \mathbf{*MV} \qquad [4.22a]$$

where

$$CE = (Tau*s)^2 - K_{12}*K_{21} \qquad [4.22b]$$

In equation 4.22 there is a single second order characteristic

equation and the damping coefficient and natural, undamped frequency are given as follows:

$$z = 1.0/[(1-K_{12}\ K_{21})]^{0.5} \qquad [4.23]$$

$$w_n = 1.0/Tau*z \qquad [4.24]$$

The damping coefficient in Equation 4.23 is a function of the gains in the cross coupling terms in the block diagram. These gains can be selected to yield step responses that are overdamped, critically damped, underdamped or unstable. Thus, a relatively large range of response characteristics are possible with a V-canonical network. Table 4.22 provides examples of the stable damping coefficients versus thumbwheel switch settings. Unstable responses result from combinations of gains that are not represented in the table.

Table 4.22 gives an indication of the wide range of dynamics possible with a V-canonical process. Nevertheless, the V-canonical structure, due to its inherent complexity, is seldom used to describe a process; this is because analysis techniques are more readily applied to the P-canonical formulation (50,51,52). The P-canonical structure is more often employed to describe process dynamics, but both forms are used in dynamic decoupling schemes. Each formulation has unique advantages that dictate when it should be employed; this will be discussed in the section on decoupler design. In the next section the topic of steady state interaction analysis will be discussed and in this regard both formulations can be proven equivalent.

Table 4.22

Damping Factor Versus Gains (in the Cross Coupling Terms)
for a V-Canonical 2x2 Process

	-1.75	-1.5	-1.25	-1.0	-0.75	-0.5	-0.25	0	0.25	0.5	0.75	1.0	1.25	1.5	1.75	2.0
-1.75	-	-	-	-	-	2.82	1.3	1	0.83	0.73	0.66	0.60	0.56	0.52	0.50	0.42
-1.5	-	-	-	-	-	2.0	1.26	1	0.85	0.76	0.68	0.63	0.58	0.55	0.53	0.5
-1.25	-	-	-	-	4.0	1.63	1.2	1	0.87	0.78	0.72	0.67	0.62	0.59	0.56	0.53
-1.0	-	-	-	-	2.0	1.41	1.15	1	0.89	0.82	0.76	0.71	0.67	0.63	0.60	0.58
-0.75	-	-	4.0	2.0	1.51	1.26	1.1	1	0.92	0.85	0.8	0.76	0.72	0.68	0.65	0.63
-0.5	2.82	2.0	1.63	1.41	1.27	1.15	1.06	1	0.94	0.89	0.85	0.82	0.78	0.76	0.73	0.71
-0.25	1.33	1.26	1.2	1.15	1.11	1.07	1.03	1	0.97	0.94	0.92	0.89	0.87	0.85	0.83	0.82
0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0

Interaction Analysis

An understanding of multivariable interactions is essential to the process control engineer because these interactions can lead to a complete loss of control. "This can be mystifying to the engineer who is unfamiliar with the concepts of interaction (34)." In this section, the relative gain array which was developed by Bristol (46) and is one of the most widely used measures of interaction (29,34,48,49,53,54,55), will be discussed and applied to multisimulator networks.

Relative Gain Array. The relative gain array (RGA) is used to characterize steady state interaction by measuring the relative strength that manipulated variables have upon controlled variables. This information can then be used to determine the best steady state pairing of variables (34,53) in control loops. However, sometimes significant dynamic interactions occur even in the absence of steady state interactions. Although a great deal of work has been devoted to the subject of dynamic interaction (47,50 ,51,52), a generally accepted standard comparable to the RGA (53) has not been developed. For this reason the following discussions will be limited to the topic of RGA analysis.

Two input, two output RGA (2X2 RGA). Discussions on the RGA are often focused on 2X2 systems (29,34,50,51). The reason for this is:

1. the array may be readily computed without resorting to matrix methods;

2. the array can be expressed in terms of the manipulated or controlled variables.

For these reasons, large systems are often decomposed into sets of 2X2 processes, and the RGAs analyzed individually (34).

Definition of the RGA. The RGA measures the relative effect which each control signal has upon each controlled variable in the system. The RGA is defined such that each element in the array (a_{ij}) is a ratio of an open loop to a relative open loop gain. The open loop gain is simply the process gain. The relative open loop gain is evaluated with all loops in automatic except the one of interest. Thus, the ratio indicates the relative degree of closed loop interaction in a process. A value of 1.0 indicates that there are no closed loop interactions and a value of 0.0 means that there is no connection (steady state) between a controlled and a manipulated variable.

One characteristic of the array is that the sum of the elements in a row or column equals 1.0. If any element is a positive fraction ($0 \leq a_{ij} \leq 1$), then every element is a positive fraction and as one element approaches 1.0 (no interaction), the others in the same row or column approach 0.0 (no effect). The greatest degree of interaction exists when all elements are 0.5 because each manipulated variable has the same relative effect (gain) on all controlled variables and loop pairings can be made arbitrarily. As the degree of interaction changes from 0.5, control loop configurations should be based on the magnitudes of the

elements in the array. Manipulated variables should be paired to the controlled variables so as to maximize the relative gain of each loop.

Determination of the 2X2 RGA. The computations required to evaluate the 2X2 RGA are straightforward because the rows and columns in the array sum to one; only one element in the array is needed to determine the complete array. There are several procedures for evaluating the RGA, but only one provides a physical and directly observable demonstration of its definition. This procedure is also practical because the minimum settling time in a simulator is 18 seconds; this is fast enough to allow a laboratory experiment to be designed around it.

Experimental procedure. Evaluation of the RGA requires one step test (four tests are required to verify that the elements sum to one). All loops are initially in manual (no controllers) and a step is introduced in one control signal (i.e. MV_1). Both process variables respond (possibly) and two open loop steady state gains can be determined. The other manipulated variable (MV_2) is then adjusted (manual control) to restore one of the process variables (PV_2) to its original steady state; the uncontrolled variable (PV_1) responds if there is any interaction. After MV_2 has been adjusted to restore PV_2, and after PV_1 has reached a new steady state, the relative open loop gain may be computed by dividing the new deviation of PV_1 from the original steady state by the magnitude of the step in MV_1. This identifies the first element in the RGA (a_{11}).

The procedure must be repeated three times to generate the complete array. The total experiment (all four tests) demonstrates the definition of the array while providing physical evidence that the elements in the rows and columns sum to one.

The RGA can also be evaluated from the steady state gain matrix of the process using the following expression:

$$a_{11} = 1.0/(1.0 - K_{12}\ K_{21}/(K_{11}\ K_{22})) \qquad [4.25]$$

Two step tests are required to evaluate Equation 4.25 and the necessary information is generated during the experiment described above. Therefore, the results from one experiment can be used to generate and compare two sets of RGAs. Tables 4.23 and 4.24 were prepared to facilitate the design of such experiments.

Tables 4.23 and 4.24 illustrate the range of RGAs that are possible using a two simulator cascade. In compiling the tables, K_{11} and K_{22} in Equation 4.25 were set to unity; this allowed the tables to be generalized in terms of the cross coupling gains. Table 4.23 is for networks with an odd number of positive signs; this yields RGA elements from 0.0 to 1.0. Networks with an even number of positive signs have both negative and positive array elements, all of which fall outside the 0.0 to 1.0 range; these are compiled in Table 4.24.

Control problems result for many of the RGAs shown in Table 4.24. If a_{11} is negative, instabilities result (34), due to the formation of a positive feedback loop through one of the cross coupling terms. For positive values of a_{11}, as the magnitude

Table 4.23

First Element in the RGA When There are an Odd Number of Positive Signs in the Process Gain Matrix ($0 < a_{11} \leq 1$) and $K_{11} = K_{22} = 1.0$

		Thumbwheel Switch Setting							
		0	1	2	3	4	5	6	7
Thumbwheel Switch	Gain	Gain							
		0.25	0.50	0.75	1.00	1.25	1.50	1.75	2.00
9	-1.75	0.69	0.53	0.43	0.36	0.31	0.27	0.24	0.222
10	-1.5	0.72	0.57	0.47	0.40	0.35	0.31	0.27	0.25
11	-1.25	0.76	0.61	0.52	0.44	0.39	0.35	0.31	0.28
12	-1.0	0.80	0.67	0.57	0.50	0.44	0.40	0.36	0.33
13	-0.75	0.84	0.73	0.64	0.57	0.52	0.47	0.43	0.40
14	-0.5	0.89	0.80	0.73	0.67	0.61	0.57	0.53	0.50
15	-0.25	0.94	0.89	0.84	0.80	0.76	0.73	0.69	0.67

Table 4.24

First Element in the RGA When There are an Even Number of Positive Signs in the Process Gain Matrix and $K_{11} = K_{22} = 1.0$

		Thumbwheel Switch Setting							
		0	1	2	3	4	5	6	7
Thumbwheel Switch	Gain	Gain							
		0.25	0.50	0.75	1.00	1.25	1.50	1.75	2.00
0	0.25	1.06	1.14	1.23	1.33	1.45	1.60	1.77	2.00
1	0.50	1.14	1.33	1.60	2.00	2.67	4.00	8.00	∞
2	0.75	1.23	1.60	2.28	4.00	16.0	-8.0	-3.2	-2.0
3	1.0	1.33	2.00	4.00	∞	-4.0	-2.0	-1.3	-1.0
4	1.25	1.45	2.66	16.0	-4.0	-1.77	-1.14	-0.84	-0.67
5	1.5	1.60	4.00	-8.0	-2.0	-1.14	-0.8	-0.62	-0.5
6	1.75	1.78	8.00	-3.2	-1.33	-0.84	-0.62	-0.48	-0.4
7	2.0	2.00	∞	-2.0	-1.0	-0.67	-0.50	-0.40	-0.33

increases, controllability decreases. Shinsky shows that, for processes with similar dynamics, an a_{11} of 2.0 yields closed loop responses which are even worse than the response when a_{11} is 0.5 (the worst case in the 0 to 1 range). As a_{11} gets larger than 5.0, closing both loops degrades responses considerably and conventional closed loop control becomes impractical. Dynamic decoupling is often employed in such cases to alleviate the problems caused by interactions. This topic will be discussed in the next section.

Decoupling with Networks of Four Simulators

In some cases the degree of interaction in a multivariable process is great enough to preclude the use of standard control even with an optimal pairing of variables (34). Under these circumstances, a compensating network is often applied (30) which cancels the interactions by simultaneously adjusting both control valves in response to a change in either control signal. This is known as decoupling and allows each control loop to be treated independently.

Decoupling networks have traditionally been classified as either ideal (56) or simplified (57). Both approaches attempt to diagonalize the process (remove interactions) by installing a computing network between control signals and valves. In ideal decoupling, the primary open loop characteristics of the process are unaffected (56), while, in simplified decoupling, the characteristics are modified. These two techniques will be briefly discussed in the sections to follow.

Ideal decoupling. The computing network for ideal decoupling has the following form:

$$\mathbf{D} = G^{-1}*\text{diag}(G) \qquad [4.26]$$

where G^{-1} is the inverse dynamic matrix of the process and diag(G) is the diagonal of G. Multiplication of D by G diagonalizes or decouples the process because the product of G times G^{-1} is an identity matrix (I); this yields an effective process as given by Equation 4.27.

$$\mathbf{PV} = I*\text{diag}(G)*\mathbf{MV} \qquad [4.27]$$

In algebraic terms (for a 2X2 plant), Equation 4.27 becomes

$$PV_1 = K_{11}*G_{11}*MV_1 \qquad [4.28a]$$

$$PV_2 = K_{22}*G_{22}*MV_2 \qquad [4.28b]$$

Equations 4.28a and 4.28b verify that ideal decoupling removes interactions while leaving the primary dynamics unchanged; thus, the closed loop response of each loop is the same as it would be if the other were in manual.

There are problems associated with ideal decoupling such as the difficulty encountered in computing the inverse dynamic matrix (for instance; a process with all gains equal does not posses an inverse). Another problem is that the decoupler may be expensive to implement because it cannot be formed using off-the-shelf control equipment. Simplified decoupling, on the other hand, has fewer

implementation problems and can sometimes be constructed from standard lead/lag controllers. For these reasons (and others), simplified decoupling is usually the preferred method (34,56) for removing interactions.

Simplified decoupling. The algebraic equations for a P-canonical simplified decoupler are

$$MV_1 = MC_1 + G_{ff_1} MC_2 \qquad [4.29b]$$

$$MV_2 = MC_2 + G_{ff_2} MC_1 \qquad [4.29b]$$

where MV_1 and MV_2 are control valve signals and MC_1 and MC_2 are from feedback controllers. If the process also has a P-canonical structure,

$$PV_1 = G_{mv_1} MV_1 + G_{dv_1} MV_2 \qquad [4.30a]$$

$$PV_2 = G_{mv_2} MV_2 + G_{dv_2} MV_1 \qquad [4.30b]$$

where $G_{mv_1} = K_{11}*G_{11}$, $G_{dv_1} = K_{12}*G_{12}$, etc.; then substitution of Equations 4.29a and b into 4.30a and b and setting the feedforward transfer functions to

$$G_{ff_1} = -G_{dv_1}/G_{mv_1} \qquad [4.31a]$$

$$G_{ff_2} = -G_{dv_2}/G_{mv_2} \qquad [4.31b]$$

yields the decoupled network given in Equations 4.32a and 4.32b.

$$PV_1 = (G_{mv_1} - G_{dv_1} G_{ff_2})*MC_1 \qquad [4.32a]$$

$$PV_2 = (G_{mv_2} - G_{dv_2} G_{ff_1})*MC_2 \qquad [4.32b]$$

Thus the simplified P-canonical decoupler removes interactions because each variable responds to only one control signal, but the open loop dynamics are changed. Even though this form of decoupling does effect the net dynamics of the process it is a more likely form with which to build a demonstration. This will be taken up in the next section.

Decoupling using multiple simulator networks. The behavior of decouplers and decoupled processes can be studied using networks of simulators. However, ideal decoupling networks cannot be studied because the inverse matrix usually cannot be constructed from the processes available in the PPS. Simplified decoupling, on the other hand, can be readily studied because of its similarity to feedforward control and multiple simulator networks have already been shown to be applicable to this mode of control.

Studies of the performance of V-canonical and P-canonical decouplers and processes may be performed using four simulators cascaded together. The interconnections for a P-canonical network are shown in Figure 4.19 and the corresponding block diagram is provided in Figure 4.20. Comparision of Figure 4.19 to Figure 4.9, on page 167 suggest that P-canonical decoupling is essentially two sided feedforward control. The block diagram shows two feedforward loops with the control signal from one acting as a disturbance to the other. Therefore, decoupling demonstrations can be readily set up using the procedure outlined in the section on feedforward control.

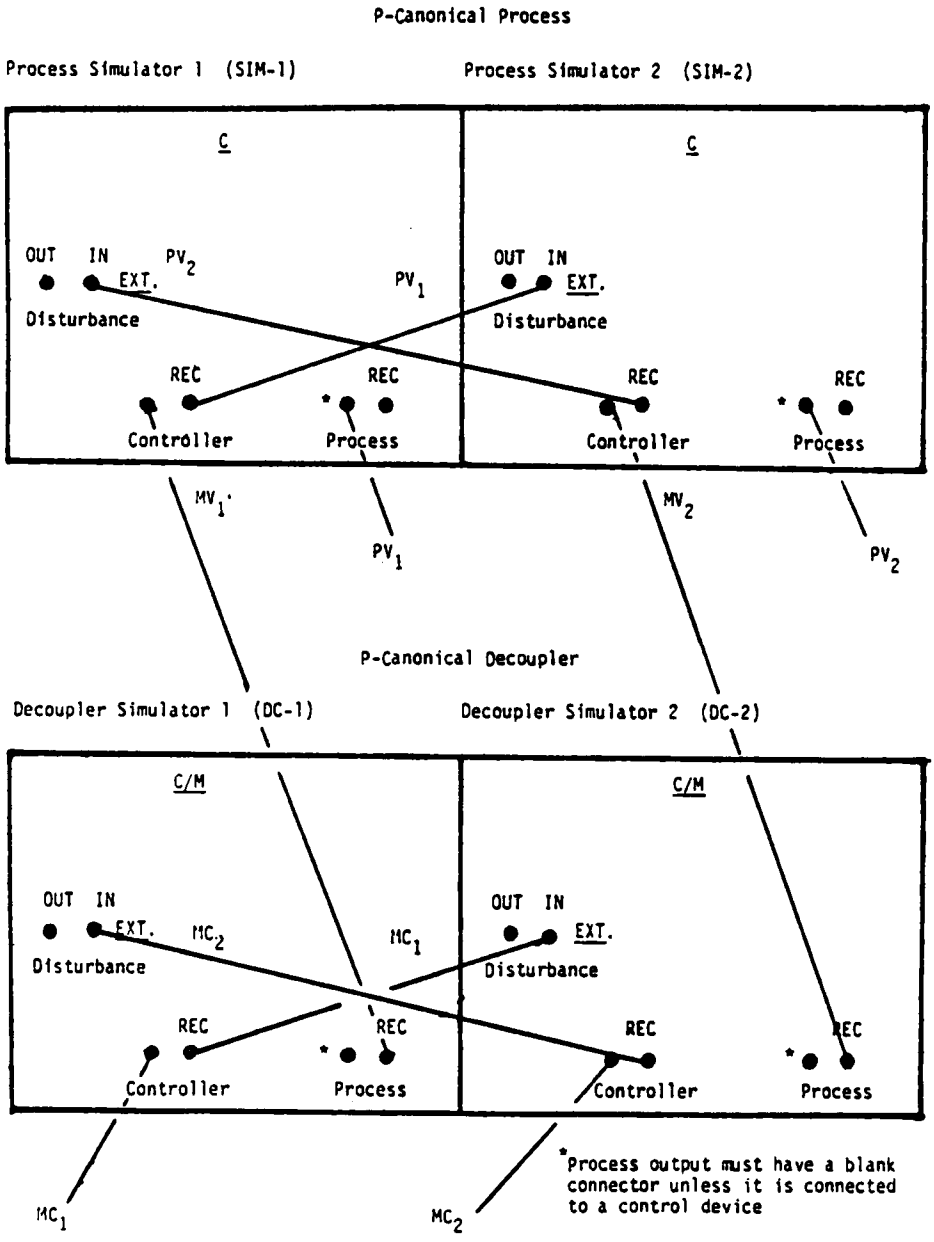


Figure 4.19. Connections required to configure four simulators as a 2X2 process with a decoupler (see Figure 4.20).

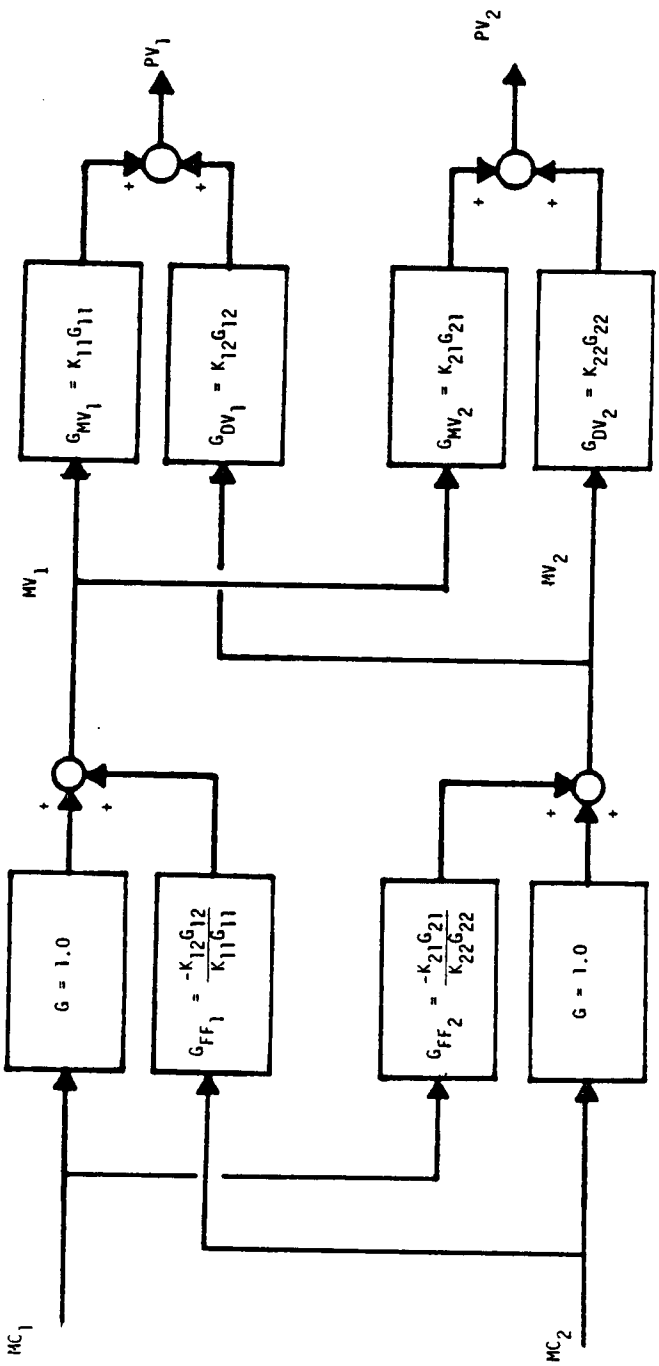


Figure 4.20. Block diagram of a four simulator network with two simulators forming a 2 x 2 process and two simulators acting as a P-Canonical Decoupler.

The P-canonical network is the most popular decoupling compensator (49), but Shinsky (34) discusses two problems that limit its applicability. The first one is the problem of initialization which results from MV_1 and MV_2 being functions of both feedback controllers. The proper initial values of MC_1 and MC_2 must be determined by solving the decoupling equations in reverse, using the known values of control valve settings. Otherwise, the control valves will be bumped when the feedback loops are closed. However, if the decoupler is constructed with two simulators, the initialization problem is not encountered because simulator outputs are based on input changes and not the absolute value of the input. Initialization requires only (a) that the control signals be set and (b) that "OP" be pressed.

The other problem with P-canonical decouplers results from the non-interactive output and involves operation around a constraint. If one control valve reaches an operational limit then only one process variable (PV_1 or PV_2) can be controlled, but the non-interactive structure insures that both controllers will attempt to control by adjusting the unconstrained valve. The result is that both valves eventually become constrained. This can be demonstrated using the network shown in Figure 4.20 and the "bias" thumbwheel switches on two of the simulators. The following procedure illustrates how such a demonstration was implemented for a pure gain network with an RGA of 0.5.

1. First, the four simulator network of Figure 4.19 was constructed.

2. Next, the thumbwheel switch settings for each transfer function block were chosen. The settings shown in Figure 4.21 form a pure gain network with an RGA of 0.5.
3. The manual/control selector switches on DC-1 and DC-2 were placed in the "M" position and the valves were adjusted to 50%.
4. The M-IC button was pressed on each simulator; this initialized the network with PV_1, PV_2 and MV_1 at mid-scale and MV_2 at 100%.
5. Finally the "OP" pushbutton on each simulator was pressed and an attempt was made to move PV_1 to 30% while holding PV_2 at 50%. The result was that eventually MC_1 constrained at 0%, MC_2 reached 100% and neither set point was satisfied.

The demonstration discussed above illustrates the problem of constraints in P-canonical networks. Shinsky suggests that the V-canonical decoupling network shown in Figures 4.22 and 4.23 avoids both the initialization and the constrained operation problems. If the experiment discussed above is performed with that network, then the saturation problem disappears because only MC_1 has any effect on either control variable. However, the V-canonical decoupler exhibits oscillatory behavior because, for the RGA chosen, the decoupler feedback loop has a damping coefficient of 0.707; this results in "ringing" or control valve chatter which is not a desirable behavior. The damping coefficient is related to the RGA as follows:

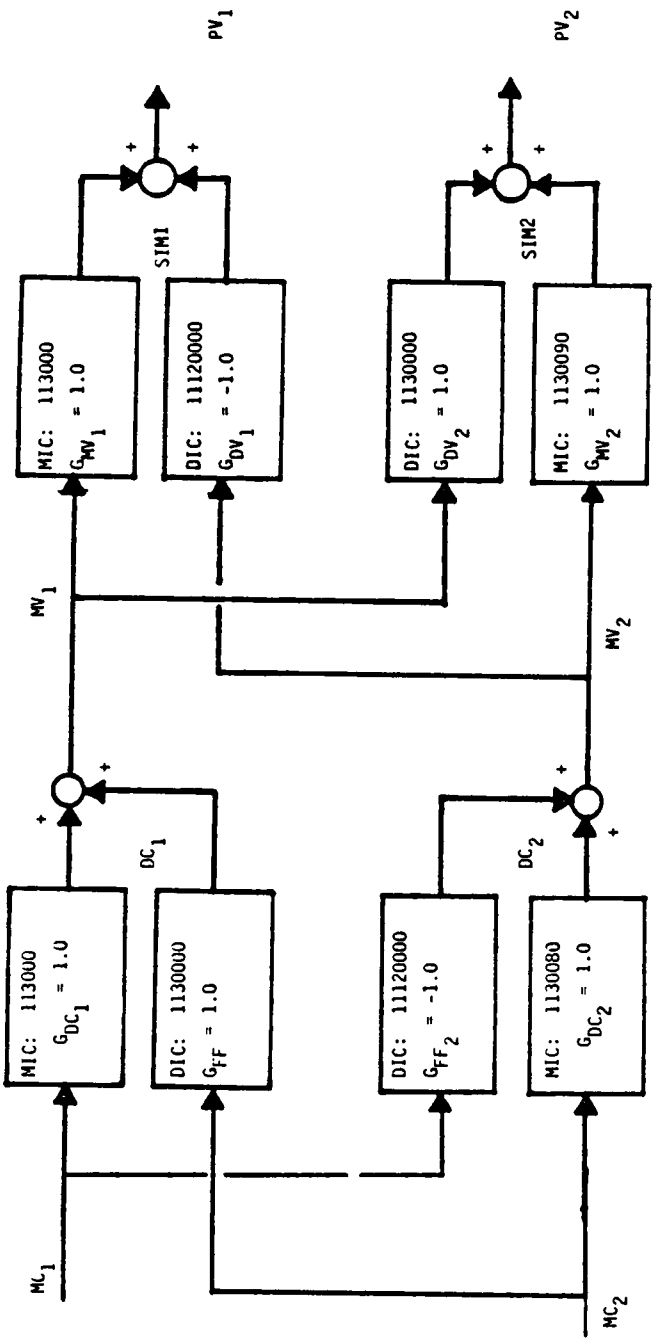


Figure 4.21. Four simulator network demonstrating a 2 x 2 process and its P-Canonical Decoupler when a_{11} in the RGA is 0.5.

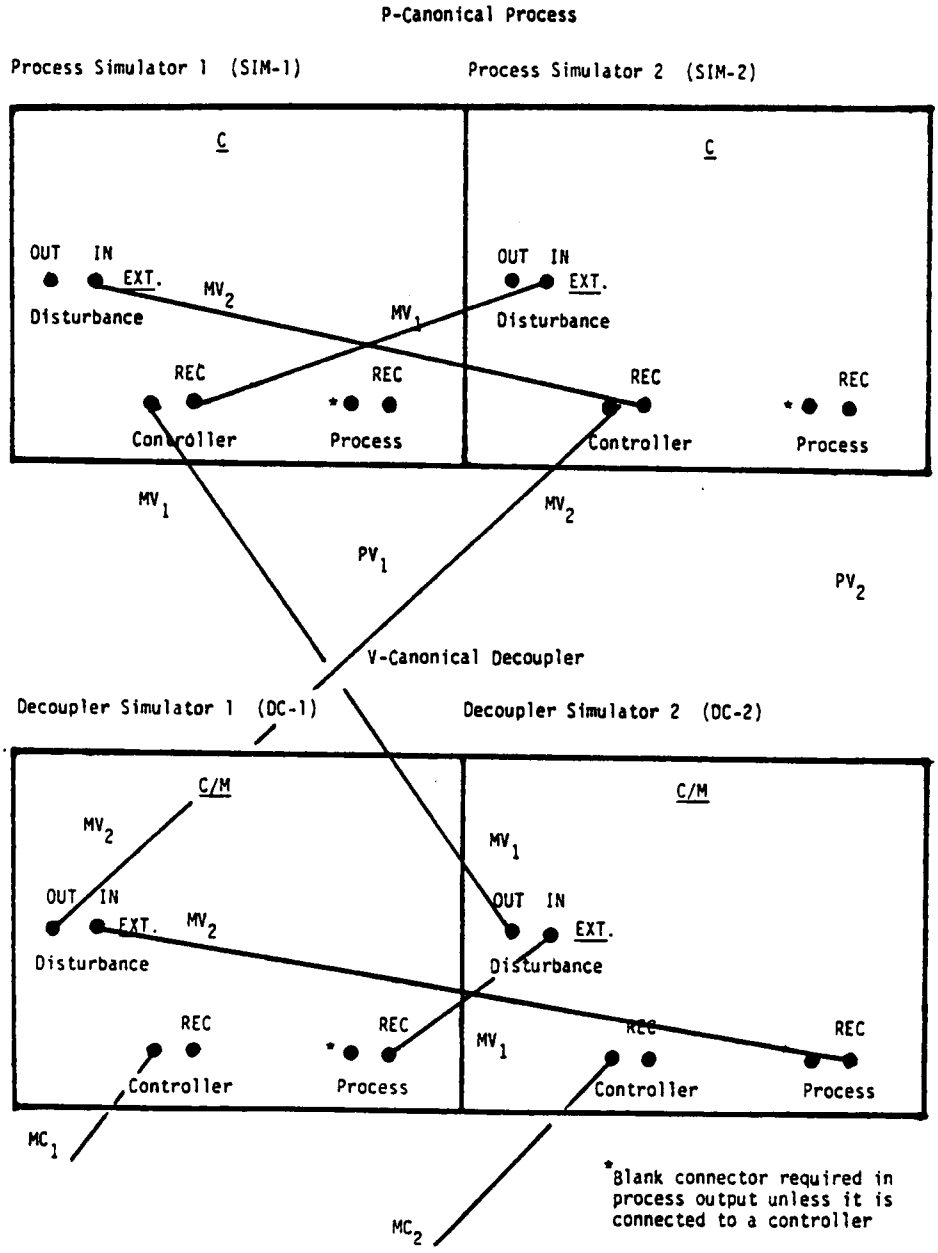


Figure 4.22. Connections to configure four simulators as a 2 x 2 process with a decoupler (see Figure 4.23).

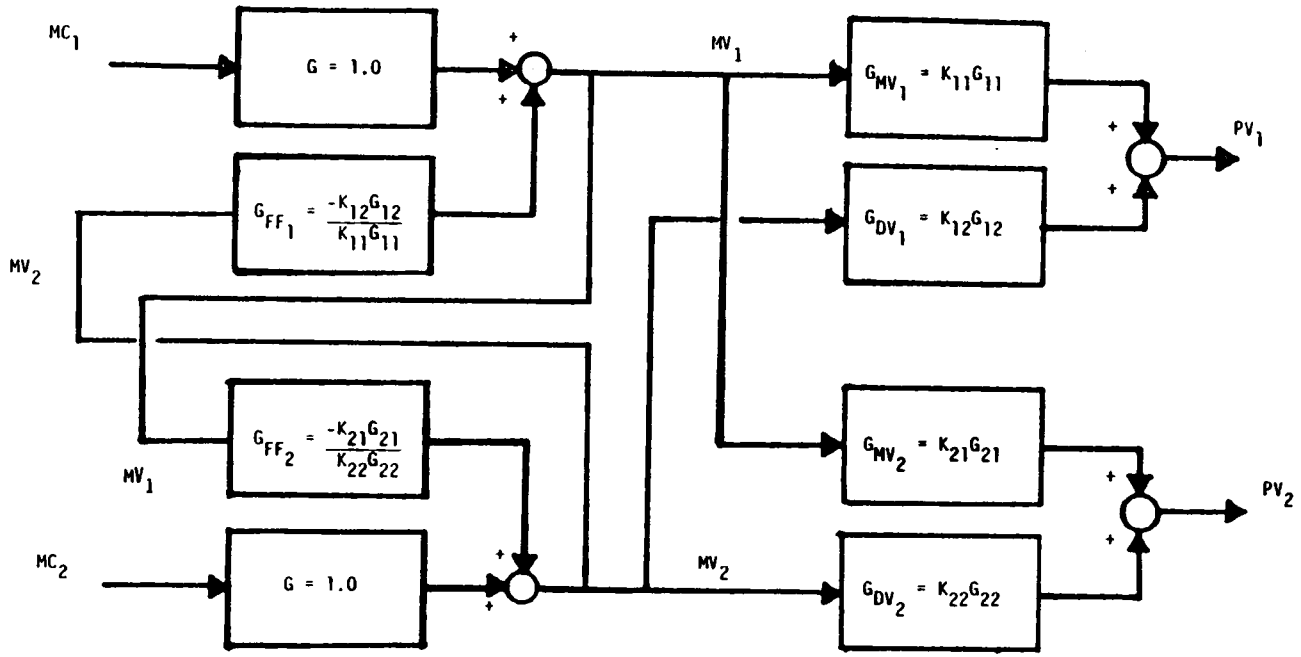


Figure 4.23. Block diagram of a four simulator network with two simulators acting as a 2 x 2 process and two simulators acting as a V-Canonical Decoupler.

$$z = [a_{11}]^{0.5} \qquad [4.33]$$

Equation 4.33 indicates that "ringing" disappears when the value of a_{11} exceeds one because then the feedback loop becomes overdamped.

The V-canonical has certain operational advantages over the P-canonical decoupler and, in spite of its inherent dynamical complexities, this decoupler is preferred to the P-canonical decoupler for some practical applications (57,58). The P-canonical decoupler is, however, the more widely used compensating network although both decouplers fail to provide satisfactory performance under some circumstances. Weischedel and McAvoy (59) have addressed this issue with regard to steady state performance of the decoupled system.

If only the steady state portions of Equations 4.32a and 4.32b are considered, these relationships reduce to the following form:

$$PV_1 = K_{11}/a_{11} \qquad [4.34a]$$

$$PV_2 = K_{22}/a_{11} \qquad [4.34b]$$

As the RGA becomes larger, the open loop gains in the decoupled process decrease, meaning that feedback controller gains must increase to maintain the performance level of the controlled process. If one of the decouplers should fail (for instance if one of the disturbance selector switches is moved to the internal position in the simulator cascade), then poor performance results, due to the tight loop tuning.

The situation described above can be readily demonstrated using the multi-simulator decoupling networks; as an aid to developing such demonstrations, Table 4.25 shows several sets of process gain matrices, the corresponding RGA and decoupled gain matrices. The processes shown in the table can be used to demonstrate the problems encountered when a decoupler fails or is constrained. The following procedure was developed to illustrate the use of a four simulator decoupling network in this area.

1. A four simulator P or V-canonical network was constructed.
2. A process with an appropriate RGA was chosen.
3. Each control loop was tuned separately.
4. One of the decouplers was disconnected by placing the internal/external selector switch on the decoupler's disturbance input in the internal position and a step change in set point was introduced.
5. The resulting process response was observed and recorded.

By performing the experiment described above, the effect of decoupler failures on the closed loop performance can be readily demonstrated by simply changing the RGA and observing the resulting closed loop response.

This concludes the discussions on multivariable control, using cacade networks of simulators. In the final section of this chapter, the topic of model algorithmic control using a three simulator network will be discussed.

Table 4.25

Process, RGA, and Decoupled Gain Matrices
for a 2 x 2 Multivariable Process

System #	Process Gain Matrix		RGA Matrix		Decoupled Gain Matrix	
1	1	1.25	0.35	0.65	2.875	0
	-1.5	1	0.65	0.35	0	2.875
2	1.0	1.0	0.5	0.5	2	0
	-1.0	1.0	0.5	0.5	0	2
3	1.0	0.5	0.8	0.2	1.25	0
	-0.5	1.0	0.2	0.8	0	1.25
4	1.0	-0.5	2	- 1	0.5	0
	-1.0	1.0	- 1	2	0	0.5
5	1.0	-0.5	8	- 7	0.125	0
	-1.75	1.0	- 7	8	0	0.125
6	1.0	-1.25	16	-15	0.0625	0
	-0.75	1.0	-15	16	0	0.0625

4.5 MODEL ALGORITHMIC CONTROL (MAC)

In previous sections on feedforward and decoupling, process models were used to compute control action in such a way as to remove the effect of disturbances, whether external (feedforward) or from other control valves (decoupling). In MAC, a model is used to remove troublesome process characteristics and improve the response of a feedback control loop. In this section, a three simulator network will be shown to be useful for demonstrating this type of control technique.

MAC algorithms. In the late 1950's O. L. Smith (60) developed a single input, single output (SISO) technique for removing dead time from a control loop. Later, Moore (61) developed an equivalent technique called the the analytical predictor for sampled data systems. In recent years, MAC algorithms such as Dynamic Matrix Control (DMC) (62) and Identification and Command (IDCOM) (63) have been developed and applied to multivariable processes. All of these techniques have one thing in common; a process model is used to predict responses to control action. The focus of the discussions to follow will be on using simulator networks for demonstrating SISO predictor algorithms such as the Smith predictor.

SISO MAC--Smith predictor. The Smith predictor algorithm is quite easy to understand and implement. Consider a process as described by Equation 4.35,

$$PV_1 = G_p*MV \qquad [4.35]$$

where G_p is a process transfer function which includes dead time. If the output from a parallel computing network with the following formulation,

$$PV_2 = (G'_{pm} - G_{pm})*MV \qquad [4.36]$$

is summmed with the process signal (PV), the resulting net process is given in Equation 4.37.

$$PV_3 = (G_p + G'_{pm} - G_{pm})*MV \qquad [4.37]$$

When G_{pm} and G_p are equal, Equation 4.37 has the form shown in Equation 4.38.

$$PV_3 = G'_{pm}*MV \qquad [4.38]$$

The net effect of adding the computing network with a perfect model is to remove dead time. If G'_{pm} is equivalent to G_p except for dead time, then the formulation shown in Equation 4.37 is called the Smith predictor algorithm.

In Figure 4.24 is shown the wiring diagram and in Figure 4.25 is the resulting block diagram of a three simulator network that can be used to demonstrate the Smith predictor algorithm. Three simulators allow the system to have a disturbance and lets the process (PV_1), the model (PV_2) and the net process response (PV_3) be observed on Sim-3. In Figures 4.26, 4.27 and 4.28 are shown examples of the types of responses that can be generated using this network. In Figure 4.26, the response of a compensated and uncompensated network to a 20% set point change is shown. Figure 4.27

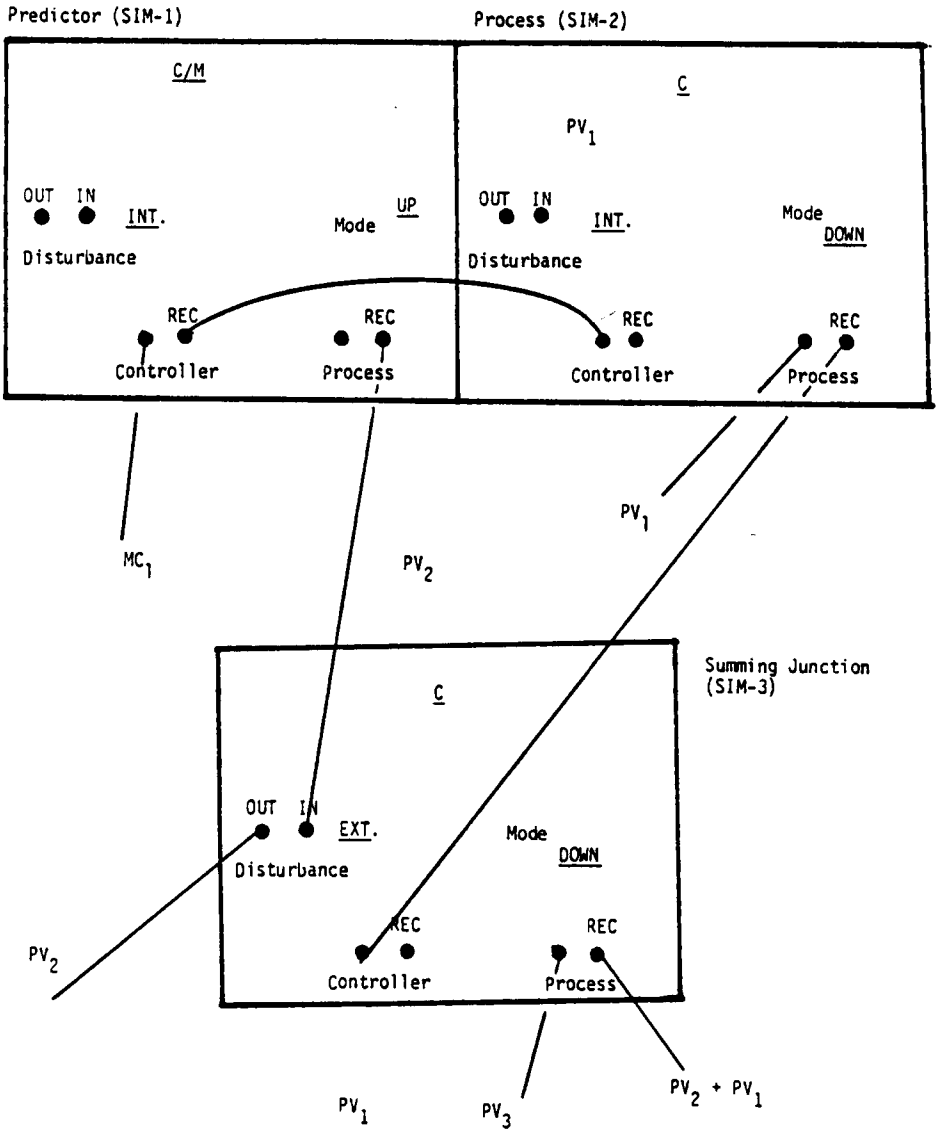


Figure 4.24. Connections required for a three simulator configuration that allows dead time compensation (Smith predictor) to be studied.

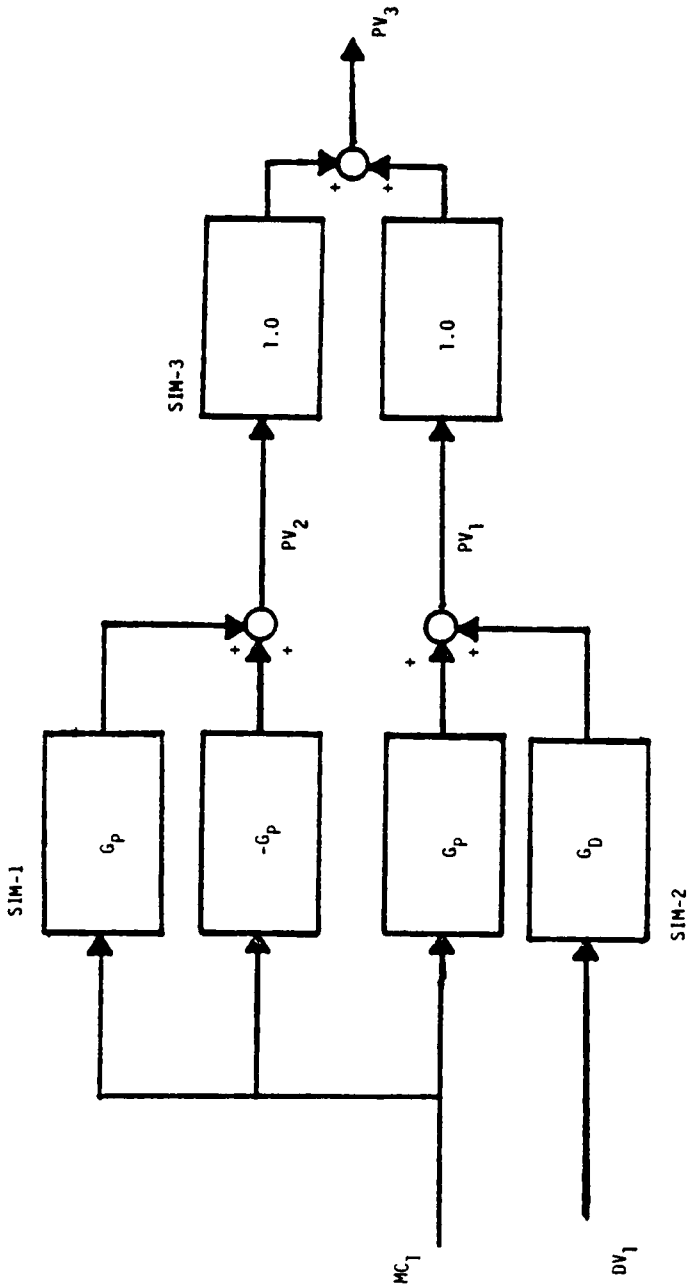
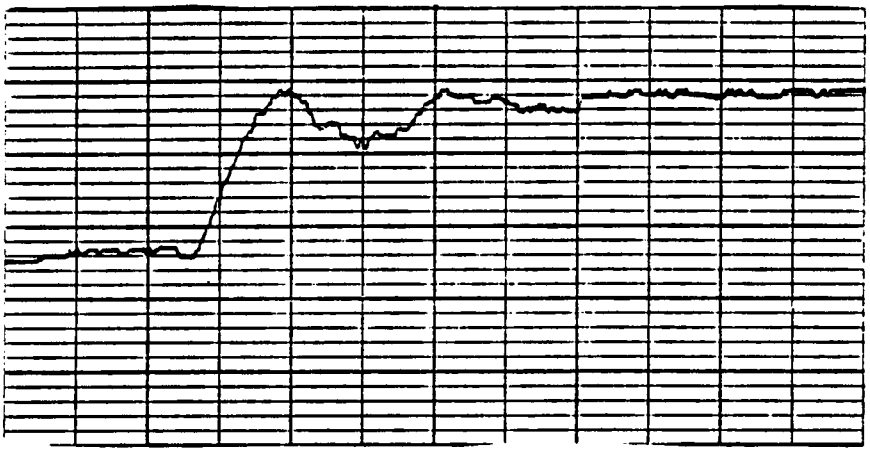
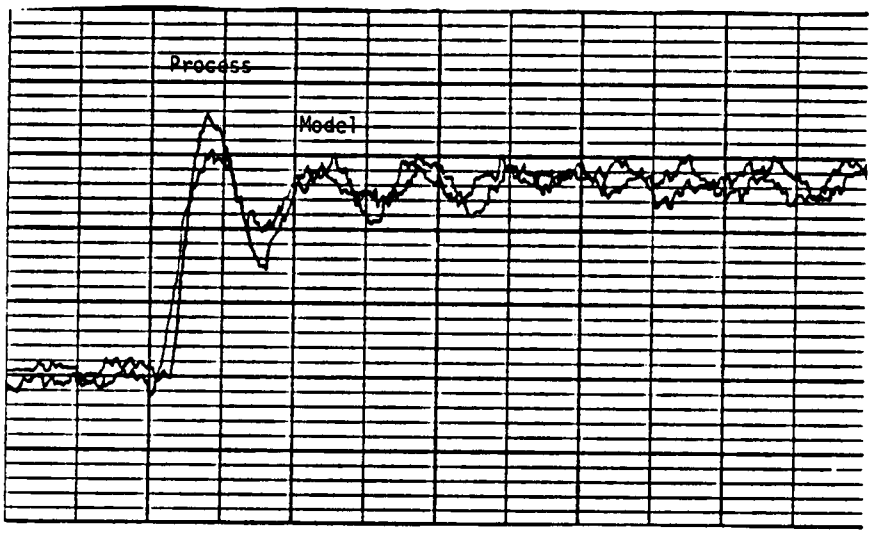


Figure 4.25. Three simulator configuration that has one simulator acting as a Smith predictor, another as the process and the third as a summing junction.

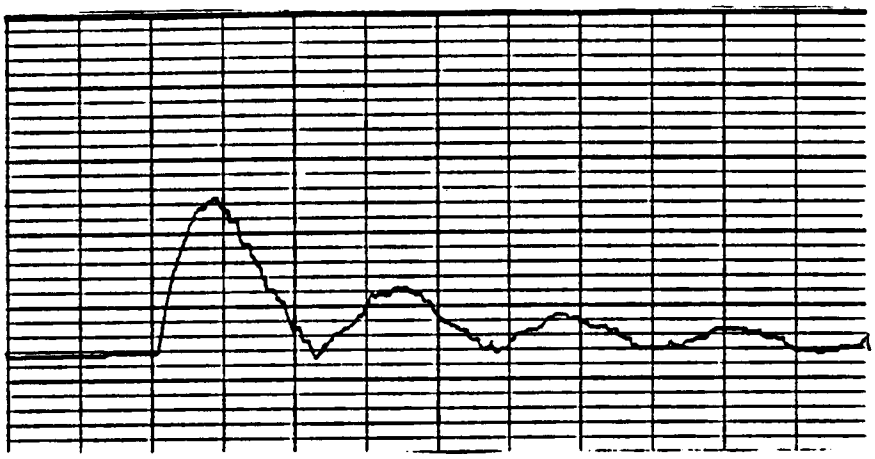


A) without compensator: PI tuning constants $K_c = 0.75$ $\tau_i = 0.38$ min

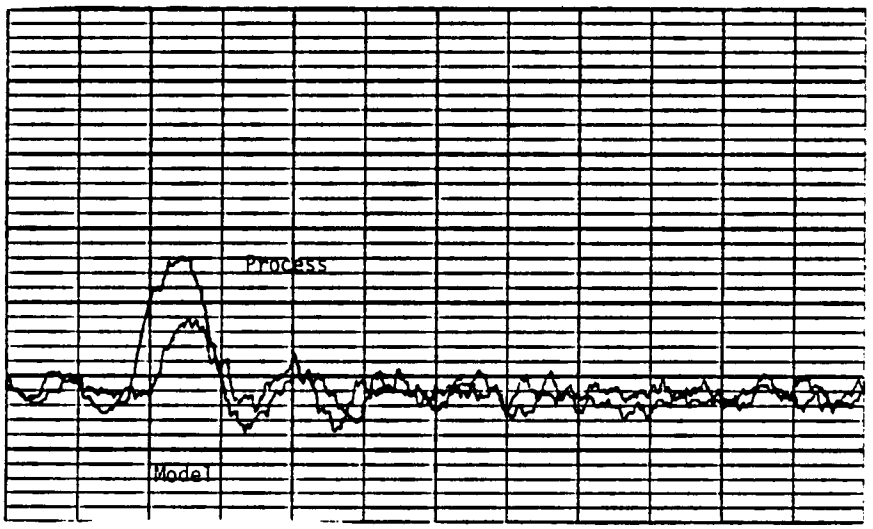


B) with conpensator: PI tuning constants $K_c = 3.0$ $\tau_i = 0.26$

Figure 4.26. Response of a first order lag plus dead time process to a step change in set point: with and without dead time compensator.

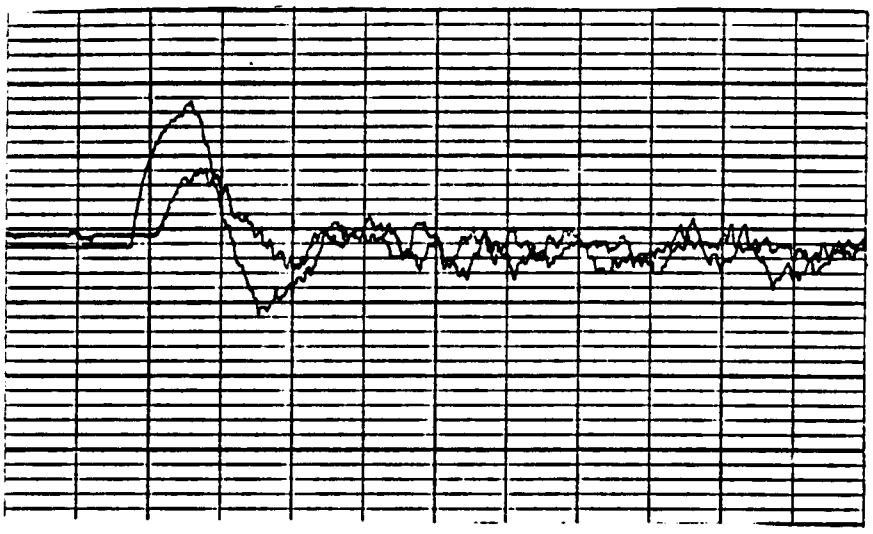


A) uncompensated process

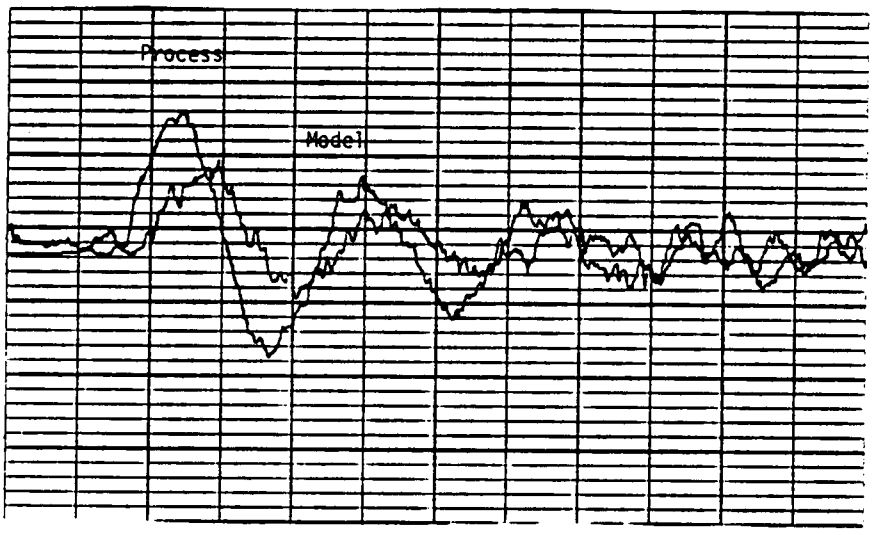


B) compensated process

Figure 4.27. Response of dead time process to 20% load disturbance: with and without dead time compensation.



(A) 30% error in compensator dead time



(B) 50% error in compensator dead time

Figure 4.28. Response of dead time process to a 20% load disturbance when the Smith predictor's dead time is in error by 30% and 50%.

shows the response to a 20% load disturbance and Figure 4.28 gives the response when the dead time is in error by 30% and 50% in the model. In all three cases, the compensated system yields a better response than when compensation was not applied and Figure 4.28 provides an indication of the extent of closed loop response degradation caused by modelling errors.

Dead time compensation is not the only process characteristic that can be removed using the predictor compensator. Iinoya and Altpeter (40) have applied this form of compensation to studies of the inverse process. In their work, only the sag was removed from the process response and this essentially changed the gain of the net process. The result was that the process variable exhibited steady state offset even with PI control. If G_{pm} is instead used to completely counteract G_p, G'_{pm} may be chosen freely in such a way as to optimize the response of PV_1. Thus, considerable improvement over straight PID control can be realized by the proper selection of G'_{pm}.

The three simulator network discussed in this section can readily be used to set up demonstrations of the basics of model algorithmic control. This and the other multi-simulator networks discussed in this chapter were presented as a guide to aid in the design of laboratory experiments.

4.6 CONCLUDING REMARKS

The purpose of this chapter has been to demonstrate uses of the PPS, alone and in tandem with other devices, in process control

studies. Some of the most common topics encountered in elementary and advanced texts have been addressed and the PPS has been shown to be a versatile and powerful tool for demonstrating these topics. In each section of this chapter, an attempt was made to provide an example of an application and to provide enough information to allow similar demonstrations or laboratory experiments. In the next chapter, some general conclusions and recommendations with regard to similar work will be presented.

CHAPTER V

CONCLUSIONS

In this work a dedicated microcomputer based simulation computer has been presented and was shown to be versatile in its operational character and in its applicability. The overall flexibility of the PPS was shown to be the result of blending analog computer features into a digital computer system; the features of the PPS that were adopted from an analog computer were the pushbutton command entry station and the patch panel design of the front panel. It was shown that the result of this hybridization was a simulation computer system that had well over 10^5 different configurations for each of the two operating modes. The pushbutton command format allowed simulations to be configured, initialized and initiated quite readily, while the patch panel design allowed other devices to be connected to the PPS in a straightforward fashion. This feature allowed multiple simulators to be cascaded together to expand the potential configurations to include multivariable networks. The result of the simulator's operating flexibility was an easy to use simulation system that could be readily employed to illustrate many typical process control concepts. The intent of this work has been to provide an understanding of the operation of the simulator and to demonstrate its use as a tool for teaching and illustrating process control.

One of the primary design criteria was low unit cost; this was achieved with the cost of one simulator being approximately 1000

dollars (1984). However, the low cost system led to an 8-bit design and some operational problems resulted from this decision. One of the problems involved the need to develop a 9-bit multiplication routine to prevent overflows in the integer arithmetic; this problem and its solution were discussed in Chapter III. Another problem involved the closed loop behavior when the PPS was connected to a controller; if the proportional gain was large the controller output tended to "chatter". The reason for this behavior lay in the inherent noise level of the system. An 8-bit system has a noise level of at least $\pm$ one bit or $\pm$ 0.4%. When a simulator is connected to a controller with a high gain, this noise level will be amplified through the controller and a great deal of control action will be observed. The solution to this problem would be to increase the word size of the system from 8-bit to 16-bit and increase the net resolution; the inherent noise level would then be decreased to $\pm$ 0.02%.

Upgrading the computer in the system to 16-bits would also modify the programming and data storage requirements. The decision to use the STM method was predicated on the fact that it was particularly suited to implementation in a small machine. Sixteen bit computers are inherently more powerful than eight bit machines and use of such a computer would allow more sophisticated arithmetic, such as fixed or floating point, to be employed. The result would be that a discrete recursive difference equation, such as a Z-transform equation, could be more readily employed. However, upgrading the system to sixteen bits would tend to increase the cost significantly because the computer, the ADCs and the DACs would all be more costly.

The tradeoff is thus between increased resolution versus cost. Use of the 8-bit version in the classroom has shown it to be adequate for demonstrating the fundamental concepts without resorting to more expensive systems. As the price of higher resolution equipment decreases it is expected that the improved computing power and accuracy will dictate basing future versions of the PPS upon sixteen or thirty-two bit machines.

BIBLIOGRAPHY

BIBLIOGRAPHY

1. Basta, Nicholas, "Equipment Wanted for Process Control Studies," Chemical Engineering, 45-47, (March 1982).

2. Worthy, W., "Chemical Education Falls on Hard Times," Chemical Engineering News, 45-49, (Feb. 1982).

3. Reklaitis, G. V., R. S. H. MAH and T. F. Edgar, "Computer Graphics in the ChE Curriculum," ASEE/NSF Position Paper, Cache Corporation, Salt Lake City, Utah, (1983).

4. Rony, Peter R., Joe D. Wright and T. F. Edgar, "Cache Microcomputer Task Force Survey on Microcomputers and Personal Computers in American and Canadian Departments of Chemical Engineering," Cache Corporation, Salt Lake City, Utah, (1984).

5. Luyben, W. L., Process Modeling, Simulation, and Control for Chemical Engineers, McGraw-Hill, New York (1973).

6. Coughanowr, D. R. and L. B. Koppel, Process Systems Analysis and Control, McGraw-Hill, New York (1965).

7. ABD-EL-BARY, M. F. and Sriram Chari, "Digital Computer Application in Process Control," Chemical Engineering Education, 16, 3, 118-121, (Summer 1982).

8. Blum, Joseph J., Introduction to Analog Computation, Harcourt, Brace and World, Inc., New York (1969).

9. Joseph, Babu and David D. Elliott, "A Microcomputer Based Laboratory for Teaching Computer Process Control," Chemical Engineering Education, 18, 3, 136-139, (Summer 1984).

10. Stevens, William F., "The Use of Microcomputers as a Teaching/Learning Aid - Applications to the Teaching of Undergraduate Process Dynamics and Control," source unknown.

11. Heist, R. H., H. Saltsburg, T. Olsen and F. U. Arcuri, "The Microcomputer in Chemical Engineering," 74th Annual AIChE Conference, New Orleans, La., (1981).

12. Yeow, Y. L., "Pulse Testing with a Microcomputer," Chemical Engineering Education, 18, 2, 78-80, (Spring 1984).

13. Kulkarni, Sheila, Armando Corripio and Authur M. Sterling, "A Laboratory Control System for Two Heat Exchangers in Series," source unknown.

14. Kershenbaum, L. S., S. D. Perkins, and D. L. Pyle, "Systems Modelling and Control," Chemical Engineering Education, 14, 1, 18-24, (Winter 1980).

15. Mellenchamp, D. A., "A Full Year Course Sequence in Real-Time Computing," Chemical Engineering Education, 14, 1, 18-24, (Winter 1980).

16. Deshponde, Pradeep B., W. L. S. Laughuf and Nandkishar G. Patke, "Advanced Process Control," Chemical Engineering Education, 14, 1, 26-31, (Winter 1980).

17. Morari, M. and W. H. Ray, "The Integration of Real-Time Computing into Process Control Teaching - The Graduate Course," Chemical Engineering Education, 11, 4, 160-162, (Fall 1979).

18. Morari, M. and W. H. Ray, "The Integration of Real-Time Computing into Process Control Teaching - The Undergraduate Course," Chemical Engineering Education , 14, 1, 32-36, (Winter 1980).

19. King, F. G., "A Flexible Self-Paced Course in Process Control," Chemical Engineering Education, 12, 3, 120-124, (Spring 1979).

20. Hittner, Phillip M. and D. B. Greenburg, "We Can Do Process Simulation: UCAN-II," Chemical Engineering Education, 14, 3, 138-148, (Summer 1980).

21. Durbin, L. D., "A Smart Simulator for Control Education," source unknown.

22. Piel, Gerard et. al., Editor, Microelectronics - A Scientific American Book, W. H. Freeman and Company, San Francisco, California, (1977).

23. Cutler, C. R., "Dynamic Matrix Control of Unbalanced Systems," ISA Trans., 21, 1, 1-5, (1982).

24. Moore, B. C., "State Space Theory for a Small Computer Control Laboratory," System Control Report No. 8005, University of Toronto, Toronto, Canada, (June 1980).

25. Garcia, Carlos E. and Manfred Morari, "Internal Model Control - A Unifying Review and Some New Results," IEC Process Design and Development, (March 1982).

26. 8080 Assembly Language Programming Manual - No. 98-004C, Rev. C, 55-56, Intel Corporation, (1976).

27. Korn, G. A., Microprocessors and Small Digital Computers for Engineers and Scientist, McGraw-Hill, New York, (1977).

28. Weber T. W., An Introduction to Process Dynamics and Control, John Wiley and Sons, New York (1973).

29. Stephanopoulis, G., Chemical Engineering Control - An Introduction to Theory and Practice, Prentice-Hall, Inc., Englewood Cliffs, NJ (1984).

30. Cohen, G. H. and G. A. Coon, "Theoretical Investigations of Retarded Control," Trans. ASME, 75, 827 (1953).

31. Miller, J. A., et. al, "A Comparision of Controller Tuning Techniques," Control Engineering, 14, 12, 72 (1967).

32. Smith, C. L., Digital Computer Process Control, International Textbook Company, Scranton, PA (1972).

33. Moore, C. F., C. L. Smith, and P. W. Murrill, "Simplifying Digital Control Dynamics for Controller Tuning and Hardware Lag Effects," Instrument Practice, 26, 1, 45 (1969).

34. Shinsky, F. G., Process Control Systems, McGraw-Hill, New York (1979).

35. D'Azzo, J. J. and C. H. Houpis, Linear Control System Analysis and Control, McGraw-Hill, New York (1975).

36. Moores, H. M., "Stand-alone Process Controllers Offer Full Features in Small Packages," Control Engineering, 92, (April 1984).

37. Kraus, T. W. and T. J. Myron, "Self-Tuning PID Controller Uses Patern Recognition Approach," Control Engineering, 106, (June 1984).

38. Nazmul, M., "Process Design in Process Control Education," Chemical Engineering Education, 18, 3, (1984).

39. Ziegler, J. G. and N. B. Nichols, "Optimal Settings for Automatic Controllers, Trans. ASME, 64, 11, 759 (1942).

40. Iinoya, K. and R. J. Alpeter, "Inverse Response in Process Control," Ind. Eng. Chem., 54, 39 (1962).

41. Waller, K. U. T. and G. H. Nygardos, "On the Inverse Response In Chemical Engineering," Ind. Engr. Chem. Fund., 14, 221 (1975).

42. Luenbuger, D. G., Introduction to Dynamic Systems, John Wiley & Sons, New York, 44 - 47 (1979).

43. Yuwana, M. and D. E. Seborg, "A New Method for On-Line Controller Tuning," AIChE Journal, 28, 3, 434 (1982).

44. Lopez A. M., et. al., "Controller Tuning Relationships Based on Integral Tuning Criteria," Instrumentation Technology, 14, 11, 57 (1967).

45. Murrill, P. W., Automatic Control of Processes, International Textbook Co., Scranton, PA. (1967).

46. Bristol E. H., "On a New Measure of Interaction for Multivariable Process Control," IEEE Trans. Auto Control, AC-11, 133 (1966).

47. Mesarovic, M. D., The Control of Multivariable Systems, John Wiley & Sons, New York (1960).

48. Smith C. R. III, "Multivariable Process Control Using Singular Value Decomposition," Ph.D. Dissertation, University of Tennessee, Knoxville, Tenn. (1981).

49. Keeton, J. M., "Multivariable Control of a Binary Distillation Column", M.S. Thesis, University of Tennessee, Knoxville, Tenn. (1981).

50. Rijnsdorp, J. E., "Interaction on Two-Variable Control Systems for Distillation Column-I," Automatica, 1, 15-18 (1965).

51. Rijnsdorp, J. E., "Interaction on Two-Variable Control Systems for Distillation Column-II," Automatica, 1, 25-29, (1965).

52. Mitchell, D. S., and C. R. Webb, "A Study of Interaction Analysis in Multiloop Control," Proc. 1st. IFAC Congress, Moscow, 342-352 (1960).

53. Ray, H. R., Advanced Process Control, McGraw-Hill, New York (1981).

54. Downs, J. J., "The Control of Azeotropic Distillation Columns", Ph.D. Dissertation, University of Tennessee, Knoxville, Tenn. (1982).

55. Lloyd, S. G., "Basic Concepts of Multivariable Control," Instrumentation Technology, 31-37 (December 1973).

56. Zalkind, C. S., "Practical Approach to Non-interacting Control," Instr. Control Systems, 40, 89, 111 (1967).

57. Greenfield, G. G., and T. J. Ward, "Structural and Terminal Analysis of Multivariable Process Control," IEC Fund., 6, 564-571 (1967).

58. Changlai, Y. S., and T. J. Ward, "Decoupler Control of a Distillation Column," AIChe J., 18, 225-227 (1972).

59. Weischedel, K., and T. J. McAvoy, "Feasibility of Decoupling Distillation Composition Control Loops," 86th Annual AIChe Meeting (1979).

60. Smith, O. J., "Close Control of Loops with Dead Time," Chemical Engineering Progress, 53, 5, 217-219 (1967).

61. Moore, C. F., "Improved Algorithm for Direct Digital Control," Instruments and Control Systems, 43, 1, 70-74 (1970).

62. Cutler, C. R., and B. L. Ramaker, "Dynamic Matrix Control - A Computer Control Algorithm," 86th Annual AIChe Meeting (April 1979)

63. Richlet, J. A., A. Rault, J. L. Testud, and J. Papon, "Model Algorithmic Control of Industrial Processes,", Digital Computer Applications to Process Control, North-Holland Publishing Co. (1977).

64. Chen, Chi-Tsong, Introduction to Linear System Theory, Holt, Rinehart and Winston, Inc., New York, 333, (1970).

65. Forsythe, G. E., M. A. Malcolm, C. B. Moler, Computer Methods for Mathematical Computations, Prentice-Hall, Inc., Englewood Cliffs, N. J., 240, (1977).

VITA

John Allen Arnold was born April 2, 1954 in Memphis, Tennessee. Before graduating from Melrose High School in Memphis, Tennessee in 1972, he attended Charjean Elementary School, East High School, and Central High School (all of Memphis). He entered the University of Tennessee at Knoxville in September of 1972 and worked seven quarters as a cooperative engineering student with Tennessee Eastman Company in Kingsport, Tennessee. He obtained his Bachelor of Science degree in chemical engineering with honors in March of 1978. He continued his education in the same field at the same university and received his Master of Science degree in March of 1982 and his doctorial degree in June of 1985. He became a member of Sigma Xi while enrolled in graduate school.