8-1-1978

Design of a Microprocessor Based Real Time Digital Controller

Venkatachalam Garimella

DESIGN OF A MICROPROCESSOR BASED REAL TIME DIGITAL CONTROLLER

BY
Venkatachalam Garimella

Doctor of Philosophy, Ohio University, 1977

A Thesis

Submitted to the Graduate Faculty

of the

University of North Dakota

in partial fulfillment of the requirements

for the degree of

Master of Science

Grand Forks, North Dakota

August
1978

This Thesis submitted by Venkatachalam Garimella in partial fulfillment of the requirements for the Degree of Master of Science from the University of North Dakota is hereby approved by the Faculty Advisory Committee under whom the work has been done.

William A. Baras
(Chairman)

John D. Olson

Bsrshajini Rao

William Johnson
Dean of the Graduate School

Permission

Title: DESIGN OF A MICROPROCESSOR BASED REAL TIME DIGITAL CONTROLLER

Department: ELECTRICAL ENGINEERING

Degree: MASTER OF SCIENCE

In presenting this Thesis in partial fulfillment of the requirements for a graduate degree from the University of North Dakota, I agree that the Library of this University shall make it freely available for inspection. I further agree that permission for extensive copying for scholarly purposes may be granted by the professor who supervised my thesis work or, in his absence, by the Chairman of the Department or the Dean of the Graduate School. It is understood that any copying or publication or other use of this thesis, or part thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to the University of North Dakota in any scholarly use which may be made of any material in my thesis.

Signature: Lin L. Lam

Date: May 19, 1978

ACKNOWLEDGEMENT

The author is deeply indepted to his advisor, Dr. William A. Bares for not only suggesting this problem but also for his constant help and encouragement during this work. Also the author would like to thank Mr. Mark E. Holle for many fruitful discussions and Mrs. Beverly Winchester for typing this Thesis.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

With the availability of microcomputers it is now possible to design low cost digital controllers. In the present work the design of a microprocessor based real time digital controller is presented. The microcomputer used was an 8080 based PROMPT 80. The necessary hardware to interface this computer with a 'plant' was designed and tested. The software for the controller was written such that it can be operated in Proportional, Proportional-Integral and Proportional-Integral-Difference modes. With minor changes in the programs the controller can be operated in Integral, Integral-Difference and Proportional-Difference modes. The necessary modifications are indicated. The controller performance was tested using a second order plant simulator. The results obtained were compared with the calculated values and it was found that the controller was performing as expected. A complete listing of programs and instructions to operate the controller are given.

CHAPTER I

INTRODUCTION

Automatic control systems are finding increasing use in the present day technology. The range of applications which require control systems are large and increasing with advances in science and technology. Thus it is necessary to continually develop new systems and improve the old systems as the demand for automatic control systems increases. Furthermore, as new components are developed, it may be possible to design systems which were previously discarded either as not feasible or too expensive. For example, with the availability of microcomputers it is now possible to design relatively inexpensive automatic digital control systems. In the present work a microprocessor based real time digital controller was developed. Even though all control problems are not the same, certain common general principles do exist and hence it is possible to develop a reasonably versatile system which can be used in different applications.

The basic objective of a control system is to 'control' the process variable in a prescribed manner by the actuating signal [1,2]. For example, consider a controller in the home heating system. The overall function of the controller is to maintain the temperature at the value set by the user. For this, the 'controller' senses the room temperature (process variable PV), compares it with the required temperature (set point SP) and generates the necessary signal (actuating signal MV) to turn on or turn off the heater. Another example of a control system is the human being himself. Suppose a person is trying to reach for an object, the movement of the hand is controlled by the brain. Here the brain is the controller and provides the activating signals to the muscles of the arm and hand.

Control systems can be divided into two broad categories as open loop and closed loop systems. Fig. 1a and 1b are block diagrams of open loop and

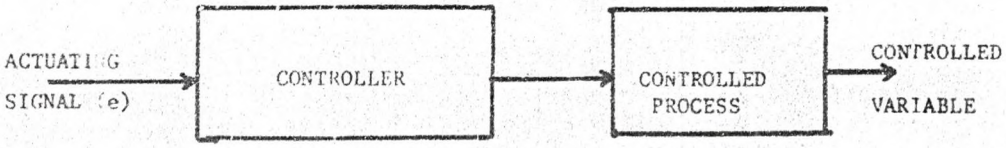


Fig. 1a

BLOCK DIAGRAM OF OPEN LOOP CONTROL SYSTEM

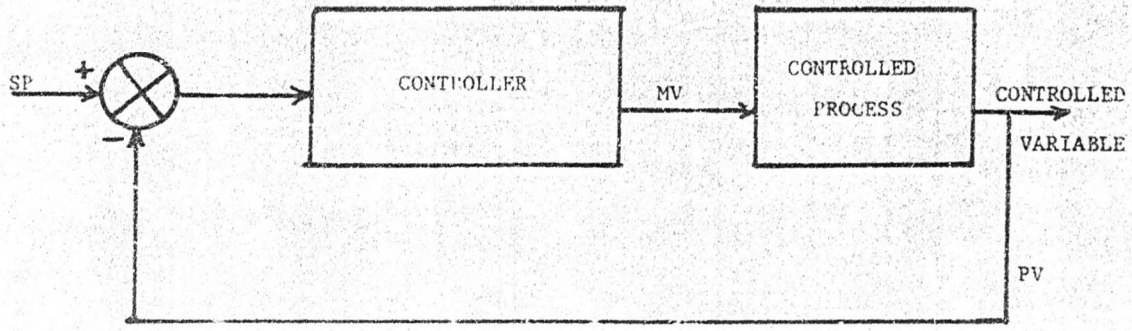


Fig. 1b

BLOCK DIAGRAM OF CLOSED LOOP CONTROL SYSTEM

closed loop control systems respectively. For an open loop control system, the actuating signal is independent of the controlled variable. In the second example, given above, if the person tried to reach for the object with eyes closed, the 'system' becomes an open loop system. However, if he reaches for the object with his eyes open the distance between the object and the hand is 'fed back' to the brain and the actuating signal to the arm and hand thus depends on the controlled variable, the distance between the hand and the object. This is an example of a closed loop system. Open loop control systems are used only in a few cases. In general most of the control systems are of the closed loop type.

DEFINITION OF TERMS

Before proceeding further some of the terms used in control systems will be defined. Fig. 2 is a block diagram of a closed loop control system. (If this looks very similar to fig. 1b the similarity is not a coincidence).

A plant can be as simple as a furnace or as complicated as a large industrial processing plant. Ogata [2] defines a plant as "..piece of equipment, perhaps just a set of machine parts, functioning together, the purpose of which is to perform a particular operation....". A process variable is the output of a plant, for example, position of a shaft. The feedback element produces some quantity (say voltage) usable by the error detector in some way related to the process variable. Set point is the desired value of the process variable. Error is the difference between the process variable and the set point. The controller is a device or devices interconnected such that, it operates on the error signal to generate the necessary signal to control the 'plant'. This signal is called the manipulated variable.

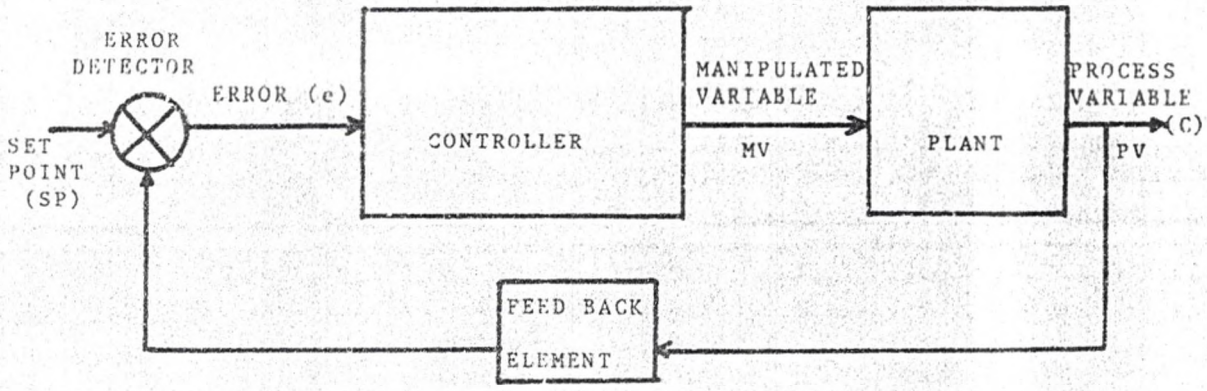


Fig. 2

BLOCK DIAGRAM OF A CLOSED LOOP CONTROL SYSTEM

CLASSIFICATION OF CONTROLLERS

Most common automatic controllers can be classified according to their control action as [2]:

(1) Two position or ON-OFF controllers

(2) Proportional (P) controllers

(3) Integral (I) controllers

(4) Proportional-Integral (PI) controllers

(5) Proportional-Derivative (PD) controllers

(6) Proportional-Integral-Derivative (PID) controllers

In a two position control system the actuating element has two positions, generally, ON or OFF. [Even though such a system can easily be implemented, in the present work it has not been done because it can be implemented with less complex controllers]. In a proportional mode (Proportional controller) the manipulated variable, MV, is related to the error, e, by

$$MV(t) = K_p\, e(t) \tag{1}$$

where K_p is a constant called proportional gain constant.

In a controller with integral control action, the manipulated variable is changed at a rate proportional to the error signal i.e

$$\frac{dMV(t)}{dt} = K_I\, e(t) \tag{2a}$$

or $$MV(t) = K_I \int_0^t e(t)\, dt \tag{2b}$$

K_I is called Integral gain constant.

For a derivative controller

$$MV(t) = K_D \frac{de(t)}{dt} \tag{3}$$

K_D is the derivative gain constant.

For a Proportional-Derivative controller

$$MV(t) = K_p\, e(t) + K_D \frac{de(t)}{dt} \tag{4}$$

and for Proportional-Integral-Derivative controller

$$MV(t) = K_p\, e(t) + K_D \frac{de(t)}{dt} + K_I \int_o^t e(t)\, dt \tag{5}$$

[Since differentiation in a digital system is achieved by taking the difference between two samples, in this thesis the derivative mode is referred to as 'Difference mode" and K_D as the difference gain constant].

ANALOG VERSUS DIGITAL CONTROLLERS

It is possible to use both analog circuits and digital circuits to achieve the desired control action. Analog controllers represent variables by continuous physical quantities whereas digital systems operate on discrete numbers. Analog controllers can operate satisfactorily in situations where there is no decision making. Since they use continuous quantities there are no errors introduced by the controller itself, but a digital system (discrete values) does introduce errors. However, it is not a very severe restriction since this can be reduced to an acceptable limit by increasing the number of digits used in computations. The advantages of a digital controller are that they can perform complex computations very accurately at high speeds and they are less susceptable to drift and temperature variations compared to an analog controller. Digital controllers are very versatile, simply by changing the software, the operation being performed can be changed. For example in this present system, the mode of operation can be changed from Proportional to PID mode by changing one number as opposed to complete rewiring of the analog controller.

In the past one of the main objections to a digital controller has been their high cost compared to analog controllers and hence, when digital computers were used for controlling plants, one large central computer was used to perform most of the tasks. With the availability of relatively inexpensive microprocessors and advances in design and manufacture of Analog to Digital and Digital to Analog converters, it is now possible to design digital controllers at a cost comparable to that of analog controllers. Furthermore, with the large scale integration techniques, the power consumption and physical size of integrated circuits is rapidly decreasing. The microcomputer can function independently or under the control of a central computer. It is possible to delegate some of the responsibility to the microcomputer based controllers, thus freeing the central computer for other uses.

The main objective of the present study is to design a versatile, microprocessor based real time digital controller. This is a real time controller since the digital computations of the manipulated variable change almost instantaneously when the process variable changes. The design of the controller (both software and the hardware) is discussed in chapter II. The performance of the controller was tested using a second order plant simulator and the results obtained are discussed in chapter III. A complete listing of the hand assembled programs is given in appendix I. Step by Step instructions to operate the controller are given in appendix II.

CHAPTER II

CONTROLLER DESIGN

It was indicated in the previous chapter that the present work deals with the design of a microprocessor based digital controller. In this chapter the design of the hardware and software of the controller will be presented. First the major components of the controller will be described. Secondly a description of the hardware will be given and the third part will deal with the software. Finally a second order plant simulator, which was used to test the performance of the controller, will be discussed.

The controller consists of an Analog to Digital (A/D) converter, Digital to Analog converter (D/A) a microcomputer and various other components to interface the controller to the simulator and vice versa. In the present context the word "hardware" implies the circuitry to achieve the interfacing of the microcomputer with the plant simulator. Since the microcomputer chosen dictates the choice of all the components that comprise the controller, it will be discussed first.

MICROCOMPUTER

The microcomputer used in this work is an 8080 microprocessor based 'PROMPT 80', manufactured by the Intel Corporation. This microcomputer was chosen for several important reasons. The PROMPT 80 is readily available and hence will enable one to proceed with the design of the controller. It has a "Monitor" Program which is sufficiently versatile, a feature very convenient for developing and testing the software. The 8080, an 8 bit processor, used by the PROMPT 80 is a versatile microprocessor with a reasonably good instruction set. [3,4]. Fig. 3 shows the general organization of this computer.

The most important component of the computer is its central processing

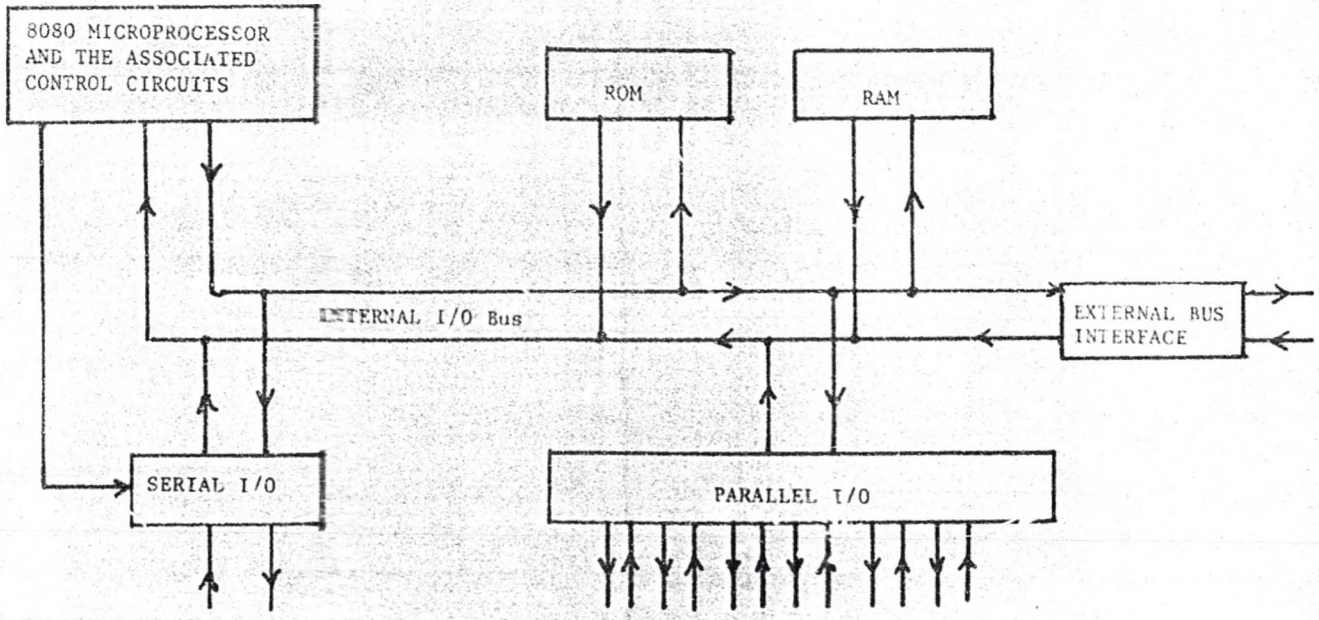


Fig. 3

ORGANIZATION OF PROMPT 80

unit (8080). The central processing unit (CPU) communicates with memory and the Input/Output interfaces.

The program counter is 16 bits long, hence up to 65k bytes of memory can be addressed. (1k Byte is 1024 locations and a byte consists of 8 binary bits). PROMPT 80 has 4k read only memory (ROM) and 1k Read/Write Memory (RAM). Of the 4k ROM, 3k is used by the Monitor programs. The 1k RAM is available for the user programs. There are serial and parallel Input/Output (I/O) Ports. The serial I/O Port was used with a Teletype.

The Parallel I/O functions are implemented using Programable Peripheral Interfaces (PPI). There are two PPI's (PPI1 and PPI2), of which one (PPI1) is used by the machine and not accessable by the user. The other PPI (PPI2) is connected to a 50 pin connector. Using PPI2 as Parallel (I/O) Ports one can establish communication between the computer and the 'outside world'. The PPI2 consists of 3 Parallel Ports, of 8 bits each, called E8, E9 and EA. These can be configured, via Software, as shown in Table 1. (The details of how to achieve a particular configuration will be discussed later). It was determined that defining E8 as an Output Port, E9 as an Input Port and bits 7-4 of Port EA as Output and 3-0 as Input is most convenient for the present work (Configuration 4).

Another important feature of the PROMPT 80 is the Monitor Program. This program is in ROM (hence permanent). It is possible to enter data and instructions using a hexadecimal key board. Besides the 8080 instruction set one can use several functions such as reading a teletype tape, etc. Also using the Monitor various register contents can be displayed on seven segment displays located on the PROMPT 80 front panel. A partial list of some of the useful instructions (Particular to PROMPT 80) are given below:

TABLE 1

CONFIGURATION	PORT E8	PORT E9	PORT EA	
	Bits 7-0	Bits 7-0	Bits 7-4	Bits 3-0
1	OUTPUT	OUTPUT	OUTPUT	OUTPUT
2	OUTPUT	OUTPUT	OUTPUT	INPUT
3	OUTPUT	INPUT	OUTPUT	OUTPUT
4	OUTPUT	INPUT	OUTPUT	INPUT
5	OUTPUT	STROBED OUTPUT	Bits 2,1,0 are STROBES	
6	OUTPUT	STROBED INPUT		

(a) DISPLAY/MODIFY MEMORY

(b) SINGLE STEP

(c) PREVIOUS/CLEAR ENTRY

(d) EXECUTE/END

(e) NEXT

(f) SCROLL REGISTER DISPLAY

(g) GO

(h) EXAMINE/MODIFY REGISTER

(i) SYSTEM RESET

(j) MONITOR INTERRUPT

(k) USER INTERRUPT

(l) CREATE HEXADECIMAL TAPE

(m) READ TAPE FROM TTY

Some of these instructions and their usage will be discussed when describing how to use this system. (See appendix II)

The 8080 CPU can accept an external interrupt. When an interrupt occurs, the Monitor Program determines the source of interrupt (System reset, Monitor interrupt, Keyboard interrupt or User interrupt) and depending on the type of interrupt, the program counter contents will be modified. For example, when a user interrupt occurs the program counter will be changed to RAM hex location 3C02 and the instructions from that location onwards will be executed. The utility of this feature comes from the fact that if the CPU is not required to continuously service a particular unit (i.e in non-dedicated applications) it is possible to use the time between requests for performing other functions (background mode). In this design the background programs were written such that, normally data from a buffer will be typed on a teletype. When an user

interrupt occurs the I/O Ports will be serviced and the typing will resume. Thus between data points the CPU will be performing other useful functions. [One can replace the typing routine with any other routine. In fact for testing purpose a DO NOTHING loop was used instead of the typing routine]. There is a protection against spurious pulses triggering the interrupt. This is achieved by means of a circuit which, after an interrupt occurs, will wait for approximately 5 milli-seconds and check to see if the interrupt request is still there. Thus triggering by noise etc. can be avoided. This feature limits the frequency of interrupts. However, for the present application this does not pose a serious problem.

Even though an attempt has been made to discuss some of the important features of the microcomputer, one should recognize that this is by no means a complete or adequate description. A more detailed discussion of the 8080 microprocessor in general and the PROMPT 80 in particular can be found in the references [3,4,5].

ANALOG TO DIGITAL CONVERTER

The analog to digital converter (A/D) used is a Monolithic CMOS 8700 A/D manufactured by the Teledyne Semiconductor Corporation. This is an 8 bit A/D converter and is available as a single, self contained 24 pin dual inline package. It required only a few passive support components, voltage or current reference and power supplies. Being a single IC package, compactness, high stability and low drift are some of its major advantages. The Analog to Digital conversion is performed by incremental charge balancing technique. An amplifier integrates the sum of the unknown analog current and pulses of reference current and the number of pulses needed to maintain the amplifier summing point near zero volts is counted. [6] This count is then latched into an 8 bit binary word. By adjusting R_{in} (see fig. 5) one can vary the voltage required to get an output of FF. It can be operated in

strobed or free running modes. With a maximum of approximately 800 conversions/sec (free running) it is more than adequate for the present application. As mentioned in the previous section since the PROMPT 80 cannot be interrupted more than once in approximately 5 milli-seconds the A/D was used in strobed mode. A frequency of 100 Hz was chosen for this purpose. It is an 8 bit A/D and hence directly compatible with the PROMPT 80. Last but not the least reason for choosing this A/D is its relatively low cost.

DIGITAL TO ANALOG CONVERTER

The Digital to Analog (D/A) Converter used is a DATEL DAC - 98BIR. The complete D/A system was developed in this department previously. Detailed drawings and circuit description are available in the department [7]. There are two input latches which are connected to the D/A. The input latches are enabled by the presence of a logical '1' on STR. SEL input on the D/A circuit. When the STR. SEL makes a transistion to '0' the data is latched and transferred to the D/A converter. The main reasons for using this D/A converter are it is an 8 bit D/A and readily available.

INTERFACING AND CONTROL CIRCUIT DESCRIPTION

The complete circuit diagram and block diagram of the interfacing and controlling circuits are shown in Figs. 4,5. One element consists of a 100 Hz oscillator made with a 555 timing circuit. The frequency of this oscillator and hence the data sample rate can be varied to suit the particular requirements. However, since the PROMPT 80 can not accept interrupt pulses faster than once in 5 milli-seconds there is an upper limit of 200 Hz on the frequency of this oscillator. There is no lower limit imposed by the controller as such. Some such limitation will come from the 'plant'. The output of the oscillator provides the initiate conversion pulses to the A/D. At the end of conversion the A/D generates an 'END OF CONVERSION' signal which in turn triggers a monostable which generates an 'INTERRUPT' signal. This signal

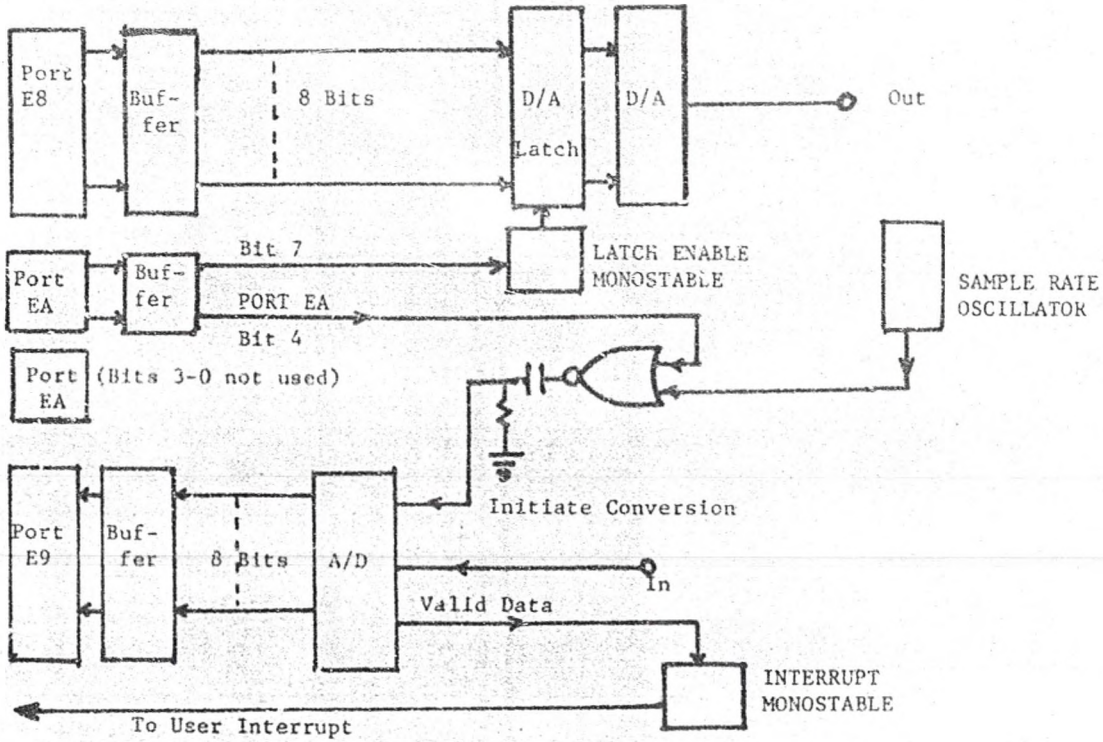


Fig. 4

BLOCK DIAGRAM OF INTERFACE AND CONTROL CIRCUIT

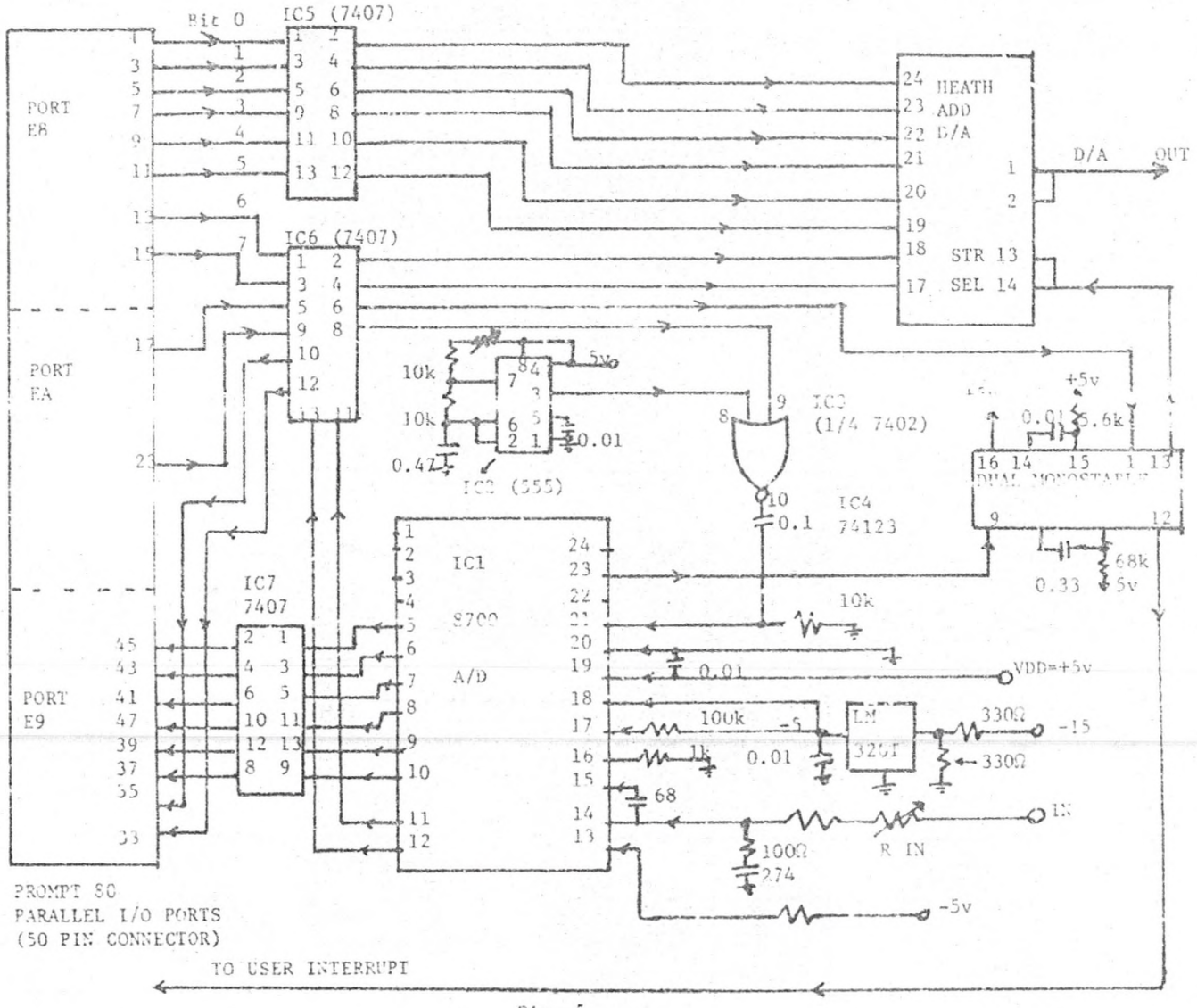


Fig. 5
CIRCUIT DIAGRAM OF INTERFACE AND CONTROL CIRCUIT

(transition from 5v to 0v and 5 milli-seconds wide) shorts the 'USR INTE' switch thus creating a condition which is similar to pressing the 'USR INTE' switch. At this point PROMPT 80 Monitor sets the program counter to 3C02. The 'Interrupt Routine' starts from this location. Hence execution of the interrupt routine will be initiated. When the data processing is complete the output appears as an 8 bit number via Port E8. A negative going pulse will be generated, using the software, and appears as bit 4 of Port EA. This pulse triggers the 'LATCH ENABLE' monostable and the data will be latched into the D/A converter.

Even though the oscillator is free running the initiate conversion pulses are controlled by the computer by means of a 'NOR' gate. One input to this 'NOR' gate comes from bit 4 of Port EA. When System Reset is pressed, due to a special feature of the PROMPT 80, this bit goes high thus terminating the initiate conversion pulses. This control will be released by lowering the bit 4 of Port EA which is done using software. The main reason for incorporating this feature is that when changes in constants or program etc. are desired the interrupt pulses must be stopped. To avoid excessive loading of the computer Input/Output gates, non-inverting buffers (7407's) were used. [It was mentioned earlier that the PPI2 was configured as Port E8 for Output, Port E9 for Input, bits 7-4 of Port EA as Outputs and bits 3-0 of Port EA as Inputs. A 7409 (Quad dual input 'AND' gates) integrated circuit was installed in socket A9 of the PROMPT 80 by the author. This change has been shown in the circuit diagrams of PROMPT 80 reference manuals 7,5. Depending on the configuration of the ports used, it may be necessary to change the gates A7 through A11 in the PROMPT 80. The necessary modifications are listed in the PROMPT 80 'HARDWARE' manual [5]].

Fig. 6 is an approximate timing diagram of the interfacing and control

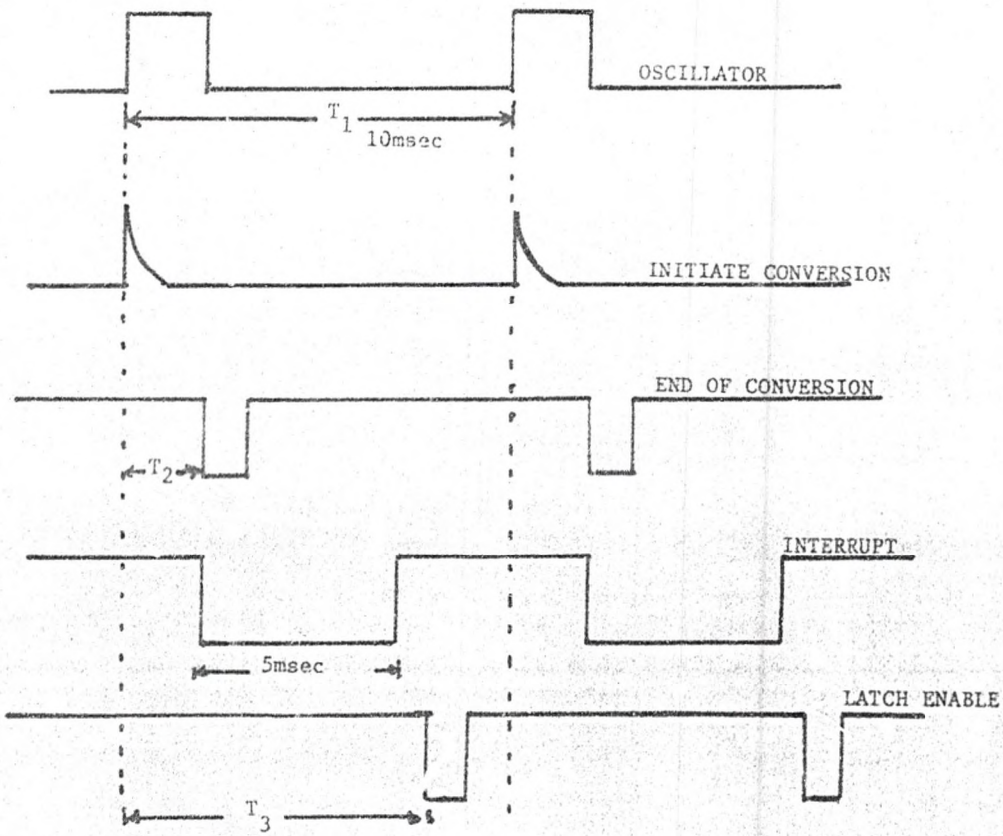


Fig. 6

TIMING DIAGRAM

[This figure shows only the sequence of events but not their exact timing relation. See text for explanation].

circuit. The reason for calling it an approximate timing diagram will be clear shortly. The oscillator frequency is 100 Hz. Thus T_1 = 10msec. 'INITIATE CONVERSION' pulse occurs at the rising edge of the oscillator pulse. The A/D, after digitizing the input generates the 'END OF CONVERSION' pulse. T_2, the time between the 'INITIATE CONVERSION' to 'END OF CONVERSION' depends on the magnitude of the input. This is one reason why the timing diagram of fig. 6 is approximate. The 'END OF CONVERSION' pulse triggers the interrupt monostable to generate 'INTERRUPT' signal. This is 5msec wide to meet the requirement of the PROMPT 80 as discussed previously. Once the Computer has been interrupted it will manipulate the data and the final results will be output via port E8 and the latch enable monostable will be triggered. Again the time, T_3, between 'INITIATE CONVERSION' to 'LATCH ENABLE' Signal depends on the magnitude of the input and the Controller mode of operation. Thus this diagram gives only the sequence of events occuring rather than exact time relation between them.

SOFTWARE

Software for the controller is written to be flexible as far as possible. This feature will be clear as the description is fully presented. The routines were divided into individual 'modules'. This is particularly important to follow the programs and debug them. Even though, as presented here it may not be possible to operate each program individually, with a few modifications they can be tested separately. As the description continues some such modifications will be suggested.

First a general flow chart will be presented and then individual programs will be discussed in detail using flow charts. Actual (assembled) programs will be presented in appendix I. Also an example of how to operate the system will be presented in appendix II.

The major routines written are listed below. They are:

(1) Start Up routine

(2) Interrupt routine

(3) Difference routine

(4) Integration routine

(5) Proportional routine

(6) Multiply and Complement routine

(7) Summing routine

(8) Output Set Point routine

(9) Typing routine

Before discussing the general flow chart, it is necessary to identify several one and two type 'registers'. These are memory locations reserved for the explicit use of various routines. They are listed below and the memory locations reserved for them are shown in parenthesis (these are hexadecimal addresses).

(1)	SET POINT	(3CE1)	One byte
(2)	K_D	(3CE2)	One byte (Difference gain)
(3)	K_I	(3CE3)	One byte (Integral gain)
(4)	K_p	(3CE4)	One byte (Proportional gain)
(5)	MC	(3CE5)	One byte (Mode code)
(6)	TEMP. STORE	(3CE6)	One byte (Temporary Data Storage)
(7)	PREVIOUS DATA	(3CE7)	One byte (Data of Immediate Sample)
(8)	DIFF OUT	(3CE8,3CE9)	Two bytes (Difference routine output)
(9)	INTG OUT	(3CEA,3CEB)	Two bytes (Integration routine output)
(10)	PROP OUT	(3CEC,3CED)	Two bytes (Proportional routine output)
(11)	DIFF CONTR	(3CEE)	One byte (Difference counter)
(12)	OUT BUFF	(3CEF,3CF0)	Two bytes (Output buffer)

(13) BUFF CONTR (3D00) One byte (Data buffer length counter)

(14) BUFF PTR (3D01,3D02) Two bytes (Data buffer pointer)

(15) DATA BUFF (3D03,3D67) 100 bytes (Data buffer).

These locations i.e. 3CE0 to 3CF0 and 3D00 to 3D67 are reserved for the use of all the routines and cannot be changed without modifying most or all the programs.

SOFTWARE ORGANIZATION

Fig. 7 is a flow chart of the Software organization. When starting the System for the first time or if the program has to be stopped to enter/modify various constants and restarted, the 'Start Up' routine must be executed. This enables the operator to initialize various buffers, counter, pointers and PPI's. It is assumed that before executing the Start Up routine all the necessary constants (SET POINT, K_D,K_I,K_p,MC, etc.) have been loaded into the respective registers. This can be done using the hexadecimal key board.

The Start Up routine after initializing all the necessary registers etc. will enter the background mode (e.g. Typing routine). Thus, between interrupts the computer executes the background mode programs. For the purposes of testing and explanation, the Typing routine has been replaced by a DO NOTHING loop (an IDLE mode). In this mode the program will enter an unending loop. This can be changed by altering a 'JUMP' address. Details of how to do this and the actual Start Up routine will be discussed later.

When an interrupt occurs, all the working registers will be stored in the stack by the interrupt routine. [The 'STACK' is memory locations 3F50 to 3F90 reserved, in this program, for storing the register contents etc.]. The output of A/D converter will be read via input port E9 and stored in 'TEMP STORE'. If the data buffer (DATA BUFF) is not full, data will be stored in it also.

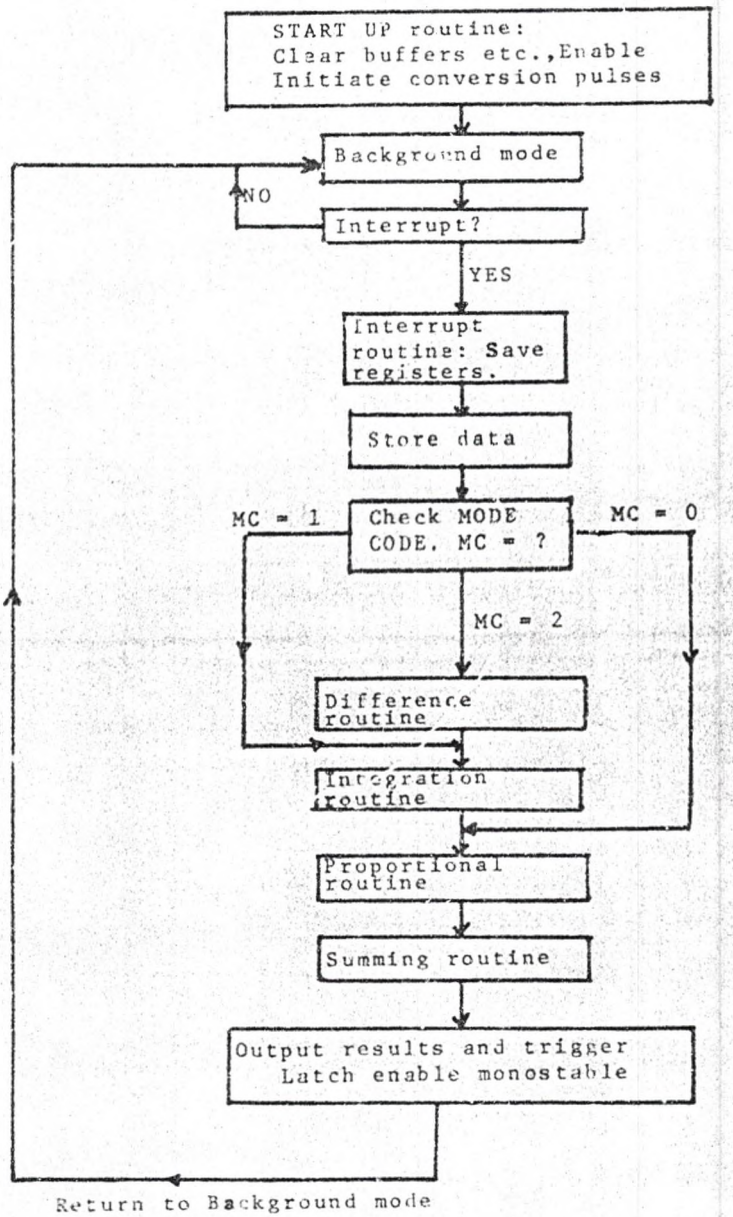


Fig. 7

SOFTWARE ORGANIZATION

The Mode code (MC) will be checked to determine the mode of operation and the program will exit to the appropriate routine. The Proportional, Integral and Difference modes were written such that they will store their results in a corresponding 16 bit output buffer (PROP OUT etc.). When the calculations are complete, the Summing routine will add the results from the three buffers and store the sum in OUT BUFF, the output buffer. The Output routine (part of the interrupt routine) will output the high order 8 bits of the OUT BUFF to Port E8, (which is connected to the D/A) and triggers the 'LATCH ENABLE' monostable, next all the working registers and program counter will be loaded from the Stack and the background mode continues.

Another important feature of the software organization is the Mode code. The Mode code (MC) will have one of three values 0, 1 and 2. The user has to enter the Mode code before executing the Start Up routine. The Controller will operate in Proportional (P) mode if MC = 0, Proportional-Integral mode (PI) if MC = 1 and Proportional-Integral-Difference mode (PID) if MC = 2. This is achieved by cascading the Difference, Integral and Proportional routines, i.e., if first the program enters the Difference routine, then on completing this, it will enter the Integral routine and then execute the Proportional routine. Thus if MC = 0 only the Proportional routine will be executed and with MC = 1 first Integral and then Proportional routine will be executed for PI mode of operation. Finally for MC =2 the Difference, Integral and Proportional modes will be executed in that order giving PID mode of operation. It is possible to execute the Integration routine alone and the necessary modification of this routine will be suggested later.

START UP ROUTINE

As the name implies this routine has to be executed to start the system for the first time or to restart after stopping the system using the system reset (SYS. RST) switch. As mentioned before the SYS. RST does not conserve the stack or save the register contents. The purpose of the Start Up routine is to clear various buffers, initiatize PPI's and to enable the 'INITIATE CONVERSION' pulses. The Start Up routine, starting and ending addresses are 3DCA and 3DFD respectively.

Fig. 8 is the flow chart of the Start Up routine. First the difference counter (DIFF CONTR), 3CDD, is cleared. [The Difference routine gives output once every N Samples for reasons discussed in the DIFFERENCE ROUTINE (P). The Difference counter counts the number of samples and is reset to zero after N samples.].

The Buffer length counter, BUFF CONTR location (3D00), is cleared to enable storage of data in the data buffer (See INTERRUPT ROUTINE). Next PREVIOUS DATA (3CE7). DIFF OUT (3CE8,3CD9), INTG OUT (3CEA,3CEB), PROP OUT (3CEC,3CED) and OUT BUFF (3CEF,3CF0) registers are cleared. Of these the INTG OUT register needs particular attention. The INTG OUT register will be required in PI or PID modes. When the system is at zero initial condition and one wishes to "move" it to a new set point the INTG OUT register must be cleared. The reason for this is, in the Integration routine new results are added to the previous ones where as in Proportional and Difference routines the respective register contents will be replaced with new results. One situation where the INTG OUT must not be cleared is when the SET POINT has to be changed. In this case if the INTG OUT is not cleared the change from one set point to the other will be 'smooth'. On the other hand if the INTG OUT is cleared the system will first return to zero and then

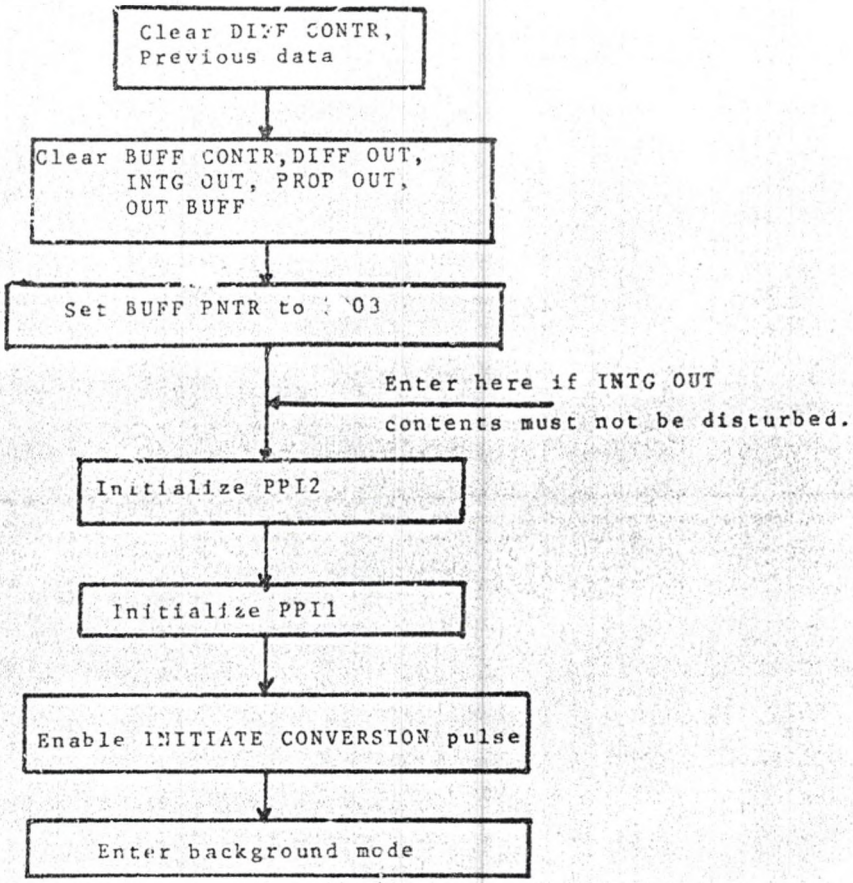


Fig. 8

FLOW CHART FOR START UP ROUTINE

move to the new set point. If the smoother transistion is required the Start Up routine has to be entered at 3DEB (See appendix II). Normally the Start Up routine will be executed from 3DCA.

Another function of the Start Up routine is to initialize PPI's. To configure PPI2 such that port E8 is Output port, E9 Input port and bits 7-4 of port EA Outputs, control word 83_H must be loaded in port EB. [4]

In PROMPT SO, bits 7-4 of Port EA are gated off. To enable these lines the following steps must be followed. First initialize PPI1 [Output 70_H to Port E6] and then output FD_H to Port E6. This enables the bits 7-4 of Port EA and they can be used as output bits.

The 'INITIATE CONVERSION' pulses will be enabled by outputting 80_H to port EA. This will put a 'low' on bit 4 of port EA. This is one input to the 'NOR' gate and since the other input is oscillator pulses, when bit 4 goes low the Initiate Conversion pulse will occur. In the actual program (see appendix I) the order in which various quantities are output to the PPI's may seem different. To clarify this, part of the program is shown below.

	ADDRESS	DATA OR INSTRUCTION	MNEMONIC	COMMENT
(1)	3DEF	3E 70	MVI A, 70_H	;LOAD A REGISTER WITH 70_H
(2)	3DF1	D3 E6	OUT E6	;OUTPUT 70_H TO PORT E6
				;THIS INITIALIZES PPI1.
(3)	3DF3	3E 80	MVI A, 80_H	;LOAD A REGISTER WITH 80_H
(4)	3DF5	D3 EA	OUT EA	;OUTPUT 80_H TO PORT EA
(5)	3DF7	3E FD	MVI A, FD_H	;LOAD FD_H TO A REGISTER
(6)	3DF9	D3 E6	OUT E6	;OUTPUT FD_H TO PORT E6

In the above, the instructions (1) and (2) initialize PPI1. 80_H will be output to port EA by executing (3) and (4). However, Pin (27) on the 50 pin connector will still be high. Finally executing the instructions (5) and (6) will enable the 'INITIATE CONVERSION' pulses.

This completes the tasks of the Start Up routine and the program can now enter background mode. The last instruction of the Start Up routine is shown below.

LABEL	ADDRESS	INSTRUCTION	MNEMONIC	COMMENT
LOOP	3DFB	3C FB 3D	JMP LOOP	JUMP TO LOOP

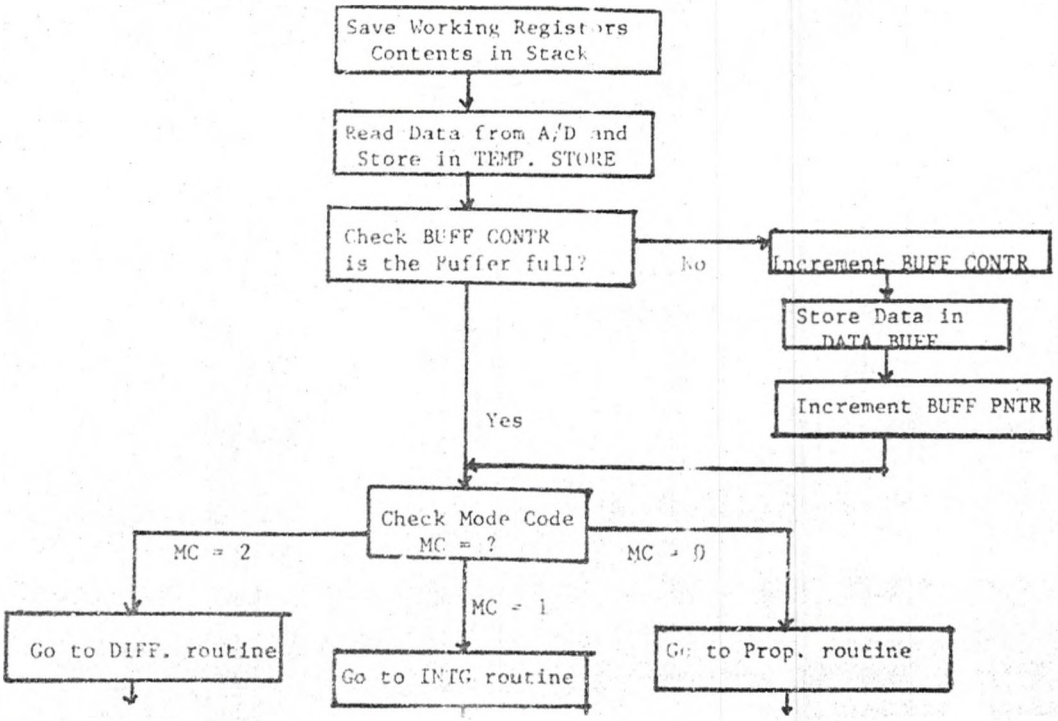
In this instruction C3 is the hex code for jump and 3DFB is hexadecimal jump address (This is a 3 byte instruction and low order address bytes must come first; thus, the instruction written as C3 FB 3D means jump to address 3DFB). Since this instruction starts and ends at 3DFB, this is an unending loop and is referred to in this thesis as a 'DO NOTHING' loop. Thus the background mode is simply this loop. If the address 3DFB is replaced with the starting address of another routine, the new routine will become the background mode. For example if the data from the DATA BUFF has to be typed on the teletype this instruction has to be modified as C3 E) 3D since the Typing routine starts at 3DE) (Notice the low address appears first in the instruction).

INTERRUPT ROUTINE

When PROMPT 80 acknowledges a user interrupt, the Monitor will set the program counter to 3C02. Hence the interrupt routine starts at 3C02. The main functions of the interrupt routine are (1) to store the return address and register contents of the routine that was interrupted so that after servicing the interrupt the computer can resume the original (background mode) program, (2) store data, (3) determine the Controller's mode of operation and exit to the proper routine, (4) when the calculations are complete, output the results, (5) restore working registers and program counter and return to background mode.

Flow chart for the interrupt routine is shown in Fig. 9. In order to facilitate the resumption of the background mode program the Monitor saves the program counter in the stack. The working registers are saved in the 'Stack' by the interrupt routine. The next step is to read the incoming data (Output of A/D) via port E8. This data will be stored immediately in TEMP STORE (3CE6).

The buffer length counter (BUFF CONTR, 3D00) will then be tested to determine if the buffer is full. The buffer length is 100 bytes (64_H). The BUFF CONTR is incremented after storing data in the DATA BUFF. (When the counter reads 64_H the buffer is full and this step will be bypassed). Also the buffer pointer (BUFF PNTR) is incremented every time data is stored. The BUFF PNTR points to the next available location in DATA BUFF for storage. Thus the first 100 points will be stored in the memory. Since the incoming data is stored in TEMP. STORE it is available for calculation whether or not it is stored in DATA BUFF. By changing 64_H to the required number, the length of the buffer can be changed. However, as in the present case, the available memory restricts the buffer length.



After completing calculations the interrupt routine will be reentered at 3C2F

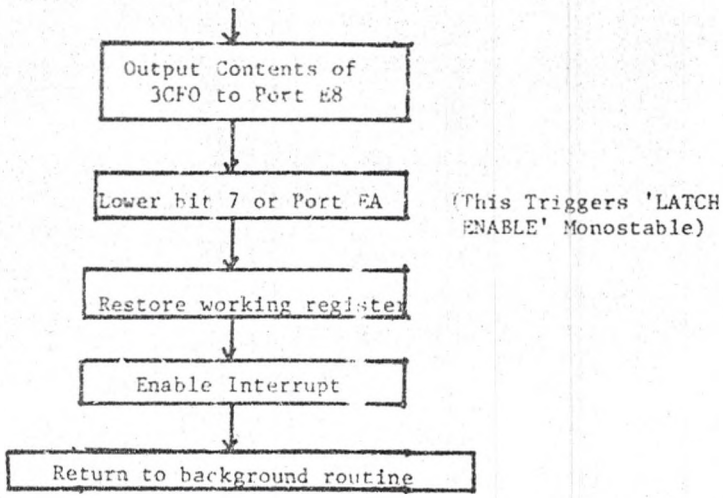


Fig. 9

FLOW CHART OF THE INTERRUPT ROUTINE

With a sample rate of 100/sec, the buffer will contain the response of the system for the first one second. Since the 'BUFF INTR' and 'BUFF PNTR' are reset by executing the Start Up routine only, data can be stored only for the first second. Once the data has been stored the mode code will be tested. This is achieved by loading the mode code in the 'A' register and rotating it right, through carry. The least significant bit in the A register will shift into the carry. Now testing for carry, the mode code is determined. On shifting the first time if the carry flag becomes '1' the mode code is one, indicating PI mode of operation. If only on shifting for the second time, the carry flag become '1', the mode code is 2 and this indicates PID mode. If the carry flag remains '0' both times the mode code is '0' and this means Proportional mode of operation.

The above described operation can easily be visualized by the following:

Effect of RAR (rotate 'A' right through carry) instruction is to move the least significant bit of the A-register to carry.

A-register	Carry	A-register	Carry
00000001	0	00000000	1

(After rotating right through carry)

Once the mode of operation is determined the program exits to the proper routine. After the manipulation of data is complete the interrupt routine will be reentered at 3C2F. Results from OUT BUFF are output via port E8. By lowering bit 7 of port EA (Output 00_H) the 'ENABLE LATCH' monostable will be triggered. This completes servicing of the interrupt. All register contents will then be restored and interrupt will be enabled. The program returns to the background mode.

DIFFERENCE ROUTINE

The Difference routine takes the difference between the present and previous sample data, multiplies it with K_D (the difference gain constant) and stores the results in the DIFF OUT register (3CEE and 3CFO) for the use of the Summing routine. If the rise time of the 'plant' involved is large, the difference between consecutive samples may not be significant even after multiplying by the largest possible difference gain constant. This is a limitation imposed by the finite word length of the computer, A/D and D/A. In such cases the difference between the first and the Nth Sample will be taken. Here 'N' is the number with which DIFF CONTR will be compared. To be able to use this feature, i.e. to change the value of N to suit the particular requirements, the desired value of N must be entered in location 3C45 before executing the Start Up routine. The Difference routine will be executed if and only if the mode code (MC) = 2.

The Difference routine starts at location 3C40 and is 64 bytes long. On completing this routine the program will automatically enter the integration routine.

The Difference routine flow chart is shown in Fig. 10. First, all flags are reset by executing an 'Exclusive OR' instruction. Then the DIFF CONTR is tested for the 5th sample. If it is not, the DIFF CONTR will be incremented and the integration routine entered. If it is the 5th sample the Difference routine execution will continue. In this case first the new data (3CE6) is loaded in the B register and the previous data from 3CE7 is loaded in the A register. The difference between A and B is taken and stored in A. Then the sign flag is tested to see if it is positive or negative.

If the difference is positive it will be multiplied by K_D. The

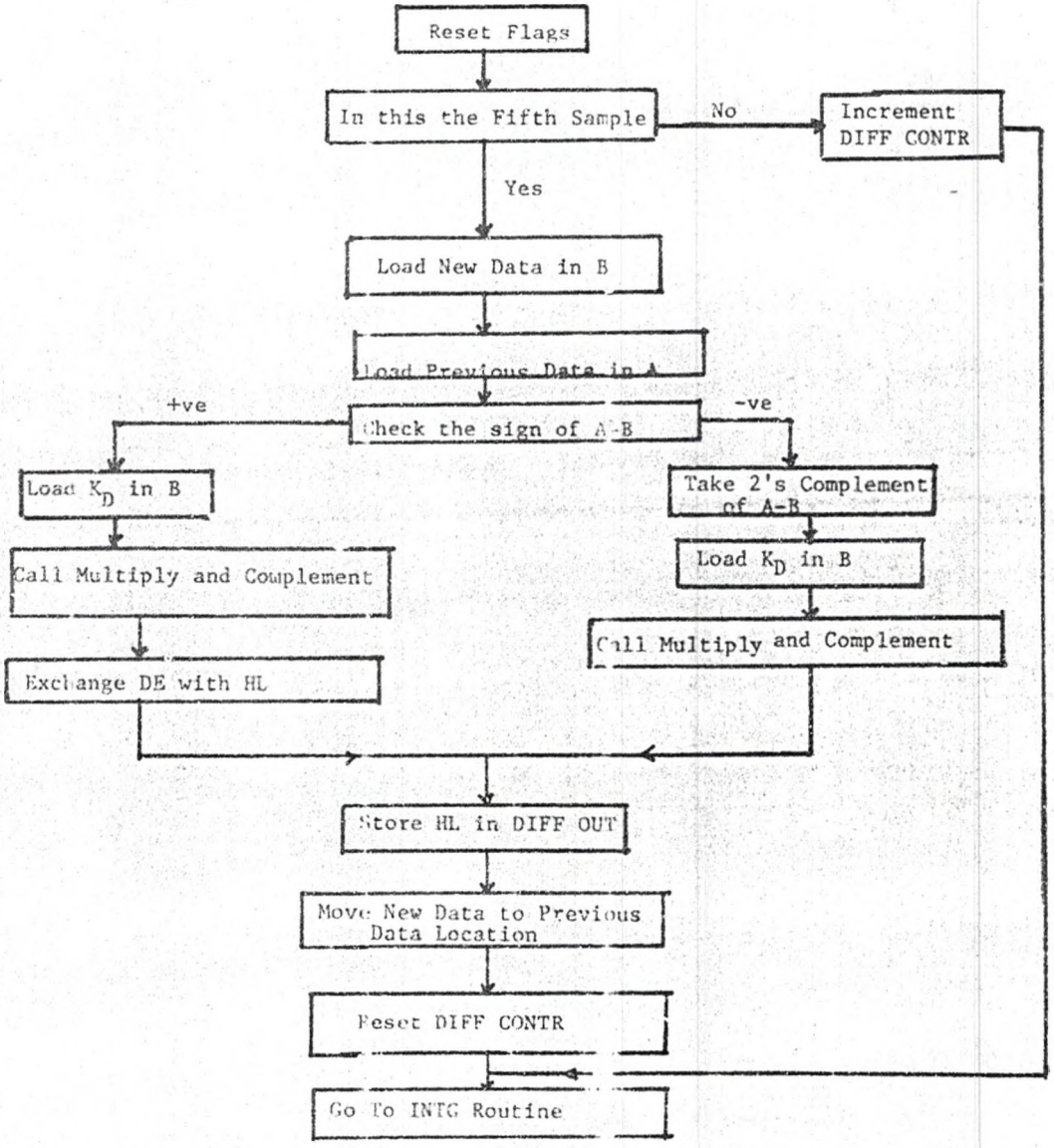


Fig. 10

FLOW CHART OF THE DIFFERENCE ROUTINE

Multiply and Complement routine stores the product in DE and 2's complement of the product in HL. On reentering the Difference routine the contents of DE and HL are exchanged and the product is stored in DIFF OUT buffer.

On the other hand if the difference is negative, A register will contain the 2's complement of (A-B). If this is multiplied by K_D the results will overflow the 16 bits. [Note that for example, the 2's complement of 0001 is 1111]. For this reason the 2's complement of (A) is taken and multiplied by k_D. The product 2's complement is stored in DIFF OUT buffer.

INTEGRATION ROUTINE

In this routine the difference between the set point and the data (i.e the error) is multiplied by K_I (integration gain constant) and added to the contents of the INTG OUT buffer. When the difference between the set point and the Data becomes zero the 'Plant' output has settled at the set point. This routine can be used as pure Integral mode, in PI mode or PID mode. To use this routine the mode code must be 1 or 2. After executing the Integral mode, the program will enter Proportional mode. The Integral mode alone can be used by making K_p and K_D = 0 thus, eliminating results of Difference and Proportional modes from the final output. In this case the mode code can be 1 or 2 thus using the controller in PI or PID modes. Another way of obtaining pure Integral mode is as follows. The last instruction in this routine is 'JMP PROP', directing the program to enter the Proportional mode. If this is changed as 'JMP SUM' the Proportional routine is bypassed and pure Integral mode of operation is obtained. The Integration routine starts at 3C80 and requires 42 bytes.

The Integration routine flow chart is shown in Fig. 11. First the flags are reset. The set point and data are loaded in the A and B registers respectively. Then the difference between A and B is taken the sign flag

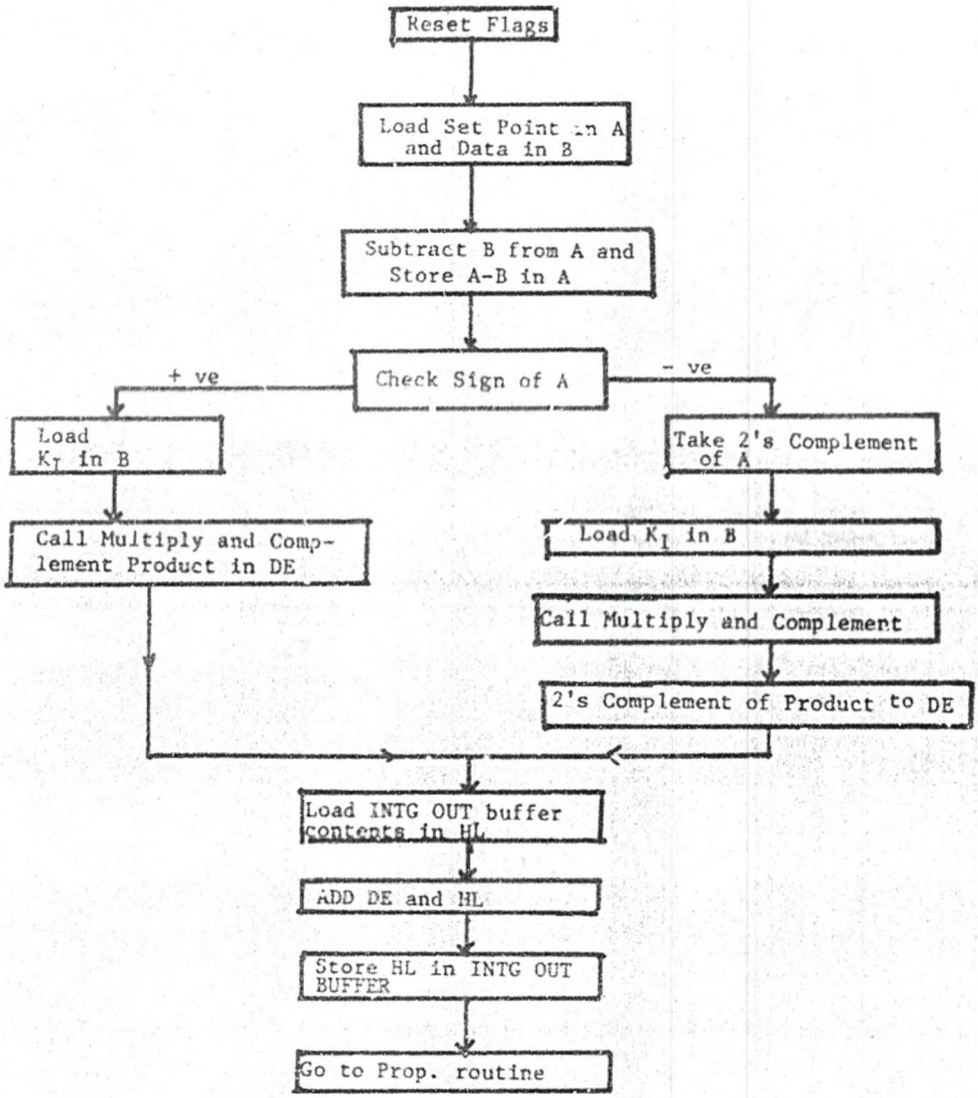


Fig. 11

FLOW CHART OF INTEGRATION ROUTINE

is tested to see if A-B is positive or negative. If the difference is positive, it is multiplied by K_I, the integration gain constant. The product will then be added to INTG OUT buffer. On the other hand if the difference, is negative, for the reasons discussed in the Difference routine, 2's complement of the difference is multiplied by K_I. The Multiply and Complement routine will return the product in DE and its 2's complement in HL registers. The 2's complement of the product will be added to the INTG OUT buffer. The program will then exit to the Proportional routine.

PROPORTIONAL ROUTINE

As the name implies the output of this routine is proportional to the error (difference of set point and data). When this routine is used the error is multiplied by K_p (the proportional gain constant) and the results will be stored in the PROP OUT buffer. The D/A can output 0 volts when the input is 00_H and 5V for FF. Thus it is not possible to output negative numbers. It is conceivable that the Proportional routine is used alone (P mode), hence if the error is negative the output of the Proportional mode is made zero. This difficulty does not arise with Integral and Difference routines because the Difference routine must be used in PID mode alone and in the Integral mode the results are added to the previous sum

In case the OUT BUFF contains a negative number the controller will not function properly. This limits the maximum values of K_I, K_D and K_P. The Proportional routine starts at location 3CB0 and requires 35 bytes.

The flow chart of the Proportional routine is shown in Fig. 12. First all flags are cleared. Next the set point and data are loaded in A and B registers respectively. Then the error (A-B) is stored in A. The sign flag is tested to see if the error is +ve or -ve. If the error is negative, PROP OUT buffer is cleared. If the error is +ve, it is multiplied by K_p and

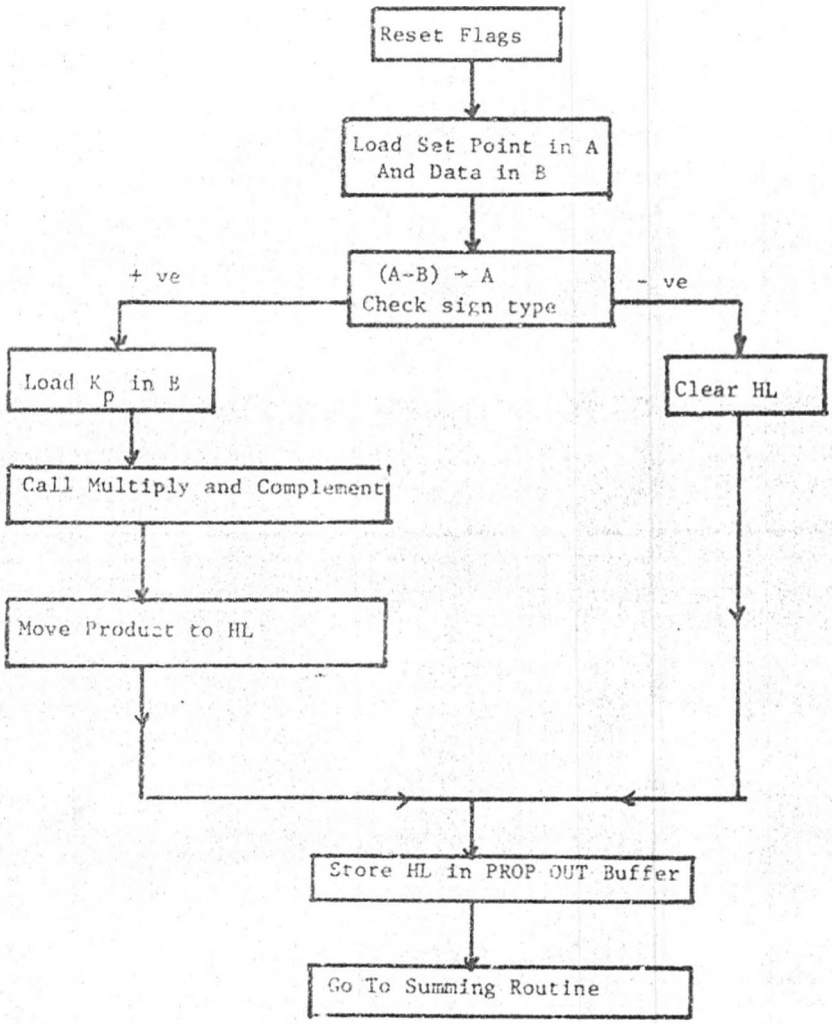


Fig. 12

FLOW CHART OF PROPORTIONAL ROUTINE

the result is stored in the PROP OUT buffer and the program will exit to Summing routine.

SUMMING ROUTINE

The Summing routine sums the results from the Difference, Integration and Proportional routines. The sum is stored in OUT BUFF and the program returns to 3C2F in the Interrupt routine which outputs results. This routine starts at 3D70 and requires 26 bytes. The flow chart of Summing routine is shown in Fig. 13. After clearing the flags the contents of DIFF OUT and INTG OUT are added. If there is no carry the PROP OUT is added to the above sum. If there is no carry the sum is stored in OUT BUFF and program returns to 3C2F. If there is a carry, the contents of OUT BUFF are not modified. This is necessary to ensure that if there is overflow, the output 'saturates'.

MULTIPLY AND COMPLEMENT ROUTINE

This routine multiplies two 8 bit numbers and stores the product in DE registers. The 2's complement of the product is stored in HL registers. The two 8 bit numbers to be multiplied must be in A and B registers before the Multiply and Complement routine is called. The starting address (Also call address) is 3DA0. This program requires 29 bytes.

Fig. 14 is the flow chart of Multiply and Complement routine. First H and L registers are defined as partial product registers and are cleared. Since this routine multiplies two 8 bit numbers the C register, called the loop counter, is loaded with 8. Data in B is loaded in E register and D register is cleared. [This transfer is necessary to use DE register pair and double byte addition instruction]. Next step is to shift partial product to left. First time this does not effect anything since HL contains zero. The multiplier (A register) is rotated left through carry. If the carry is 1, the DE contents are added to HL and the loop counter is decremented. If

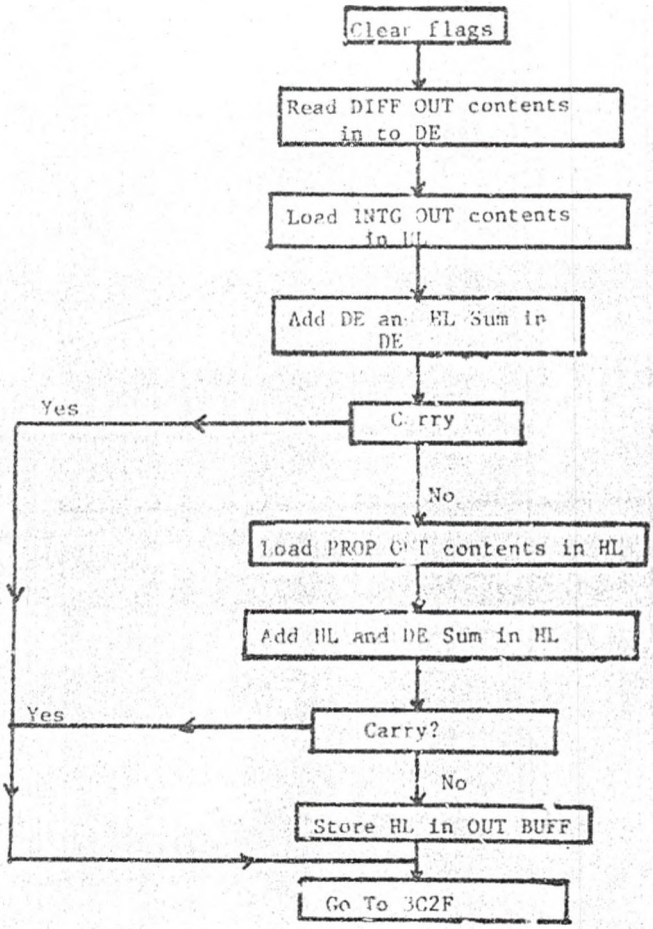


Fig. 13

SUMMING ROUTINE

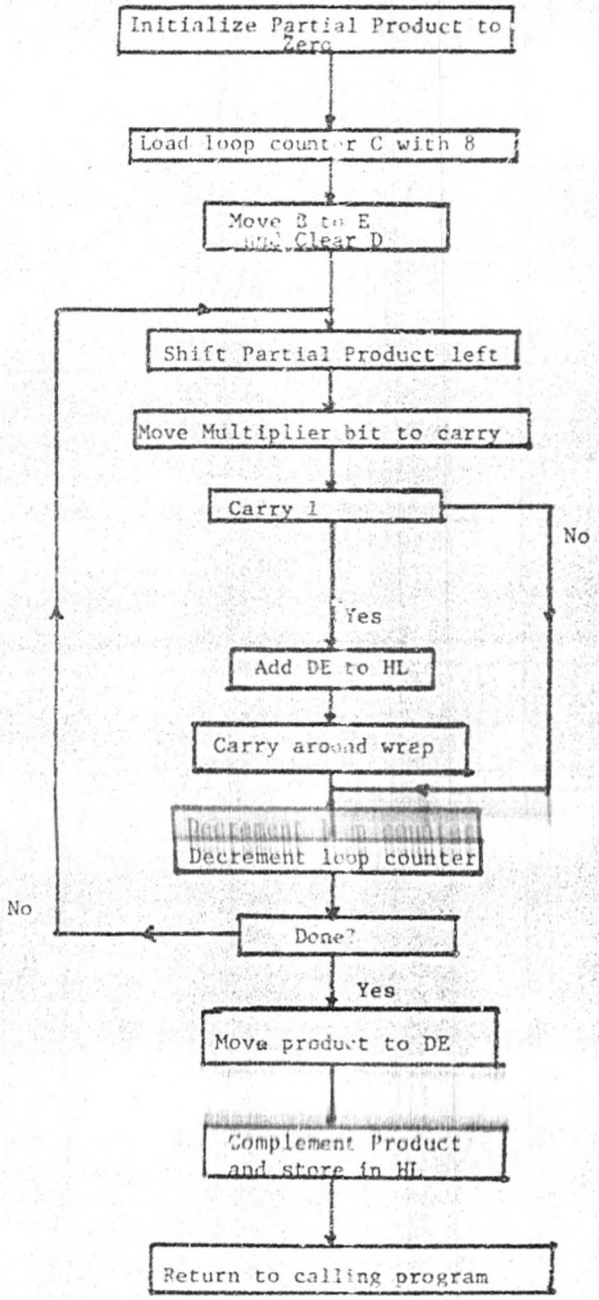


Fig. 14

FLOW CHART OF MULTIPLY AND COMPLEMENT ROUTINE

the carry bit is zero there will be no addition, only the partial product is shifted. This process continues till the loop counter reads zero. At this point the multiplication is complete and the product is in HL registers. Next the product is moved to DE registers and the 2's complement of the product is taken. The 2's complement is loaded in HL and control is returned to the calling program.

OUTPUT SET POINT ROUTINE

The main use of this routine is to provide a step input to test the 'plant' response (with no control). As far as the controller is concerned this routine is not necessary. However, this program is useful to compare the plant response, with and without control, to a step input. Since it is written to be independent, this program can be used without executing the Start Up routine. The set point value must be entered in 3CE1 before executing this routine.

Fig. 15 is the flow chart of the Output Set Point (OSP) routine. First the PPI2 is initialized. The Set Point is output to Port E8. To trigger the 'LATCH ENABLE' monostable bit 7 of Port EA is lowered. As mentioned before to be able to use bits 7 to 4 of Port EA, first the PPI1 is initialized and the FD is output to Port E6. At this point the D/A Output will 'Jump' to set point. This is equivalent to giving a Step Input to the plant. The program will then enter a DO NOTHING loop. One can replace the DO NOTHING loop with a 'HALT' instruction.

TYPING ROUTINE

This is another program that is not necessary for the controller operation but provided for the convenience of the user. Even though PROMPT 80 has a Typing routine in the Monitor, the format in which this routine types data is not very convenient for use. Monitor Typing routine

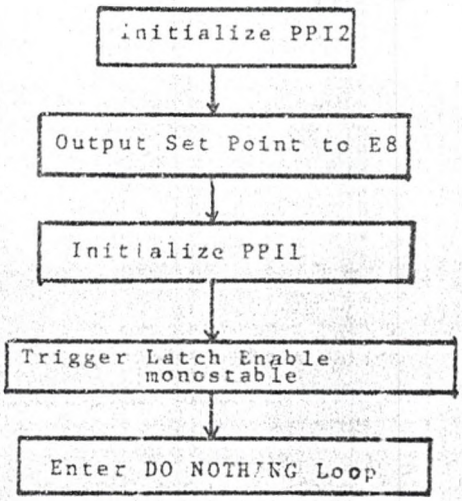


Fig. 15

FLOW CHART OF OUTPUT SET POINT ROUTINE

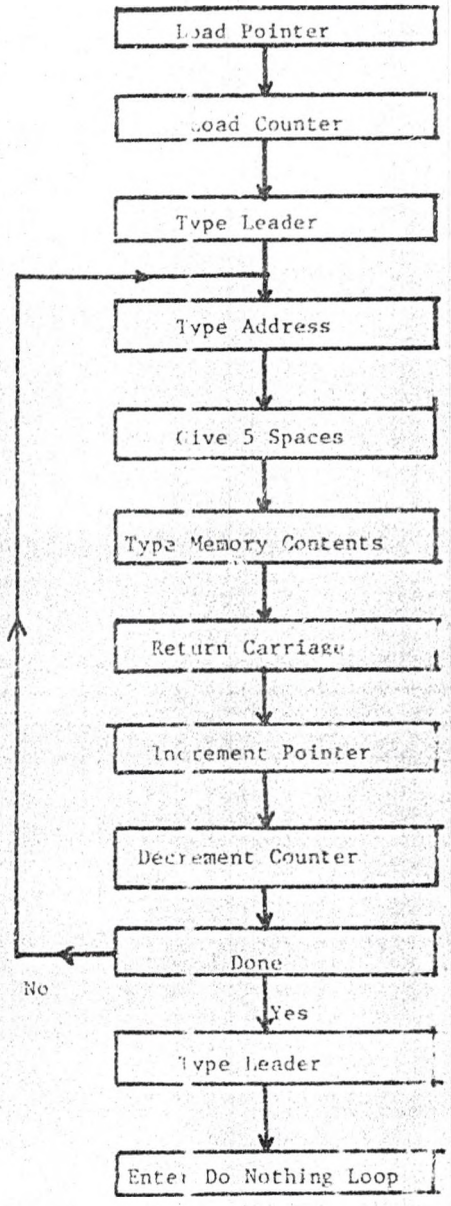


Fig. 16

FLOW CHART OF TYPING ROUTINE

must be used for punching programs. The reason for this is when a program is loaded in the PROMPT 80 via Teletype paper tape, it requires a special format [4]. The Typing routine developed here types the address and contents of the location in the following format.

ADDRESS CONTENTS

This is particularly useful when typing data from the DATA BUFF. Wherever possible some of the Monitor subprograms were used and one subprogram called 'Space' is written. Program 'Space' will generate five spaces between address and contents. The Typing program requires the starting address must be loaded in 3ED1 and 3ED2 (low address first) and the number of locations to be typed in 3ED4 before executing this routine. This program starts in location 3ED0 and requires 47 bytes. The maximum number of locations that can be typed at a time are 256. This limit comes from the fact that the maximum number the counter can be loaded with is 256.

Fig. 16 is the flow chart of the Typing routine. The H and L registers are used as pointer and the C register as counter. A leader tape is punched in the beginning and at the end. If the leaders are not necessary '0' must be loaded in locations 3ED7, 3ED8, 3ED9, 3EEF and 3DF0, prior to execution. While the leader is being punched nothing will be typed. First the address is typed and after giving 5 spaces the contents of the location are typed. Next the carriage is returned and line feed will be given. The pointer is then incremented and the counter is decremented. After this the counter is tested for zero. If it is not zero, the typing continues. If it is zero 'end' leader is punched and the program enters a DO NOTHING loop.

PLANT SIMULATOR

A second order plant was simulated using three Burr-Brown 3318/14 operational amplifiers. This simulator was used to test the operation of the controller. A general linear second order system transfer function is given by

$$\frac{C(s)}{R(s)} = \frac{\omega_n^2}{S^2 + 2\zeta\,\omega_n S + \omega_n^2} \qquad (6)$$

where C is the process variable, R, the input, ω_n^2 is natural frequency, ζ is the damping constant and S is Laplace operator. The differential equation governing such a system is

$$\frac{d^2\,c(t)}{dt^2} + 2\zeta\omega_n \frac{dc(t)}{dt} + \omega_n^2 c(t) = \omega_n^2\, r(t). \qquad (7)$$

For the values chosen for the components of the simulator the natural frequency is approximately 3 radians/sec. or 0.5Hz. The flow chart for this system is shown below

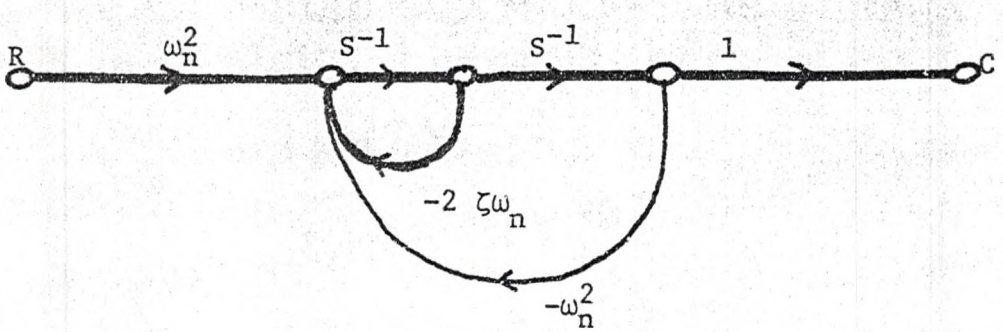
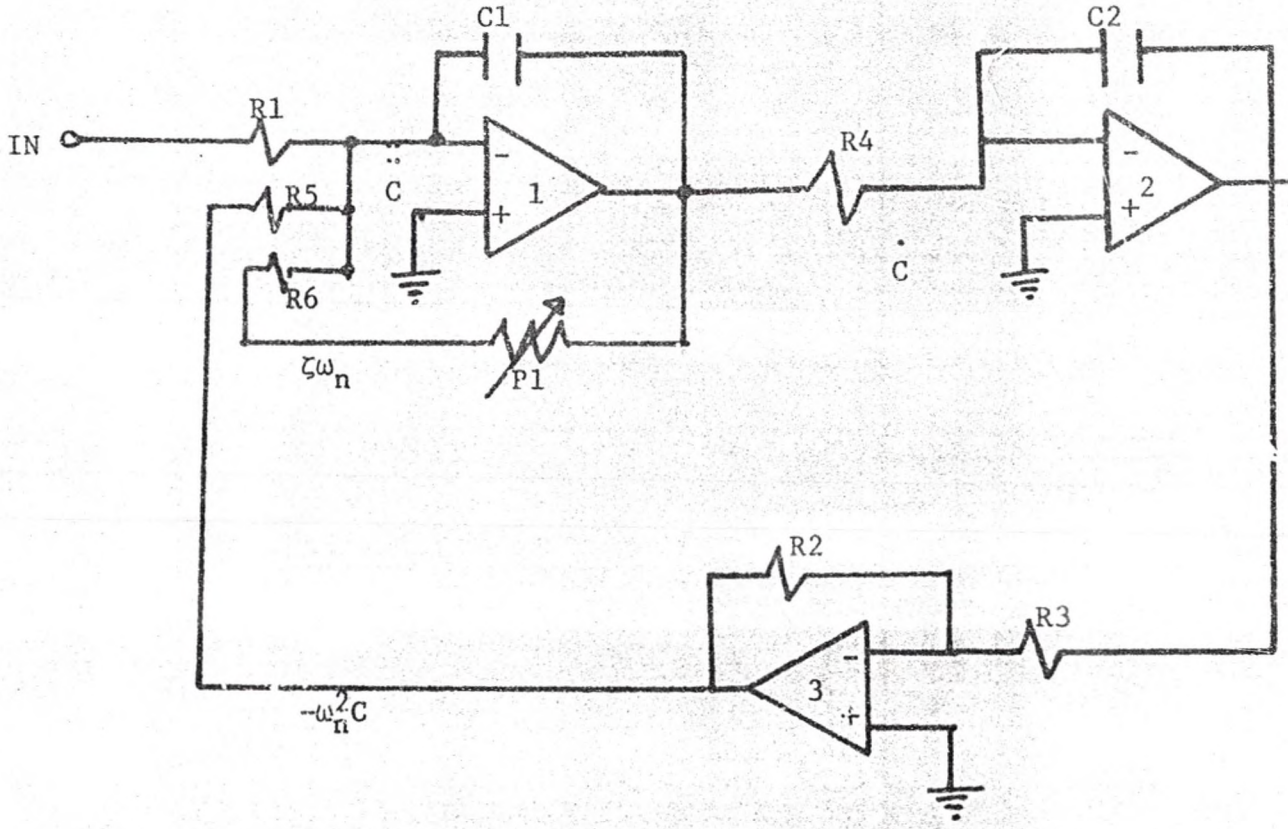


Fig. 17 is a Schematic diagram of the simulator.



R_1 = 100KΩ
R_2 = 10KΩ
R_3 = 10KΩ
R_4 = 100KΩ
R_5 = 100KΩ
R_6 = 10ɔKΩ
C_1 = 1.0μF
C_2 = 1.0μF

OP AMPS BURR-BROWN
3318/14
P_1 is damping potentiameter

Fig. 17

SECOND ORDER PLANT SIMULATOR

CHAPTER III

RESULTS AND DISCUSSION

In this chapter results of the controller performance are presented and discussed. The plant simulator was used for testing so that the plant's transfer function is known exactly and this makes it convenient to calculate the closed loop responses and compare with experimental values. Again the response of the system was tested for a step input for the same reason.

It was already mentioned that the natural frequency is approximately 0.5Hz. Thus sample rate used is 100/sec. Due to the sufficiently large sampling rate, the response can be calculated using analog techniques [root locus, for example [1,2,8]] even though the controller is digital. From the response of the plant to a step input, the value of ζ in equation 6 can be determined using a graph of maximum percent overshoot vs ζ, the damping constant [1]. From Fig. 19 the maximum percent overshoot is found to be 50% and this corresponds to

$$\zeta = 0.23$$

thus, with $\omega_n^2 = 10$, equation 6 becomes

$$\frac{C(s)}{R(s)} = \frac{10}{S^2 + 1.45S + 10} \qquad (8)$$

The characteristic equation for this system is

$$S^2 + 1.45S + 10 = 0$$

and has roots at

$$S_{1,2} = -0.73 \pm j\ 3.08$$

The roots are indicated as plant poles in Fig. 18.

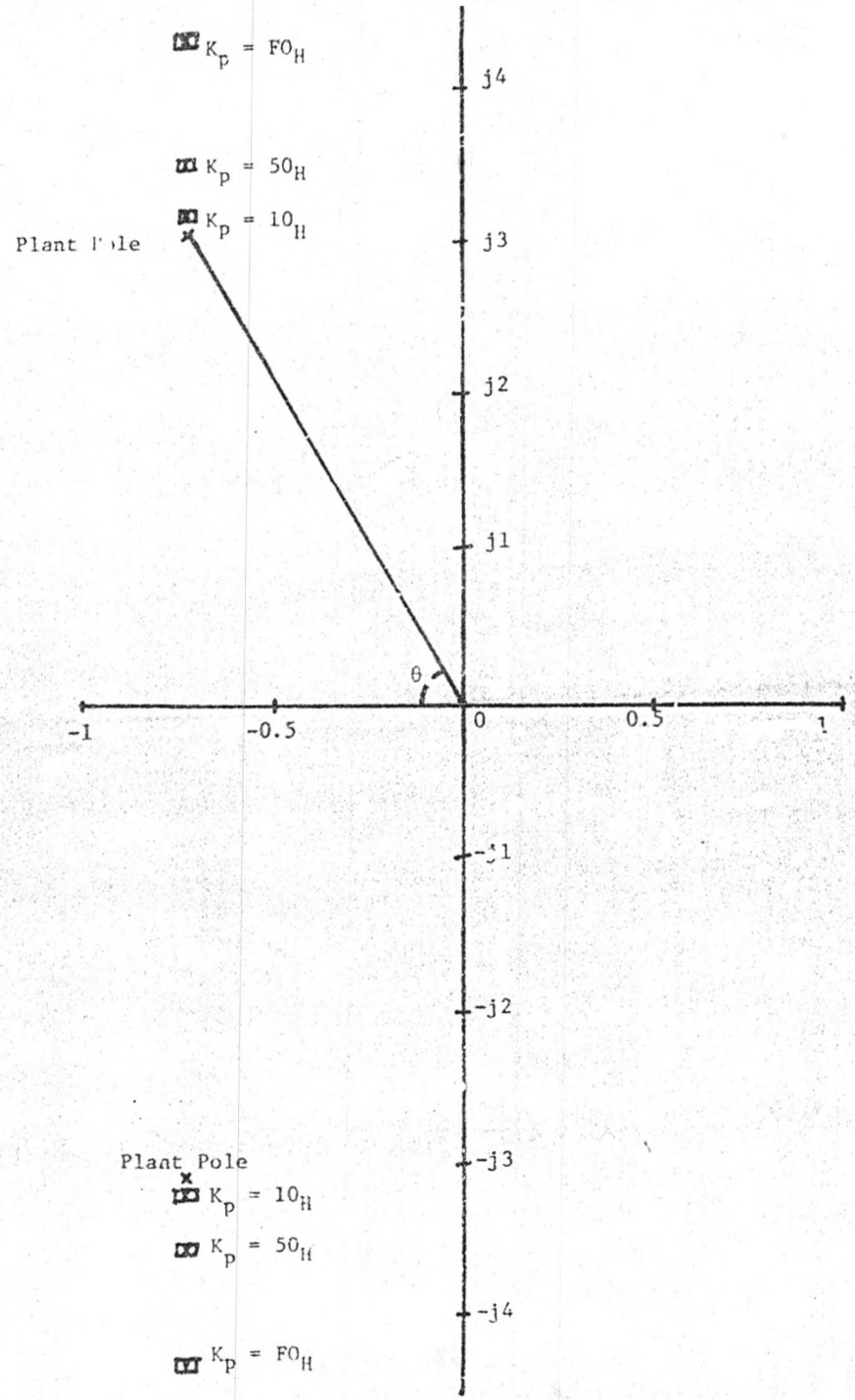
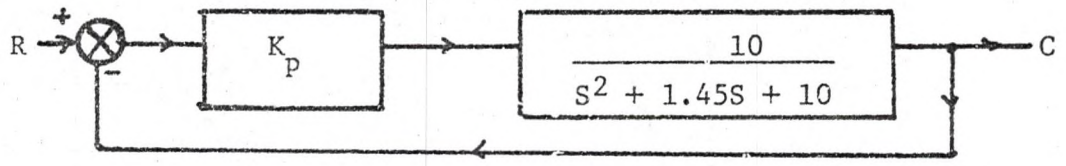


Fig. 18

PLANT POLES AND CLOSED LOOP POLES FOR PROPORTIONAL MODE

PROPORTIONAL MODE

For Proportional mode the system block diagram is shown below.



Closed loop transfer function of the system is

$$G_p(s) = \frac{C(s)}{R(s)} = \frac{10K_p}{S^2 + 1.45S + (10 + 10K_p)} \qquad (9)$$

for a Unit step input

$$C(s) = \frac{1}{S} \frac{10K_p}{S^2 + 1.45S + (10 + 10K_p)}$$

using the final value Theorem the final value of output is $K_p/(K_p+1)$. Figs. 19b, 19c and 19d are the responses of the system in the Proportional mode with $K_p = 10_H$, 50_H, and $F0_H$ respectively. (Subscript H stands for hexadecimal number). Since only the higher order bits of the final results are output, this is equivalent to dividing K_p by 256. Thus the actual K_p's are 0.06, 0.31 and 0.94 for Figs. 19b, 19c and 19d respectively. For these K_p's the final values are 0.06, 0.24 and 0.48 for a unit step input. The corresponding experimental values are 0.055, 0.233 and 0.477. The small, but finite, difference is due to the tolerances of the components used with A/D and D/A circuits. Using equation (9) the location of the closed loop poles, for the three values of K_p chosen are shown in Table II.

TABLE II, K_p and closed loop poles

K_p	S_1	S_2
0.06	- 0.73 + j 3.17	- 0.73 - j 3.17
0.31	- 0.73 + j 3.54	- 0.73 - j 3.54
0.94	- 0.73 + j 4.34	- 0.73 - j 4.34

Set point 70_H

(a) No control

(b) $K_p = 10_H$

(c) $K_p = 50_H$

(d) $K_p = F0_H$

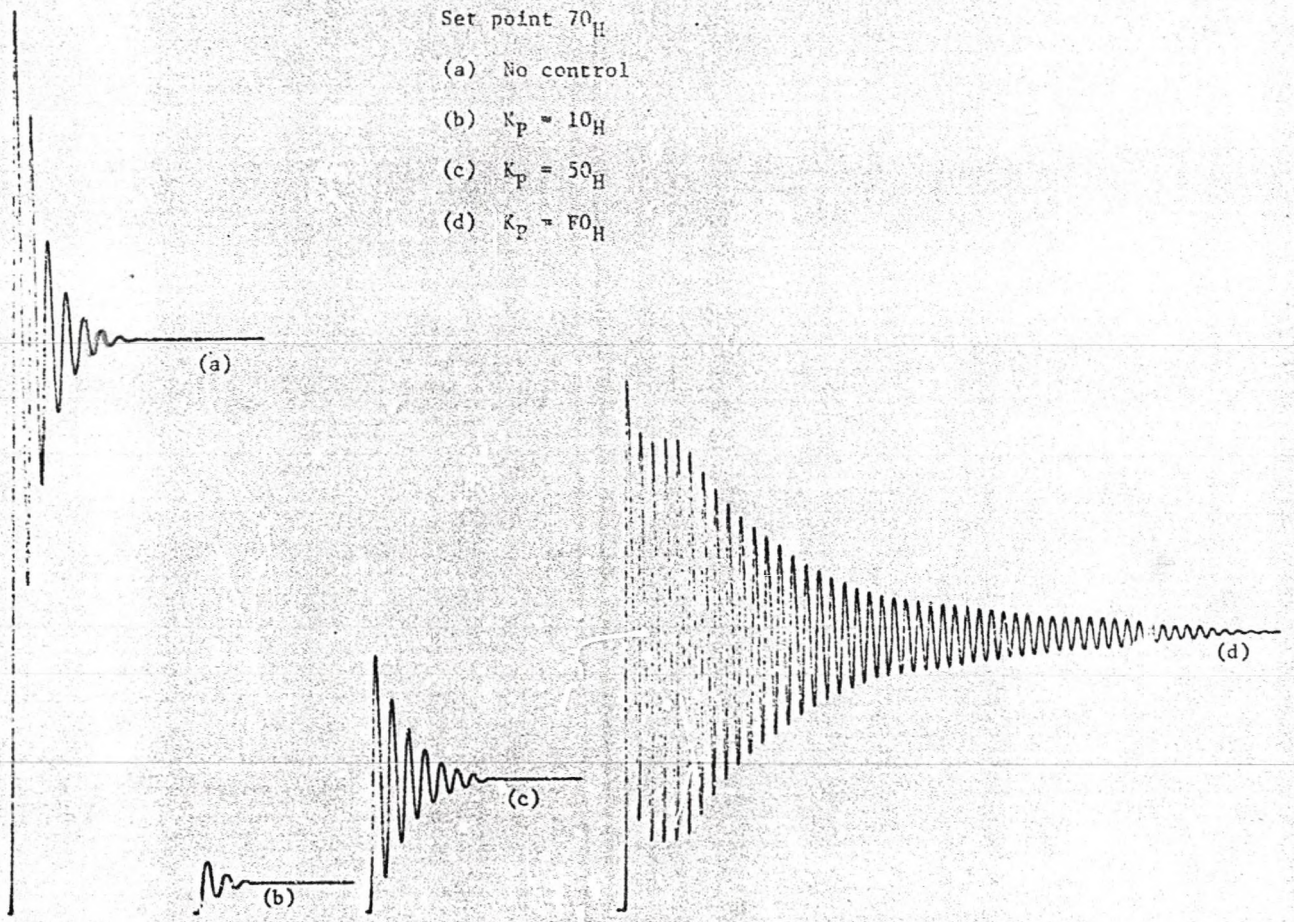


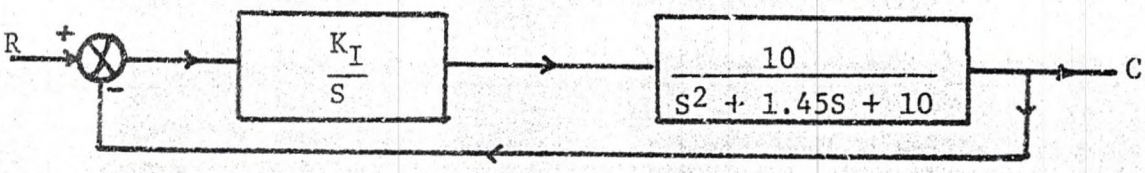
Fig. 19

Proportional Mode

The closed loop locations are plotted in Fig. 18 enclosed in squares and identified with K_p values. As can be seen from Fig. 18 the poles move, with increasing K_p, such that the damping constant, ζ, decreases. This gives rise to increasing, overshoot and longer setting time. These conclusions agree very well with the system behavior as can be seen from Fig. 19.

INTEGRAL MODE:

The system block diagram with the Integral Mode controller is shown below.



The closed loop transfer function for this system is given by

$$G_I = \frac{C(S)}{R(S)} = \frac{10K_I}{S^3 + 1.45S^2 + 10S + 10K_I} \qquad (11)$$

Using the final value Theorem the final value of the output can be shown to be C = 1 for a unit step. i.e. after a sufficiently long time the output will be same as input. This is seen to be the case from Figs. 20b, 20, and 20d. Fig. 20a is the plant response for a step input and is shown in all graphs for comparison. Again the finite difference between the controlled and the uncontrolled 'outputs' is due to the A/D. The characteristic equation of the system is

$$S^3 + 1.45S^2 + 10S + 10K_I = 0 \qquad (12)$$

values of K_I entered in the program are 1, 2 and 3. However, since only the high order 8 bits are used the K_I's in equation 12 are 0.004 (=1/256) 0.008 (=2/256) and 0.012 (=3/256) respectively.

Set point 70_H

(a) No control

(b) $K_I = 01_H$

(c) $K_I = 02_H$

(d) $K_I = 03_H$

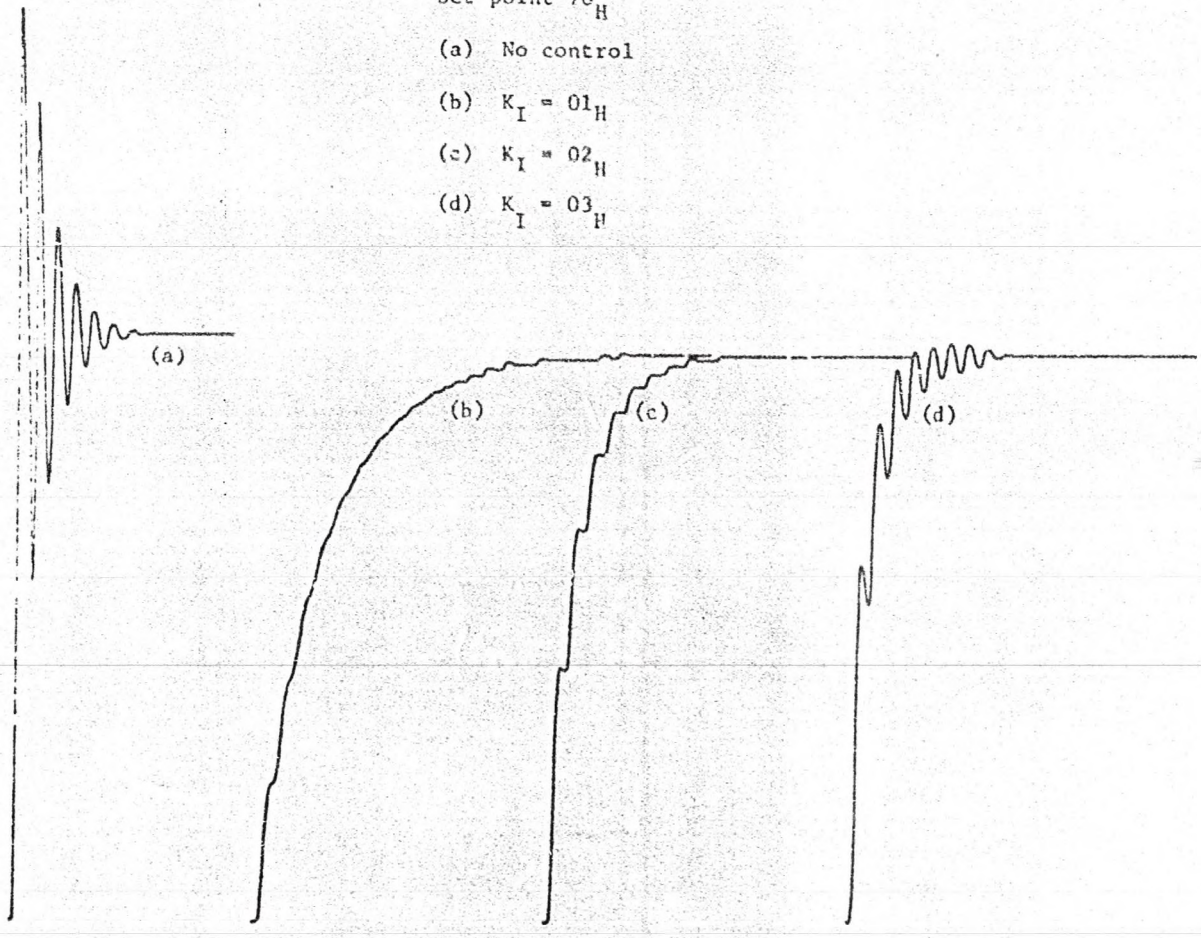


Fig.20

Integral Mode

The roots of equation 12 are shown in Table III.

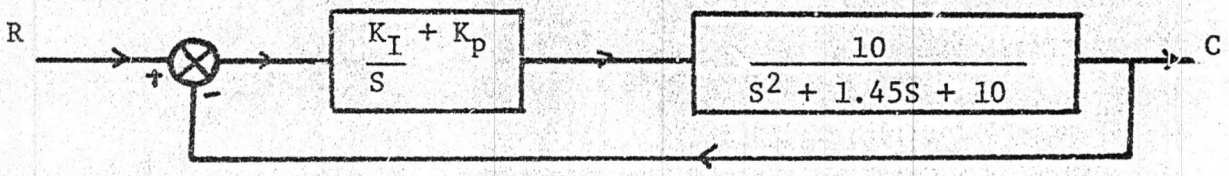
TABLE III

K_I	Root 1	Root 2	Root 3
0.004	- 0.004	-0.727 + j 3.078	-0.727 - j 3.078
0.008	- 0.008	-0.721 + j 3.077	-0.721 - j 3.077
0.012	- 0.012	-0.714 + j 3.077	-0.714 - j 3.077

First of all the three real roots are very close to the origin. The presence of the real root gives the exponential rise towards the final value. However the damping is also decreasing with increasing K_I. Two factors contribute to a faster rise time and oscillation as K_I increases. This behavior is seen from Fig. 20b, 20c and 20d.

PROPORTIONAL PLUS INTEGRAL MODE

For a PI mode the block diagram is



The closed loop transfer function is

$$G_{PI}(S) = \frac{C(S)}{R(S)} = \frac{10\,[K_p\,S + K_I]}{S^3+1.45S^2 + 10(1+K_p)S+ 10K_I} \qquad (13)$$

The final value of the output will be 1.0 for a unit step input, (From final value Theorem). Figs. 21b, 21c and 21d show the response of the system for constant K_p = 0.31 with changing K_I. On analyzing the roots (given in Table IV) it is seen that the complex roots stay almost the same as K_I increases

giving rise to the same oscillating behavior but faster rise time. This is seen to be the case from Figs. 21b, 21c and 21d.

TABLE IV

$K_p = 0.31$

K_I	Root 1	Root 2	Root 3
0.004	- 0.003	- 0.724 + j 3.55	- 0.724 - j 3.55
0.008	- 0.006	- 0.722 + j 3.55	- 0.722 - j 3.55
0.012	- 0.0089	- 0.721 + j 3.55	- 0.721 - j 3.55

When K_I is kept constant and K_p is varied Figs. 22b, 22c, 22d. The roots are shown in Table V.

TABLE V

$K_I = 0.004$

K_p	Root 1	Root 2	Root 3
0.062	- 0.0037	- 0.723 + j 3.178	- 0.723 - j 3.178
0.312	- 0.003	- 0.724 + j 3.55	- 0.724 - j 3.55
0.938	- 0.002	- 0.724 + j 4.34	- 0.724 - j 4.34

Here the real root did not change very much compared to the complex roots. Hence it is expected that the rise time will be same but the oscillations and hence the settling time increases with increasing K_p which is seen to be the case.

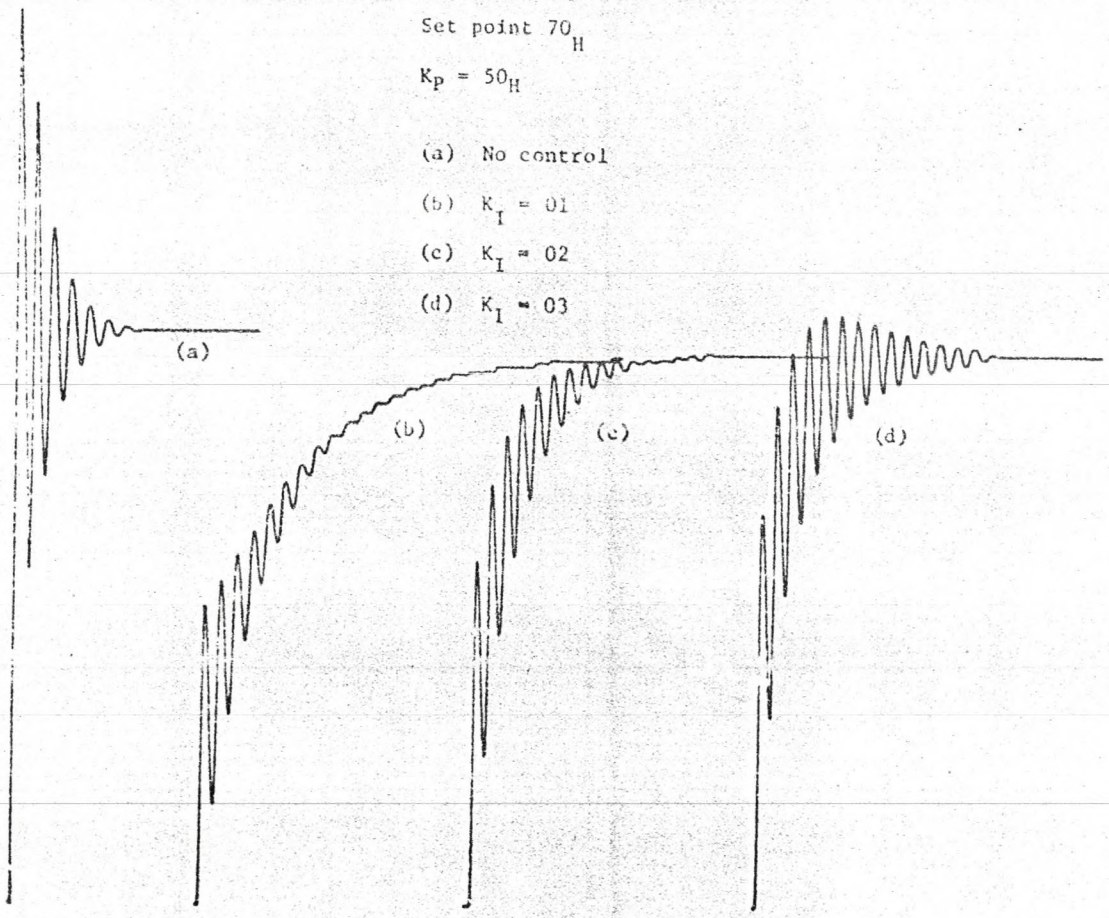


Fig. 21

Proportional - Integral Mode

Set point 70_H

$K_I = 01_H$

(a) No control

(b) $K_P = 10_H$

(c) $K_P = 50_H$

(d) $K_P = F0_H$

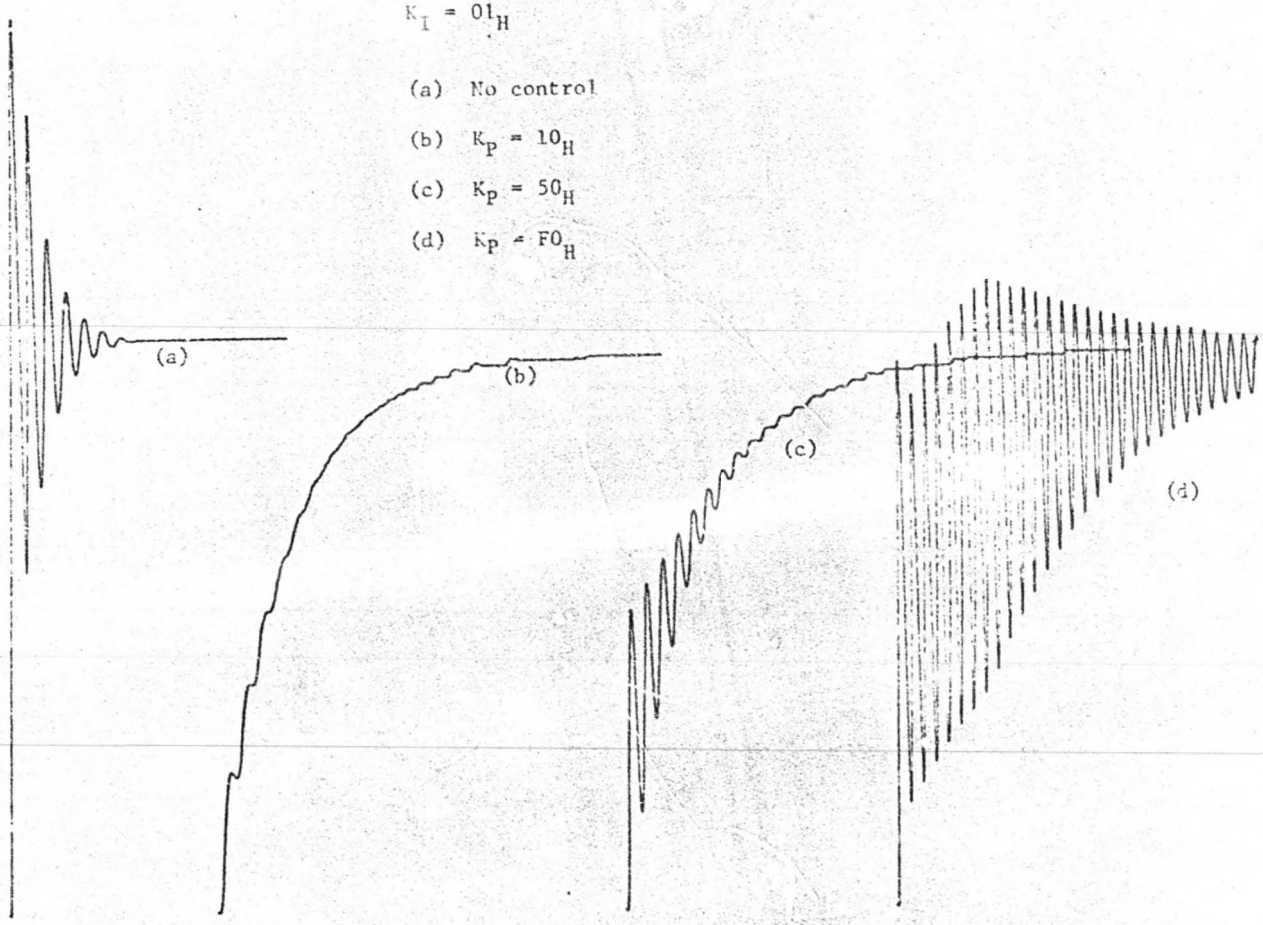
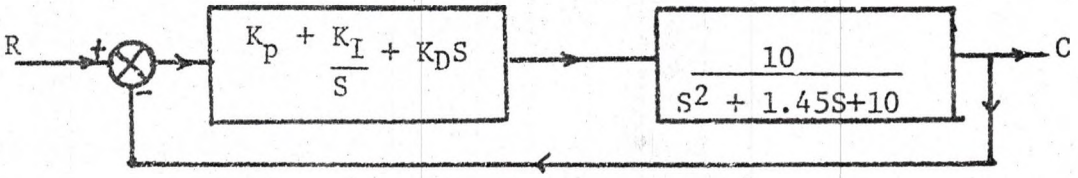


Fig. 22

Proportional - Integral Mode

PROPORTIONAL-INTEGRAL-DIFFERENCE MODE

For a PID mode the system block diagram is shown below.



The closed loop transfer function is given by

$$G(S) = \frac{C(S)}{R(S)} = \frac{10\,[K_D\,S^2 + K_p\,S + K_I]}{S^3 + (1.45+10K_D)\,S^2 + 10(1+K_p)\,S + 10K_I} \qquad (14)$$

Using the final value Theorem and a unit step input, the output can be seen to be unity. A root locus analysis can be done for this system as before. The roots are given in Table VI.

TABLE VI

$K_p = 0.31,\ K_I = 0.004$

K_D	Root 1	Root 2	Root 3
0.0625	- 0.00298	- 1.036 + j 3.47	- 1.036 - j 3.47
0.31	- 0.00298	- 2.29 + j 2.808	- 2.29 - j 2.98
0.938	- 0.00298	- 9.433	- 1.387

These indicate increased stability and Figs. 23b, 23c and 23d show this to be the case.

SUMMARY AND CONCLUSIONS

In this study a versatile real time digital controller was designed. The performance of the controller was tested on a second order plant simulator and was found to be operating as expected. One of the major advantages of this system is its versatility. Even though the mode code allows for P,

Set point 70_H

$K_I = 01_H$

$K_P = 50_H$

(a) No control

(b) $K_D = 01_H$

(c) $K_D = 50_H$

(d) $K_D = F0_H$

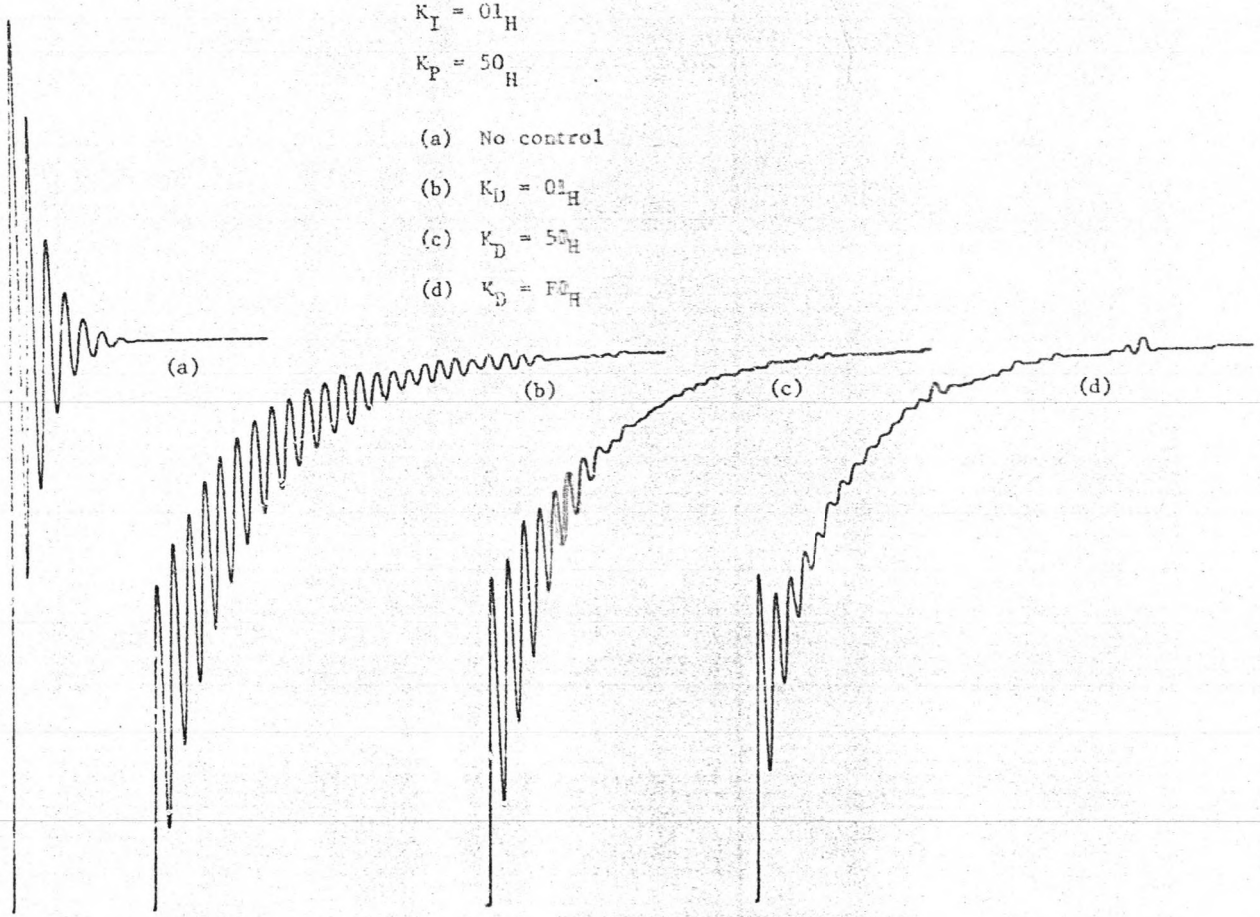


Fig. 23

Proportional - Integral - Difference Mode

PI and PID modes of operations, it is possible to operate this controller in I, ID and PD modes also. This can be achieved by making the appropriate constants zero and running the controller in PID mode. For example if $K_p = 0$ on operating the system in PID mode it can easily be seen that the controller functions as in ID mode.

One of the major restrictions is the size of A/D and D/A. If 12 or 16 bits are used higher gains can be obtained. If the memory is expanded it is possible to control more than one system. An assemble would prove to be an asset in reducing the time required for the development of software.

APPENDIX I

INTERRUPT ROUTINE

```
; When there is an interrupt request the Monitor will set progra
; counter to 3C02.  Hence the Interrupt routine starts at 3C02.
; This routine will save all working registers.  The data from A/D
; will be read and stored in location 3CE6 for further use.  The
; 'Buffer Length Counter' (location 3D00) will be tested to de-
; termine if the buffer is full.  If the buffer is not full data
; will be stored in the buffer.  The next step is to test the
; 'Mode Code' in location 3CE5.  The program enters Proportional
; mode if the code is 0.  It enters Integral or Differential modes
; if the code is 1 or 2 respectively.  When the calculations are
; complete all programs enter Interrupt routine at 3C2F to output
; results to D/A.  Total number of (Bytes) required 61.
```

LABEL	ADDRESS	DATA OR INSTRUCTION	MNEMONIC	COMMENT
INTER	3C02	C5	PUSH BC	;SAVE REGISTER CONTENTS
	3C03	D5	PUSH DE	;
	3C04	E5	PUSH HL	;
	3C05	F5	PUSH PSW	;
	3C06	DB E9	READ PORT E9	;READ INPUT FROM A/D
	3C08	21 E6 3C	LXI HL 3CE6	;LOAD TEMP. DATA STORAGE ADDRESS
	3C0B	77	MOV M,A	;TO H L. STORE DATA
	3C0C	47	MOV B,A	;STOR DATA IN B
BUFLN	3C0D	21 00 3D	LXI H,L	;FETCH BUFFER LENGTH COUNTER
	3C10	7E	MOV A,M	;AND CHEK TO SEE IF BUFFER
	3C11	D6 64	SUI D6	;IS FULL
	3C13	CA 1F 3C	JZ 3C1F	;JUMP ON ZERO TO MODCK
	3C16	34	INR M	;INCREMENT BUFFER LENGTH COUNTER
	3C17	2A 01 3D	LHLD 3D01	;LOAD BUFFER POINTER IN HL
	3C1A	70	MOV M,B	;STORE DATA IN BUFFER
	3C1B	23	INX HL	;INCREMENT BUFFER POINTER
	3C1C	22 01 3D	SHLD 3D01	;STORE BUFFER POINTER
MODCK	3C1F	97	SUB A	;CLEAR FLAGS
	3C20	21 E5 3C	LXI HL 3C25	;LOAD MODE CODE ADDRESS IN HL
	3C23	7E	MOV A,M	;FETCH MODE CODE
	3C24	IF	RAR	;CHECK IF MODE CODE IS 1
	3C25	DA 80 3C	JC INTG	;IF 1 PI MODE, JUMP TO INTG.
	3C28	IF	RAR	;CHECK IF MODE CODE IS 2
	3C29	DA 40 3C	JC DIFF	;IF 2 PID MODE, JUMP TO DIFF.
	3C2C	C3 B0 3C	JMP PROP	;JUMPT TO PROP.

LABEL	ADDRESS	DATA OR INSTRUCTION	MNEMONIC	COMMENT
OUT	3C2F	21 F0 3C	LXI HL 3CF0	;LOAD DATA BUFFER ADDRESS
				;3C F0 TO HL
	3C32	7E	MOV A,M	;FETCH DATA TO OUTPUT
	3C33	D3 E8	OUT E8	;OUTPUT DATA TO PORT E8
	3C35	3E 00	MVI 00	;ENABLE D/A LATCHES BY
	3C37	D3 EA	OUT EA	;OUTPUT 0 TO PORT EA
	3C39	F1	POP PSW	;RESTORE REGISTER CONTENTS
	3C3A	E1	POP HL	;
	3C3C	C1	POP BC	;
	3C3D	FB	E1	;ENABLE INTERRUPT
	3C3E	C9	RET	;RETURN
				;END OF INTERRUPT ROUTINE

DIFFERENCE ROUTINE

;In this routine the difference between the present
;data and previous data is multiplied by K_D, the
;difference gain constant. The product is stored in
;DIFF OUT. For reasons discussed in text this multi-
;plication occurs once every 5 samples. If the present
;sample number is not a multiple of 5 the multiplication
;is bypassed and the difference counter (DIFF CONTR)
;(which keeps track of the number of samples) is incre-
;mented. The number of samples required before
;multiplication (N) can be changed by entering (N-1) in
;location 3C45 (here M=5 hence location 3C45 contains 4).
;This routine requires that the value of K_D in 3CE2 be
;entered before execution. Mode code to execute this
;routine is 2. Starting address is 3C40.

LABEL	ADDRESS	DATA OR INSTRUCTION	MNEMONIC	COMMENT
DIFF	3C40	AF	XRA	;CLEAR FLAGS
	3C41	3A EE 3C	LDA 3CEE	;DIFF CONTR TO A
	3C44	FE 04	CPI 04	;CHECK TO SEE IF THIS IS ;5TH POINT
	3C46	C2 79 3C	JNZ INC	;IF IT IS NOT 5TH POINT ;JUMP TO INC
	3C49	AF	XRA	;CLEAR FLAG
	3C4A	3A E7 3C	LDA 3CE6	;LOAD OLD DATA IN A
	3C4D	47	MOV B,A	;OLD DATA TO B

LABEL	ADDRESS	DATA OR INSTRUCTION	MNEMONIC	COMMENT
	3C4E	3A E6 3C	LDA 3CE7	;NEW DATA TO A
	3C51	90	SUB B	;A-B → A
	3C52	F2 61 3C	JP PSLP	;IF A IS +ve JUMP
	3C55	2F	CMA	;TAKE 2's COMPLEMENT OF A
	3C56	3C	INR A	;
	3C57	21 E2 3C	LXI HL 3CE2	;FETCH K_D AND LOAD
	3C5A	46	MOV B,M	;IT IN B
	3C5B	CD AO 3D	CALL MULT	;RETURNS WITH PRODUCT IN DE
				;AND 2's COMPLEMENT OF
				;PRODUCT IN HL
	3C5E	C3 69 3C	JUMP BUFF	;JUMP TO BUFF
PSLP	3C61	21 E2 3C	LXI HL, 3CE2	;K_D TO B
	3C64	46	MOVE B,M	;
	3C65	CD AO 3D	CALL MULT	;
	3C68	EB	XCHG	;
BUFF	3C69	22 E8 3C	SHLD 3CE8	;STORE DIFF. PRODUCT
	3C6C	3A E6 3C	LDA 3CE6	;STORE NEW DATA IN
	3C6F	32 E7 3C	STA 3CE7	;OLD DATA LOCATION
	3C73	AF	XRA	;CLEAR DIFF CONTR
	3C73	32 EE 3C	STA 3C EE	;
	3C76	C3 70 3C	JUMP EXIT1	;
INC	3C79	3C	INR A	;INCREMENT DIFF CONTR
	3C7A	32 EE 3C	STA 3CEE	;
EXIT1	3C7D	C3 80 3C	JUMP INTG	;ENTER INTEGRATION ROUTINE

INTEGRATION ROUTINE

;This routine integrates the error signal (Set point-data).
;Each time this routine is executed the difference between
;set point and data is multiplied by K_I, the integration
;gain constant and the result will be added to the value in
;INTG OUT register (3CEA, 3CEB). The set point must be in
;3CE1,K_I in 3CE3 and data in 3CE6 before this routine is
;executed. Starting address of this routine is 3C80.

LABEL	ADDRESS	DATA OR INSTRUCTION	MNEMONIC	COMMENT
INTG	3C80	AF	XRA	;CLEAR FLAGS
	3C81	3A E1 3C	LDA 3CE1	;FETCH SET POINT
	3C84	21 E6 3C	LXI HL 3CE6	;DATA ADDRESS TO HL
	3C87	46	MOV B,M	;DATA TO B
	3C88	90	SUB B	;A-B → A
	3C89	F2 99 3C	JP UP	;GO TO UP IF +ve
	3C8C	2F	CMA	;TAKE 2's COMPLEMENT OF A
	3C8D	3C	INRA	;
	3C8E	21 E3 3C	LXI HL 3CE3	;LOAD K_I IN B
	3C91	46	MOV B,M	;
	3C92	CD AO 3D	CALL MULT	;MULTIPLY ERROR WITH K_I
	3C95	EB	XCHG	;2's COMPLEMENT OF PRODUCT TO DE
	3C96	C3 AD 3C	JMP ADD	;JUMP TO ADD
UP	3C99	21 E3 3C	LXI HL 3CE3	;K_I TO B
	3C9C	46	MOV B,M	;
	3C9D	CD AO 3D	CALL MULT	;
	3CAO	2A EA 3C	LHLD 3CEA	;INTG OUT SUM TO HL
ADD	3CA3	19	DAD DE	;ADD HL AND DE

LABEL	ADDRESS	DATA OR INSTRUCTION	MNEMONIC	COMMENT
	3CA4	22 EA 3C	SHLD 3CEA	;STORE IN INTG OUT
	3CA7	00	NOP	;
	3CA8	00	NOP	;
	3CA9	00	NOP	;
	3CAA	3C BO 3C	JMP PROP	;GO TO PROPORTIONAL ROUTINE

START UP ROUTINE

;This routine must be used when first starting the system or if
;the system is stopped to change set point etc. This routine
;clears all buffers, initializes PPI'S and enables the INITIATE
;CONVERSION PULSES. Starting address is 3DCA.

LABEL	ADDRESS	DATA OR INSTRUCTION	MNEMONIC	COMMENT
START	3DCA	00	NOP	;
	3DCB	3E 00	MVI A,0	;
	3DCD	32 EE 3C	STA 3CEE	;CLEAR DIFF. CONTR.
	3DD0	00	NOP	;
	3DD1	32 E7 3C	STA 3D00	;CLEAR BUFFER LENGTH COUNTER
	3DD4	32 E7 3C	STA 373C	;CLEAR PREVIOUS DATA REG.
	3DD7	67	MOV H,A	;CLEAR H AND L
	3DD8	6F	MOV L,A	;
	3D09	22 E8 3C	SHLD 3CE8	;CLEAR DIFF. OUT BUFFER
	3DDC	22 EA 3C	SHLD 3CEA	;CLEAR INTG. OUT BUFFER
	3DE0	22 EC 3C	SHLD 3CEC	;CLEAR PROP. OUT BUFFER
	3DE2	22 EF 3C	SHLD 3CEF	;CLEAR OUT. BUFF
	3DE5	21 03 3D	LXI H,L 3D03	;
	3DE8	22 01 3D	SHLD 3D01	;RESET DATA BUFF. PNTR

LABEL	ADDRESS	DATA OR INSTRUCTION	MNEMONIC	COMMENT
	3DEB	3E 83	MVI A, 83_H	;INITIALIZE PPI2
	3DED	D3 EB	OUT EB	;
	3DEF	3E 70	MVI A, 70_H	;INITIALIZE PPI1
	3DF1	D3 E6	OUT E6	;
	3DF3	3E 80	MVI A, 80_H	;ENABLE INITIATE CONVERSION
	3DF5	D3 EA	OUT EA	;PULSES
	3DF9	D3 E6	OUT E6	;
LOOP	3DFB	C3 FB 3D	JMP LOOP1	;JUMP TO LOOP 1

MULTIPLY AND COMPLEMENT

;This routine multiplies two 8 bit numbers and returns the product ;to DE register pair (low order bits in E register). The 2's ;complement of the product will appear in HL register pair (low ;order bits in L). This program requires that the multiplier ;and multiply card be in A and B registers (order is immaterial) ;before calling this program. On completion control will be ;returned to the calling program. Starting address of this ;routine is 3DA0 and requires 29 bytes.

LABEL	ADDRESS	DATA OR INSTRUCTION	MNEMONIC	COMMENT
MULT	3DA0	21 00 00	LXI H, 00	;CLEAR PARTIAL PRODUCT
	3DA3	0E 08	MVI C, 8	;LOAD LOOP COUNTER WITH 8
	3DA5	58	MOVE, B	;
	3DA6	16 00	MVI D, 00	;

LABEL	ADDRESS	DATA OR INSTRUCTION	MNEMONIC	COMMENT
LOOP2	3DA8	29	DAD H	;
	3DA9	17	RAL	;MULTIPLIER BIT TO CARRY
	3DAA	D2 BO 3D	JNC DCR	;DO NOT ADD IF NO CARRY
	3DAD	19	DAD D	;ADD DE AND HL IF CARRY
	3DAE	CE	ACI OO	;WRAP CARRY AROUND
DCR	3DBO	OD	DCR C	;DECREMENT LOOP COUNTER
	3DB1	C2 A8 3D	JNZ LOOP2	;JUMP TO LOOP 2 IF C
				;IS NOT ZERO
	3DB4	EB	XCHG	;PRODUCT TO DE
	3DB5	7B	MOV A,E	;TAKE 2's COMPLEMENT OF
	3DB6	2F	CMA	;PRODUCT
	3DB7	6F	MOV L,A	;
	3DB9	7A	MOV A,D	;
	3DB9	2F	CMA	;
	3DBA	67	MOV H,A	;
	3DBB	23	INXHL	;
	3DBC	C9	RET	;RETURN TO CALLING PROGRAM

PROPORTIONAL ROUTINE

;In this program the error is multiplied by K_p, the Proportion-
;ality gain constant, and the results are stored in PROP OUT
;buffer. (3CEC and 3CED) high order bits in 3CED. If the error
;is negative the PROP OUT buffer will contain zero. Before
;executing this program K_p and set point must be loaded in
;their respective locations. The starting address is 3CBO.

LABEL	ADDRESS	DATA OR INSTRUCTION	MNEMONIC	COMMENT
PROP	3CBO	AF	XRA	;CLEAR A
	3CB1	21 E1 3C	LXI HL, 3CE1	;SET POINT TO A
	3CB4	7E	MOV A,M	;
	3CB5	21 E6 3C	LXI HL, 3CE6	;DATA TO B
	3CB8	46	MOV B,M	;
	3CB9	90	SUB B	;A-B → A
	3CBA	F2 C5 3C	JP POS	;JUMP TO POS IF POSITIVE
	3CBD	97	SUB A	;CLEAR FLAGS
	3CBE	57	MOV D,A	;ERROR -ve
	3CBF	5F	MOV E,A	;CLEAR HL TO
	3CCO	00	NOP	;OUTPUT ZERO
	3CC1	00	NOP	;
	3CC2	C3 CC 3C	JMP STORE	;
POS	3CC5	21 E4 3C	LXI HL, 3C24	;FETCh K_p AND
	3CC8	46	MOV B,M	;LOAD 1T IN B
	3CC9	CD AD 3D	CALL MULT	;
STORE	3CCC	EB	XCHG	;
	3CCD	22 EC 3C	SHLD 3CE1	;STORE HL IN 3CEC
	3CDO	C3 70 3D	JMP SUM	;GO TO SUMMING ROUTINE

SUMMING ROUTINE

```
;In this routine results of Proportional, Integral and Difference
;routines are added and the results are stored in OUT BUFF (3CEF,
;3CF0). All registers will be summed irrespective of the mode of
;operation. Since 'Start Up' routine clears all three buffers
;(PROP OUT, INTG OUT and DIFF OUT). Summing all buffers, say in
;Proportional mode, does not cause errors. If the results are
;such that the sum exceeds 16 bits, the previous contents of OUT
;BUFF are not altered. Before entering this routine the three
;buffers (DIFF OUT, INTG OUT and PROP OUT) must contain data to
;be Summed (or zero).
```

LABEL	ADDRESS	DATA OR INSTRUCTION	MNEMONIC	COMMENT
SUM	3D70	AF	XRA	:CLEAR FLAGS
	3D71	2A E8 3C	LHLD	;DIFF OUT TO HL
	3D74	EB	XCHG	;DIFF OUT TO DE
	3D75	2A EA 3C	LHLD INTG	;INTG OUT TO HL
	3D78	19	DAD DE	;HL+DE → HL
	3D79	DA 81 3D	JC NOADD	;IF SUM EXCEEDS 16 BITS
				;GO TO NOADD
	3D7C	EB	XCHG	;SUM TO DE
	3D7D	2A EC 3C	LHLD PROP	;PROP OUT TO HL
	3D80	19	DAD DE	;DE+HL → HL
	3D81	DA 87 3D	JC NOADD	;IF SUM EXCEEDS 16 BITS
				;GO TO NOADD
	3D84	22 EF 3C	SHLD OUT BUFF	;STORE SUM IN OUT BUFF
NOADD	3D87	C3 2F 3C	JUMP OUT	;JUMP TO OUT.

TYPING ROUTINE

```
;In this routine the contents of selected memory locations
;are typed. The format of typing is first the address and
;memory contents after 5 spaces. One location per line is
;typed. The starting address of the locations to be typed
;must appear in locations 3ED1 and 3ED2, number of locations
;(in hex) to be typed must be entered in location 3ED4 prior
;to execution. If leaders in the beginning and in the end are
;not needed, enter zero in 3ED7, 3ED8, 3ED9 and 3EEE, 3EEf
;and 3FF0. After typing all locations the program enters a
;'DO NOTHING' loop. The maximum number of locations that
;can be typed using this program once are 256. The starting
;address is 3ED0. This requires Monitor Subprograms PADDR,
;PBYTE, PEOL, CO and Lead. Subprogram 'Space' is part of
;this routine. To test this program load this program and
;execute from 3ED0. It should reproduce the addresses and
;contents of this program.
```

LABEL	ADDRESS	DATA OR INSTRUCTION	MNEMONIC	COMMENT
TYPE	3ED0	21 D0 3E	LXIHL, 3ED0	;LOAD STARTING ADDRESS IN HL
	3ED3	01 30 00	LXIBC, 0030	;# OF LOCATIONS TO BE TYPED
				;TO BC (LOCATION COUNTER)
	3EDC	C5	PUSH BC	;SAVE BC
	3ED7	CD 07 04	CALL LEAD	;PUNCH LEADER

LABEL	ADDRESS	DATA OR INSTRUCTION	MNEMONIC	COMMENT
ADDR	3EDA	CD 32 04	CALL PADDR	;TYPE ADDRESS
	3EDD	CD F4 3E	CALL SPACE	;GIVE 5 SPACES
	3EE0	7E	MOV A,M	;FETCH MEMORY CONTENTS
				;A AND LOAD IN 'A'
	3EE1	CD 61 04	CALL PBYTE	;TYPE CONTENTS
	3EE4	CD A4 04	CALL PEOL	;GIVE CARRIAGE RETURN
				;AND LINE FEED
	3EE7	23	INX HL	;INCREMENT HL (POINT
				;TO NEXT LOCATION)
	3EE8	C1	POP BC	;
	3EE9	0D	DCR C	;DECREMENT LOCATION COUNTER
	3EEA	C5	PUSH BC	;
	3EEB	C2 DA 3E	JNZ ADDR	;IF NOT DONE CONTINUE
	3EEE	CD 07 04	CALL LEAD	;PUNCH END LEADER
LOOP4	3EF1	C3 F1 3E	JMP LOOP4	;ENTER DO NOTHING LOOP
SPACE	3EF4	0E 20	MVI C,20	;LOAD HEX CODE FOR SPACE IN C
	3EF6	0605	MVI B,05	;ENTER # OF SPACES
CO	3EF8	CD FA 07	CALL CO	;GIVE SPACE
	3EFB	05	DCR B	;
	3EFL	C2 F8 3D	JNZ CO	;IF NOT DONE JUMP TO CO
	3EFF	C9	RET	;RETURN TO CALLING PROGRAM

OUTPUT SET POINT

;In this routine the set point in location 3CE1 is output

;via Port E8 and the 'LATCH ENABLE' monostable is triggered.

;Once this is done the program enters a 'DO NOTHING loop'.

;The set point must be entered before executing this routine.

;Starting address of this routine is 3F00.

LABEL	ADDRESS	DATA OR INSTRUCTION	MNEMONIC	COMMENT
o: p	3F00	3E 83	MVI A, 83	;INITIALIZE PPI2
	3F02	D3 EB	OUT EB	;
	3F04	21 E1 3C	LXI HL 3CE1	;MOVE SET POINT TO A
	3F07	7E	MOV A,M	
	3F08	D3 E8	OUT E8	;OUT PUT SET POINT VIA PORT E8
	3F0A	3E 70	MVI A, 70	;INITIALIZE PPI1
	3F0C	D3 E6	OUT E6	;
	3F0E	3E 70	MVI A, 70	;TRIGGER LATCH ENABLE
	3F10	D3 EA	OUT EA	;MONOSTABLE
	3F12	3E FD	MVI A, FD	;ENABLE BITS 7-4 OF PORT EA
	3F14	De E6		;
LOOP3	3F16	C3 16 3F	JMP LOOP3	;ENTER DO NOTHING LOOP

APPENDIX II

INSTRUCTIONS TO OPERATE THE SYSTEM

In this appendix, instructions to operate the controller are given. Since the 'PROMPT 80' is the main component of the controller it is necessary to know how to enter programs and execute them. Figure 24 is a sketch of the PROMPT 80 front panel. A complete description of the controls can be found in references 4 and 5 hence only a brief description will be given here.

The SYS RST, (System Reset), resets the system, initializes the registers and enters the monitor. The 'Monitor' is a program that resides in three 1K PROMS (Programmable Read Only Memories). MON INT, Monitor Interrupt, interrupts user program and enters the monitor. The USR INT traps the PROMPT 80 to 3C02. Since a condition similar to pressing the USR INT is created with the hardware of the controller, the Interrupt program starts at 3C02. Using the HEX DATA/FUNCTIONS (Referred as Hex Key Board) data, addresses and function codes can be entered (This point will be discussed shortly). There are 8 commands. The DISPLAY/MODIFY MEMORY together with the command is used to check and change the memory contents as well as to enter programs manually. In the following the Symbol [3] means press the key labeled 3. When the DISPLAY/MODIFY MEMORY key is pressed in the display labeled 'HEX DATA/FUNCTION' (See Fig. 24) 'dn' will appear. Thus [dn] stands for press DISPLAY/MODIFY MEMORY command. Next the required address is entered via hex key board ([3] [C] [0] [2] for example). Now pressing 'NEXT' [,], displays the contents of the location 3C02 in DATA field. Hence to enter 10_H in location 3C02 the sequence is as follows.

[dn] [3] [C] [0] [2] [,] [1] [0] [.] the last key shown above is END/EXECUTE, [.]. If consecutive memory locations are to be examined/modified, the END/EXECUTE [.] must be replaced by NEXT ([,]). Instead if NEXT is

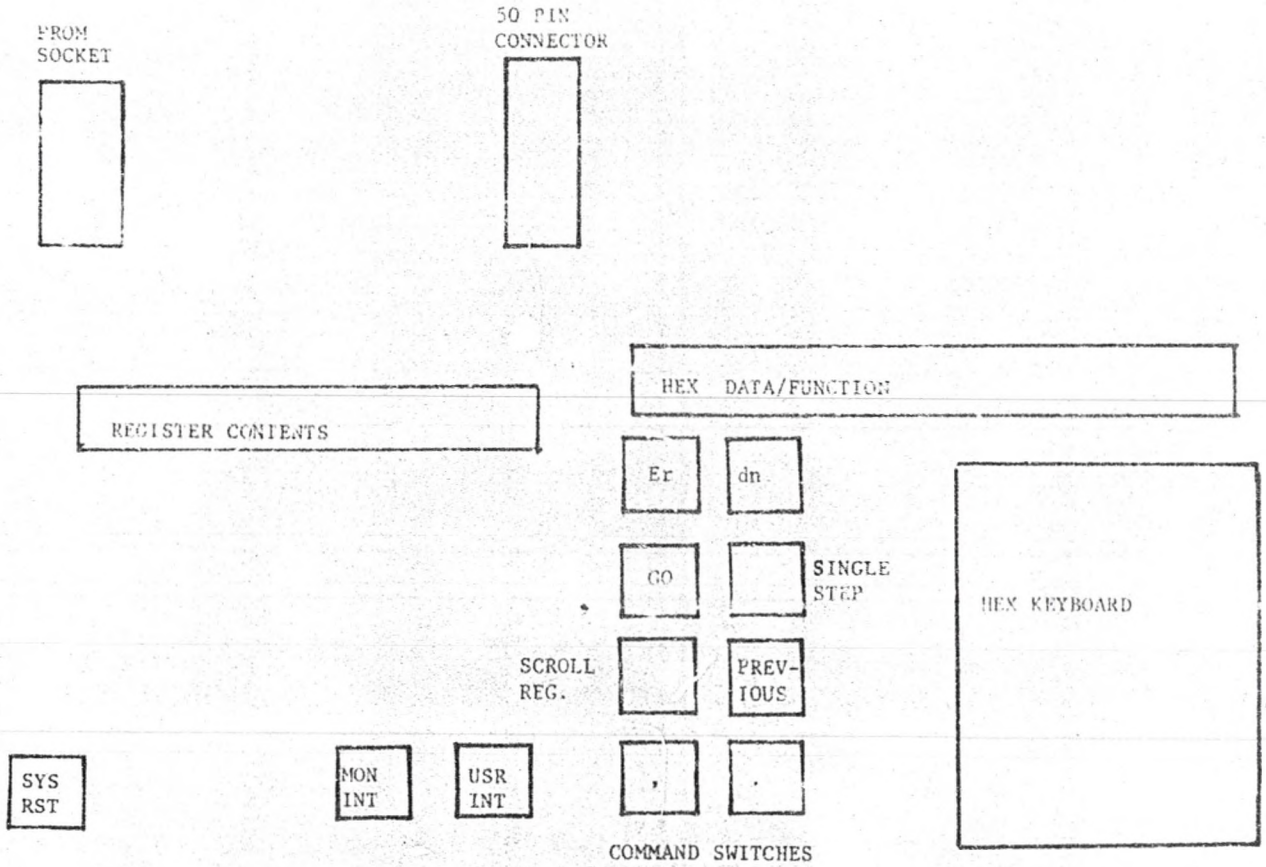


Fig. 24

SKETCH OF PROMPT 80 FRONT PANEL

replaced by PREVIOUS/CLEAR ENTRY, the previous location will be displayed. If the PREVIOUS/CLEAR ENTRY is pressed before entering the complete command, the command will be cleared. The 'SCROLL REGISTER DISPLAY' command is used to display various register contents on the 'REGISTER CONTENTS' display. A light emitting diode (LED) next to the three register groups indicate which registers are being displayed in the 'REGISTER DISPLAY GROUP'. The GO command is used to execute programs. To execute the controller programs (after they are loaded the following sequence is used.

[GO] [3] [D] [C] [A] [.]

In this case the the Start Up routine will be executed from the location 3DCA. If, as mentioned in Chapter II (See Integration routine), it is necessary to execute from 3DEB the following sequence

[GO] [3] [D] [E] [B] [.]

tarts execution from location 3DEB. (This is where the Start Up routine must be entered so that INTG OUT buffer is not cleared). The computer can be stopped at any time by pressing [SYS RST].

So far only one method of entering the program (via keyboard) is given. The PROMPT 80 has a serial I/O port and hence can be used with a teletype. The teletype connects to a connector on the back of PROMPT 80. It is advisable to turn on the PROMPT 80 first and then to turn the teletype LINE-OFF-LOCAL switch to LINE. To load a program using teletype paper tape reader it must be punched in a special format (See reference 4). Assuming that the tape is already in this format enter the following

[0] [0] [.]

when the END/EXECUTE key is pressed the paper tape will be read. The paper tape on which the controller programs are punched can be read into the computer using the above sequence. To punch paper tapes in the required format, first load the program in PROMPT 80 using the hex key board and command switches.

Then press the following

[1] [Starting address of the program to be punched] [,] [Ending address] [.] (note the first instructions for reading and punching paper tapes are functions hence the function field will show F0 and F1 respectively). If the sequence for reading paper tape is entered the display will show

F0 0

FUNCTION ADDRESS

The controller programs were also put on EPROM (Erasable Programmable Read Only Memory). The EPROM is a MOS device and must be handled very carefully. To transfer programs from an EPROM into PROMPT 80 memory first disconnect the 50 pin connector, place the EPROM in the PROM Socket and lock it. The EPROM contains the controller program from 3C00 to 3F1F. Enter the following via hex key board

[4] [3] [C] [0] [0] [,] [3] [F] [1] [F] [,] [0] [.]

The reading will be very fast. It is advisable to check the contents of at least a few locations (to use the controller the EPROM must be removed and the 50 pin connector reconnected). To put a program on the EPROM, the required sequence is [2] [STARTING ADDRESS OF PROGRAM] [,] [ENDING ADDRESS] [,] [0] [.]. The restrictions on the address are not clear from references 4,5. However, the sequence used to program the EPROMS in the present work is

[2] [3] [C] [0] [0] [,] [3] [F] [1] [F] [,] [0] [.].

The A/D and interface circuits were assembled on a single card (Referred here as A/D card) of HEATH ADD SYSTEM. Figure 25a shows the locations of the components on this card. The D/A is on another card (Referred as D/A card) of the HEATH ADD SYSTEM. Since a 50 pin connector is mounted on the A/D card the port E8 connections for the D/A card must be given via jumpers

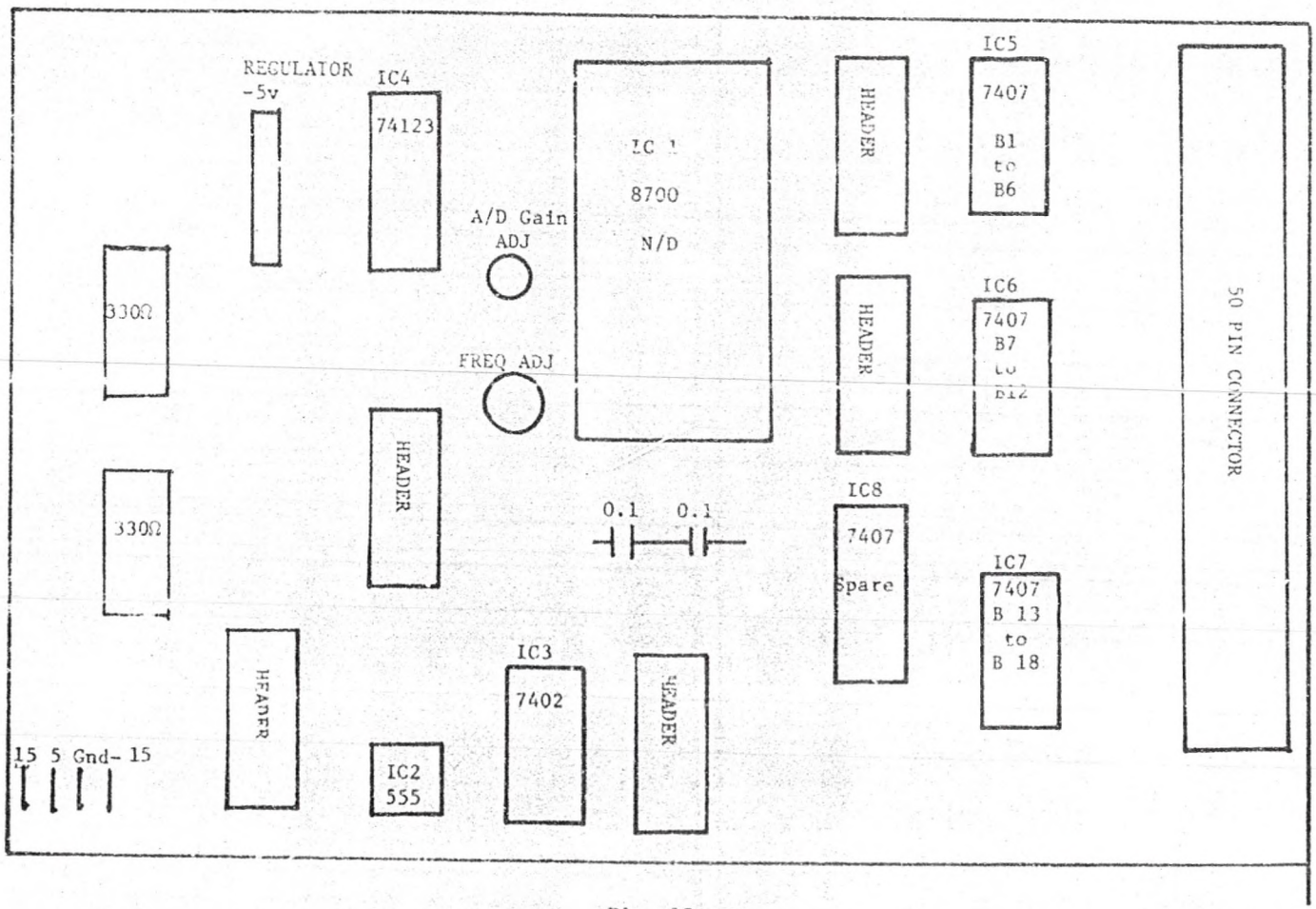


Fig. 25a

COMPONENT LAYOUT

between A/D and D/A cards. The necessary connections are shown in Figure 25b. The HEATH ADD SYSTEM provides the necessary power supplies by means of connectors on the lower side of the A/D and D/A cards.

The controller can be operated by following the steps given below.

(1) Connect the A/D and D/A cards with the necessary jumpers.

(2) Connect Output of the plant to the A/D input and input of the plant to D/A output.

(3) Enter the programs into PROMPT 80 either manually or using paper tape and teletype or from EPROM. If EPROM is used the 50 pin connector must be disconnected.

(4) Reconnect the 50 pin connector if it was disconnected in the previous step.

(5) Enter values of the necessary constants such as Set Point, K_D, K_I, K_p, and Mode code (MC).

(6) Execute Start Up routine by entering the following via hex key board

[GO] [3] [D] [C] [A] [.].

If the INTG OUT should not be disturbed use [GO] [3] [D] [E] [B] [.] instead of the above sequence.

The OUTPUT SET POINT routine can be executed by entering

[GO] [3] [F] [0] [0] [.].

To execute the typing routine alone, enter the starting address and the number of bytes to be typed (in hex) via hex key board (See Typing routine) followed by

[GO] [3] [E] [D] [0] [.].

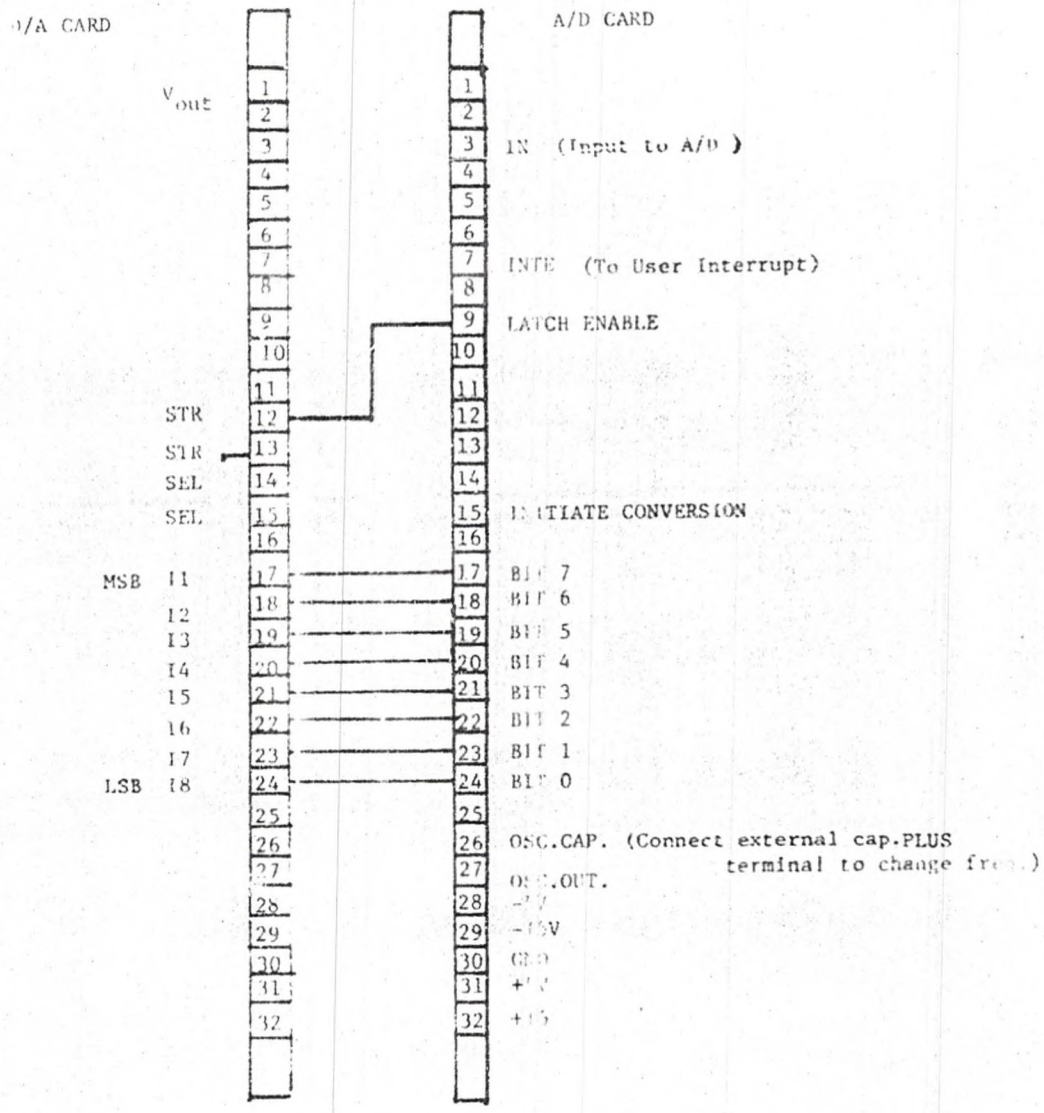


Fig. 25b

PIN OUT OF A/D AND D/A CARDS AND INTER CONNECTIONS

REFERENCES

(1) Benjamin C. Kuo, Automatic Control Systems, 3rd edition, Prentice Hall Inc., 1975.

(2) Kalsuhiko Ogata, Modern Control Engineering, Prentice Hall Inc., 1970.

(3) Adam Osborne and Associates, An Introduction to Micro Computers, Adam Osborne and Associates Inc.

(4) Intellec Prompt 80 Micro Computer Users Manual, Intell Corporation, 1976.

(5) (a) SBC 80/10 Single Board Computer Hardware Reference Manual, Intel Corporation, 1976.

(b) SBC 80/10 Reference Schematic Drawings.

(6) Monolithic CMOS A/D Converters, 8700 Series, Teledyne Semiconductor, 1977.

(7) 8 Bit D/A Converter. (The necessary information is in the Electrical Engineering Department, University of North Dakota, Grand Forks, North Dakota.)

(8) Peter Dransfield and Donald F. Haber, Introducing Root Locus, Cambridge University Press, 1973.