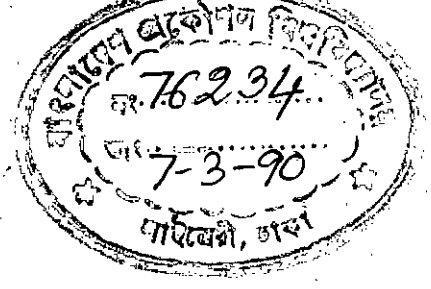
ONLINE MICROCOMPUTER CONTROL OF MULTIVARIABLE SYSTEM FOR CHEMICAL PROCESS

A THESIS

Submitted to the Department of Computer Science & Engineering, Bangladesh University of Engineering and Technology, Dhaka, in partial fulfilment of the requirements for the degree

of

MASTER OF SCIENCE IN COMPUTER ENGINEERING



by

MOHAMMAD ZAHIDUR RAHMAN.

December 31, 1989.

Bangladesh University of Enginering and Technology, Dhaka.

ONLINE MICROCOMPUTER CONTROL OF
MULTIVARIABLE SYSTEM FOR CHEMICAL PROCESS

A THESIS
by

MOHAMMAD ZAHIDUR RAHMAN.

Approved as to style and contents by:

i. ---------- 31/12/89

Dr. Syed Mahbubur Rahman
Associate Professor & Head
Department of Computer Science and
Engineering, Bangladesh University
of Engineering and Technology, Dhaka.

Chairman
and
Supervisor

ii. ---------- J. Zaman

Dr. Jasimuz Zaman
Professor
Department of Chemical Engineering
Bangladesh University of Engineering
and Technology, Dhaka.

Member
and
Co-supervisor

iii. ---------- M.S. Alam

Dr. Md. Shamsul Alam
Associate Professor
Department of Computer Science and
Engineering, Bangladesh University
of Engineering and Technology, Dhaka.

Member

iv. ----------

Mr. A.R. Khan
Senior General Manager
Electrical Instrumentation & Construction,
Bangladesh Chemical Industries Corporation,
BCIC Bhaban, Dhaka.

Member
(External)

CERTIFICATE

This is to certify that the work presented in this thesis was done by me under the complete supervision of Dr. Syed Mahbubur Rahman, Associate Professor and Head, Department of Computer Science and Engineering, and Dr. Jasimuz Zaman, Professor, Department of Chemical Engineering, Bangladesh University of Engineering and Technology, Dhaka, Bangladesh. It is also certified that neither this thesis nor any part thereof has been submitted elsewhere for the award of any degree or diploma.

Counter signed by
supervisor

(Dr. Syed Mahbubur Rahman)

Signature of the
candidate

Mohammad Zahidur Rahman

(MOHAMMAD ZAHIDUR RAHMAN)

Counter signed by
co-supervisor

(Dr. Jasimuz Zaman)

CONTENTS

ACKNOWLEDGMENT

This work has been prepared under the constant guidance and supervision of Dr. Syed Mahbubur Rahman, Associate Professor and Head, Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, Dhaka and, Dr. Jasimuz Zaman, Professor, Department of Chemical Engineering, Bangladesh University of Engineering and Technology, Dhaka. The author express his deepest gratitude to them.

The author takes this opportunity to express his heart-felt gratitude and thanks to Mr. Md. Fajlur Rahman and Mr. Mirja Rejaul Karim, Assistant Engineer, Chittagong Urea Fertilizer, Chittagong, for their kind cooperation and valuable information of process control.

The author is also thankful to Mr. Minhaz of Electronics laboratory, EEE for his kind help in the development of hardware. The all-out support and services rendered by the faculty members and staff of Computer Science and Engineering Department and Chemical Engineering Department, BUET are acknowledged with sincere thanks. Last, but not the least, I thank my parents for constant encouragement for my work.

ABSTRACT

Rapid technological development in digital computing system coupled with significant reduction in their cost have had a profound effect on how chemical plants can and should be controlled. In this work, a control system for a laboratory fixed bed catalytic reactor for methanation of carbon dioxide was designed with a PC as data logger and controller.

To handshake digital computer with methanation plant a hardware interface for input-- flow and temperature and output-- heater power control, stepper motor controlled flow control valve and solenoid valve control, was designed. The raw data in chemical process system is often corrupted by noise. To reduce the noise a digital filter program was written. To obtain desired system performance adaptive controller program was written and tested by simulation.

The complete hardware and software were developed for the Methanation project of Chemical Engineering Department and process computer used was XT compatible personal computer of Computer Science and Engineering Department.

CHAPTER 1
INTRODUCTION

CHAPTER 1 INTRODUCTION

1.1 INTRODUCTION

Computer can perform arithmetic calculation, can change order of calculations based on intermediate result, can store large amount of data and can transfer data from one place to another. These powers of a digital computer can be utilized using special equipment interfacing the chemical process to perform variety of jobs.

The computer and process can be interconnected in three ways -- off line, in line and on line (fig 1.1). In the off-line applications, process data are entered manually in to the computer and calculations are carried out by the computer. With the in-line configurations, data collections and entry require human intervention, but the computer operation can be interpreted by the operator and necessary calculation made immediately. In on-line applications, the computer directly connected to the process which oversees the automatic collection and transfer of data into the computer. On-line may be open loop or closed loop - depending on whether or not the computer controls the process.

The control of a chemical process plant is intricate because of

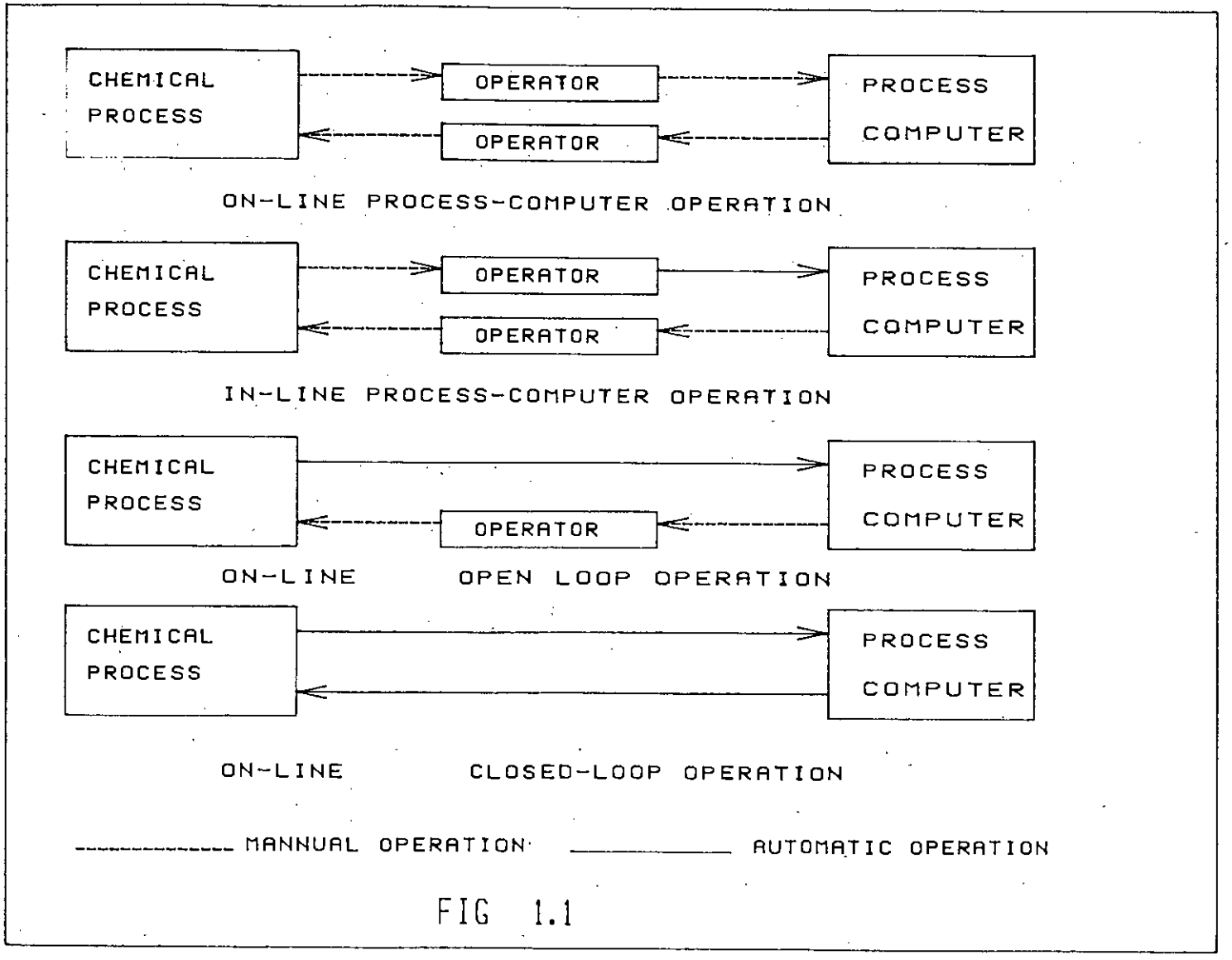

CHEMICAL PROCESS
OPERATOR
OPERATOR
PROCESS COMPUTER
ON-LINE PROCESS-COMPUTER OPERATION
CHEMICAL PROCESS
OPERATOR
OPERATOR
PROCESS COMPUTER
IN-LINE PROCESS-COMPUTER OPERATION
CHEMICAL PROCESS
OPERATOR
PROCESS COMPUTER
ON-LINE OPEN LOOP OPERATION
CHEMICAL PROCESS
PROCESS COMPUTER
ON-LINE CLOSED-LOOP OPERATION
MANNUAL OPERATION
AUTOMATIC OPERATION

FIG 1.1

the nonlinearities in the process variable and a need to satisfy a number of control objectives at the same time such as safety, process economy, specified throughput and quality. A chemical plant basically consists of a collection of mixers, separators, reactors, splitters and heat exchangers interconnected in a variety of ways to profitably produce output materials from given raw materials. Of all the units in a process plant, the chemical reactor plays the most significant role and is often regarded as the heart of a chemical plant. In addition to this significance, the chemical reactor often poses a nontrivial control problem. The most common variables which determine the performance of the reactor are the temperature, pressure and concentration of the chemical species. The overall behavior of the chemical reactor is often critically dependent on one or more of these variables. A small change in such variables can lead to drastic change in performance. The temperature of the reactor most often is the significant variable affecting the performance of the reactor in a profound manner. The control of the temperature is then the key to the satisfactory operation of a chemical reactor. The reactor most often requires a two dimensional mathematical representation. The design of a control system based on state space representation of the reactor results in a complex mathematical problem difficult even to solve computationally. The use of a computer to control such a reactor provides an opportunity to use a simple algebraic model for the process and then continually upgrade the model based on process

identification techniques and introduce optional control based on adaptive approach.

The incorporation of a computer in the control of a chemical reactor in place of a conventional analog controller introduces a number of difficulties as well. First of all, it introduces an incompatibility in signals; the process sends out analog signals while the digital computer can receive only discrete signals. In particular, the control system designer is now confronted with the following questions:

1. Analog signal produced by the measuring sensor or transducer is very low, and requires amplification to the prespecified range for A/D conversion.
2. The computer is physically located some distance from the controlled process. If analog signal is transmitted over a short distance, I/O interface may be close to the computer. Else to reduce noise, voltage loss and cable capacitance I/O interface is placed near the controlled process because digital signal is less sensitive to external noise.
3. Digital computer uses, processes and produces information in discrete time form. Hence we need to develop a mechanism that will convert differential equations describing the process to difference equations.
4. How fast will be the sampling time to produce discrete time equivalent of measured variable and what is the effect of sampling rate on the quality of control?
5. To analyze the stability of discrete time system new

analytical tools is required and these will be provided by the z-transformation.

6. What will be the stability conditions and tuning of a loop?
7. How can we use the tremendous computational power of a computer to implement some advanced notions of process control such as feed forward, adaptive, inferential, optimizing, expert system and so on?
8. Software implementation of the control laws is a new control design problem introduced by the digital computer. Operating systems provide efficient use of resources of computer system. Utility programs support the development of the application program. The application programs are written by user for the specific functions required by the control problem such as

   a) Monitoring the measured process variables at specified time interval.

   b) Executing the algorithms of the control laws.

   c) Coordinating the control actions to the various final control elements.

   d) Computing and changing set points.

   e) Computing and changing the values of the adjustable control parameter.

   f) Calling alarms if process variable exceeds preset limits and so on.

1.2 OBJECTIVE

In this work, it is intended to develop control system for a laboratory fixed bed reactor for the catalytic methanation of carbon dioxide and hydrogen.

During the time of the normal operation of the plant control action is necessary in the case of minor upset. In such a case, the disturbance is to be minimized by necessary corrective action with the help of control system. This avoids any serious incident, shut down and/or wastage of material/energy. Hence it is important that the control system be sensitive to the system variables.

The major objective of the work is therefore to design, construct and implement a digital control system for a chemical process involving the following tasks:

a) Hardware interface between microcomputer and process plant: The computer handles digitized values but the data available from the process are in the form of analog signals. These signals are to be converted to digital signals. Again digital signals sent from the computer to manipulate different process variables are to be converted to analog signals. The interfacing circuits composed of sampler, hold element, analog to digital converter and digital to analog converter have to be designed and constructed.

b) Necessary software to control the above hardware also has to be developed.

c) Signal conditioning software: The process data are usually corrupted by external noises and requires the use of filters to reduce noise. A digital filter has to be designed for this purpose.

d) Process control software: To maintain process at a given system performance, a PID process control algorithm will be drawn up with a provision to incorporate adaptive control.

CHAPTER 2
PROCESS CONTROL SYSTEM

CHAPTER 2 PROCESS CONTROL SYSTEM

2.1 INTRODUCTION

The design of the chemical process control becomes a tedious task because a lot of important and difficult problems arise from the different aspects of the design procedure [1]. Chemical processes are characterized by nonlinear, multiple couplings among variables, physically realizable variable measurement are often limited to a small number, picture of the control objective are sometimes not clear to the designer, even unknown and considerable uncertainty in the prediction of process behavior exists. Inspite of these shortfalls control system of a chemical process is called upon to satisfy source some or all the following features:

a)safety;

b)reliability;

c)goodness of control;

d)ease of start up and shut down;

e)ease of operation of the system;

f)production specification;

Now it is to be clear that to design a control system the designer has to consider the above points. But it also should be kept in mind that the controller, that is, the output of this design is always a matter of compromises and trade-offs [2].

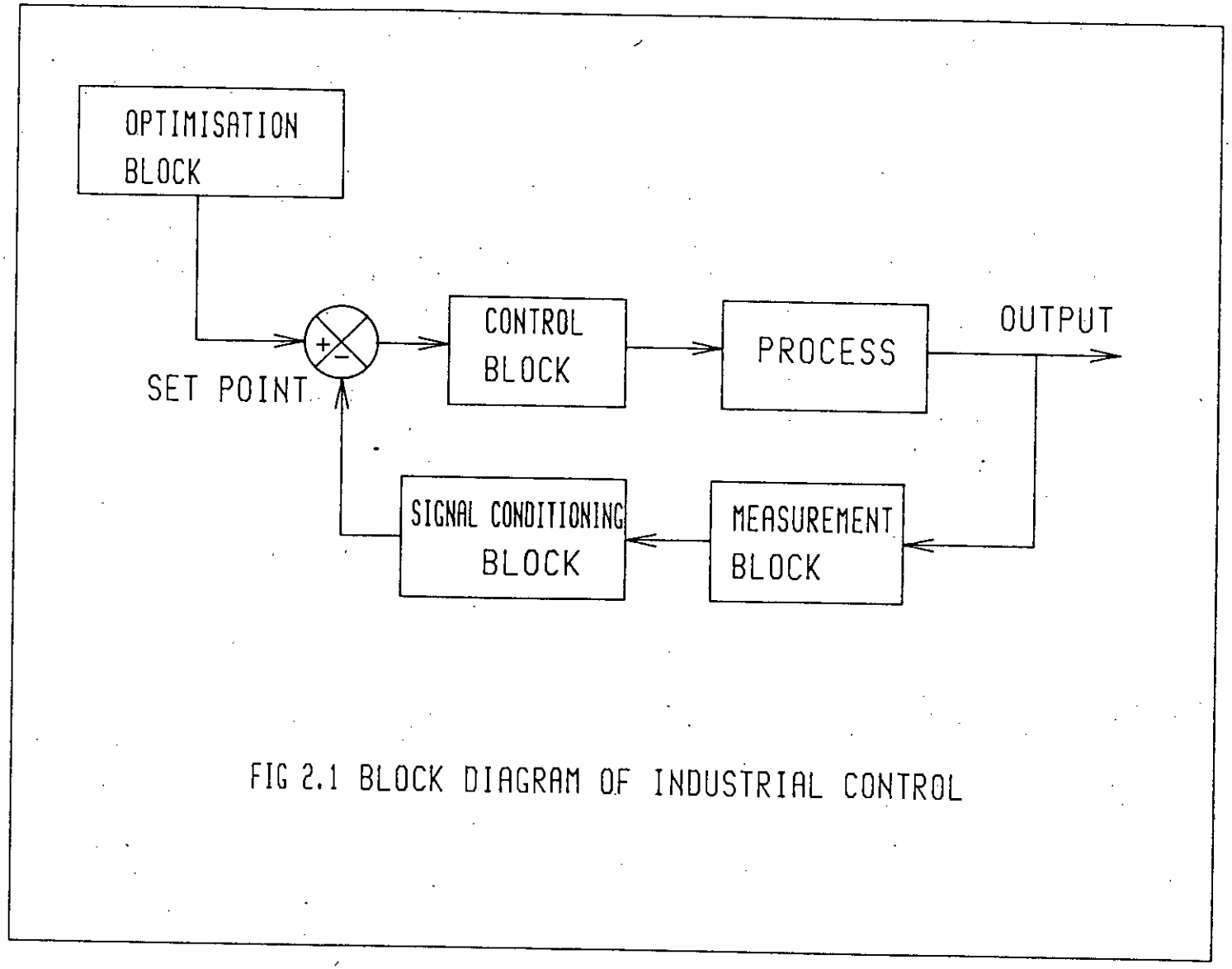


FIG 2.1 BLOCK DIAGRAM OF INDUSTRIAL CONTROL

Considerable work has been done on different chemical processes relating to the catalytic chemical reactor control. Jutan et.al [3] gave a view of collecting data from the process.

From a survey [4] it is shown that the most important issues in the technical maintenance of the control system are understanding the general process, improving yield, the availability of good analytical measurements, reactor dynamics, and the product quality. A large portion of the control of a chemical process can be done by using traditional methods based on past practice. In the case of the difficult nontraditional control problem, the designer has to use modeling, estimation, multivariable frequency analysis, adaptive control and steady state optimization. Most of the industrial reactors are of large size and yield a large volume, and yield and quality are the major economic incentives for control. Different types of reactors lead to different solution for the controller.

The building block of industrial control is shown in figure 2.1. This consists of measurement block, signal conditioning block, control block and optimization block.

2.1.1 MEASUREMENT BLOCK

The basic building block of industrial control is measurement block. Conventional measurements pressure, temperature, flow, and level are used for reactor regulatory control. The basic

elements of the measurement block are :

1. Standard PTLF(pressure,temperature,level,flow) devices;
2. Product analysis;
3. Density;
4. Surface tension;
5. Refractive index;
6. Viscosity;
7. pH,and
8. Laboratory sample results.

2.1.2 SIGNAL CONDITIONING BLOCK

Signal condition block processes raw measurement signals using scaling and filtering techniques. Conditioning cleans up the signal from both the external noises and internal noises from the sensor in order to use the data effectively in a closed loop reactor control scheme. The common techniques for cleaning up are the exponential filtering (RC filters), flow data reconciliation, Kalman filtering and other digital filters.

Exponential filtering involves passing the input signal through a first order lag in order to eliminate the high frequency noise.

Flow data reconciliation uses redundant information to improve the quality and utility of the total measurement system. The technique works by fitting a measurement to a simple model using a least square technique. The output of the model calculations are used as the measurements.

Kalman filtering can be used a filter, a signal smoother or an estimator block which calculates control variables from the conditioned measurement signal. Digital filter uses the different types of algorithm to smooth out unexpected noises.

2.1.3 CONTROL BLOCK

Control block contains the algorithm for the control scheme. Options for this block are:

1. Traditional control: feedforward (including ratio and impulse), feedback(including PID and cascade);

2. Advanced traditional control: overrides, adaptives, and nonlinear feedback;

3. Non traditional control: compensator design using multivariable frequency analysis, linear quadratic gaussian control, nonlinear model based control;

4. Steady state control: steady-state, model-based techniques, cumulative sum (CUSUM) techniques, product wheel scheduling and accounting; and

5. Others: pole placement model, dynamic matrix, optimal servo, dynamic programming, statistical and expert system, state variable, model-based, stochastic and selftuning control.

2.1.4 OPTIMIZER BLOCK

Optimizer block is the highest level in the control hierarchy. It determines the optimal operating conditions for the lower level control blocks and attempts to maximize or minimize an economically based objective function. Optimizers are useful for processes where high yield is an economic justification for improved control. For specialty product, improving quality is the main goal and meeting this objective usually involves good regulation and supervisory control. Feedforward techniques are perhaps the most effective way of regulating the feed to the reactor because of the disturbance rejection they provide. The optimizer block has to perform the following tasks :

a) Identify that current setpoints no longer correspond to optimization.

b) Find new optimum operating conditions.

c) Bring process to the new optimum operation by changing set point values.

2.1.5 PROCESS CONTROL: FROM HARDWARE POINT OF VIEW

The process control diagram from the hardware point of view is shown in figure 2.4. The sensor and transducer transform measured variable to electrical signal which is sampled by sampler. The analog to digital converter ADC converts the sampled data to digitized data and feeds to computer control algorithm.

The computer control algorithm gives out a digital data to digital to analog converter (DAC) which converts a digital data

to discretized signal. The hold element changes the discretized data to analog data compatible to final control element.

2.2 OVERVIEW OF ADVANCED CONTROL

Advanced control is important for processes with true multivariable problems as well as for processes with nonlinearities such as lengthy dead times, inverse response or time varying behavior.

Advancement in computer technology, information processing, automatic analyzer and intelligent transducer give a thrust to control theory. These advances on the hardware side are accompanied by the new software and algorithm break through [2].

Besides the conventional methods, there are new area of control theory, for example, model predictive control and robust control. Model predictive control (MPC) has become widely known as dynamic matrix control (DMC) and model algorithmic control (MAC). As the MPC algorithm includes the dynamic model of the process to be controlled, hence on the knowledge of previous control actions and current output measurement, the control algorithm predicts the future and present control actions on the future process output. The success of the MPC is mainly based on the fact that

this technique does not require state space models of transfer matrix models but utilizes the impulse response as a simple and intuitive process description. MPC has the inherent on line optimization, which makes it possible to deal easily with 1) system having an unequal number of inputs/outputs, 2) constraints on the manipulated or controlled variables and 3) the failure of the actuator. MPC (model predictive control) is suitable for the multivariable systems with objectives and constraints instead of the simpler situations. The MPC methodology consists of the following elements.

-Linear model relating the manipulated variables and measurable disturbances to the output of interest.

-Prediction of the outputs of interest over the future time horizon, corrected via feedback.

-Computation of the future manipulated variable moves to make the prediction of the outputs and manipulated variables satisfy some performance criteria.

Robust process control uses the information of allowable change in operating conditions, degree of mismatch tolerated by the model/plant, or the characteristics of the good model for control system. In monitoring and controlling process of chemical engineering the knowledge of these quantities which are not directly measurable is often required or atleast desirable. The structured singular value (SSV) approach of Doyle employs the

correct philosophy of approaching the robust issue: use an uncertainty description of the process at the design stage to obtain a controller that will not perform worse than desired on the true process. In this method the designer has to specify uncertainties in the gains, time constants etc. in given elements of the system transfer function matrix [27]. As a result a less conservative stability criterion is obtained.

In practice a decentralized control (multi loop) structure is preferred for ease of start up, bumpless auto/manual transfer and fault tolerance in the event of actuator or sensor failure. Practical design starts with selection of the variables that are to be manipulated and measured. After the variables are chosen, the structure of the controller is selected. This determines which manipulated variables are changed and on which errors the changes are based. Proper control system structuring can have a significant effect on the reliability. Steady state gains, singular value decomposition, relative gain array and Niderlinski indices are used for control structure selection.

2.3 THE PROPOSED CONTROL SYSTEM

2.3.1 PROCESS DESCRIPTION

The process for the computer control is the catalytic methanation in a fixed bed reactor. The process uses H_2, N_2 and CO_2 as the process gases. A schematic diagram of the process is shown in figure 2.2. The gas flows can be controlled by the stepper motor controlled valves. The ratio of the gases is selected by the requirement of the reaction. The solenoid valve is used for shutting off the gas flow to the reactor at emergency. Preheater is provided with electric heater to heat the gas mixture before entering the reactor vessel. To study the reactor vessel dynamics the reactor is provided with thermocouples. To select pressure of the reaction the valve-5 is provided. For the emergency to release the pressure of reactor, the solenoid valve-2 is placed in parallel to the outlet.

The monitoring parameters for the plant are the flow rates of the gases, temperatures of the reactor and the preheater, and pressure of the reactor vessel (Q_1, Q_2, Q_3, Q_4, T_p, T_r, P_r). The controlled parameters are inlet valves of the gases; control valve for mixture; pressure control valve; heater power controller and emergency solenoid valves.

The control system of the process is shown in figure-2.3. The

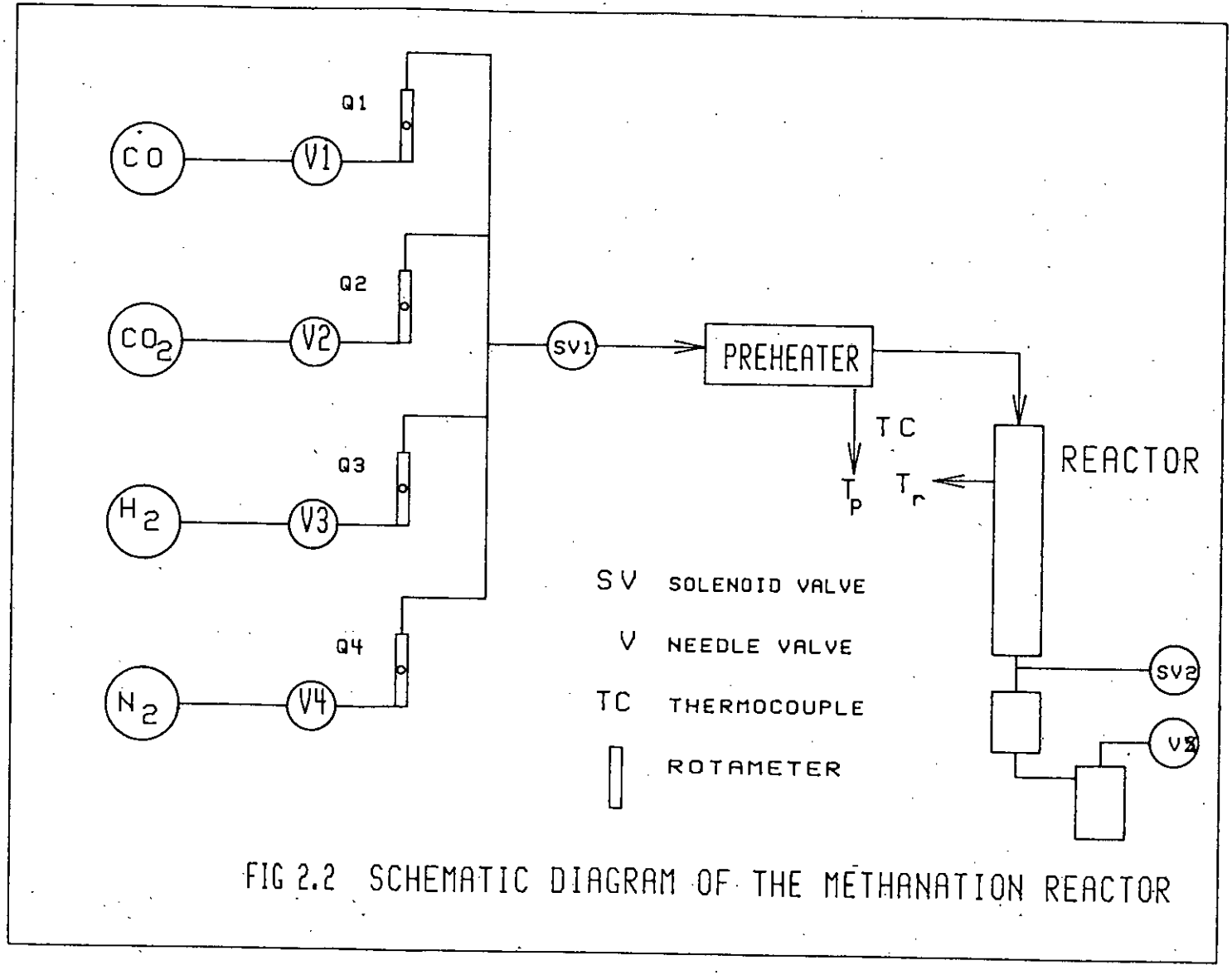


FIG 2.2 SCHEMATIC DIAGRAM OF THE METHANATION REACTOR

process controller is the microcomputer. The data logger and the processor chosen for this purpose is the microcomputer itself. The raw data which is acquired from the system is passed to the microprocessor. The data is filtered digitally and passed to the process controller at the same time to the data logger.

To understand the process controller action it is necessary to know 1) what units are in the plant and their sizes; 2) how they are interconnected; 3) the range of the operating conditions; 4) possible disturbances, available measurements and manipulations and 5) what problems may arise during startup or shutdown of the plant.

A general description of the plant is as follows :

1. The feed streams (H_2 and CO_2) with nitrogen are mixed.
2. The mixture above is preheated by the electric heater before entering the reactor.
3. The reactor previously stabilized at the reaction temperature, develops a temperature profile because of the exothermicity of the reaction.
4. The reactants leave through the pressure control valve to the atmosphere. Part of this is sent through a sampling line to the gas chromotograph for analysis.

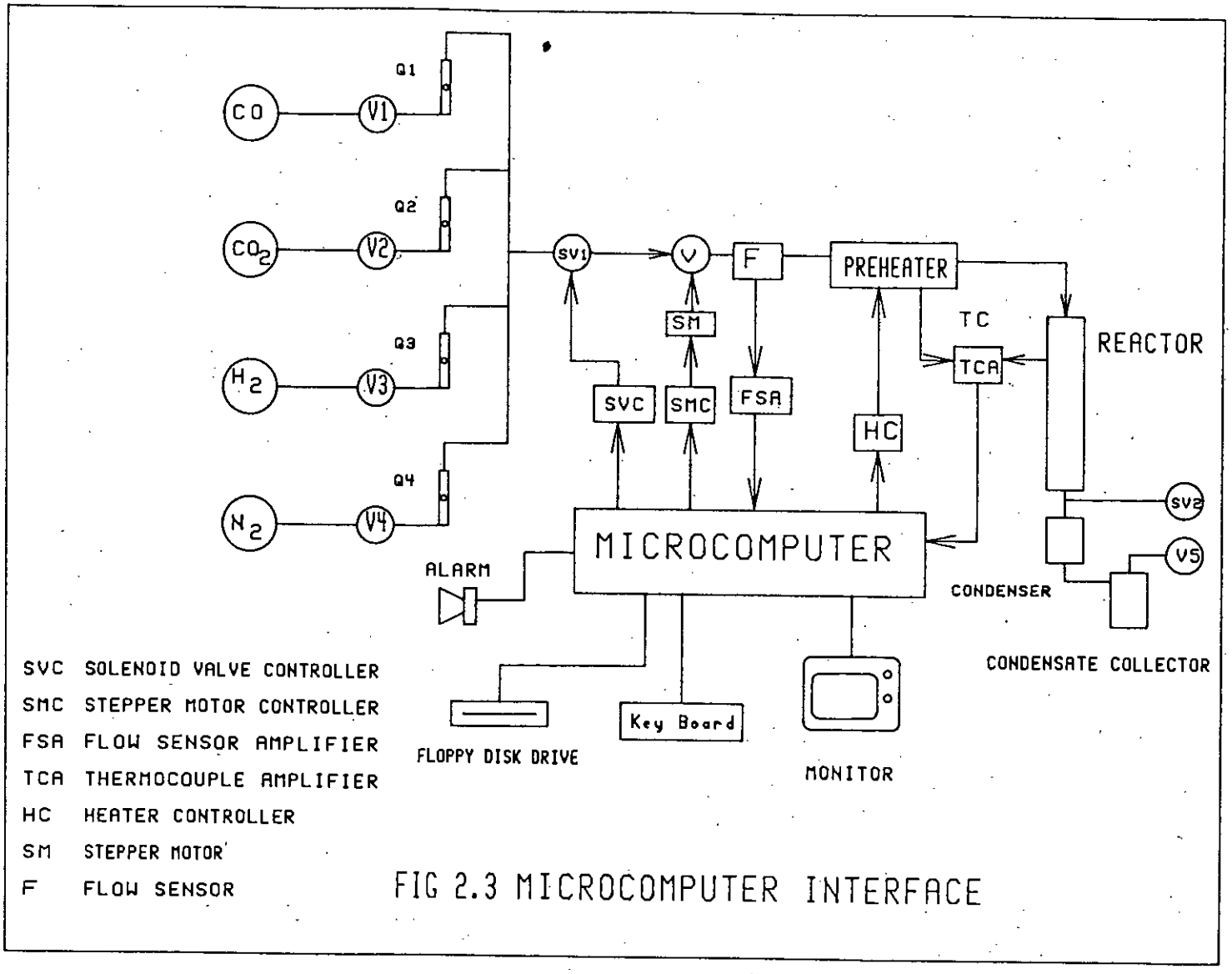


FIG 2.3 MICROCOMPUTER INTERFACE

2.3.2 HARDWARE AND SOFTWARE BLOCKS

For the process described above,it is necessary to design hardware for heater power control (HC), stepper motor control (SMC) for controlling flow and solenoid valve control(SVC) (figure 2.3). To measure flow, flow sensor and required hardware, and to measure temperature thermocouple amplifier are to be designed and constructed. This sensor data is to convert to digital data by IBM Analog to Digital converter (ADC) card resided in computer slot. The flow sensor amplifier (FSA) and thermocouple amplifier (TCA) output is made compatible with the ADC card.

The hardware cannot independently control the process. The total process control is done with close cooperation of hardware and software blocks as shown in figure 2.4.

The software block is the heart of the operation of the process. These softwares can be classified in three categories: a)filter software, b)controller software, and c) hardware interface software.

a) Filter software: The raw data from the analog to digital converter card is corrupted by noise, hence to reduce the noise of flow and temperature data, it has to design a filter and to develop filter program. The sampling time for temperature and flow is to be chosen to optimize the use of the computer.

b) Controller software: To operate the total process in desired

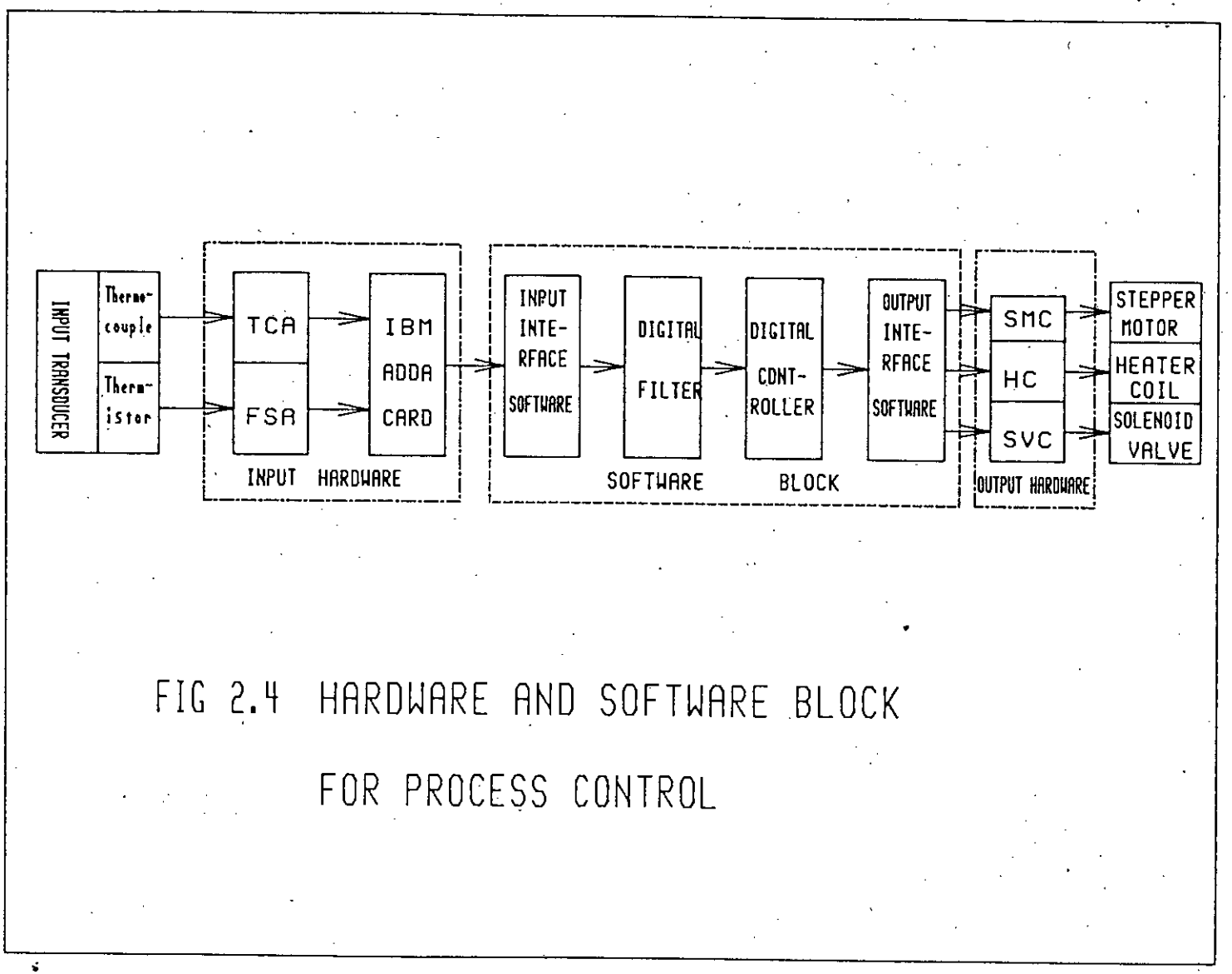


FIG 2.4 HARDWARE AND SOFTWARE BLOCK FOR PROCESS CONTROL

operating environment, controller algorithm processes the system input data and produces the output for the final control elements, stepper motor controller and heater power controller. Adaptive control, selftuning and autotuning method, are used for control algorithm development.

c) Hardware interface software: To convert analog data to digital a software requires to operate analog to digital converter card (ADC). To operate stepper motor controller (SMC), heater controller (HC) and solenoid valve controller (SVC) special programs are required.

CHAPTER 3
THE PROCESS CONTROL SYSTEM HARDWARD DESIGN

CHAPTER 3 THE PROCESS CONTROL SYSTEM HARDWARE DESIGN

3.1 STUDY OF HARDWARE REQUIREMENTS

The process of methanation plant requires the measurement of flow and temperature of the reactants.

Flow measurement is an important parameter for all chemical process plants. A thermistor based flow meter has been designed for the flow measurement [5][6][7]. The instrument is based on the theory of self heating of the thermistor and the resistance of the thermistor is changed with the change of temperature. The thermistors used in this flow meter are 3 mm, 1 Kohm. One of the thermistors is suspended in the tube of flow stream and the other thermistor is placed in a T-section at zero flow. This is the reference thermistor. Both thermistors are supplied with a constant current source. The heat produced by self heating soon brings thermistors to steady state. At this state the voltage difference of the thermistors is zero. This signal is amplified and sent to analog to digital converter (ADC). The main advantage of this type of the flow measurement is the linearity of the transducer i.e. the flow is directly proportional to the voltage of the instruments. The time constant of the instrument is in the range of a few seconds. This can be reduced to the range of few hundred millisecond by use of miniature thermistor. The differential amplifier used for this purpose is such that the

noise introduced in the line connecting the thermistors and the amplifier can be partially eliminated.

The temperature is measured by the thermocouple and an instrument amplifier is designed to amplify the voltage of the thermocouple to match the voltage range of the ADC .

The heater power sources commonly used are air, water, steam, electricity, oil and gas. The control element for the supply of the energy are contactors, blowers, electric motor or pneumatically-operated dampers and valves, motor operated variacs, saturable reactors, and time proportioning or phase fired SCRS . For computer control, any one of the above source and control element can be chosen. In the present work, heat is supplied by electrical element and SCR control is used.

The SCR controls the power flows to the heating elements of the heater. The power control can be achieved by the phase control of the AC or allocating a time proportion. The last method operates on the fact that the power is on for a period and off for another period. i.e. power is applied to the load for a percentage of a fixed cycle. If cycle time is reduced to one half of the power line period, then the proportioning action is referred to as the phase fired control.

Both of these methods have some advantages over the other. But to

control the heater power accurately phase fired control is preferable. For the control purpose the phase fired control to the load is selected. To control the heater power a special DAC is constructed. This gets an eight bit data from the computer. According to the data received by the DAC the firing angle of the SCR is selected and power flow to the heating element is controlled. The response of the DAC completely depends upon the heater coil.

To control the flow of the gas, the stepper motor controlled valve is used. The stepper motor steps are selected by the computer data steps and stepper motor controller is hardware interrupt driven.

At emergency, to stop the gas flow and to release the gas to atmosphere a solenoid valve controller circuit is designed.

3.2 DESIGN OF HARDWARE UNITS

3.2.1 HEATER D/A CONVERTER

The circuit for phase fired control of SCR/TRIAC is shown in figure 3.2-3.7. The input to the controller is digital firing angle data. This data is used to set the firing angle of the SCR/TRIAC. This controller design consists of 1) Firing angle data register, 2) Zero cross detector, 3) Phase shift circuit, 4) SCR driver and 5) Reference clock generator.

i) Data register : It consists of an eight bit register 74198. It stores data from computer upon request.

ii) Zero cross detector : This section sends a pulse to the phase shift circuit when the power cycle crosses zero average level. A 220-6V step down transfer output is applied to inverting input of an operational amplifier 741 and non-inverting input is connected to ground. This makes a zero cross detection circuit, the output of which is a 50Hz square wave. A 74123 is connected to monostable circuit such that it responds to both rising edge and falling edge. As the output of zero cross detector is connected, a 2 micro-second width pulse at each edge of 50 Hz square wave is the output of this circuit.

iii) Phase shift circuit : This consists of two divided by 16 counter, 74163, so that divided by 256 synchronous counter is

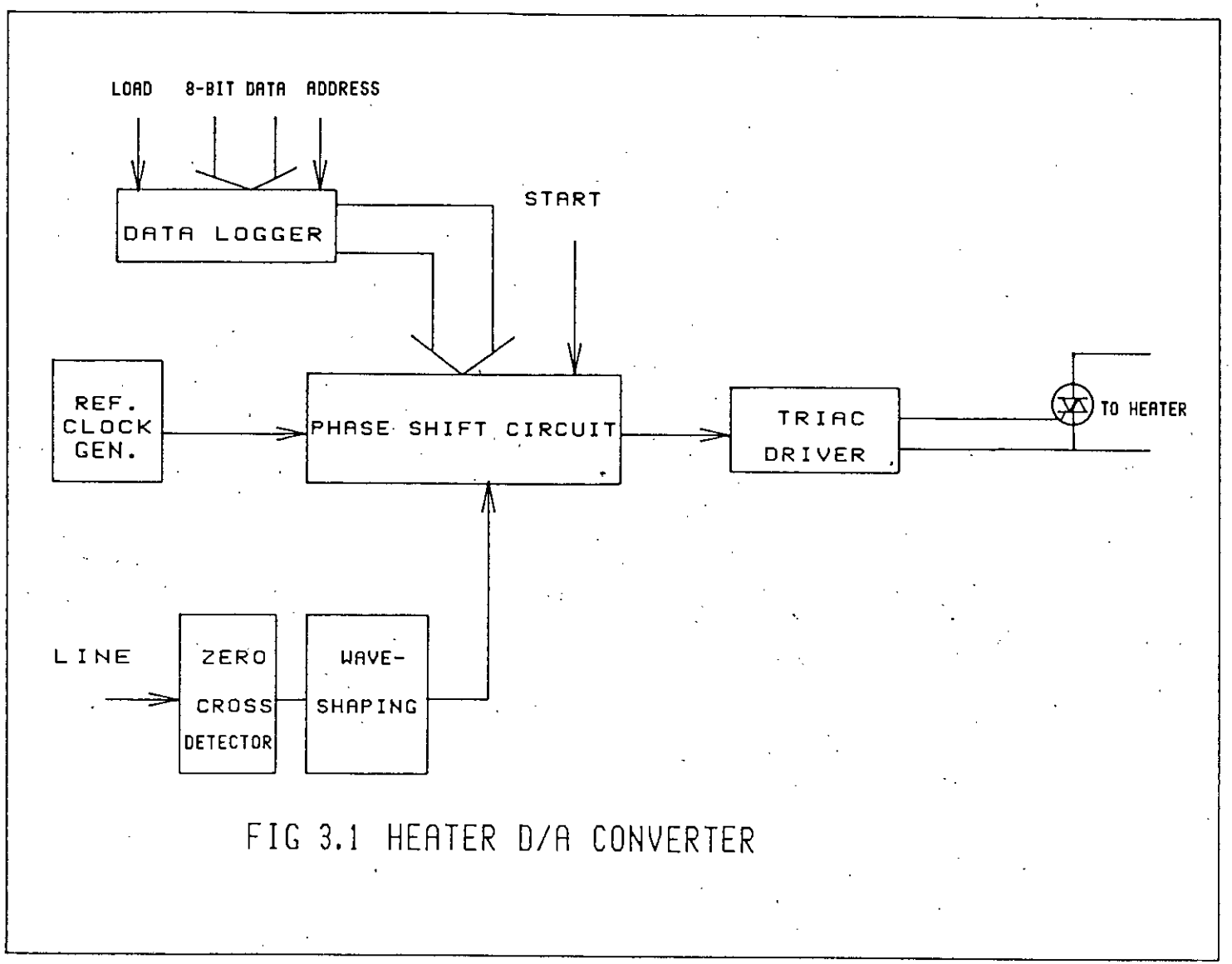


FIG 3.1 HEATER D/A CONVERTER

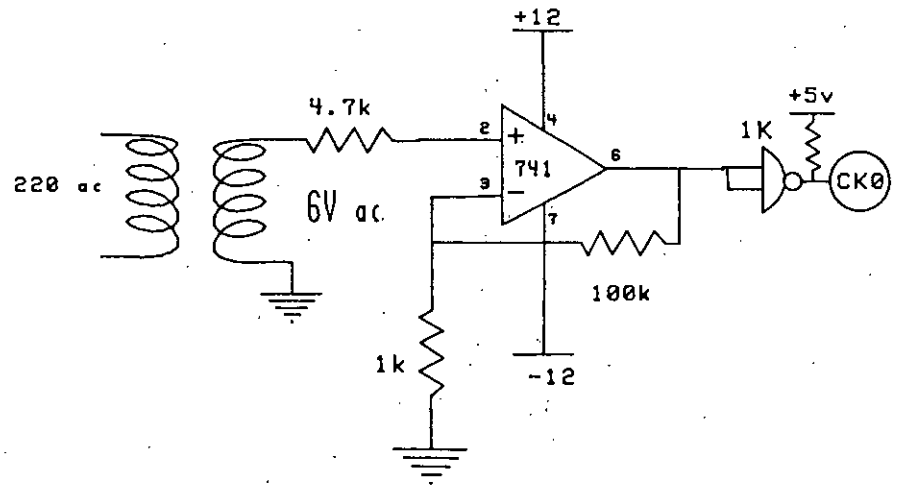
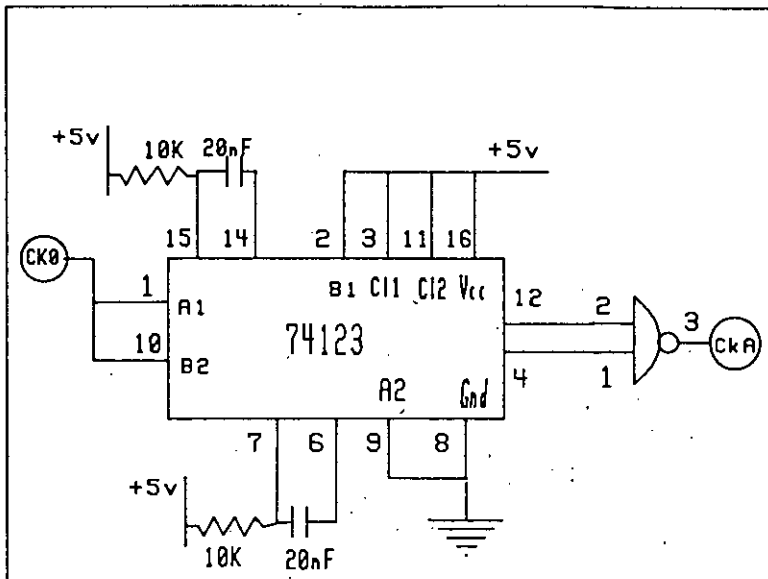
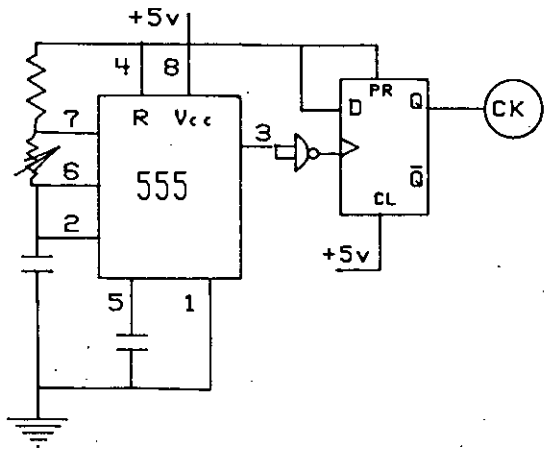


FIG 3.2 ZERO CROSS DETECTOR



WAVE SHAPING

(a)



REFERENCE CLOCK GENERATOR

(b)

FIG 3.3

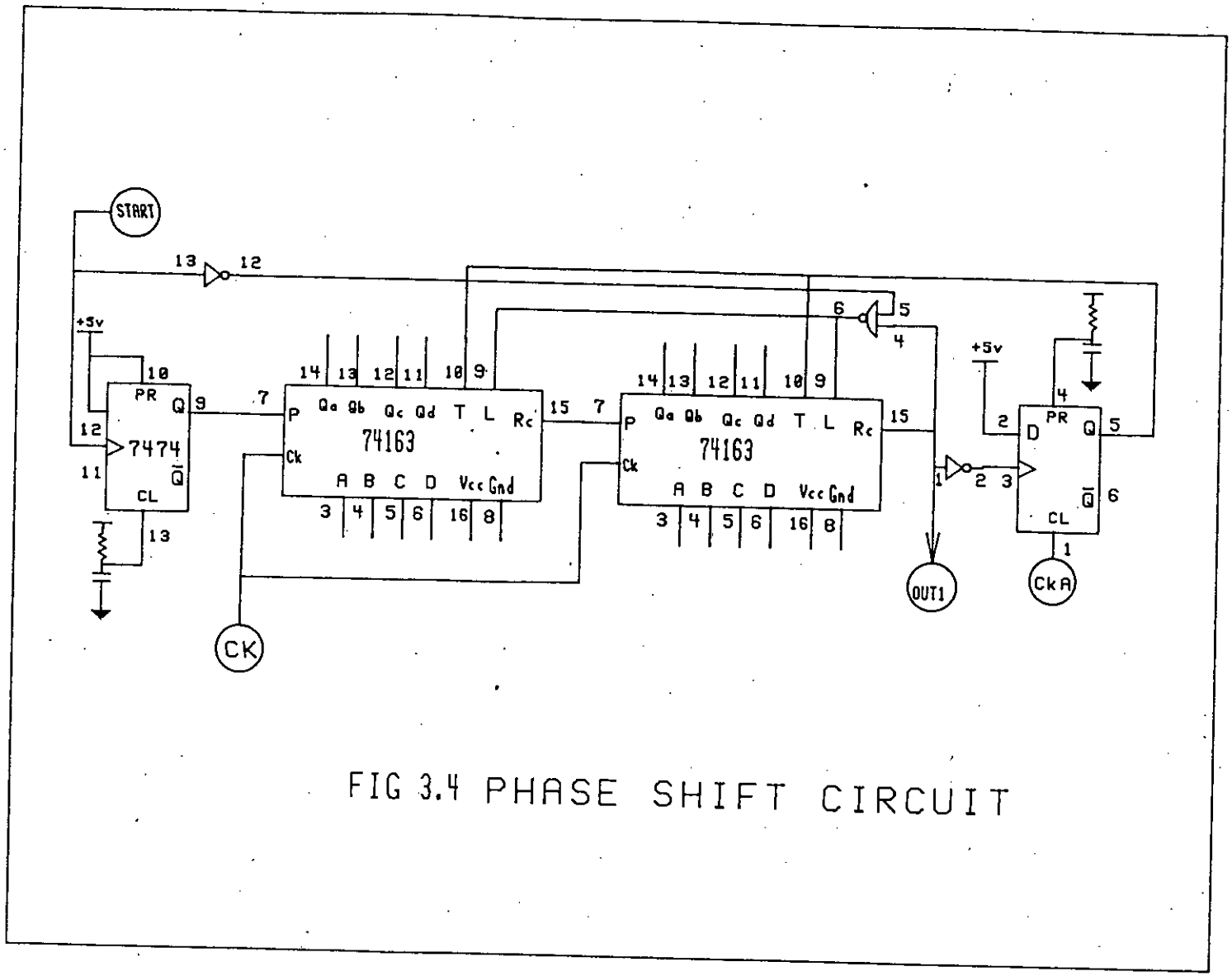


FIG 3.4 PHASE SHIFT CIRCUIT

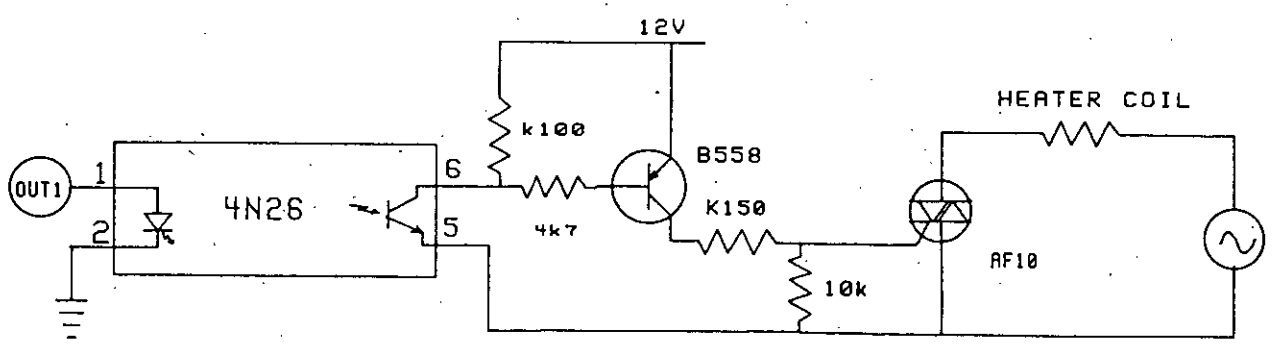


FIG 3.5 TRIAC DRIVER

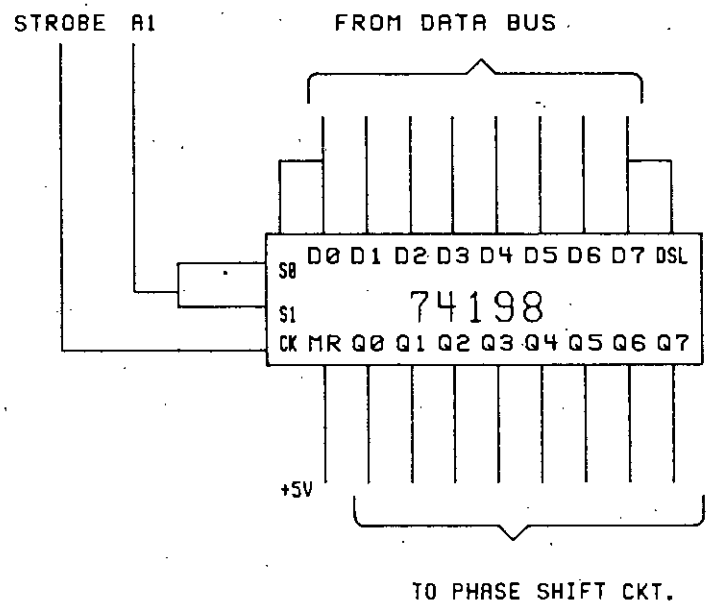


FIG 3.6 DATA LOGGER

available. At the start, a start pulse loads 8 bit data from data register and P enable input of 1st 74163 is high. But T enables are low as T enables are connected to $\bar{Q}$ of 1/2 7474. When CkA goes low, $\bar{Q}$ is high and count starts. While count terminates, 74163 are loaded and $\bar{Q}$ goes low and count disables. Again when CkA goes low cycle repeats. The terminal count output of last 74163 i.e. a delayed pulse according to the data from firing angle data register is passed to the next section, SCR driver.

iv) SCR/TRIAC driver : The SCR/TRIAC is fired by the pulse from shift circuit. Optocoupler is used to isolate the high voltage section from the low voltage input circuit. A PNP transistor is used to drive the gate of SCR/TRIAC.

v) Reference clock generator : 555 timer is used for the reference clock generator. The frequency of clock is 51.2 KHz which is halved by a T flip-flop because the 10 millisecond time period is divided in 256 division.

The pulse trains of different sections are shown in the figure 3.7 and 3.8.

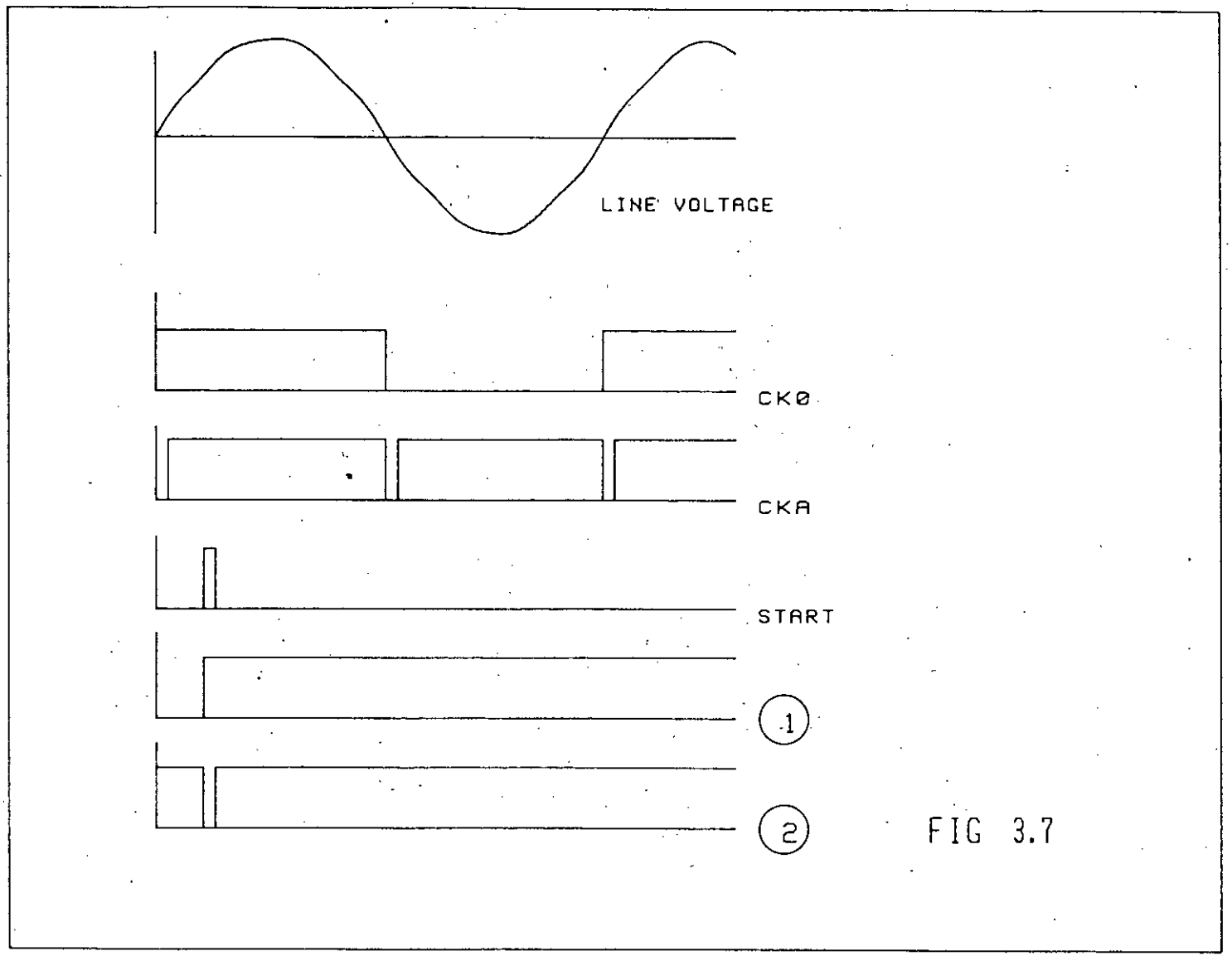


FIG 3.7

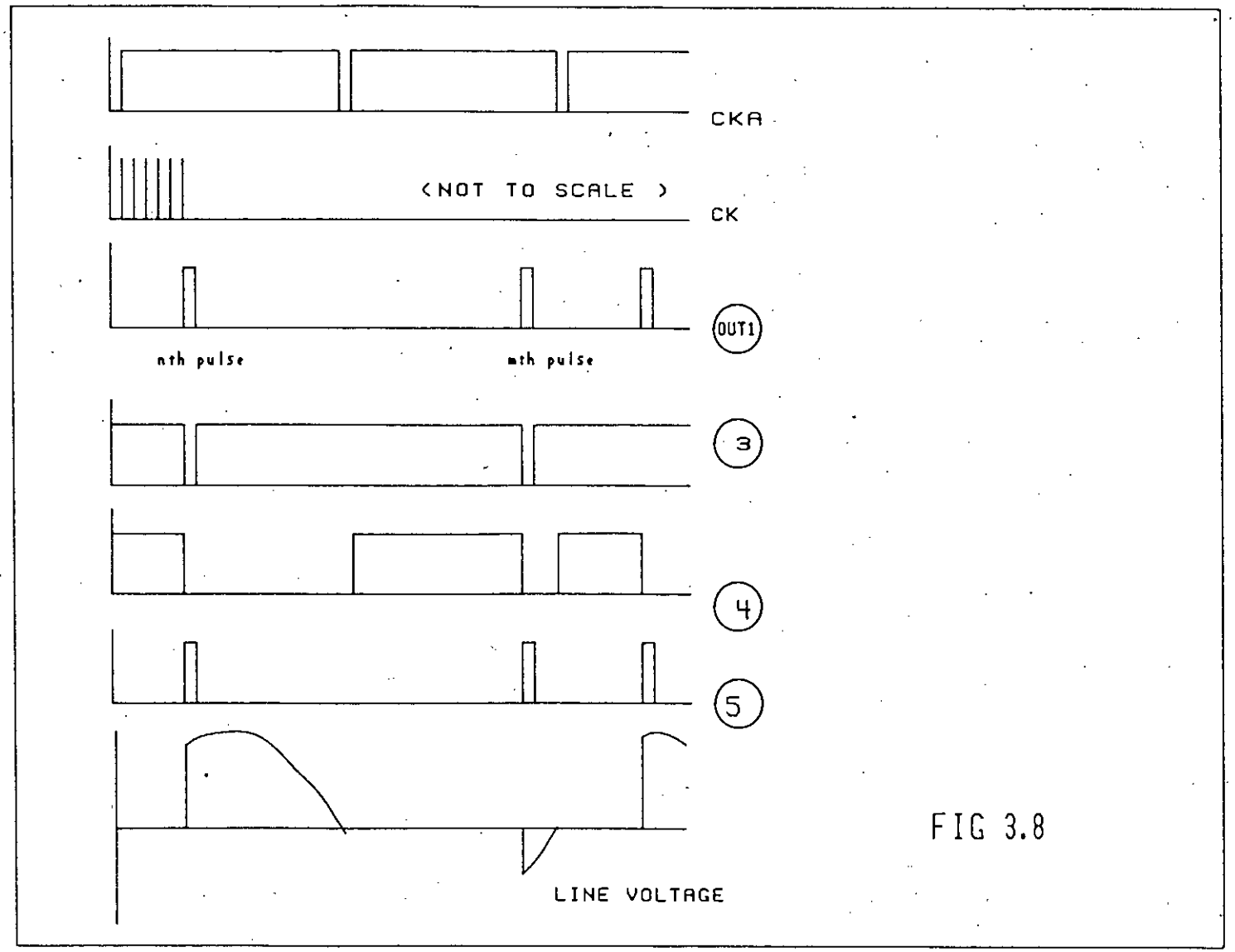


FIG 3.8

3.2.2 FLOW TRANSDUCER

To measure flow of the gas there are several methods. Thermal conductivity flow measurement can measure the flow in the range of 1cc to 1000,000 cc/min. The instrument is based on the fact that the cooling effect of the gas crossing a hot surface is directly proportional to the fluid velocity and volumetric flow rate [6] and that the resistance of the thermistor changes if the temperature of thermistor changes [11]. The thermistor has a property of self heating i.e. current through the thermistor makes thermistor hot and its resistance decreases and current increases. This cycle goes on until it comes to an equilibrium with the environment of the thermistor. At steady state, there is constant voltage across the thermistor depending on the environment. The environment of the thermistor changes, heat dissipation equilibrium changes and current through the thermistor is affected. This effect is reflected on the voltage across the thermistor.

To construct flow sensor, two thermistors were placed at a distance. One was placed so that its environment cannot be changed and it was surrounded by the sample of the gas whose flow was to be measured. This thermistor is known as reference thermistor. The other thermistor was placed in the sample of gas (figure 3.9) called measurement thermistor.

At zero flow the voltage difference of two thermistors is zero if

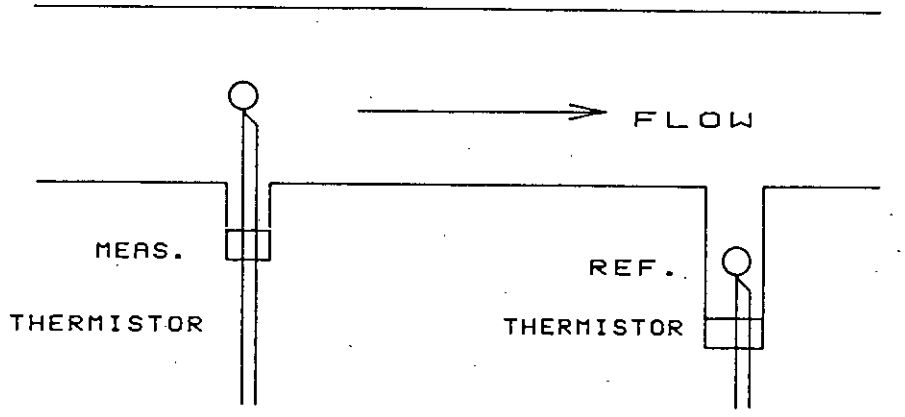


FIG 3.9 THERMISTOR PLACEMENT

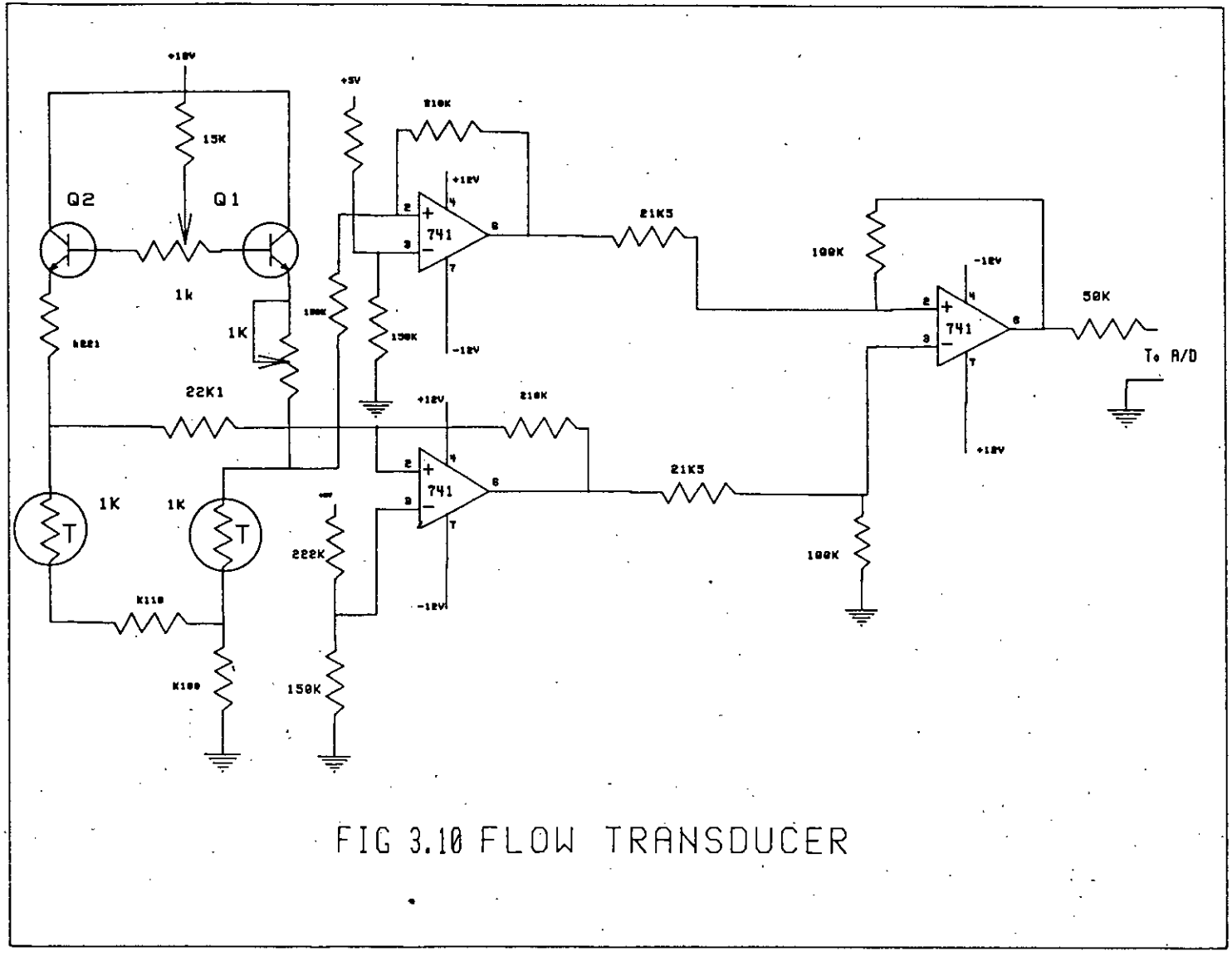


FIG 3.10 FLOW TRANSDUCER

thermistors are perfectly matched pair. The current through the thermistor was regulated by the constant current supply by Q_1 and Q_2. When a flow of gas is introduced, the environment of the measurement thermistor changes with respect to the reference thermistor and a voltage difference occurs. This voltage difference is directly proportional to the flow of the gas. To reduce the noises from the connecting wire, instrument amplifier, composed of three 741 operational which has large input impedance was used. The amplification of the amplifier can be adjusted as the analog to digital converter demands (figure 3.10).

The advantage of this type flow measurement is the reliability and low response time and can have an improved precision. Due to the relatively simple construction detail, a very satisfactory mathematical model of the characteristic behavior can be obtained.

3.2.3 VALVE CONTROLLER

For the gas flow control valve, opening and closing are controlled by the stepper motor. To control the stepper motor a special stepper motor controller is designed. A detail description of stepper motor control is given in [9]. The valve controller composes of a stepper motor, stepper motor controller and data register. The stepper motor used has the stepping angle 18 degree. The motor is coupled with the valve to be controlled.

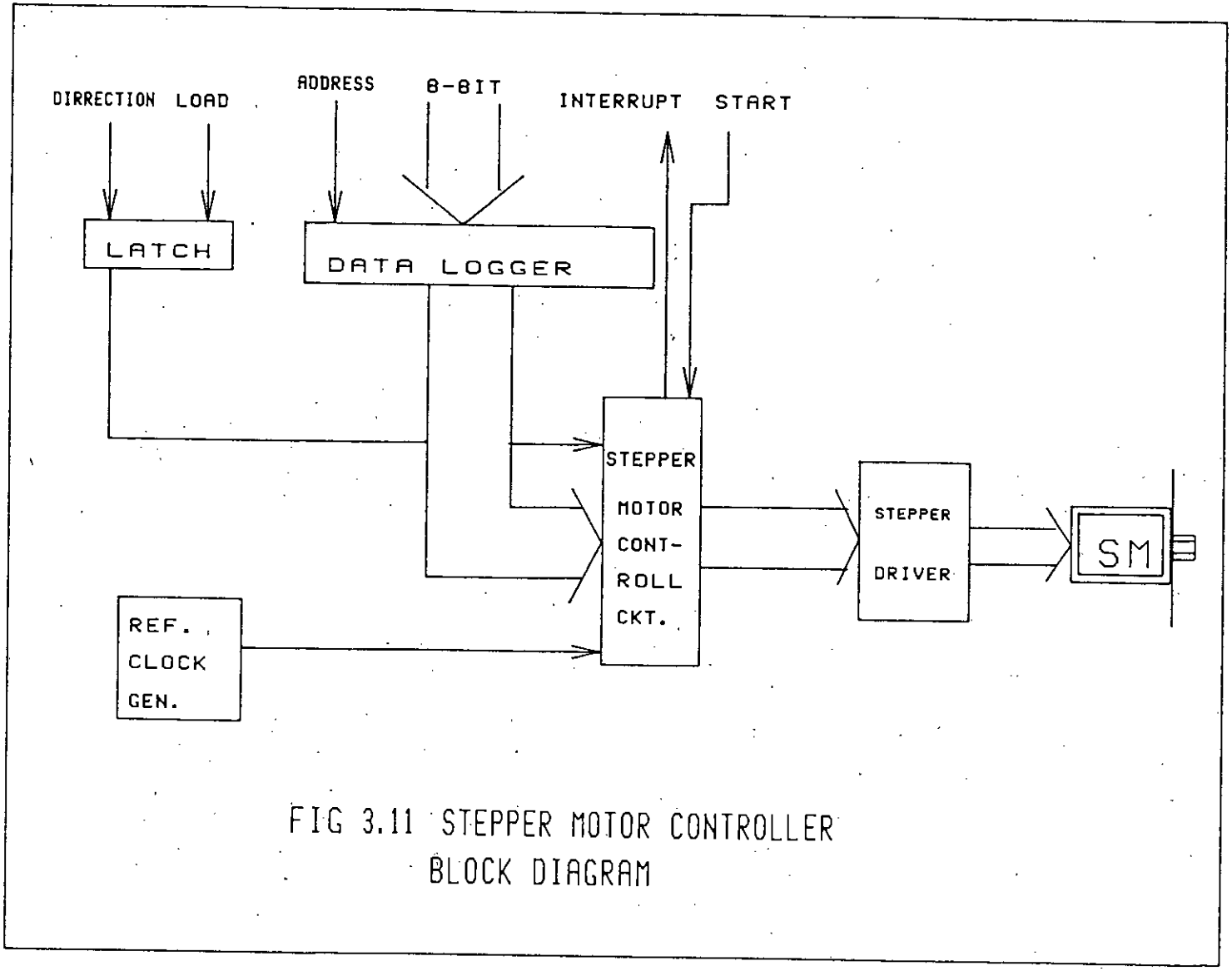


FIG 3.11 STEPPER MOTOR CONTROLLER
BLOCK DIAGRAM

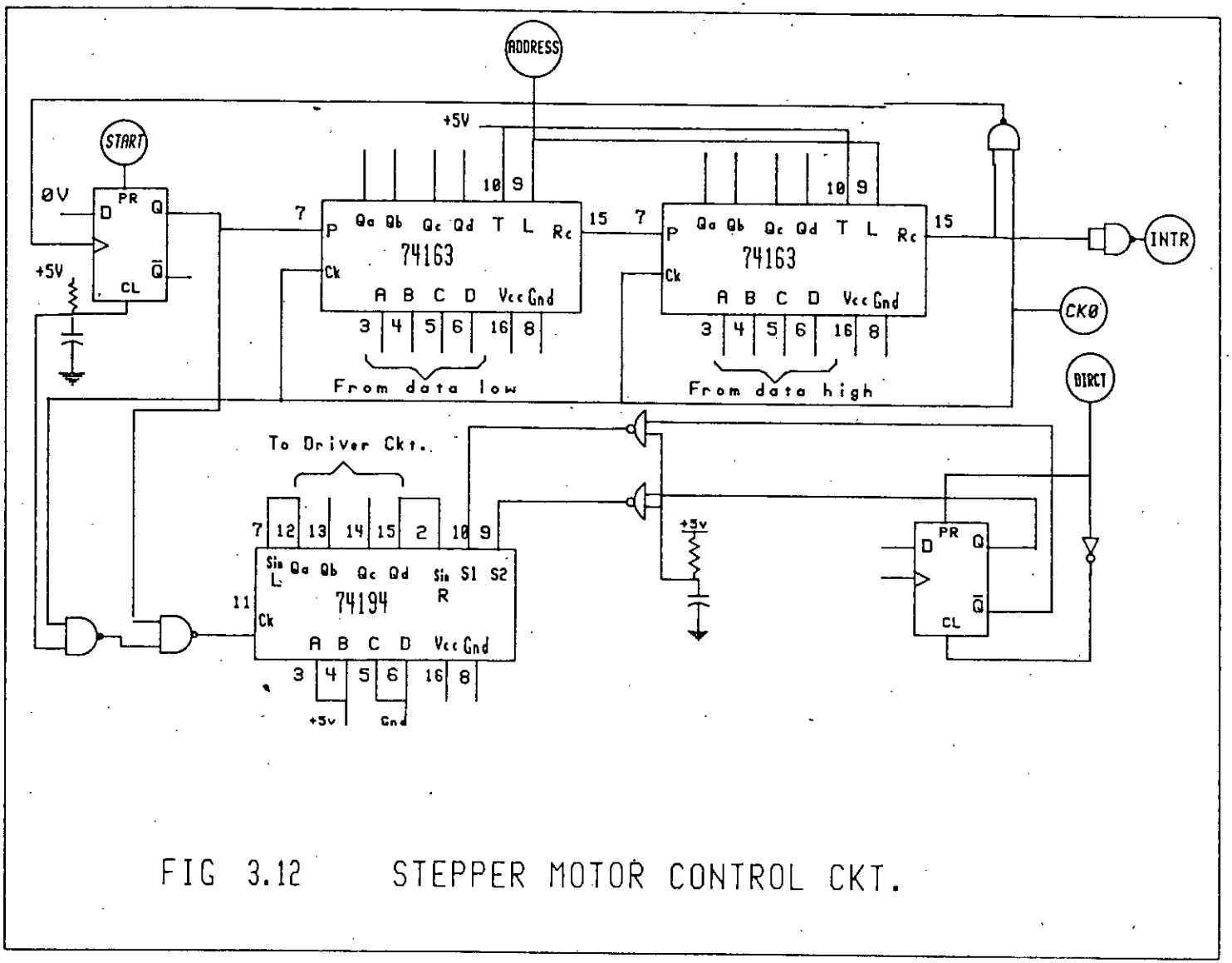
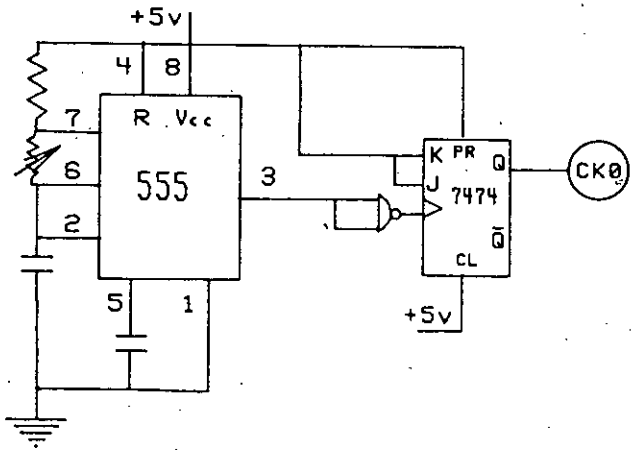
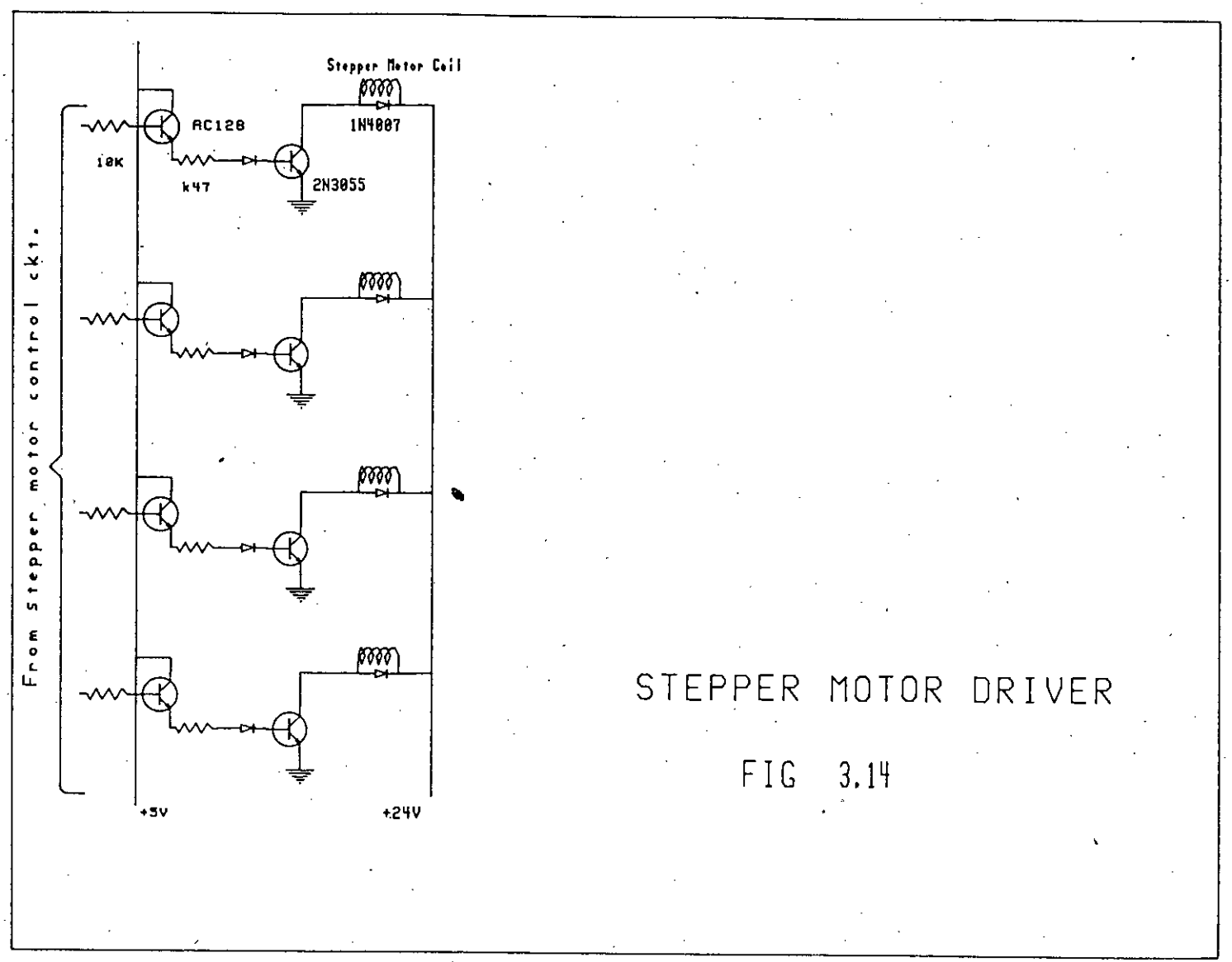


FIG 3.12 STEPPER MOTOR CONTROL CKT.



STEPPER MOTOR REFERENCE CLOCK

FIG 3.13



STEPPER MOTOR DRIVER

FIG 3.14

The stepper motor controller decodes the data received from the computer a)number of steps and b)direction of rotation. To reduce load of the computer,stepper motor controller receives the direction and the number of steps to rotate. The motor rotates accordingly when it receives a start pulse from computer (figure 3.11). The stepper motor controller card composes of (figure 3.12-3.14).

i.Data logger: This section stores data of number of rotation of the stepper motor set by the computer. When a low at load of 74163 is applied, the data at the data bus of the printer port is loaded.

ii.Reference Clock generator: 555 timer of 30 Hz is used for this purpose. This frequency is in the limit of the stepper motor maximum operating frequency.

iii.Direction latch: The direction of rotation is latched in 74198. For the counter clockwise rotation the direction flag is set to high and for anti-clockwise rotation flag is reset to low.

iv. Stepper motor control circuit: Two 74163 4-bit counter is connected to form an 8-bit binary counter. After getting start pulse it starts counting because P enable of 74163s are high. The shift circuit 74194 rotates left or right according to the status of the direction latch. When preassigned number of steps are completed i.e. the counter is zero an interrupt flag is set high and P enables goes low and clock to shift circuit is disabled. On acknowledging the interrupt the microcomputer has to reset the

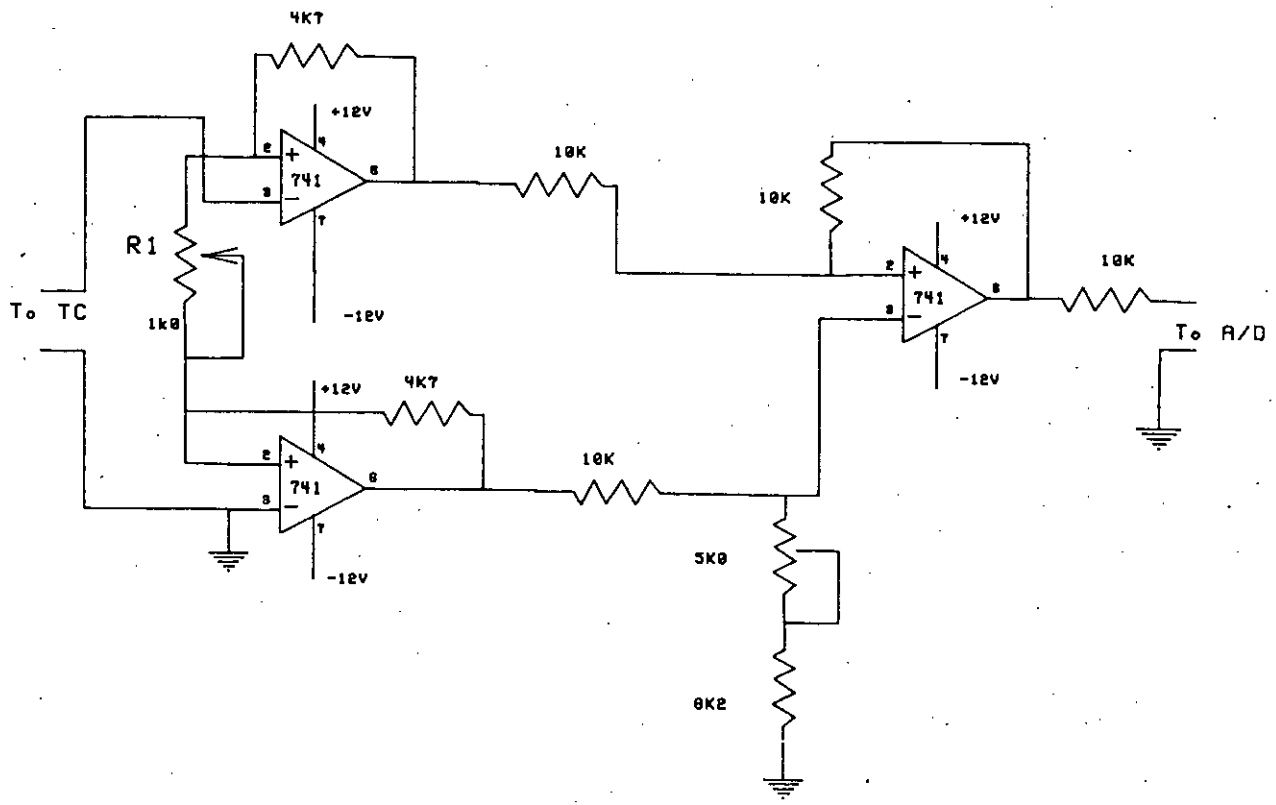


FIG 3.16 THERMOCOUPLE AMPLIFIER

flag, else it remains high.

iv. Drive circuit: Drive circuit boosts low power output of control circuit by darlington transistor AC128 and 2N3055 transistor amplifier.

3.2.4. THERMOCOUPLE AMPLIFIER

The voltage produced by the thermocouple is too small to be fed to ADC of the computer. Hence an instrument amplifier was designed to amplify the voltage. The amplifier uses LM741 opamp to have high input impedance (figure 3.16). The gain of the amplifier can be adjusted by potentiometer R1.

3.2.5 SOLENOID VALVE CONTROLLER

The solenoid valve controller receives 3 bit data of solenoid valve and returns 3 bit status of relay to computer. This card consists of following sections(fig 3.17):

i.Data storage: This stores data of the valve which are to on/off. A zero means valve is to be off and high means to open. A 74198, an 8 bit register, was used for this purpose.

ii.Relay driver circuit: The relay driver circuit capable of operating three 12V relay. Transistor circuit is used to drive a relay with free wheeling diode across the relay coil for the protection of relay coil against high voltage produced when coil is deenergised.

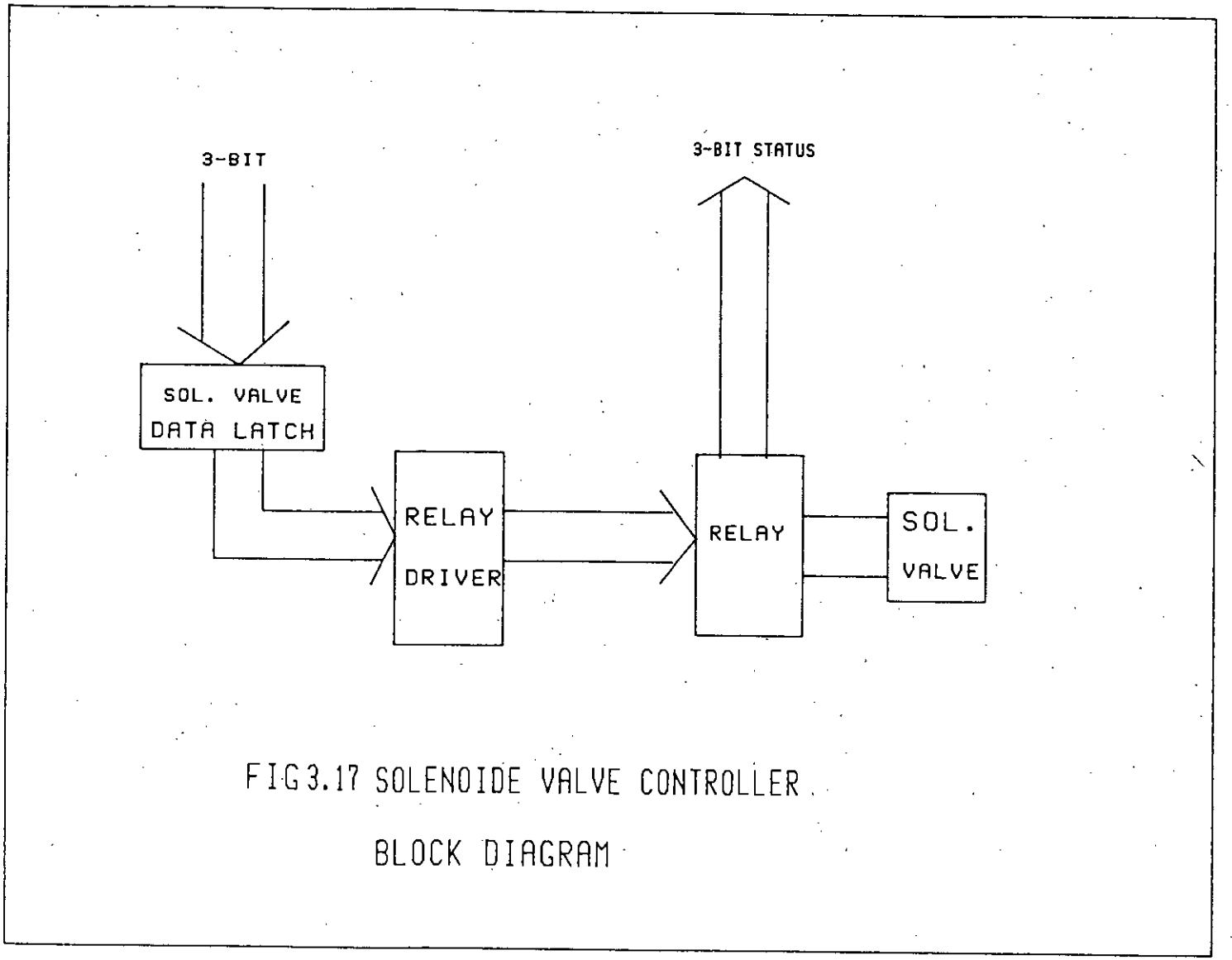


FIG 3.17 SOLENOIDE VALVE CONTROLLER

BLOCK DIAGRAM

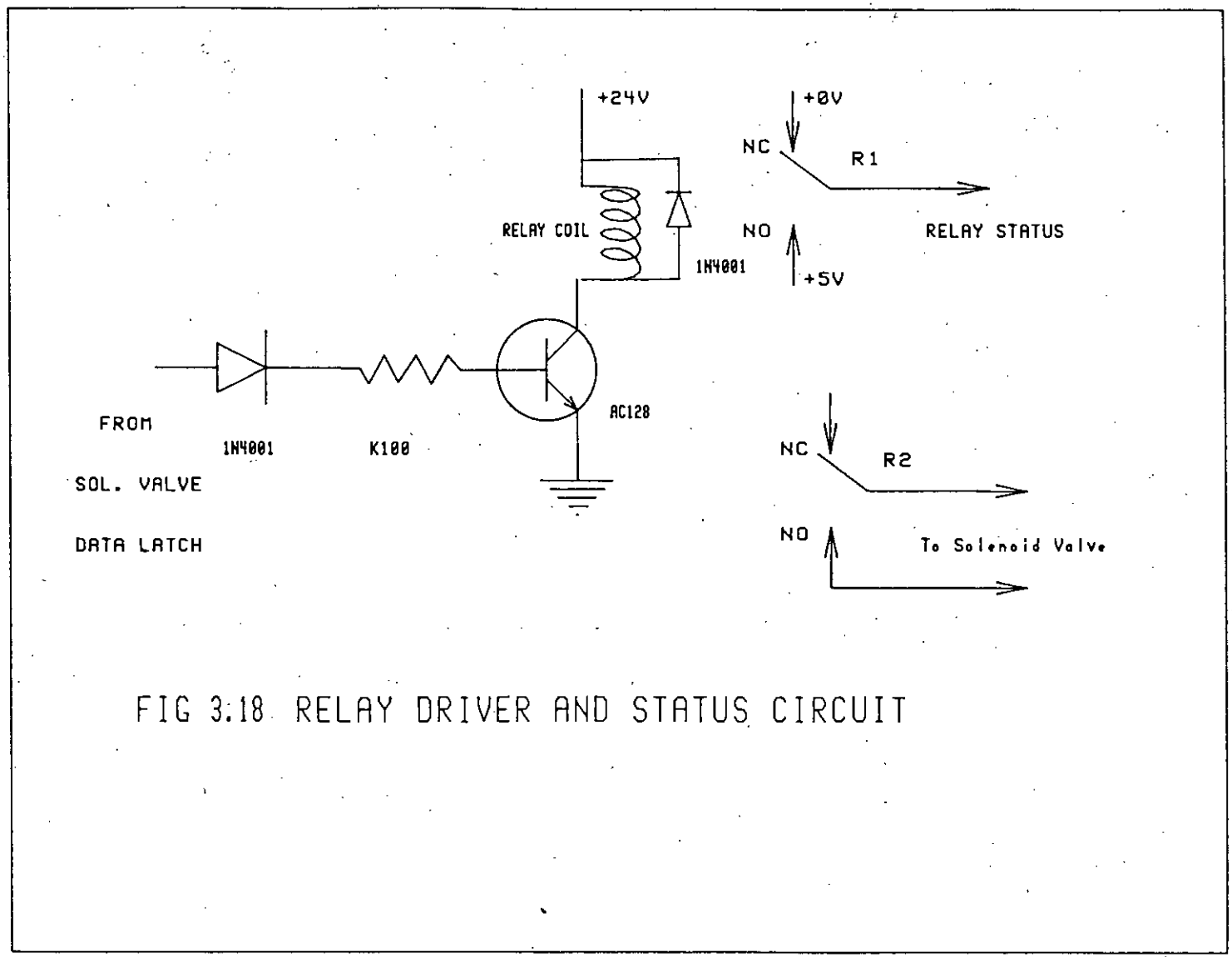


FIG 3.18 RELAY DRIVER AND STATUS CIRCUIT

iii. Solenoid valve status circuit: This returns the status of relay i.e. returns high when closed and returns low when opened. The connection for this is shown in figures 4.18.

3.2.6 POWER SUPPLY

The power supply has regulated +5v,+12v and -12v using 7805, 7812 and 7912 and unregulated +12V and +24V for heater controller and stepper motor control.

3.2.7 COMPUTER INTERFACE

The computer is interfaced with the modules given above via printer port and ADDA converter port. Printer port handshaking protocol was changed as shown in figure 3.19. The printer port now has 8 bit uni directional data bus, three bit address bus, one bit strobe line and three bit status line. An interface card connecting to printer port is shown in figure 3.20.

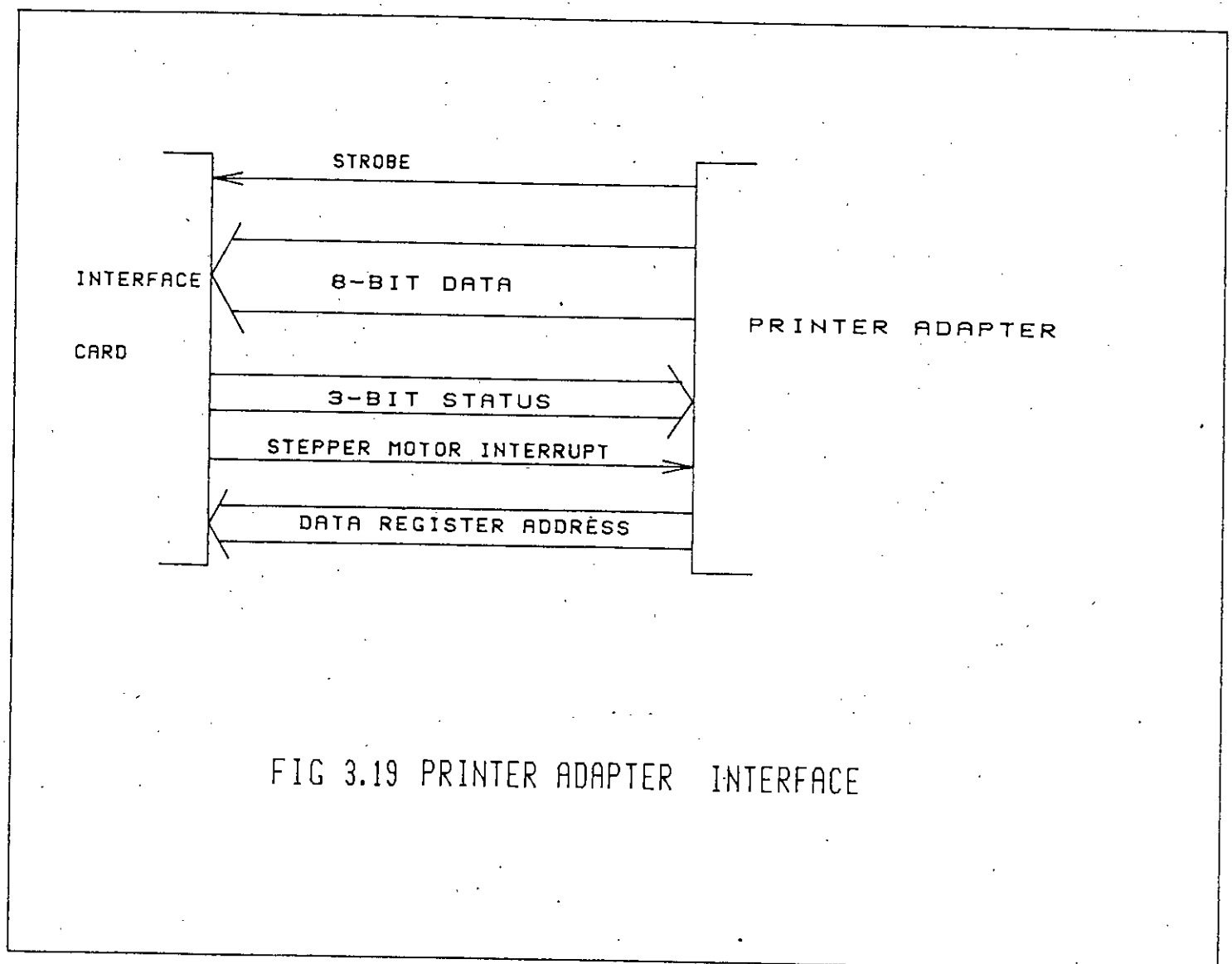


FIG 3.19 PRINTER ADAPTER INTERFACE

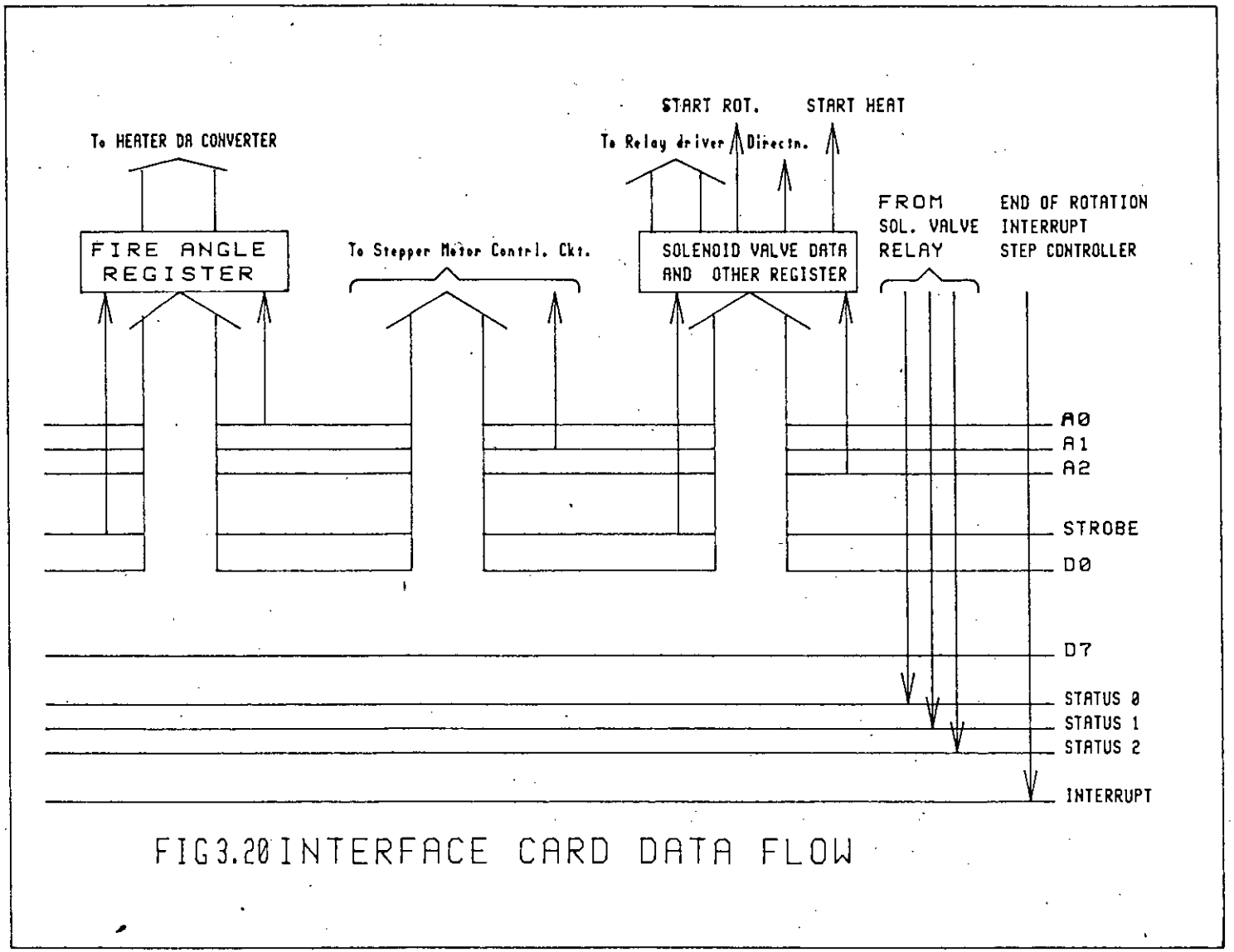


FIG3.20 INTERFACE CARD DATA FLOW

CHAPTER 4
DIGITAL FILTER DESIGN

CHAPTER 4 DIGITAL FILTER DESIGN

4.1 SAMPLING TIME SELECTION

The selection of sampling time requires answer to the question

a)What is economic sampling time?

b)What is the best sampling time from a process dynamics or data acquisition point of view?

If sampling rate is high enough the process information does not get lost and also cost of sampling (the number of different measurement points which can be monitored with computer) decreases. On the other hand, the economic cost due to control system performance deterioration increases as the sampling time increases as shown in figure 4.1 [17]. For large dead time ratios sampling time has less effect on the control performance due to the poor performance of a feedback controller for a dead time process. For small dead time ratios, sampling time selection can be important.

According to T.F.Edgar [17] dominant time constant of process model may be the key parameter for choosing sampling time. If τ_1 is the dominant time constant, $0.1\tau_1$ may be the sampling period (a detailed analysis is done in [17]). If the system model contains dead time t_d which is about the same magnitude as τ_1, then the sampling time should be selected in accordance with

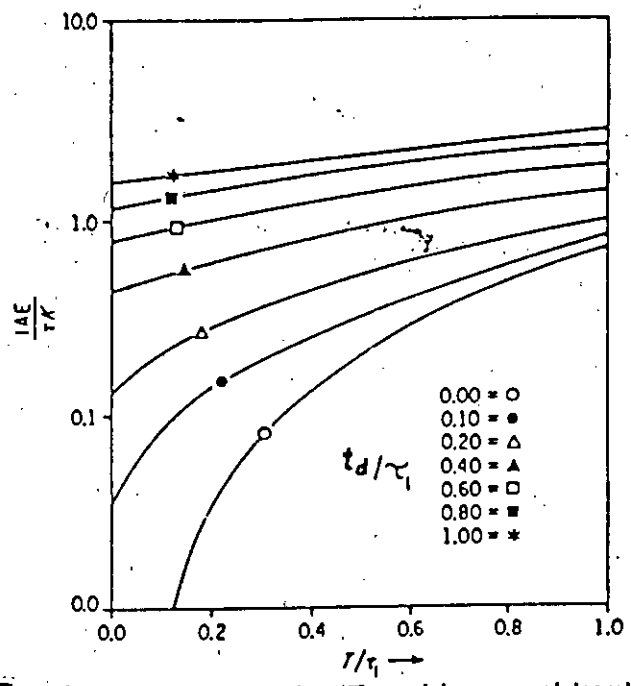


Fig. 4·1 Effect of sampling time (T) on minimum error integral IAE Criterion, PI control, for a first-order plus deadtime process, ratio of deadtime to time constant (t_d/τ) shown as parameter. [17]

figure 4.1. If $t_d << \tau_1$ dead time effect can be neglected. In selecting the sampling time it should be kept in mind that faster sampling time is a waste of computing power.

4.1.1 SAMPLING TIME FOR TEMPERATURE

For the heater in the experiment, using Cohen and Coon method t_d and τ_1 was obtained experimentally.

t_d =2.5 min.

τ_1 =33 min.

From the figure 4.1 for t_d/τ_1=0.076 , T/τ_1 is 0.2;

Hence T= 0.2 τ_1=6.6 min.

Thus sampling time for temperature is 6.6 min.

This is a very slow system, as the most heat exchangers are.

4.2 INTRODUCTION TO DIGITAL FILTER

Signal received by the computer from process is corrupted by the noise. The noise that makes the process unpredictable is mainly of two kinds: process noise and measurement noise. Measurement noise is produced by the measuring device and the connecting wires. This can be easily eliminated by the proper choice of the instrument and shielding of the cable. The process noise is introduced by different environmental phenomena. To

remove this type of noise the analog or digital filters are used.

The analog filters are composed of the RC circuits, either active or passive, to smooth the experimental data.

The equation of first order analog filter is

$$\tau_f \frac{dx}{dt} + x(t) = u(t) \qquad \text{--4.1}$$

where x(t) = input;

u(t) = output;

τ_f =time constant.

For step input,the response of filter is given by

$$x(t)=1-e^{(-t/\tau_f)} \qquad \text{--4.2}$$

The choice of the filter device depends on the constant of the filter. Analog filter is sufficient for the case τ_f less than 3 second. But if τ_f is greater than this limit, analog filter with amplifier or digital filter can be used. The analog filter with an amplifier is not cost effective in comparison to the digital filter. In general for the chemical process the time constant is greater than 3 second and hence the digital filter can be used for filtering purpose.

A comparative study of the filters is given so that the choice of digital filter is made clear.

DIGITAL FILTER	ANALOG FILTER
1.Easy to tune filter parameters to filter the process noise and can easily be modified.	1. To redesign it requires to change all components.
2.Digital filter uses computer which samples the process variables and analyze the measurement.	2. Analog filter performs continuously.
3.Limited by the computation time, data length, computating power of the data acquisition.	3. Analog filters are indepe-ndent items. It is not necessary to interact with other computational aspect of the process.
4.Instead of increasing the sampling rate, filtering the low frequency component of the signal can be performed.	4. Filtering the high frequency noise can be performed with analog elements.

The above comparison shows clearly that the digital filter is much more efficient, flexible and easy to implement than the analog filter.

To use the computer, the difference equation is applied for the digital filter construction.

4.3 DIGITAL FILTER REALIZATION

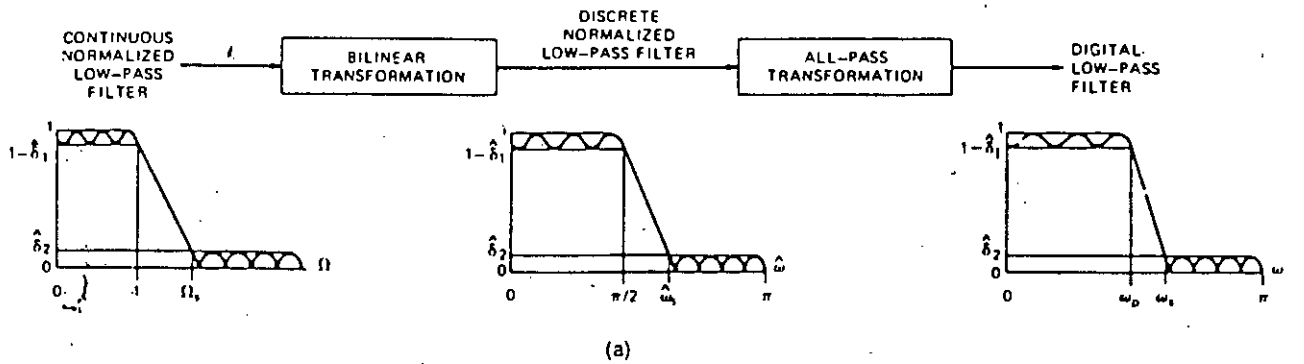
To design a digital filter there are two methodologies : infinite impulse response (IIR) and finite impulse response (FIR) [12].

By starting with an analog filter we are assuming an infinite duration unit pulse response as the basic theme behind the IIR filter design. FIR filter, on the other hand, uses the Fourier series and windowing method for its realization.

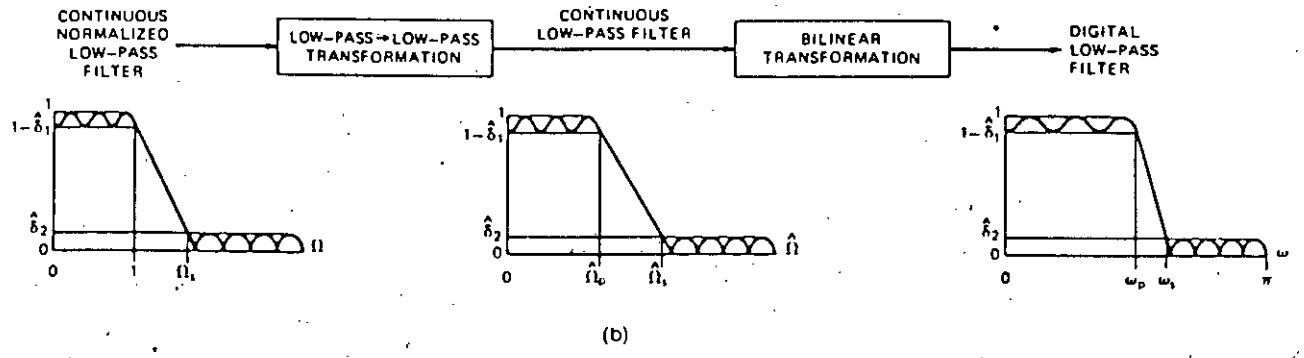
Using IIR filter the first order popular filters are exponential filter, double exponential filter or moving average filter. In the case of the higher order filter Butterworth, Chevishev and elliptical filters are a few popular filters.

The design techniques for IIR filters are easy to apply and it directly follows from the analog prototype, but the FIR filters have perfectly linear phase and have no poles which makes the filter always stable. Fast Fourier transformation (FFT) gives the filter designer a very simple tool for determining the filter weight.

The analysis done by Robiner et. al [13] showed that the IIR filter can be more efficiently realized than the FIR filter in the case of low pass filters with quasi-equiripple magnitude

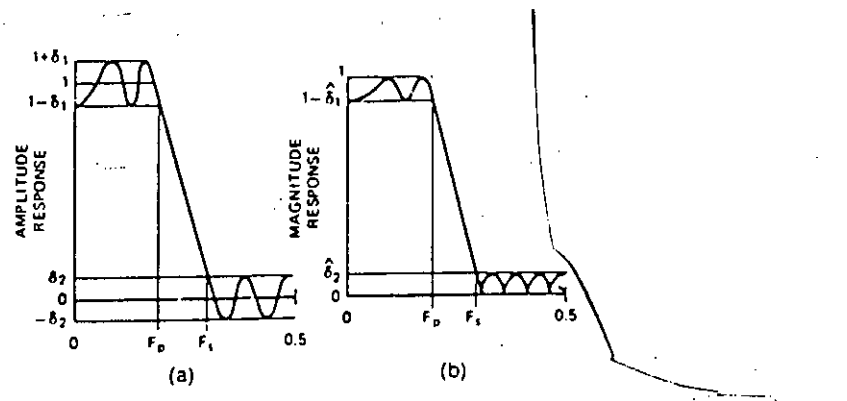


(a)



(b)

Fig 4.2 —Two techniques for transforming a continuous normalized low-pass filter to a digital low-pass filter. [17]



(a) (b)

Fig 4.3 Terminology used to describe low-pass filter characteristics. [17]

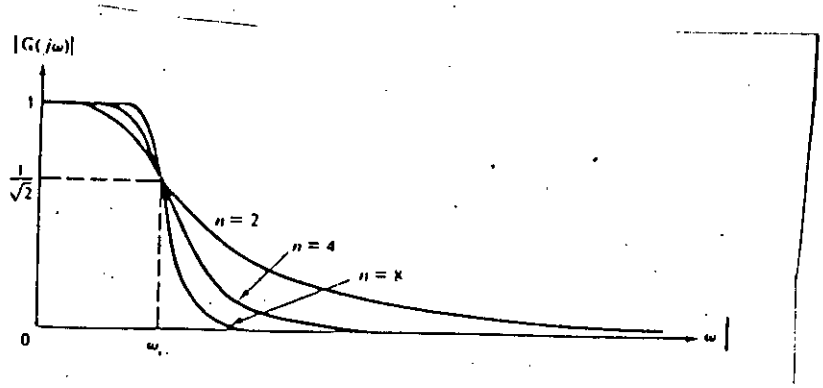


Fig 4.4 General Butterworth frequency response. [17]

characteristics. IIR filters have the recursive realizations that are very economical in terms of computational complexity. If the phase is not the criteria, FIR filters in general requires more computation to achieve a given accuracy of approximation to the desired amplitude response than do IIR filters.

To design the IIR digital filter there are many techniques, such as impulse invariant design, bilinear transformation or matching poles and zeros to transform analog filter to digital filter [25]. The design of the IIR filters proceeds through the bilinear transformation of an appropriate continuous filter. There are two techniques to design the IIR filter using bilinear transformation.

In the first procedure the continuous normalized low pass filter is first transformed using bilinear transform,then this discrete normalized low-pass filter is passed through an all-pass transformation with pass band frequency and cutoff frequency (figure 4.2a).

The second technique begins with the identical analog low-pass filter as the first technique and immediately performs a low-pass-to-low-pass filter transformation to give an analog filter and then resulting filter transformed to a digital filter using the bilinear transformation (figure 4.2b).

4.3.1 FILTER ORDER SELECTION

Using the most general procedure for IIR digital filter design the bilinear transformation, it is required to have different parameters for digital filter i.e. the order of the digital filter, transition ratio for a given pass band ripple and stop band loss and other relative key data. Different parameters for filter are shown in typical low pass filter amplitude response and magnitude response (figure 4.3). A generalized frequency response of low pass Butterworth filter of different order is shown in figure 4.4. The design parameters for the low-pass is given below and from the different curves presented in figure 4.5, order of the filter is selected.

a)pass-band ripple δ_1=0.01 ;

b)stop-band loss δ_2=0.02;

c)pass-band cutoff frequency =150Hz;

d)stop band frequency =350 Hz;

e)sampling frequency =1000Hz;

The sampling frequency is limited by the analog to digital conversion (A/D) card limitations and the speed of computation. The A/D card that was used and the speed of computation limit the sampling frequency to 1000 Hz and from thumb rule cutoff frequency is of about 0.1 times of sampling frequency which leads cutoff frequency to 150 Hz.

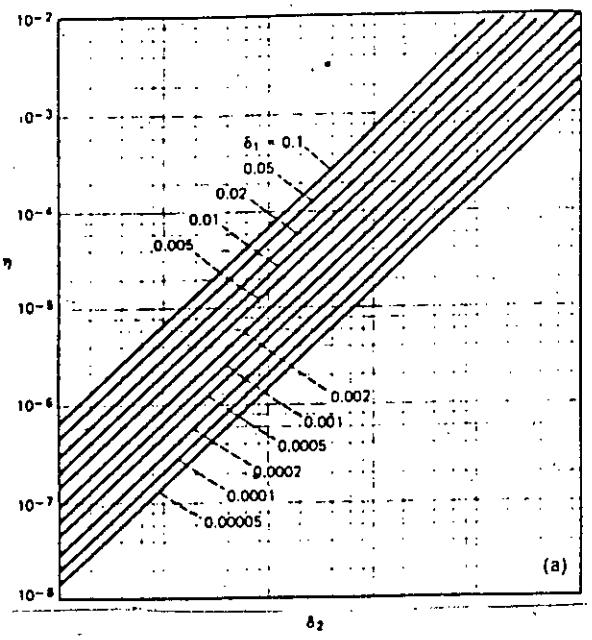


Fig - 4.5 A -Plots of η versus stopband specification, with parameter passband specification for low-pass filters. [17]

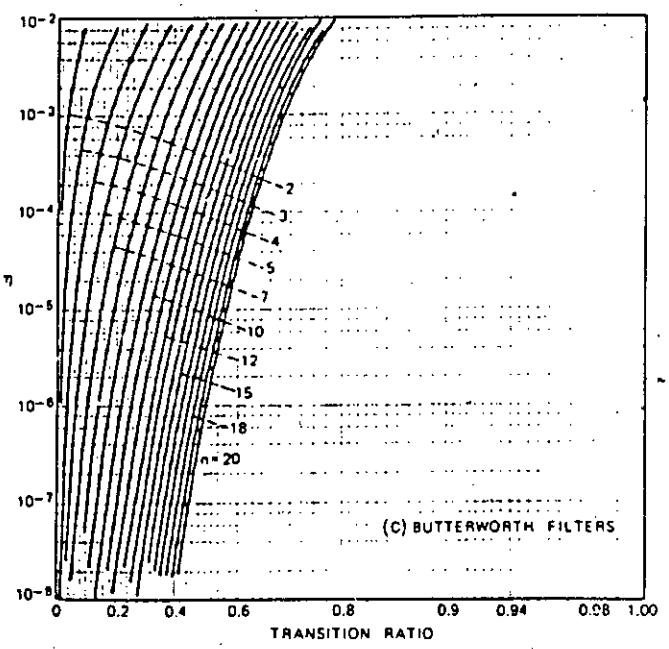


Fig 4.5 B -Plots of η versus transition ratio as a function of filter order n for Butterworth low-pass filters. [17]

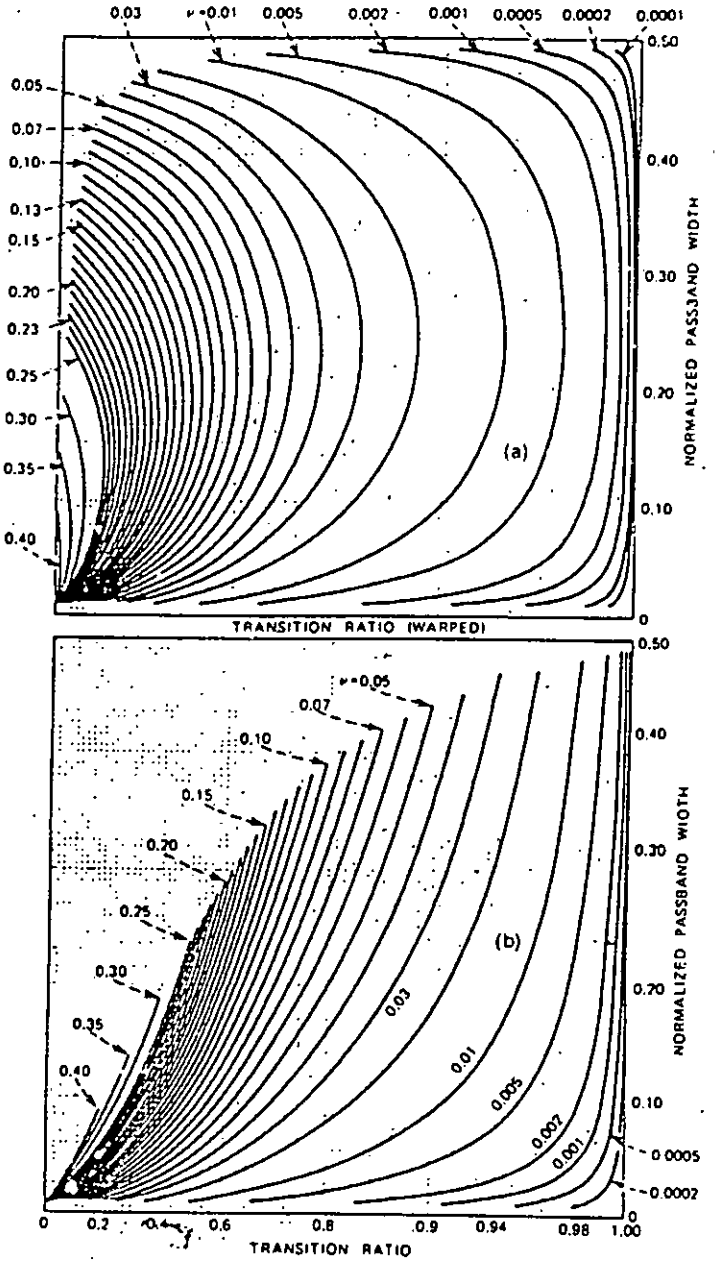


Fig. 4.5 c –Plots of passband cutoff frequency versus transition ratio as a function of transition width for discrete and continuous low-pass filters. [17]

Normalized pass band frequency F_p and stop band frequency F_s are

$$F_p = \frac{150}{1000} = 0.15;$$

$$F_s = \frac{350}{1000} = .35$$

The μ for δ_1 and δ_2 is 4e-3 (from fig 4.5 a);

The transition ratio for $F_s - F_s = 0.25$ with $F_p = 0.35$ is 0.38 (fig 4.5c);

Order of Butterworth filter n using intersection of lines μ and transition ratio in figure 4.5b is 4;

4.3.2 DIGITAL BUTTERWORTH FILTER REALIZATION

The Butterworth filter of order 4 are calculated for

Cutoff frequency 150 Hz.

Sampling frequency 1000Hz.

a)The transfer function of fourth order Butterworth filter is given by

$$H_B = \frac{1}{B_4(s)} \qquad --4.3$$

where $B_4(s) = s^4 + 2.6131259\ s^3 + 3.4142136\ s^2 + 2.6131259\ s + 1$;

b) Convertion into low pass with bandwidth $f_c = 150$ Hz is done by putting $s = (s / 2\ f_c)$

Hence

$$H(s)=\frac{5.0625\ E8}{s^4+3.9167E2\ s^3+7.8198\ E4\ s^2+8.81929\ E6\ s+5.0625\ E8}$$

c)Digital filter transfer function H(z) can be formed from bilinear transform by putting

$$s= C\ \frac{1- Z^{-1}}{1+ Z^{-1}}$$

where $C=w_c \cot(w_c\ T\ /2)$

$$=w_c \cot(\pi\ f_c/f_s)$$

Hence

$$H(z)=\frac{0.01856+0.74252z^{-1}+0.111378z^{-2}+0.074252z^{-3}+0.018563z^{-4}}{1-1.57039z^{-1}+1.275613z^{-2}-0.484403z^{-3}+0.76197z^{-4}}$$

$$=\frac{X(z)}{U(z)} \qquad \text{--4.4}$$

Hence

$X(z) =1.5704\ z^{-1}\ X(z)-1.275\ z^{-2}\ X(z)+0.4844\ z^{-3}\ X(z)-0.76197\ z^{-4}$ $X(z)+0.01856\ U(z)+0.74252\ z^{-1}\ U(z)+0.11137\ z^{-2}\ U(z)+0.7425$ $z^{-3}U(z) + 0.01856\ z^{-4}U(z);$

The filter in difference form is

$x(nT) = 1.5704\ x((n-1)T) - 1.275\ x((n-2)T) + 0.4844\ x((n-3)T) -$ $0.76197\ x((n-4)T) +0.01856\ u(nT) + 0.74252\ u((n-1)T) + 0.11137$ $u((n-2)T) + 0.7425\ u((n-3)T) +0.01856\ u((n-4)T).$ --4.5

4.3.3 FILTER RESPONSE

The filter performance of 4th order Butterworth filter as designed earlier for step input and random noise are given in figure 4.6a and 4.6b respectively. In figure 4.6b, actual signal of one unit is corrupted by Gaussian random noise generated by software.

STEP RESPONSE

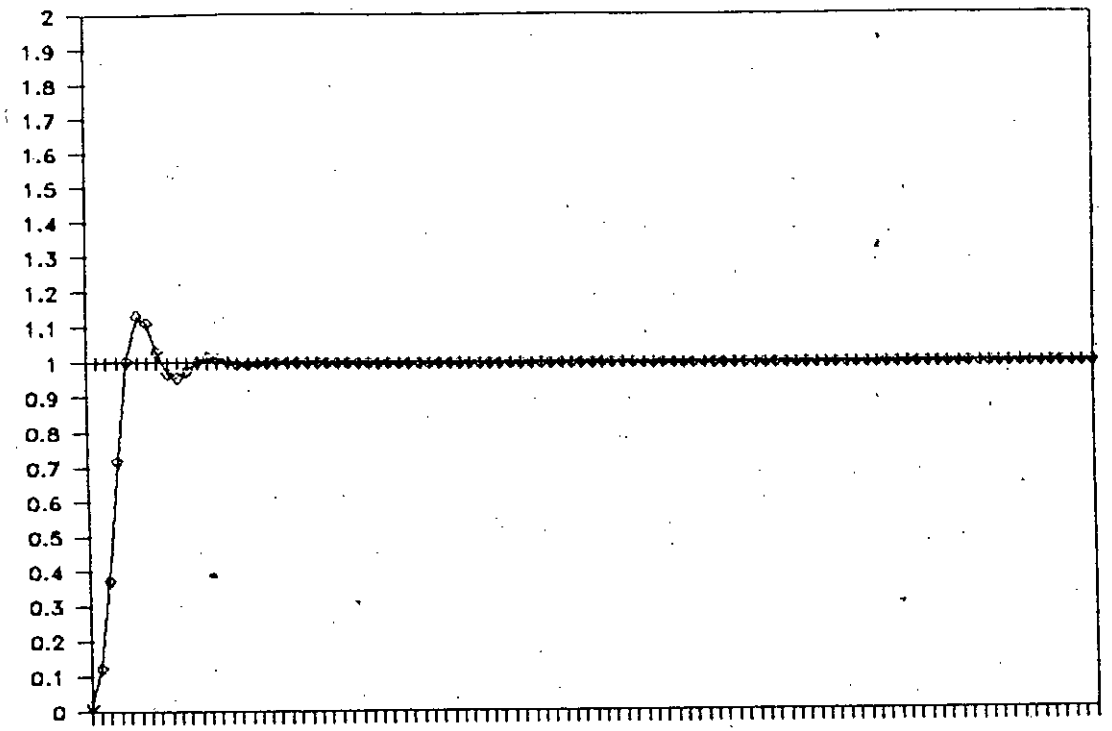


Fig - 4.6.A.

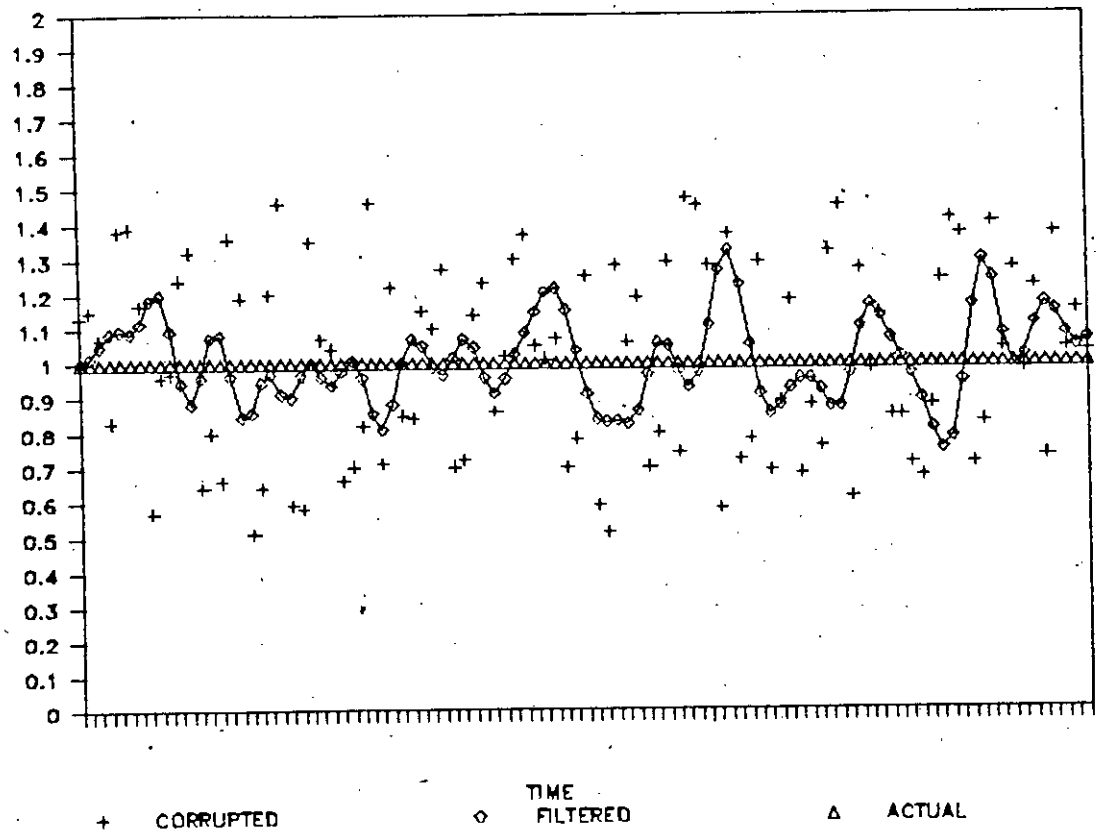

RANDOM RESPONSE
2
1.9
1.8
1.7
1.6
1.5
1.4
1.3
1.2
1.1
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
TIME
+ CORRUPTED
◇ FILTERED
△ ACTUAL

Fig-4.6 B

CHAPTER 5
CONTROLLER DESIGN

CHAPTER 5 CONTROLLER DESIGN

5.1 GENERAL IDEA OF CONTROLLER DESIGN

To improve the system response to a desired specification one of two common methods, cascade or feedback compensation may be used (figure 5.1). The control system is required to compensate for one of the following four discriminations of the system due to step input:

i) A given system is stable and its transient response is satisfactory, but its steady state error is too large;

ii) A given system is stable, but its transient response is not satisfactory;

iii) A given system is stable, but both transient and steady state error is unsatisfactory;

iv) A given system is unstable for all values of gain.

The additional equipment that is inserted into the system in order to obtain the desired system performance is referred as controller or compensator.

There are three basic type of compensators or controller and they are a)proportional type or P type; b)integral type or I type; c) derivative or D type. A compensator that is generally used in the process control industry is the PID controller.

To design the controller, the system transfer function and hence the controller parameter can be derived either by Cohen-Coon method or by Zigler-Nichols method. Another procedure for cascade compensator design is Guilleman-Truxal method. This method is commonly called pole-zero placement technique. The adaptive controller selftuning and autotuning use these methods to find controller parameter. Both these tuning methods update the PID controller parameters.

5.1.1 COHEN COON METHOD

To find the control parameters such as PID, an empirical method such as Cohen-Coon method can be used. In this method, the controller is taken out of the loop. A step input of known magnitude is applied to activate the final element. A record of output with time known as process reaction curve. It has been shown that from the process reaction curve, an approximate delayed first order system can be developed having transfer function

$$G(s) = \frac{Ke^{t}\, s}{\tau s + 1} \qquad \text{--5.1}$$

where K = static gain;

t_d = dead time;

τ = time constant;

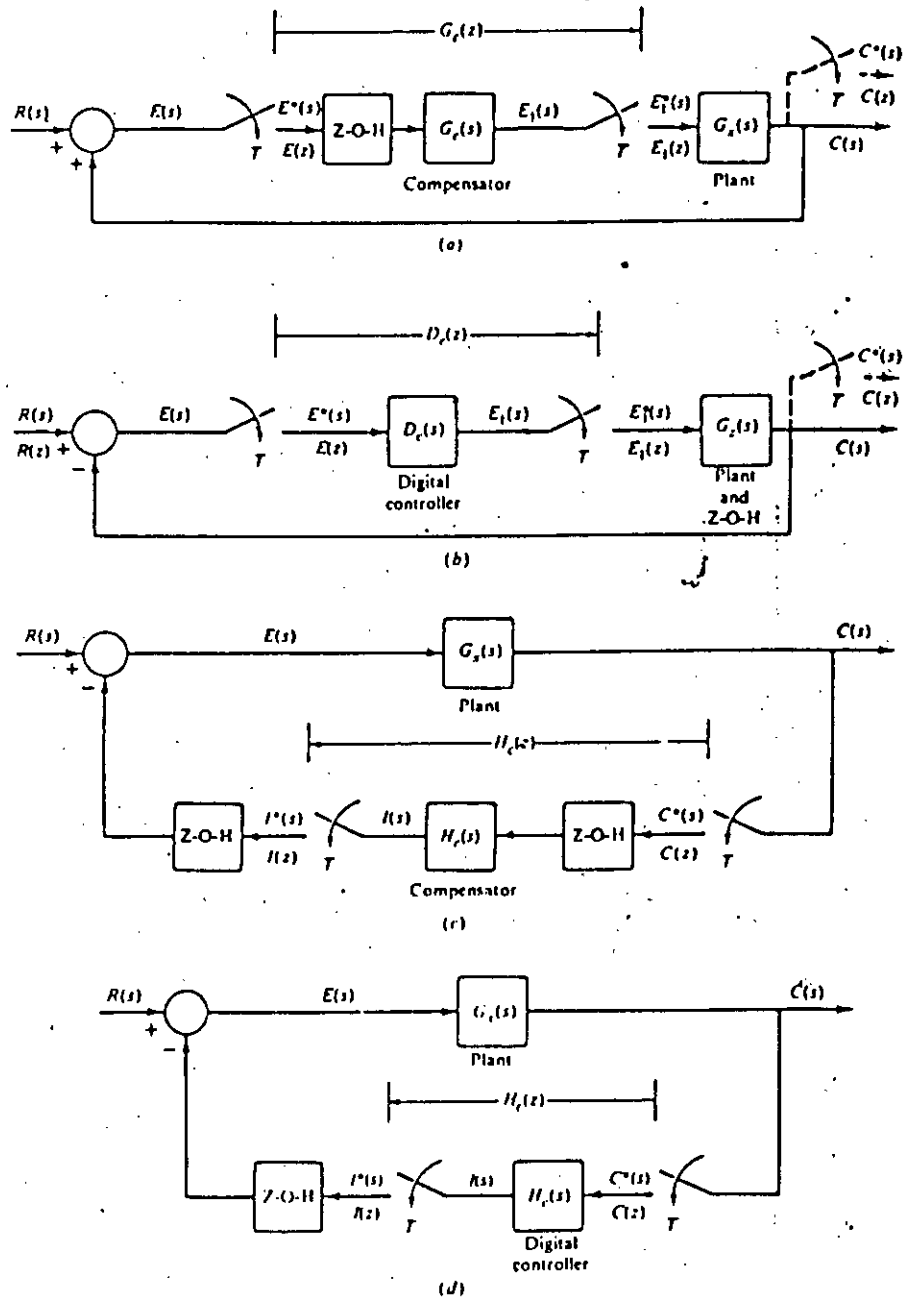
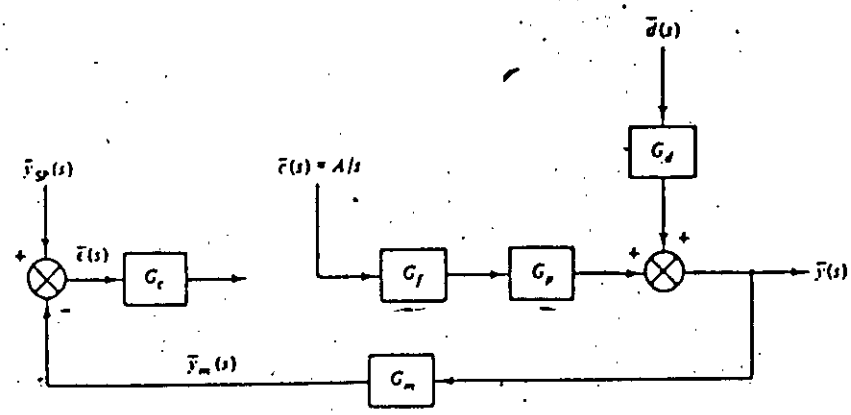


FIG. 5.1. A compensated sampled-data control system. (*a*), (*c*) Cascade and feedback analog compensators, respectively; (*b*), (*d*) cascade and feedback digital controllers, respectively. [25]



"Opened" control loop.

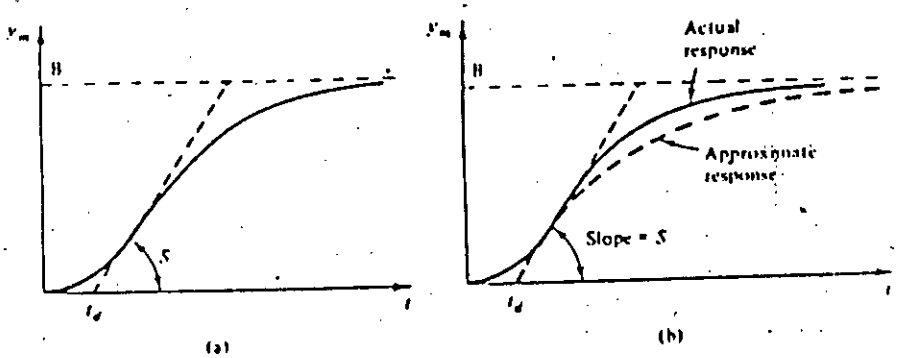


FIG 5.2 | (a) Process reaction curve; (b) its approximation with a first-order plus dead-time system. [20]

thus $$K=\frac{\text{output (at steady state)}}{\text{input (at steady state)}}=\frac{B}{A}$$

t_d=time elapsed until the system respond;

τ_0=B/S, where S is the slope of sigmoidal respond at point of inflection.

The graphical construction for the identification of the parameter of the model is shown in fig 5.2.

Having one quarter decay ratio, minimum offset, minimum integral square error for the PID controller, the different constants are

$$K_c=\frac{1}{K}\ \frac{\tau}{t_d}\ \left(\frac{4}{3}+\frac{t_d}{4}\right); \qquad 5.2a$$

$$\tau_I= t_d\ \frac{32+6\ t_d/\tau}{13+8\ t_d/\tau}\ ; \qquad 5.2b$$

$$\tau_D= t_d\ \frac{4}{11+2\ t_d/\tau} \qquad 5.2c$$

5.1.2 ZIGLER NICHOLS TUNING TECHNIQUE

By this method, the tuner uses closed loop system. In this method the system is brought to desired operational level. The proportional control gain is increased such that the system starts oscillating. At this point the ultimate gain K_u and ultimate period of sustained oscillation T_u are obtained. Using this data, Zigler and Nichols recommended that the PID controller has the parameters

$$K_c=(k_u / 1.7); \qquad 5.3a$$

$$\tau_I=(T_u/ 2.0); \qquad 5.3b$$

$$\tau_D=(T_u /8.0); \qquad 5.3c$$

5.1.3 DISCRETE FORM OF PID CONTROLLER

From the theoretical PID controller, the transfer function is

$$D_{PID}(s)= \tau_D s + Kc + \tau_I /s;$$

Using z-transformation and rearranging, z-transformed PID controller is given

$$D_{PID}(z)=\frac{K_{PID}(z^2-a\ z+b)}{z(z-1)}=\frac{C(z)}{E(z)} \qquad 5.4$$

where

$$K_{PID}=K+T\ \tau_I+\tau_D/T;$$

$$a=(T\ K_c +2\ \tau_D)/T\ K_{PID};$$

$b = \tau_D / T K_{PID}$;

T=sampling period;

$$C(z)=K_{PID}(E(z) - a E(z) z^{-1} + b E(z)z^{-2}) + C(z) z^{-1};$$

Hence the PID controller in difference form

$$C(nT)=K_{PID}(E(nT)-aE((n-1)T) +b E((n-2)T)) +C((n-1)T); \quad --5.5$$

The controller equation is directly transformed from the analog controller. The main objection about this is that this does not take care of the delay and no delay compensation is utilized. Chemical process is a nonlinear and time dependent process and hence the constants derived for PID controller will soon become obsolete due to environmental changes and/or operating point changes.

5.2 AUTOTUNING

5.2.1 BASIC THEORY

Zigler-Nichols method requires that the ultimate gain and ultimate frequency are found out from closed loop experiments. This can be done by using proportional regulator with increasing gain until oscillation occurs. The gain (K_u) when oscillation occurs is the critical gain and the period is critical period (T_u). But in practice difficulty arises in keeping amplitude of oscillation under control [21].

The autotuning is based on the idea that the ultimate gain and frequency can be determined by introducing relay feedback. The ultimate period T_u is simply the period of oscillation and critical gain can be calculated by applying Fourier expansion of relay amplitude and amplitude of oscillation. If d is relay amplitude, then the amplitude of first harmonics is

$$4d/\pi$$

If output amplitude = a, then ultimate gain

$$K_u=(4d) / (\pi a);$$

5.2.2 CONTROLLER DESIGN

Using Ziegler.Nichols rule [21]

$$K_c = K_u/2;$$

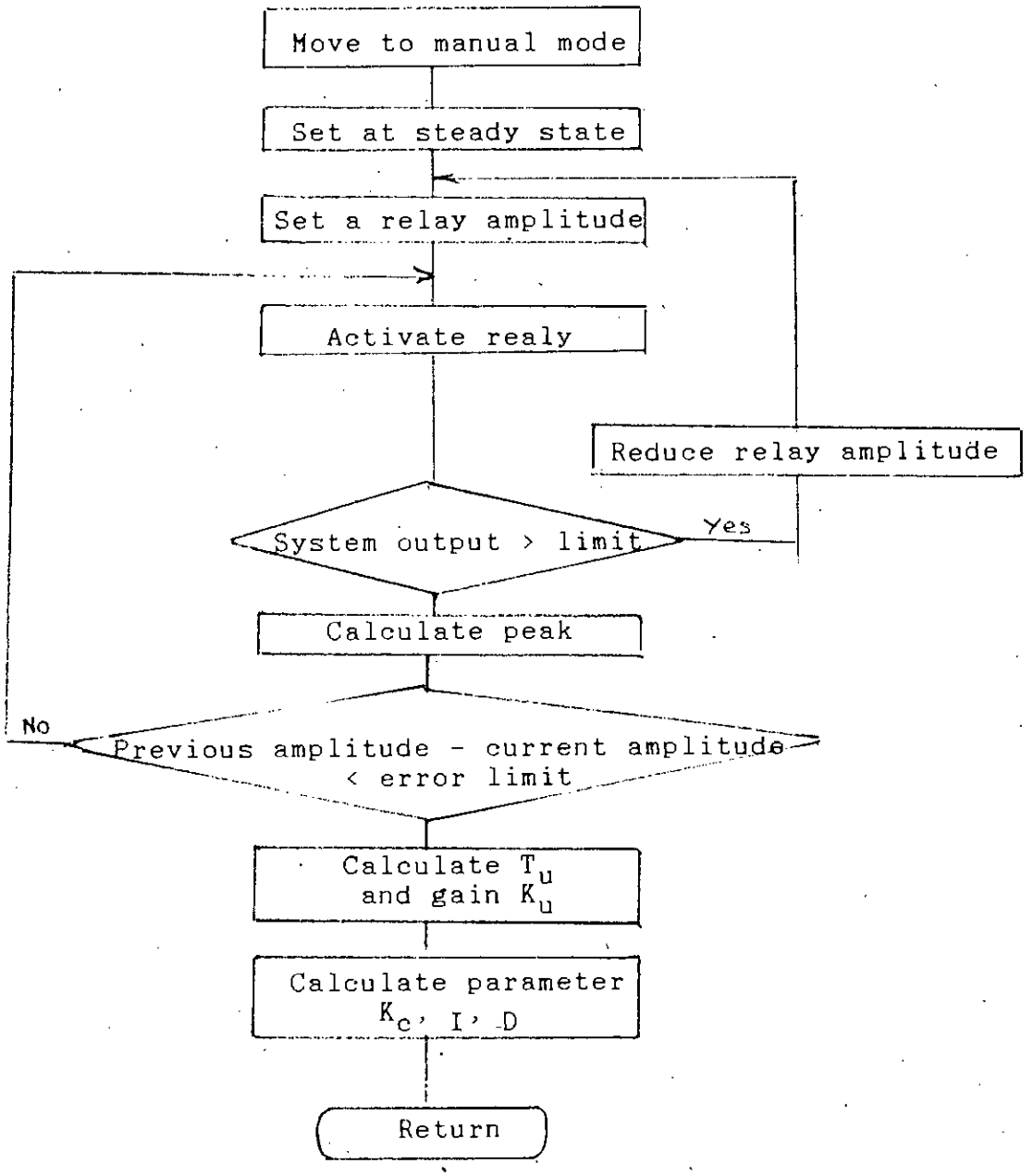
$$\tau_I = T_u/2;$$

$$\tau_D = T_u/8;$$

These PID constants are transformed to digital form as given earlier (5.5).

5.2.3 IMPLEMENTATION OF AUTO TUNER

The diagram of auto tuner is given figure 5.3. The flow diagram of the auto tuner is shown below.



To implement autotuner the following problems have to be solved.

a)Removal of measurement noise;

b)Level adjustment;

c)Saturation of actuator;

d)Automatic adjustment of the amplitude of oscillation.

Measurement error poses errors in detection of peaks and zero crossing. Filtering or hysteresis relay reduces the influence of measurement noise.

If relay amplitude is not sufficiently large for a process with finite low-frequency gain, there is no guarantee that the desired steady state will be achieved with relay control.

5.3 ADAPTIVE CONTROL

Often chemical processes are nonlinear and process dynamics vary with changes in operating conditions. Thus with the change of time and operating conditions the controller need to be retuned in order to provide satisfactory performance at the new operating conditions. The adaptive control might be attractive in the following applications.

1.Catalytic process where unmeasureable change occur;

2.Heat exchanger with fouling problem;

3.Frequent equipment change (failure,startup,shutdown);

4.Large, frequent disturbance (feed compensation, fuel quality etc.);

5.Ambient conditions(rain storms,24 hour daily cycle);

6.Product specification i.e. grade changes.

There are two general categories of adaptive control problems [22]. In the first category the different parameters of the controller are stored for different conditions or a model for constants are prepared for different conditions. If the process is reasonably well understood, then it may be feasible to adjust the controller parameters by measuring process changes. This type is known as programmed adaptation since effects of the process changes are predictable.

Where the process changes cannot be measured nor predicted, in this difficult situation the adaptive control strategy use the feedback method to calculate controller parameter. This type of

adaptive control is referred to as self tuning control.

A wide variety of adaptive control strategy that has been proposed can be conveniently placed into four categories [22].

1.Adaptive controllers designed using a quadratic cost function;

2.Design methods based on stability theory;

3.Pole assignment techniques;

4.Miscellaneous approach such as pattern recognition.

5.4 SELF TUNING CONTROL

If the process changes cannot be measured nor anticipated, selftuning control is applied as stated earlier. From the process data, parameters of controllers are updated (figure 5.4).

To implement self tunning control, the system must

1. Estimate the process parameter;
2. Calculate new controller parameter;
3. Implement the control.

Most real time parameter estimation techniques requires the introduction of an external forcing signal to allow process identification. Such an input signal can be introduced through the set point change or superimposed on the controller output.

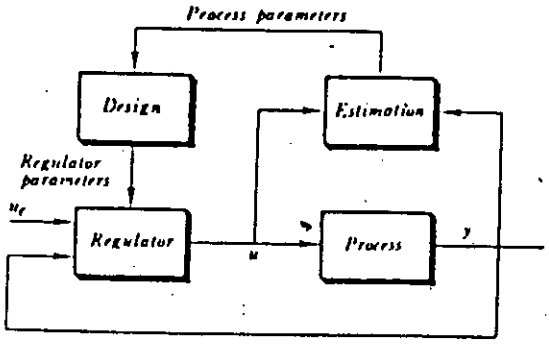
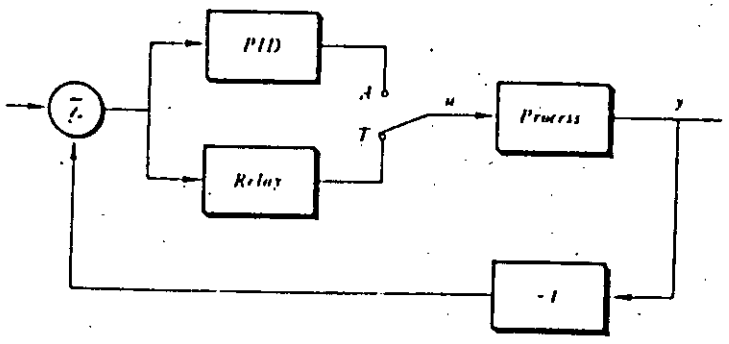


FIG 5.4 Block diagram of a conventional adaptive regulator. [22]



Block diagram of an auto-tuner.

FIG 5.3 The system operates as a relay controller in the tuning mode (T) and as an ordinary PID regulator in the automatic control mode (A). [22]

If the process control function is known then desired process function is derived. The criterion of the feedback process first is set and accordingly the desired transfer function is derived. Guilleman Truxal method is used to find the control transfer function. This can be done in two ways (figure 5.5).

5.4.1 THEORETICAL BASIC OF CONTROLLER

For most of the chemical process have the first order delayed system transfer function, hence for the selftunning controller the system transfer function can be given by

$$G_p = \frac{K_p\, e^{-t_d s}}{\tau s + 1} \qquad --5.6$$

where t_d= delay time;

τ=time constant;

K_p=gain;

Generally hold element have zero order with sampling time T, then transfer function of hold circuit is

$$H(s) = \frac{1 - e^{-sT}}{s} \qquad --5.7$$

The pulse transfer function for the combination of zero order hold H(s) and process transfer function G_p (figure 5.6) is

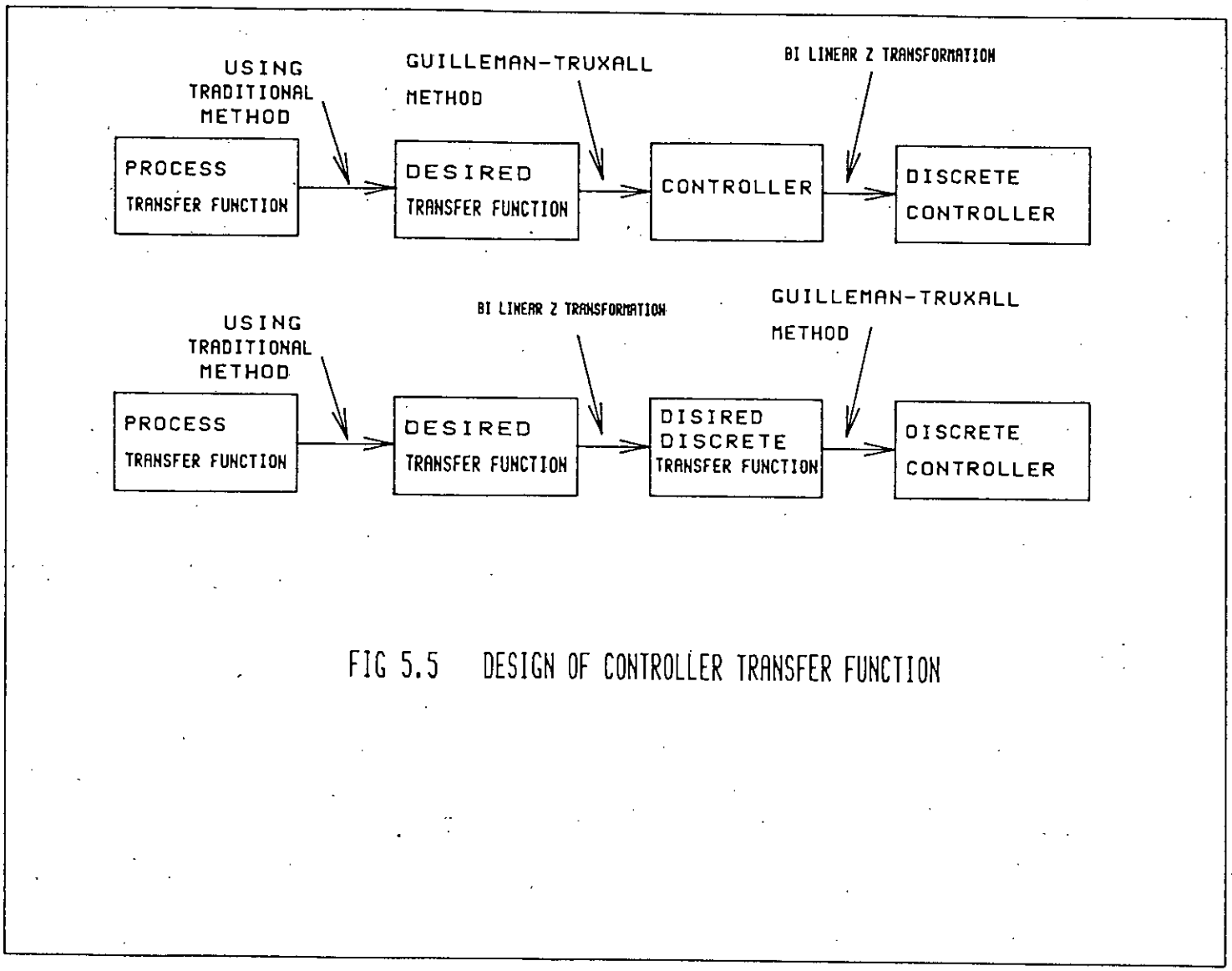


FIG 5.5 DESIGN OF CONTROLLER TRANSFER FUNCTION

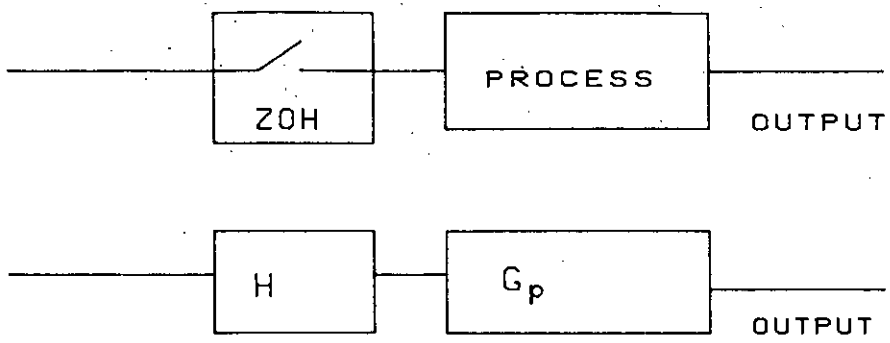


FIG 5.6 PROCESS AND ZERO ORDER HOLD

$$HG_p(z) = [H(s)G_p(s)] = \frac{K_p(1-a)z^{-N-1}}{1-a\,z^{-1}} \quad --5.8$$

where $a=e^{(-T/\tau)}$

N=nearest integer of t_d/T;

Using Dahlin algorithm, then the closed loop transfer function

$$\frac{Y}{R} = \frac{(1-q)z^{-N-1}}{1 - q\,z^{-1} - (1-q)z^{-N-1}} \quad --5.9$$

where $q=e^{-\lambda T}$

λ =Dahlin tunning parameter;

if the Dahlin tuning parameter is increased, q approaches zero, error is reduced faster, and the Dahlin algorithm approaches the Deadbeat algorithm. For small value of λ, the value of q is near 1.0 and the control is very slow. Thus q can be adjusted to obtain any desired speed of control.

Using Guilleman-Truxall method (pole-zero-placement technique) obtain controller pulse transfer function is obtained as

$$D(z) = \frac{1}{HG_p}\,\frac{C(z)}{R(z) - C(z)}$$

$$= \frac{1-az^{-1}}{K_p(1-a)z^{-N-1}}\,\frac{(1-q)z^{-N-1}}{1-qz^{-1}-(1-q)z^{-N-1}}$$

Simplifying and rearranging

$$D(z) = \frac{1-az^{-1}}{K_p(1-a)(1-qz^{-1}-(1-q)z^{-N-1})} = \frac{U(z)}{E(z)} \quad --5.10$$

Hence the difference equation of the controller

$$u_k = q\,u_{k-1} + (1-q)\,u_{k-N-1} + \frac{1-q}{K_p(1-a)}(e_k - a\,e_{k-1}) \quad --5.11$$

It requires a and K_p and tunning parameter q to identify. For this type of controller, the closed loop response time remains constant, although their controller setting varies when the process model changes.

5.4.3 MODEL IDENTIFICATION

The first order model of the process derrieved earlier (5.8) is given by

$$HG(z) = \frac{K_p(1-a)z^{-N-1}}{1-az^{-1}} = \frac{Y(z)}{E(z)};$$

Then the difference equation of process with hold element is

$$y_k = K_p(1-a)\,e_{n-N-1} + a\,y_{k-1}$$

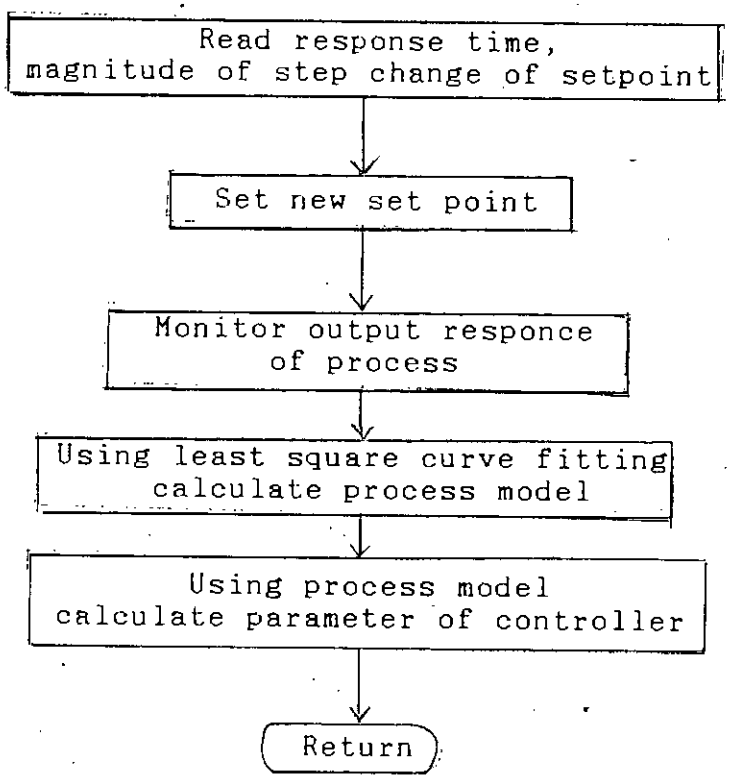
$$= b\,e_{n-N-1} + a\,y_{k-1}$$

where $b = K_p(1-a)$;

This equation has two unknown a and b. Using least square curve fitting the unknown a and b can be found out .

5.4.4 IMPLEMENTATION OF SELF-TUNING CONTROL

The sequence of model Identification is as follows:



5.5 SOFTWARE FOR INTERFACE

Interfacing the hardware developed with the computer there requires handshaking software. This software can be categorized in two section a)program for data acquisition and b) program for hardware control.

To control the total process the data flow diagram is shown in appendix. To control the heater power, stepper motor, and solenoid valve and to get the temperature and flow digitized data taken from ADC, special programs are developed.

5.5.1 PROGRAMMING FOR DATA ACQUISITION

The first step in data acquisition is to scan the process sensing instruments. This is done either on a fixed time schedule, on demand by the program or by the operator if conditions so warrant, or the scan may be initialized by a signal from the process. The sampling frequency is completely depends on the nature of sampled data and nature of the process. A detailed analysis is done in sampling time selection of chapter 4. The programming for data acquisition follows the sequence followed.

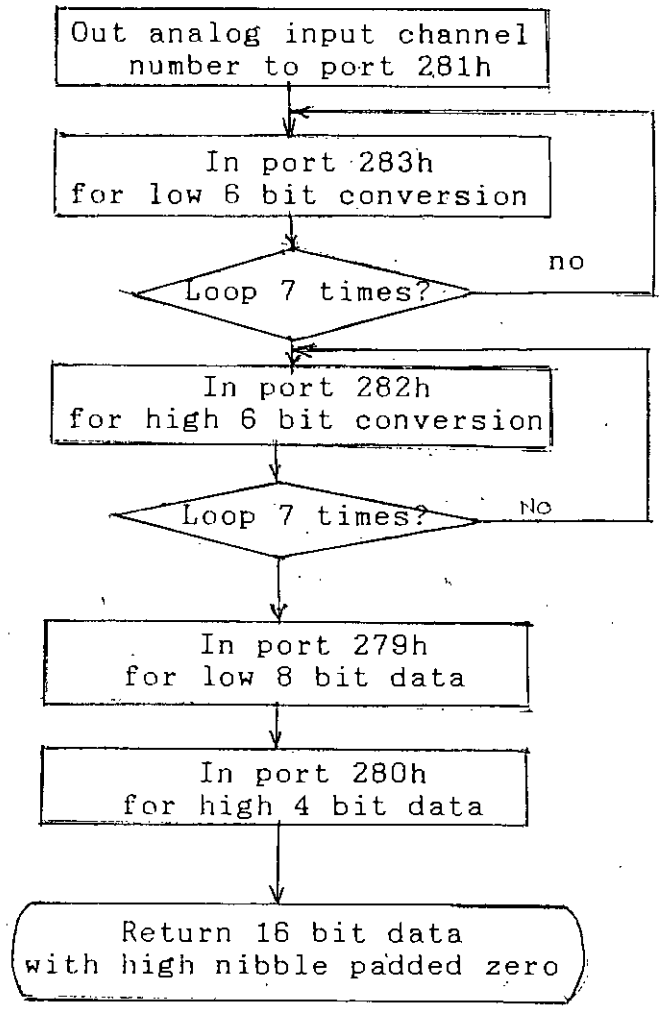
1. Channel selection and conversion initialization:

The IBM analog to digital converter (ADC) card has internal analog multiplexer with 16 input channels which may receive signals from 16 different sensors. To convert signal from the temperature and flow sensor, the channel number connected to the sensor should be selected and this is out to port hex 278. It requires to loop 7 times to start analog to digital conversion process for low 6 bit and high 6 bit.

2. Reading data from ADC:

After conversion initialization, from port 279h low 8 bit and from port 280h high 4 bit data is read i.e. a 12 bit AD data is now available.

Flow chart for reading data from ADC card is:



5.5.2 SOFTWARE FOR HARDWARE CONTROL

The data out from the computer to the hardware is through the printer port. The printer port has 8 bit data out line (port 378h), 3 bit address line A0,A1 and A2 and one strobe line (port 37Ah). To out 8 bit data on the data bus the out put port is 123 and to out address and strobe out put port is 123. To get status of the solenoid valve and interrupt of end of motor rotation, the

input port 379h is used.

1. Initialization of stepper motor environment:

The stepper motor uses the interrupt line of the printer card of the personal computer. This interrupt line is connected through the interrupt controller 8259. Before starting the process control, the interrupt controller 8259 requires to set interrupt enable bit high. As interrupt 15 is connected to the interrupt line of the printer port, initial interrupt vector location is first saved and int_15 routine address is loaded. This is done by calling initmotor().

2. Stepper motor control program:

The stepper motor control program uses the technique that to start motor with step data and direction of rotation it calls soft interrupt 15 and interrupt 15 calls runmotor function.

a)The runmotor function: It first load 8 bit rotation data to stepper motor controller (SMC) by placing 8 bit data to data bus and lowering A1 momentarily. Now load the direction data in solenoid valve control and other (SVCO) register. Start rotation bit is set high temporarily by loading this bit first high in SVCO register and subsequently loading low in SVCO register.

b)The stepper routine: When interrupt 15 is called it first check whether the higher 8 bit of step data is zero. If it

finds this data is zero it simply load dummy data to SMC to lower the interrupt of SMC and disable further hard interrupt call, else set step data 0 such that stepper motor rotates 256 and calls run motor.

3.Heater power control:

The program of the heater power control is written in assembly. The heater power is controlled by setting firing angle of the SCR/TRIAC. The fire angle is first stored in the register 74198 by sending 8 bit fireangle on the data bus and setting A0 high, then bringing strobe line temporarily high and finally lowering A0 low. Now start heat bit of solenoid valve controller and other register (SVC0) 74198 is stored high using address A2 and strobe. During data loading interrupt is disabled because interrupt is called by motor controller which uses this resister.

4.Solenoid valve control:

To control solenoid valve, the solenoid valve data is stored SVC0 register by placing bit 1,2 and 3 to data bus accordingly and lowering A0 and strobe temporarily. To on the solenoid valve the corresponding bit is set high and done by routine svon and to off the solenoid valve the corresponding bit is reset and done by routine svoff.

5. Solenoid valve status:

To get the solenoid valve status the data from the port 379h is read and returned. The data returned is in the form that if

solenoid valve 1 is on then bit 0 is 1 and if solenoid valve 2 is on bit 1 is 1 and so on.

CHAPTER 6
CONCLUSION AND SUGGESTION FOR FUTURE WORK

CHAPTER 6 CONCLUSION AND SUGGESTION FOR FUTURE WORK

6.1 CONCLUSION

The work involved development of the necessary hardware for the measurement and control of the controlled variables and the necessary interfaces. To measure the flow of gases, thermistor based flow sensor was designed and to control flow stepper motor controlled valve actuator was used. To measure temperature thermocouple amplifier and to control heater power phase fired heater controller were developed. To interface solenoid valves with the computer, a special interface circuit was designed.

Handshaking programs were written to interface with these hardware developed . All data acquisitions, implementations of control commands were written in assembly to perform at high speeds and repeatedly. Complex part of control programs and to filter noise digital filter programs were written in high level language C. To represent data, status and alarm graphical representation program is written in C. The control algorithm and the filter algorithm were verified by simulation.

The autotuner requires little prior information. On the other hand selftuning regulators can use more complex control laws with better performance.

6.2 SUGGESTION FOR FUTURE WORK

To enhance the controller and filter algorithm as an extension of the present work or as an independent work, the following work may be undertaken.

1. In filter design, low order filter was chosen . For process with noisy environment, higher order digital filter may be used. Hardware implementation of digital filter may be carried out as an independent work.

2. If process model is known, the Kalman filter can be implemented to have better result.

3. The autotuner and selftuning regulator will however not work if the prior guesses are too far off. For better results autotuner and selftuning algorithm can be combined for better performance. An expert system can be implemented for this purpose.

4. If the process model is known, Model Predictive Control can be implemented with high speed computation facility.

5. To handle alarm, detect and diagnose failure and optimize control expert systems can be developed.

BIBLIOGRAPHY

BIBLIOGRAPHY

1.	Rakesh Govind,G.J. Power	"Control System Synthesis Strategies" AIChE Journal Vol 28, No.1,, 1982;
2.	M. Morary	"Advances in Process Control Theory" Chemical Engineering Progress, Oct 1988.
3.	A.Jutan, J.F.MacGregor & J.D. Wright	"Multivariable Computer Control of a Butane Hydrogenolysis Reactor Part II" Chemical Engineering Progress, Oct 1988.
4.	J.R. Richard & P.D. Shmeller,Jr.	"Perspective of Industrial Control" Chemical Engineering Control Oct 1988.
5.	Driscol	Inustrial Electronic Device Circuits and Application. McGraw Hill Inc.
6.	D. Patrabinous	Principles of Industrial Instrumets.
7.	J. J. Carr	Elements of Electronic Instrumentation & Measurement.
8.	Intel Corporation	Measurement & Control Computer Hardware Ref.Mannual. Intel Corporation.
9.	L.P.S. Bebington & S. Annadural.	"Microprocessor Based Stepper Drove With Enhenced Available Torque" Electronioc For You Nov 1988;
10.	O. Ernest & Doeblin	System Modeling & Response-Theoretical and Experimental Approch.
11.		Phillips Data Hand Book Part 2a.

12. R.E. Ziemer, W.H. Tranter & D.R. Famin — Signals and System-Continuous and Discrete MacMillan Publishing, N.Y.

13. L.R. Rabiner and et. al. — "Some Comparison Between FIR and IIR Digital Filter" Selected Papers In Digital Signal Processing II MacMillan Publishing, N.Y.

14. Tsai, Lane & Lin — Modern Control Techniques for the Processing Industries.

15. R.W. Hamming — Digital Filters, Theory and Practice. Prentice Hall, Inc, N.J., 1977.

16. Paul Katz — Digital Control Using Microprocesss Prentice Hall International 1981.

17. T.F. Edger — "Sampling & Filtering for Discrete Time System." AIChE Modular Instruction Vol 3 1983.

18. L.R.Rabiner, N.Y.Graham, H.D.Helms — "Linear Programming Design Of IIR Digital Filter With Arbitrary Magnitude Function." Selected Papers In Digital Signal Processing II, IEEE Press.

19. F.P Lee — Loss prevension in the process industry.

20. G.Stephanopoulos — Chemical Process Control Prentice Hall, Englewood Cliff. N.J. 1983.

21. K.J.Astrom — "Adaptive, Autotuning and Smart Control." Chemical process Control CPC III- CACHE Pub.

22. T.F. Edgar & D.E. Seborg — Adaptive Control AIChE Modular Instruction Series Vol IV, 1986.

23. T.F. Edgar — "Conversion of Continuous Time Models to Discrete-Time Models AIChE Modular Instruction Series Vol III.

24. A.B. Corripio — Digital Control Techniques AIChE Modular Instruction Series Vol III.

25. C.H. Houpis — Digital Control System, Theory Hardware and Software McGraw Hill.

26. C.E. Garcia, D.M. Prett — Advances in Industrial Model Predictive Control Chemical Process Control CPC III.

APPENDICES

APPENDIX A

DATA FLOW DIAGRAM OF THE PROCESS CONTROL

The data flow diagram of the process control is shown in figures A-1 to A-5. The data dictionary of the program is given below.

DATA DICTIONERY:

Name and Number: Discretize data: DFD 1.0

Composition: AtoD 1.1

Get_temp 1.2

Get_flow 1.3

Variable names:

Channel number: Which channel to be digitised, range 0 to 15.

Raw_temp_data: Digitised temperature data after 12 bit adda converter, 16 bit long with highest nibble zero.

Raw_flow_data: Digitised flow data after 12 bit adda converter, 16 bit long with highest nibble to zero.

Thermo_no: Thermocouple number.

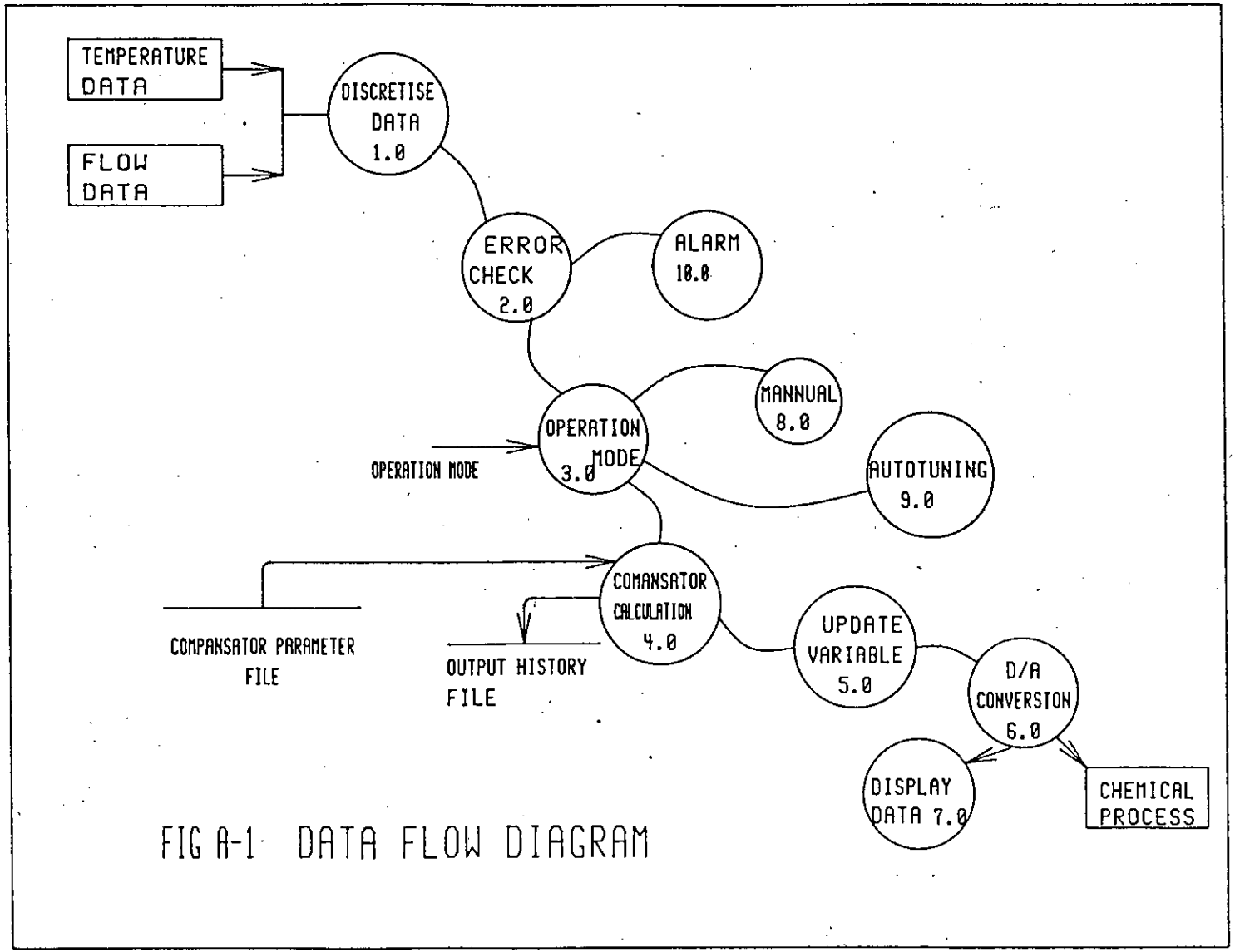


FIG A-1 DATA FLOW DIAGRAM

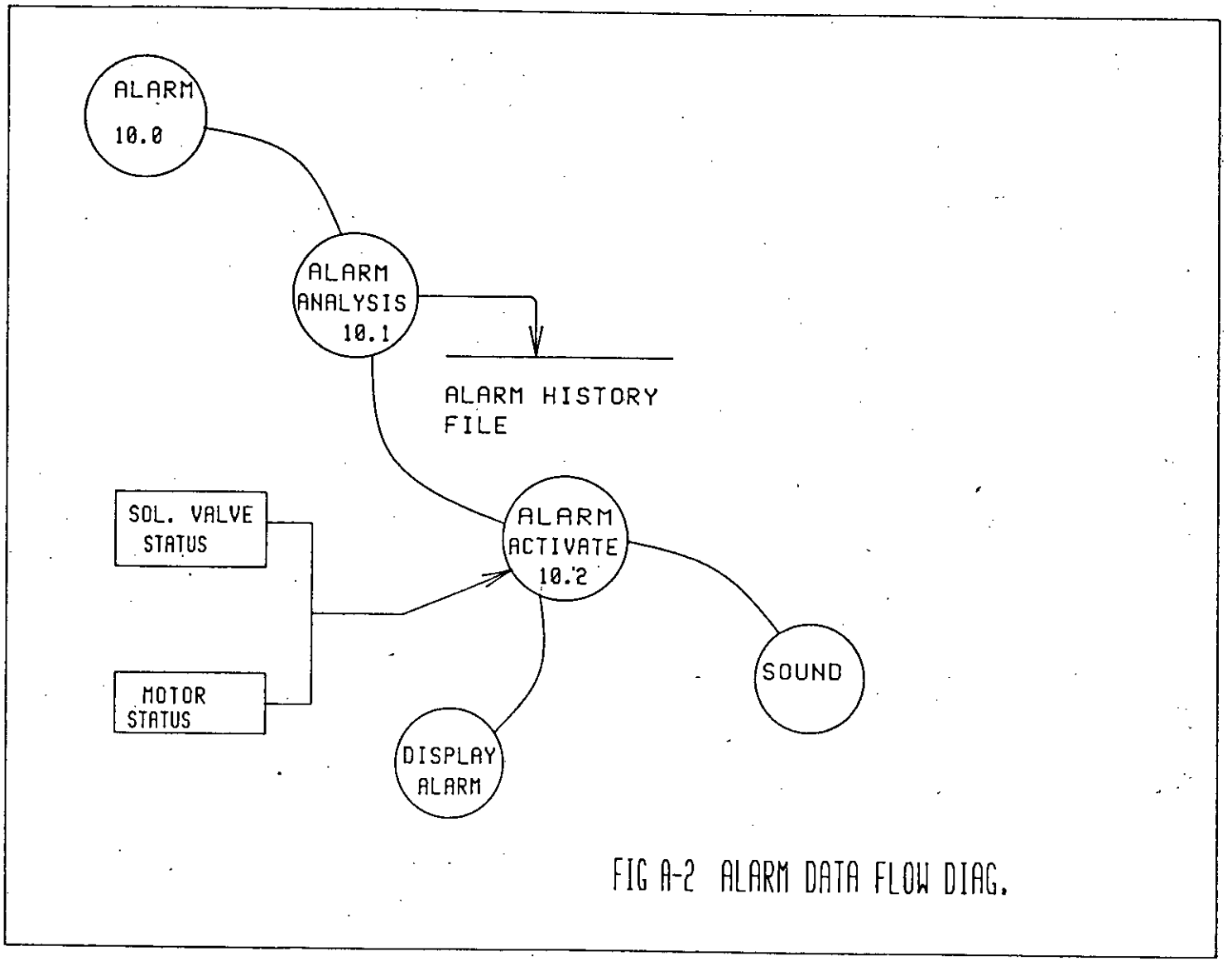


FIG A-2 ALARM DATA FLOW DIAG.

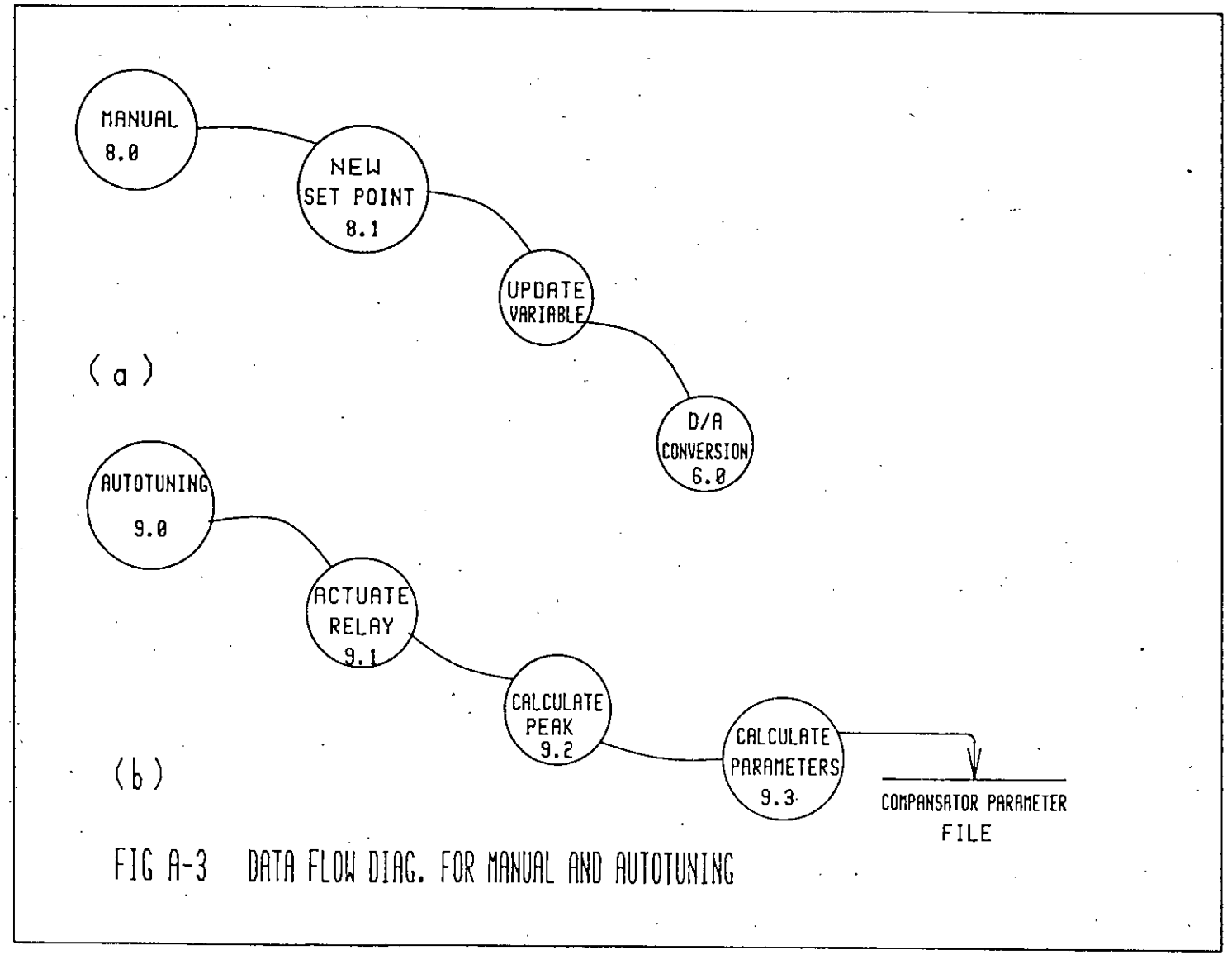


FIG A-3 DATA FLOW DIAG. FOR MANUAL AND AUTOTUNING

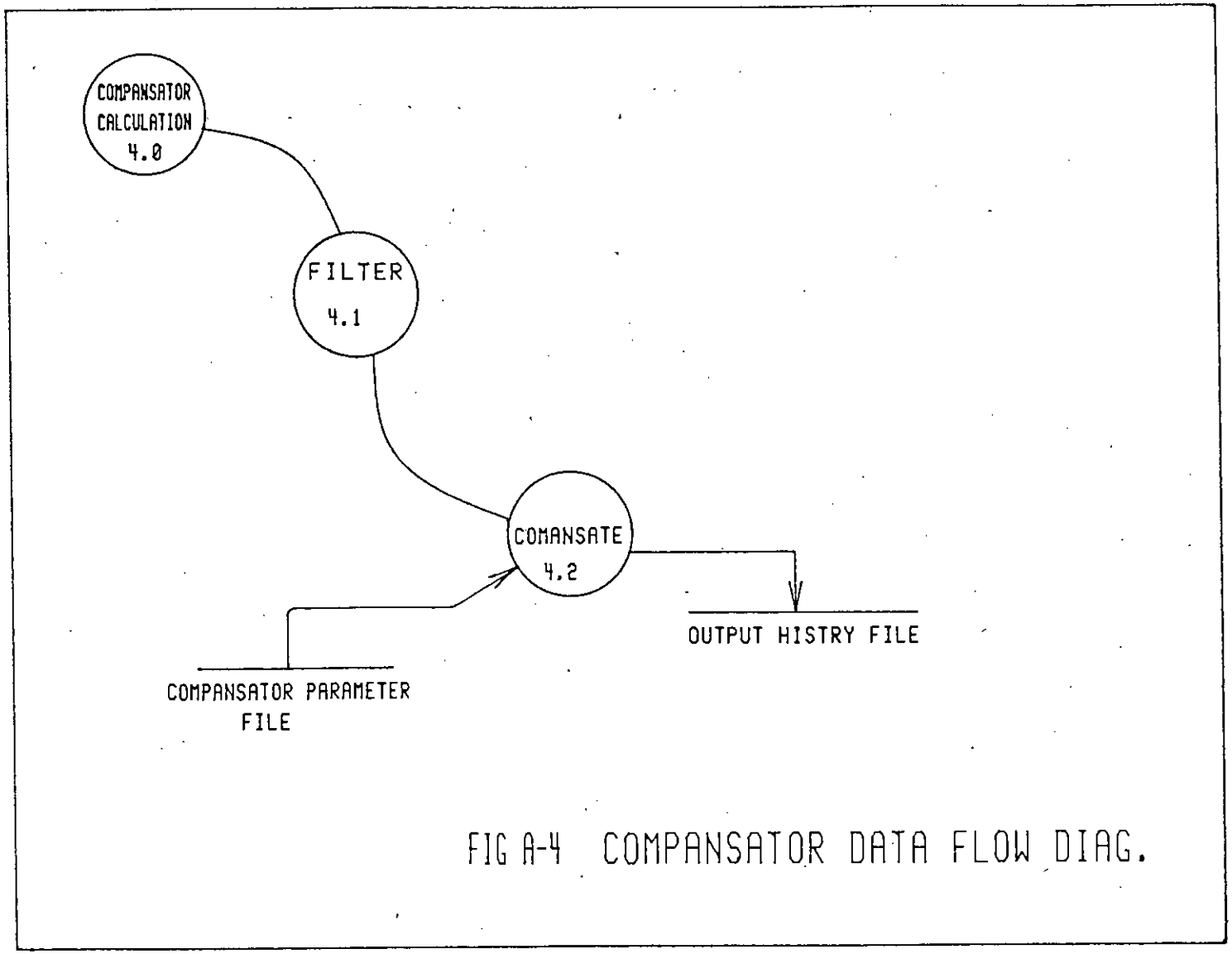


FIG A-4 COMPANSATOR DATA FLOW DIAG.

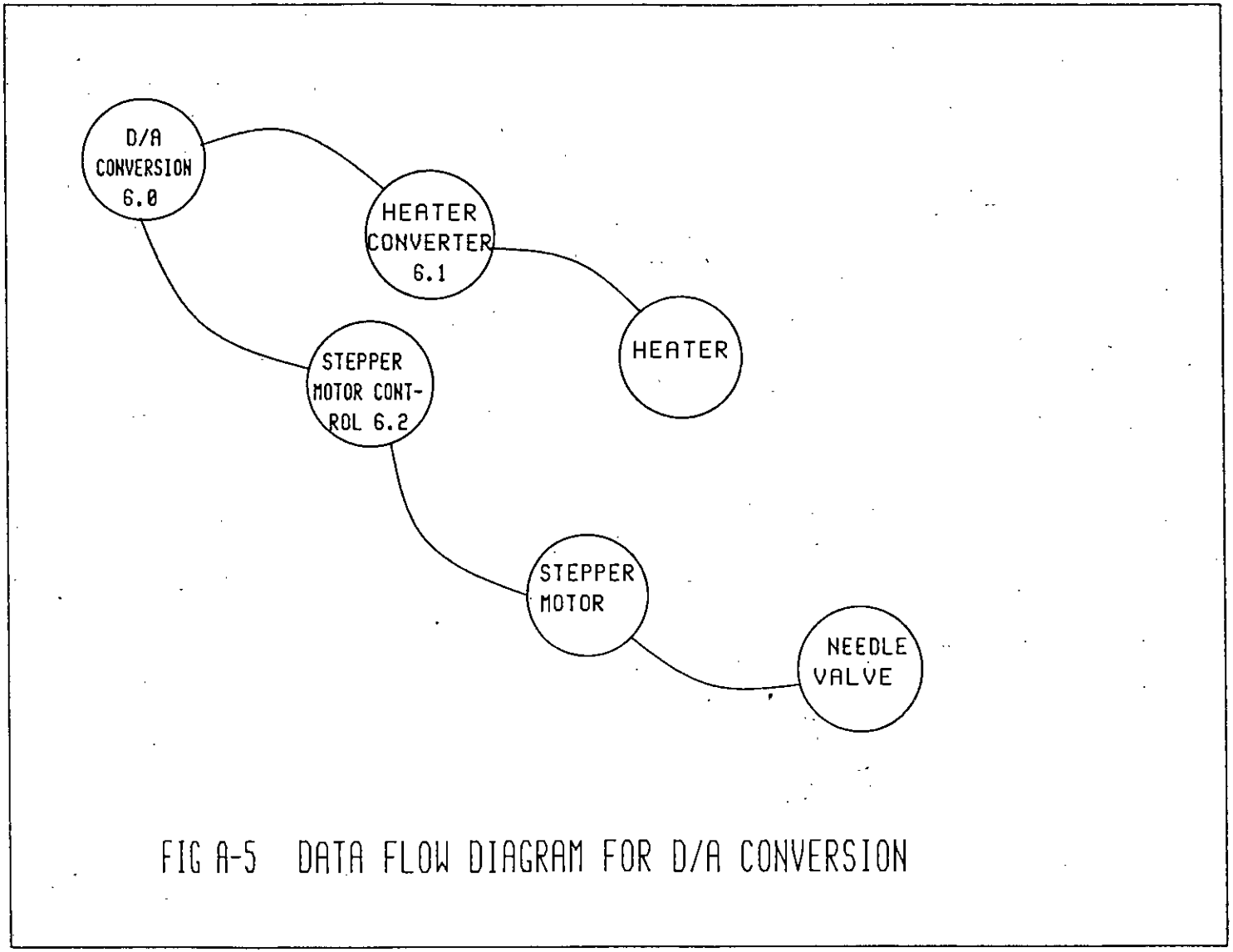


FIG A-5 DATA FLOW DIAGRAM FOR D/A CONVERSION

Process description:

```
AtoD.
{
Set channel number ;
Clear AD register ;
Loop 7 times for low 6 bit ;
Loop 7 times for high 6 bit ;
Read high order 4 bit ;
Read low order 8 bit ;
Store in 16 bit with high nibble zero ;
Return this data. ;
}
```

```
get_temp
{
Get thermo_no;
Call atod with thermo_no connected channel number;
store in raw_temp_data ;
}
Get_flow
{
call atod with flow sensor connectred channel_no ;
store in raw_flow_data;
}
```

Name and Number: Limit Check DFD 2.0

Composition: Lmt_chck_Temp 2.1

Lmt_chck_Flow 2.2

Variable Name:

Raw_Temp_data	A/D output of thermocouple amplifier
Raw_flow_data	A/D output of flow sensor.
Temp_limit	The upper limit of teperature
Flow_limit	Upper limit of flow;
Count_temp	Temporary count register to store how many times temerature remains higher than the upper limit continuously.
Count_flow	Same as Cont_temp but for flow.
Thermo_no	Thermocouple number.

Process Description:

lmt_chck_temp 2.1

{

Get raw_temp_data for given thermo_no;

If raw_temp_data > temp_limit increment count_temp by 1;

Else set count to zero;

If count_temp > 3 call alarm;

If count_temp <> 0 get temperature data again ;

Else return raw_temp_data;

}

lmt_chck_flow 2.1

```
{
Get raw_flow_data ;
If(raw_flow_data >flow_limit)increment  count_flow
by 1 ;
Else set count to zero;
If count_flow > 3 call alarm;
If count_flow = 0 call ATOD ;
Else return raw_flow_data;
}
```

Name and Number: Operation mode selection DFD 3.0

Composition: Op_mode_select 3.1

Variable names:

Op_mode The operator enters the operation mode which have a value 0 to 4. 0= automatic, 1= manual, 2= tuning.

Prev_mode Previous mode of operation entry.

Process description:

```
Op_mode_select 3.1
{
If  operator has no entry continue on with prev-
mode entry;
Else call manual for 1;
          tuning for 2;
}
```

Name and Number: Compansator calculation: DFD 4.0

Composition: Prefilter1 4.1

Prefilter2 4.2

Compansator1 4.3

Variable names:

Raw_temp_data: Digitised value of temperature measurement.

Raw_flow_data: Digitised value of flow measurement.

Filtered_temperature: Raw temperature data is filtered by 4th order butterworth filter.

Filtered_flow: raw flow data is filtered by 4th order Butterworth filter.

Compansator output:Filtered temperature is compansated by PID compansator.

File names:

Output history file: Collects the filtered data,compansator ouput with time.

Composition: Time of sampling,value sampling,compansator output.

Process description:

Prefilter1 4.1

{

Initialise filter parameters

Get current data ;

```
    Calculate new output ;
    Update old data ;
    Update old output ;
    Return new output;
          }
Prefilter2:   4.2
    {Same as Prefilter1.
    }
Compansator1      4.3
    {
    Initialise compansator parameters;
    Compute data for heater fire angle;
    Store compansated data;
    Return compansated data;
    }
```

Name and Number: Update variable DFD 5.0

Composition: Update1 5.1

Update2 5.2

Variable names:

Update-flow:Updated flow data

Updated-temperature: Updated temperature.

Process description:

Update1: 5.1

{

Get current data;

Update old temperature data;

}

Update2: 5.2

{Same as Update1for flow data.

}

Name and Number: D/A conversion DFD 6.0

Alises:

Composition:

Heater DA 6.1

Stepper DA 6.2

Start motor 6.2.1

Rum motor 6.2.2

Int-15 6.2.3

*Notes: Stepper motor contol uses hardware interrupt.

Variable names:

No_Of_Steps:Stepper motor steps,16 bit data.

Direction_of_rotation: Direction of rotation is anticlockwise if value is 0 else clockwise.

Fire_angle: 8 bit fireangle for heater D/A converter.

Process description:

Heater 6.1

```
{
  Get firing angle;
  Store firing angle in resister;
  Set heat start bit high ;
 }
```

Stepper motor 6.2

Start motor 6.2.1

```
{
```

```
        Get no_of_step.
        Divide number of step by 256 .
        Store        in rest_rot.
        Start  rotation  with  modulus  of  division
        calling run motor.
        }

Run motor   6.2.2
        {If  rotation is not completed  by  checking
        rest_rot
            Store  step  8  bit  no_of_rotation  to
            stepper register ;
            start stepping;
        else store dummy data to stepper register;
        }
Int_15      6.2.3
{
Set no_of_step 256;
decrese rest_rot by 1;
call run motor;
}
```

Name and Number: Display Data & Status 7.0

Composition: Display-status 7.1

Display-alarm 7.2

Display-temperature 7.3

Display-flow 7.4

Display-power to heater 7.5

Set-status 7.6

Reset-status 7.7

Set-alarm 7.8

Reset-alarm 7.9

Set-temperature 7.10

Set-flow 7.11

Set-power 7.12

Variable name:

Status The status of the plant, 0=Automatic mode, 1=Manual mode, 2= Tuning mode.

Alarm 0=Temp. Hi, 1=Flow High, 5=Temp. low, 6=Flow low

3 extra high and 3 extra low alarm for future expansion.

Themo_no Thermocouple number, ranging from 0 to 3.

filtered_Temp Temperature data.

Filtered-flow Flow data of gas from filter.

Power Heater power calculating from firing angle .

Process description:

```
Set_status      7.7
{
Get mode of operation;
Display mode;
}
Set_alarm       7.8
{
Get alarm number;
Clasify high or low;
Display on screen with sound;
}

Reset_alarm     7.9
{
Get alarm number;
Classify high or low;
Remove  alarm;
}
Set_temperature      7.10
{
Get temperatur data and thermocouple number;
Scale data;
Reset old pointer;
set pointer to new position;
}
```

Name and Number: Manual 8.0

Composition:

Get_new_fireangle 8.1

Get_new_step 8.2

Process description:

Get_new_fireangle 8.1

{

read fireangle from console;

store Fire_angle;

Call heater;

}

Get_new_step 8.2

{

Read step and direction from console;

Store them in No_of_steps and direction_of_rotation;

Call start motor.

}

Choice 8.3

{

Get choice from console

If choice is 1 call get_new_step

If choice is 2 call get_new_fireangle;

If choice is 0 return to main stream;

Repeat

}

Name and Number: Autuning 9.0

Composition:

Put new data	9.1
Relay	9.2
Peak	9.3
Parameter calc	9.4

Variable names:

New_error: set_point - measured and filtered data;

process_array: stores previous data of relay output;

amp_err_range: Allowable range of error for two consecutive peak of output;

measur_array: store of previous measured data

Relay_amplitd: Amplitude of relay oscilation;

KU,TU: Critical gain and critical period;

Peak_data: amplitude of measured data;

KC,TAO_I,TAO_D: Parameters of PID controller.

File names:

Process description:

```
Put_new_data
{
get new data
Shift data of array
Put new data in last position of array
}
```

```
Relay
{
Get error data;
If error_data > 0 set present_flag=1;
Else set present_flag=0;
If (previous_flag != present_flag) set amplitude of relay
previous_flag = present flag;
return amplitude of relay;
}
Peak
{
Get new data
initialize peak;
if new data > previous data set flag to 0
if  new data < previous data and flag equals 0 set  peak  to
previous data and set flag to 1 and get new_time;
period = old_time - new_time;
old_time = new_time;
return peak and period;
}
parameter_calc
{
Call relay with set_point - measured data
Call peak
If previous_peak - new_peak less than error_range
     {calculate KU and TU
```

```
        calculate KC,TAO_I,TAO_D from TU and KU
        Stores all in compansator coefficint file.
        return to main stream }
    Else repeat
    }
```

```
#include <process.h>
#include <stdlib.h>
unsigned int channel,step,direct,valve_no,fire_angle;
int i,j,k;
double scale;
float qq;

/*   This routine returns average of 10 data from the
     channel indicated */
float filter_avg1(int channel_no){
int kk;
channel=0;

for(kk=0;kk<10;kk++)
        {
        channel+=atod(channel_no);
        }
channel/=10;
return(channel);
}


/*   This routine displays four filtered temerature data on the
     screen and stores data in file data.log until q is pressed  */

void chanel(void)
{
float chan,mx;
int kk;
for(;;)
     {
     for(kk=0;kk<4;kk++)
          {
          chan=filter(kk); /* fourth order Butterworth filter */
          if(k % 4 ==0)
               printf("\n ch(%2d) = %5.4f ", kk, qq = scale * chan);
          else
               printf("   ch(%2d) = %5.4f  ",kk, qq = scale * chan);
          }
     store(&qq);
     printf("\nTo quit press q\n");
     if(getch()=='q')break;
     }

}       /*end of channel*/

/*   This routine starts running stepper motor with keyboard entered
     step data and direction data and returns the sucess of
     motor rotation*/
```

```
int     motor(void)
{
int sucess;
printf("\n\tNo of Steps?      ");
scanf("%d",&step);
printf("\n No of steps?=%d",step);
printf("\n\tDirection?        ");
scanf("%d",&direct);
printf("\nStepper motor sucess=%d\n",sucess=stepper(step,direct));
        /*stepper routine is an external routine which starts motor
        after loading number of steps and direction of rotation
        properly. */
getch();
return(sucess);
}       /* end of motor */


/*   This routine get firing angle data, ranging from 10 to 244, from
     key board.  */
int heat(void)
{
printf("\n\tEnter firing angle?   ");
scanf("%d",&fire_angle);
heater(fire_angle);

     /* heater is an external routine which loads firing angle data
     to heater controller circuit.      */

}


/*   This routine shows the condition of the solenoid valve in coded
     form i.e. sol. valve 1 status * 1 + sol. valve 2 status * 2 +
               sol. valve 3 status * 4.
     To on the required solenoid valve the valve number is entered
     through key board. The sv_on routine is an external routine which
     loads high in proper bit of solenoid vave control register.  */

void valveon(void){
printf("\nCondition of valves is  %d   \n",getsolset());
printf("What is your valve to on?          ");
scanf("%d",&valve_no);
sv_on(valve_no);
printf("\ncondition of sv%2d is %d        ",valve_no,sv_on(valve_no));

}       /*end of valveon */

/*   This routine shows the condition of the solenoid valve in coded
     form i.e.  sol. valve 1 status * 1 + sol. valve 2 status * 2 +
                sol. valve 3 status * 4.
```

```
     To off the required solenoid valve the valve number is entered
     through key board. The sv_off routine is an external routine
     which loads low in proper bit of solenoid vave control
     register.          */

void  valveoff(void){
printf("\n Condition of valves is  %d  ",getsolset());
printf("\nWhat is your valve to off?  ");
scanf("%d",&valve_no);
printf("\ncondition of sv%2d is %d   ",valve_no,sv_off(valve_no));
sv_off(valve_no);

}       /* end of valveoff  */

/*   The menu routine displays the menu of operations i.e. stepper
     motor on, solenoid valve on/off, setting heater fire angle or
     display of thermocouple voltage data.  */

int menu(){

int choice;
clrscr();
gotoxy(5,5);
printf("YOUR OPTIONS ARE...");
printf("\n\t1.\tMotor On..");
printf("\n\t2.\tSolenoid valve on..");
printf("\n\t3.\tSolenoid valve off..");
printf("\n\t4.\tHeater fire angle..");
printf("\n\t5.\tHeater off..");
printf("\n\t6.\tThermocouple voltage..");
printf("\n\t7.\tQuit..");
do{
        gotoxy(5,20);
        printf("\t\t\tYOUR CHOICE IS...");
        scanf("%2d",&choice);
        }while(choice<0 || choice >7);

return(choice);

}       /* end of menu */

/*   The menup routine calls the required operation routine according
     to menu routine choice.  */

menup(void){
int choice;
while((choice=menu()) !=7){
        switch(choice){
                case 1:clrscr();motor();break;
                case 2:clrscr();valveon();break;
```

```
                case 3:clrscr();        valveoff();break;
                case 4:clrscr();heat();break;
                case 5:clrscr();stopheat();break;
                case 6:clrscr();chanel();
                default:break;
                }

        }
printf("\n\t\tPress any key.....");
getch();

}       /* end of menup  */

manual()
{
char ch;
scale=9.000/4095;       /* scaling of the 9 volt by 12 bit data */
k=0;

fileintl();     /*Initialisation of file to store data */
initmotor();    /*To initialize stepper motor start environment */
initfilter();   /*The filter parameters are initialised */

menup();

/*   Before closing mannual mode following routines are called. */

closemotor()    /*To retrive previous environment */
stopheat();     /*To stop heating of heater     */
clrscr();
printf("press any key.....");
getch();

}

main()
{
manual();
}
```

```
#include <math.h>
#include <stdio.h>
#include <graphics.h>
#include <dos.h>
#include <stdlib.h>
#include <a:\process\filter.h>

struct time *timess;     /* start time pointer    */
struct time *timee;      /* end time pointer      */
extern float channel(int channel_no);

float xreal[250],value[250],value_exp[250]; /*Array for display */
float x[7],y[7],coeff_y[7],coeff_x[7];  /* Array for filter routine */
int order=4;


/* To initialise graphics mode to display filter response  */

void initial()
{int graphdriver,graphmode;
graphdriver=DETECT;
initgraph(&graphdriver,&graphmode,"\\bgi");
}       ./*end of initial */


/* To plot an array,value in y direction,maximum of 250 points */

void plotting(int axis,float array[],int point,float mult)
{
int i,x,y;
x=0;
y=axis-array[0] * mult;                 /* mult times enlarged */
moveto(x,y);
for(i=0;i<point;i++){
        y=axis-array[i] * mult;     /* mult times enlarged */
        x+=2;                       /* increament in x axis by 2 */
        lineto(x,y);
        }

}       /* end of plotting routine */

/*   Put a new data at the end of the array and shift the previous
     data to immediate previous position     */

void putnew( float array[],float data )
{
int i;
for(i=0;i<order;i++)
        array[i]=array[i+1];
array[order]=data;
}       /* end of putnew routine*/
```

```
/*   nth order filter calculation,Chebicheve or Butterworth depending
     on coefficient supplied */

float filter_calc(void)
{
int i;
y[order]=0;

for (i=0;i<=order;i++)
        y[order]+=coeff_y[i] *y[order-i] +coeff_x[i] * x[order-i] ;

return (y[order]);

}       /* end of filter routine */


/*   Initialisation of the coefficient for 4th order butter filter
     type I    */

void set_coeff_butter(int a)
{
coeff_y[0]=0.0;
coeff_y[1]=1.5703989;
coeff_y[2]=-1.2756133;
coeff_y[3]=+0.4844033;
coeff_y[4]=-0.076197;
coeff_x[0]=0.018563;
coeff_x[1]=0.074252;
coeff_x[2]=0.111378;
coeff_x[3]=0.074252;
coeff_x[4]=0.018563;

} /* end of set_coeff_butter for 4th order */


/*   initialize an array with data d */

void start(float array[], float d)
{    int i;
for (i=0;i<=order;i++)
        array[i]=d;

}       /*end of start */
```

```
/*  Initialisation of filter parameters and x data and y data array */

void initfilter()
{
set_coeff_butter(0);
start(x,0);start(y,0);

}       /* end of initfilter    */


/*    Filter routine takes data from given channel number of AD card
      and uses 4th order Butterworth filter equation to filter out
      noise and returns filtered data.    */

float filter(int chanel_no){
int i;
float yx;
for(i=0;i<20;i++){
        putnew(x,atod(chanel_no));
        yx=filter_calc();
        putnew(y,yx);
        }
printf("\n Filtering %d samples %f \n",i,y[order]);
return(yx);

}       /*end of filter  */



/*
        This section was used for tesing the routines when developed

main()
{ int i,j,k=250,r;
float z,d;
float scale;                /* scale for data read    */
order=4;
scale=100.0/4.0950;
timess=malloc(sizeof(struct time));
timee=malloc(sizeof(struct time));
initfilter();               /* Initialisation of filter parameters */
for (j=0;j<5;j++){
     gettime(timess);
     filter(2);
     gettime(timee);
     printf("START TIME = %d:%d:%d.%d",timess->ti_hour,\
          timess->ti_min,timess->ti_sec,timess->ti_hund);
     printf("\nSTOP  TIME = %d:%d:%d.%d",timee->ti_hour,\
          timee->ti_min,timee->ti_sec,timee->ti_hund);

        getch();
```

```
        initial();        /*Initialisation of graphics session */
        plotting(250,value,k,par);       /* Filtered data */
        plotting(100,xreal,k,par/10.);  /* Real data    */
        getch();
        closegraph();
        }

}  end of main */
```

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
int wdataptr,rdataptr;  /* write data track and read data track */
int new,c,a;
float q=5.2231;
float *buff;            /*buffer of temprary data read or write */
FILE *in,*out;          /* input file and output file pointer   */

/*   This routine initialize file where the data to be read or write;
     the data stored is in binery form to reduce file size.     */

void fileintl(void)
{
clrscr();
buff = calloc(sizeof(float),10);
in=fopen("a:data.log","wb");
fclose(in);
if((in=fopen("a:data.log","r+b"))==NULL)
        {
        fprintf(stderr,"Cannot open file.\n");
        return(1);
        }
wdataptr=0;     /* write data track and read data track */
rdataptr=0;     /* are initialized       */

}       /*end of fileinitl      */

/*   This routine stores float temerature data; wdataptr keep track of
     data to write      */

void  store(float *data)

fseek(in,wdataptr*sizeof(float),0);
wdataptr+=fwrite(data,sizeof(float),1,in);
printf(" %d\n ",ftell(in));
rdataptr=wdataptr;

        /*end of store  */

/*      retrieves data that stored earlier and shows in screen  */
void retrieve(void)

int n;

fseek(in,((rdataptr-10) < 0 ? 0 : (rdataptr-10) )*sizeof(float),0);
if( (fread (buff,sizeof(float),10,in))==NULL)  printf("Oh no!");
for (n=0;n<10;n++)
printf(" %5.4f",buff[n]);
        /* end of retrieve  */
```

```
/* Closes the the file openned  */

void  fileclose(void)
{
fclose(in);
}

/*
        This section is used for testing of routines.

main()
{
fileintl();
while(getch()!='q')
        {
        q+=1.0;
        store(&q);
        retrieve();
        }
fileclose();
}*/
```

```
#include <graphics.h>

/*   This module can be used to display data in monitor.The
     functions used are:
        void init(void);
        void display(void);
              void display_status(void);
              void display_alarm(void);
              void display_temp(void);
              void display_flow(void);
              void display_power(void);
        void setstatus(int status);
        void resetstatus(int status);
        void setalarm(int alarm_no);
        void resetalarm(int alarm_no);
        void settemppoint(int therm_no,float temerature);
        void setflowpoint(float flow);
        void setpowerpoint(float power);

The global variables used for different functions
        TOP_X,TOP_Y,BOTTOM_X,BOTTOM_Y   status box size;
        DISPLACE_BOX            displacement of status box;
        T_START_X,T_START_Y     teperature monitor position;
        F_START_X,F_START_Y     flow monitor position;
        A_START_X,A_START_Y     alarm monitor start position;
        P_START_X,P_START_Y     power monitor start position;
        TEMP_START,TEMP_END     temperature initial & end value ;
        FLOW_START,FLOW_END     flow initial & end value for meter;
        POWER_START,POWER_END   power initial & end value for meter;
        scaleset,pointerset     scale and pointer data;
        status,alarm            status and alarm data;
        prev.....               previous state of .....

*/

int TOP_X =10, TOP_Y= 10,BOTTOM_X=50,BOTTOM_Y=30;
int DISPLACE_TEMP=50;
int T_START_X=550,T_START_Y=10,T_WIDTH=35;
int F_START_X=625,F_START_Y=140;
int A_START_X=555,A_START_Y=280;
int P_START_X=550,P_START_Y=140;
int O_START_X=350,O_START_Y=25;
float TEMP_START=0,TEMP_END=10,FLOW_START=0,FLOW_END=10;
float POWER_START=30,POWER_END=60;

int * scaleset,*pointset;
int *status,*alarm;
int prevstatus[5]={0},prevalarm[10]={0};
int prevpos[4]={105,105,105,105};
int prevpower=0,prevflow=0;
```

```
/* To initialise the graphics display */
init()
{
int driver=DETECT,mode,g_error;
initgraph(&driver,&mode,"\\bgi");
if((g_error=graphresult())!=0)
        printf("initgraph error: %s",grapherrormsg(g_error));
}


/*   To display the status of the solenoid valve,heater or motor
          0 = solenoid valve R
          1 = Solenoid valve S
          2 = Solenoid valve T
          3 = Heater
          4 = Flow i.e. stepper motor   */

display_status()
{
int i,j;
status= malloc(imagesize(TOP_X,TOP_Y,BOTTOM_X,BOTTOM_Y));
setfillstyle(11,1);
rectangle(TOP_X,TOP_Y,BOTTOM_X,BOTTOM_Y);
floodfill(15,15,WHITE);
getimage(TOP_X,TOP_Y,BOTTOM_X,BOTTOM_Y,status);
for(i=0;i<5;i++)
        {
        j=i*DISPLACE_TEMP;
        putimage(j+10,10,status,COPY_PUT);
        switch(i){
                case 0:outtextxy(j+15,15,"SV R");break;
                case 1:outtextxy(j+15,15,"SV S");break;
                case 2:outtextxy(j+15,15,"SV T");break;
                case 3:outtextxy(j+15,15,"HEAT");break;
                case 4:outtextxy(j+15,15,"FLOW");
                }
        }
putimage(10,35,status,XOR_PUT);
outtextxy(15,40,"ON");
getimage(10,35,10+BOTTOM_X-TOP_X,35+BOTTOM_Y-TOP_Y,status);
putimage(10,35,status,XOR_PUT);
j=0;
for(i=0;i<5;i++)
        {
        putimage(j+10,35,status,COPY_PUT);
        j+=DISPLACE_TEMP;
        }
}
```

```
/* To set the status of the motor and others described above */

setstatus(int i)
{
if(prevstatus[i]==0)
          putimage(i*DISPLACE_TEMP+10,35,status,XOR_PUT);
prevstatus[i]=1;
}

/*   To reset the status of the motor and others described
     earlier   */
resetstatus(int i)
{
if(prevstatus[i]==1)
          putimage(i*DISPLACE_TEMP+10,35,status,XOR_PUT);
prevstatus[i]=0;
}

/*   To display operation mode screen preparation routine  */

display_operation_mode()
{
int i,j;
outtextxy(O_START_X-15,O_START_Y-10,"OPERATION MODE");
setlinestyle(SOLID_LINE,0,THICK_WIDTH);
rectangle(O_START_X,O_START_Y,O_START_X+80,O_START_Y+30);
setlinestyle(SOLID_LINE,0,NORM_WIDTH);
rectangle(O_START_X+7,O_START_Y+7,O_START_X+70,O_START_Y+25);
setfillstyle(XHATCH_FILL,1);
floodfill(O_START_X+3,O_START_Y+3,WHITE);
}

/*   Operation mode setting routine
     0 = Auto;
     1 = Mannual;
     2 = Tune;
     3 = Start;
     4 = Shut down;                              */

setoperationmode(int mode)
{
setviewport(O_START_X+8,O_START_Y+8,O_START_X+69,O_START_Y+24,0);
clearviewport();
setviewport(0,0,getmaxx(),getmaxy(),0);
moveto(O_START_X+13,O_START_Y+13);
switch(mode)
        {
        case 0:{outtext(" AUTO "); break;}
        case 1:{outtext("MANUAL");break;}
        case 2:{outtext(" TUNE ");break;}
```

```
        case 3:{outtext(" START"); break;}
        case 4:{outtext(" SHUT ");break;}
        }
}

/*   To display alarm section */
display_alarm()
{
int x,y,i,j;
x=getmaxx();y=getmaxy();
setlinestyle(SOLID_LINE,0,THICK_WIDTH);
rectangle(A_START_X=x-200,A_START_Y=y-80,x,y);
setlinestyle(SOLID_LINE,0,NORM_WIDTH);
j=40;
for(i=0;i<4;i++)
        {
        line(A_START_X+j,A_START_Y,A_START_X+j,y);
        j+=40;
        }
line(A_START_X,A_START_Y+40,x,y-40);
outtextxy(A_START_X+80,A_START_Y-10,"ALARM ");
alarm=malloc(A_START_X+2,A_START_Y+2,A_START_X+38,A_START_Y+38);
rectangle(A_START_X+2,A_START_Y+2,A_START_X+37,A_START_Y+37);
rectangle(A_START_X+4,A_START_Y+8,A_START_X+36,A_START_Y+18);
rectangle(A_START_X+4,A_START_Y+18,A_START_X+36,A_START_Y+28);
setfillstyle(SOLID_FILL,WHITE);
floodfill(A_START_X+3,A_START_Y+3,WHITE);
getimage(A_START_X+2,A_START_Y+2,A_START_X+37,A_START_Y+37,alarm);
putimage(A_START_X+2,A_START_Y+2,alarm,XOR_PUT);
}

/*   To set alarm. Alarm no. from 0 to 4 for lo
                              and 5 to 9 for hi   */
setalarm(int alarm_no)
{
int x,y;
if (alarm_no>4)
        {y=A_START_Y+42;
        x=A_START_X+(alarm_no-5)*40+2;
        }
else
        {y=A_START_Y+3;
        x=A_START_X+(alarm_no)*40+2;
        }

if(prevalarm[alarm_no]==0)putimage(x,y,alarm,COPY_PUT);
x+=2;
y+=8;
if(alarm_no >4)outtextxy(x,y+10,"HIGH");
```

```
        case 3:{outtext(" START"); break;}
        case 4:{outtext(" SHUT ");break;}
        }
}


/*   To display alarm section */
display_alarm()
{
int x,y,i,j;
x=getmaxx();y=getmaxy();
setlinestyle(SOLID_LINE,0,THICK_WIDTH);
rectangle(A_START_X=x-200,A_START_Y=y-80,x,y);
setlinestyle(SOLID_LINE,0,NORM_WIDTH);
j=40;
for(i=0;i<4;i++)
        {
        line(A_START_X+j,A_START_Y,A_START_X+j,y);
        j+=40;
        }
line(A_START_X,A_START_Y+40,x,y-40);
outtextxy(A_START_X+80,A_START_Y-10,"ALARM ");
alarm=malloc(A_START_X+2,A_START_Y+2,A_START_X+38,A_START_Y+38);
rectangle(A_START_X+2,A_START_Y+2,A_START_X+37,A_START_Y+37);
rectangle(A_START_X+4,A_START_Y+8,A_START_X+36,A_START_Y+18);
rectangle(A_START_X+4,A_START_Y+18,A_START_X+36,A_START_Y+28);
setfillstyle(SOLID_FILL,WHITE);
floodfill(A_START_X+3,A_START_Y+3,WHITE);
getimage(A_START_X+2,A_START_Y+2,A_START_X+37,A_START_Y+37,alarm);
putimage(A_START_X+2,A_START_Y+2,alarm,XOR_PUT);
}

/*   To set alarm. Alarm no. from 0 to 4 for lo
                           and 5 to 9 for hi   */
setalarm(int alarm_no)
{
int x,y;
if (alarm_no>4)
        {y=A_START_Y+42;
        x=A_START_X+(alarm_no-5)*40+2;
        }
else
        {y=A_START_Y+3;
        x=A_START_X+(alarm_no)*40+2;
        }

if(prevalarm[alarm_no]==0)putimage(x,y,alarm,COPY_PUT);
x+=2;
y+=8;
if(alarm_no >4)outtextxy(x,y+10,"HIGH");
```

```
else outtextxy(x,y+10,"LOW");
switch(alarm_no)
        {
        case 0:{outtextxy(x ,y ,"TEMP");break;}
        case 1:{outtextxy(x ,y ,"FLOW");break;}
        case 2:{outtextxy(x ,y ,"DUMY");break;}
        case 3:{outtextxy(x ,y ,"DUMY");break;}
        case 4:{outtextxy(x ,y ,"DUMY");break;}
        case 5:{outtextxy(x ,y ,"TEMP");break;}
        case 6:{outtextxy(x ,y ,"FLOW");break;}
        case 7:{outtextxy(x ,y ,"DUMY");break;}
        case 8:{outtextxy(x ,y ,"DUMY");break;}
        case 9:{outtextxy(x ,y ,"DUMY");break;}
        }

}

/*    Reset alarm routine */
resetalarm(int alarm_no)
{
int x,y;
if (alarm_no>4){y=A_START_Y+42;x=A_START_X+(alarm_no-5)*40+2;}
else{y=A_START_Y+2;x=A_START_X+(alarm_no)*40+2;}
setviewport(x,y,x+36,y+36,0);
clearviewport();
setviewport(0,0,getmaxx(),getmaxy(),0);
}


/*To display temperature digital and analog read out */
display_temp()
{
int triangle[]={305,110,310,115,310,105,305,110};
int i,j,x,del;
char temp_array[10];
scaleset=malloc(imagesize(295,10,310,115));
pointset=malloc(imagesize(305,105,310,115));

line( 300,10,300,110);
for(j=1;j<=11;i++){
        j=i*10;
        line(296,j,304,j);
        }
drawpoly(sizeof(triangle)/(2*sizeof(int)),triangle);
getimage(295,10,310,115,scaleset);
getimage(305,105,310,115,pointset);
putimage(295,10,scaleset,XOR_PUT);
outtextxy(T_START_X+10,T_START_Y-8,"TEMPERATURE");
del=(TEMP_END-TEMP_START)/10;
x=TEMP_END;
```

```
j=-5;
for(i=0;i<11;i++){
        sprintf(temp_array,"%3d",x);
        outtextxy(T_START_X-30,T_START_Y+j,temp_array);
        x-=del;
        j+=10;
        }
for(i=0;i<4;i++)
     {putimage(T_START_X+i*T_WIDTH,T_START_Y,scaleset,COPY_PUT);
     prevpos[i]=T_START_Y+95;
     }
}

/* To display flow in analog and digital */
display_flow()
{
int del,x,y,i,j;
char temp_array[10];
outtextxy(F_START_X,F_START_Y-10,"FLOW");
putimage(F_START_X,F_START_Y,scaleset,COPY_PUT);
prevflow=F_START_Y+95;
del=(FLOW_END-FLOW_START)/10;
x=FLOW_END;
j=-5;
for(i=0;i<11;i++){
        sprintf(temp_array,"%3d",x);
        outtextxy(F_START_X-30,F_START_Y+j,temp_array);
        x-=del;
        j+=10;
        }
}


/* To display power in analog and digital */
display_power()
{
int del,x,y,i,j;
char temp_array[10];
outtextxy(P_START_X,P_START_Y-10,"POWER");
putimage(P_START_X,P_START_Y,scaleset,COPY_PUT);
prevpower=P_START_Y+95;
del=(POWER_END-POWER_START)/10;
x=POWER_END;
j=-5;
for(i=0;i<11;i++){
        sprintf(temp_array,"%3d",x);
        outtextxy(P_START_X-30,P_START_Y+j,temp_array);
        x-=del;
        j+=10;
        }
```

```
}


/* Display data of temerature in analog and digital display */
settemppoint(int no,float temperature)
{
char cnvrtd_data[10];     /* Array for converted float data
                                 in character   */
int j,new_position;       /* New position of pointer     */
j=T_START_X + 10 + no * T_WIDTH;
if(temperature >= TEMP_START && temperature <=TEMP_END)
 {
 new_position=(temperature-TEMP_START)*100/(TEMP_END-TEMP_START);
 putimage(j,prevpos[no],pointset,XOR_PUT);
 prevpos[no]=T_START_Y+95-new_position;
 putimage(j,prevpos[no],pointset,XOR_PUT);
 }
     /* convertion of float data to string of chracter */
sprintf(cnvrtd_data,"%4.1f",temperature);
setviewport(j-15,T_START_Y+105,j+30,T_START_Y+115,0);
clearviewport();
setviewport(0,0,getmaxx(),getmaxy(),0);
outtextxy(j-15,T_START_Y+105,cnvrtd_data);
          /*   Displays converted data  */

}


/*To display flow data in analog and digital form */

setflowpoint(float flow)
{
int j;
char cnvrtd_data[10];

j=F_START_X+10;
if(flow >= FLOW_START && flow <= FLOW_END){
 putimage(j,prevflow,pointset,XOR_PUT);
 prevflow=F_START_Y+95-100*(flow-FLOW_START)/(FLOW_END-
       FLOW_START);
 putimage(j,prevflow,pointset,XOR_PUT);
 }
sprintf(cnvrtd_data,"%5.2f",flow);
setviewport(F_START_X,F_START_Y+105,F_START_X+50,F_START_Y+115,0);
clearviewport();
setviewport(0,0,getmaxx(),getmaxy(),0);
outtextxy(F_START_X,F_START_Y+105,cnvrtd_data);
}
```

```
/*To display power data in analog and digital form */

setpowerpoint(float power)
{
int j;
char cnvrtd_data[10];

j=P_START_X+10;
if(power >= POWER_START && power <= POWER_END){
     putimage(j,prevpower,pointset,XOR_PUT);
     prevpower=P_START_Y+95-100*
               (power-POWER_START)/(POWER_END-POWER_START);
     putimage(j,prevpower,pointset,XOR_PUT);
     }
sprintf(cnvrtd_data,"%5.2f",power);
setviewport(P_START_X,P_START_Y+105,P_START_X+50,P_START_Y+115,0);
clearviewport();
setviewport(0,0,getmaxx(),getmaxy(),0);
outtextxy(P_START_X,P_START_Y+105,cnvrtd_data);
}

/*To display all the displayable section */
display()
{
display_operation_mode();
display_status();
display_temp();
display_flow();
display_power();
display_alarm();
}


/*
     This section was used at the time of development of this
     module.

main()
{
float tc1,flow,power;
int i,j;
init();
display();
getch();

for(j=0;j<1;j++){
        power=0.0;
        for(i=0;i<80;i++)
                {
```

```
                    tc1=1.0/409.50*atod(1);
                    flow=1.0/409.50*atod(5);
                    power+=1.2;
                    settemppoint(1,tc1);
                    setflowpoint(flow);
                    setpowerpoint(power);
                    }
        getch();
        }
getch();
for(i=0;i<1;i++){
        setstatus(i);
        getch();
        setstatus(i);
        getch();
        resetstatus(i);
        getch();
        }

for(i=0;i<10;i++){
        setalarm(i);
        setoperationmode(i);
        getch();
        resetalarm(i);
        getch();
        }

closegraph();
}           end of main     */
```

```
#include <a:\process\kohen.h>
#include <math.h>
#include <stdio.h>
#include <dos.h>

struct time *timen;      /* end time      */
struct time *timep;      /* temporary time       */
struct time *times;      /* start time    */
int DELAY_SEC=15,DELAY_MIN=30;  /* delay in minute and second    */
FILE *fp;        /* file pointer to store time and system output */
int firingangle,flag;    /* fire angle for heater         */
int sec_flag,min_flag;   /* flag for second and minute    */


/*   Checking elasped  prespecified time; if prespecified time
     elasped returns 1 else returns 0.   */

int checktime()
{
gettime(timep);
sec_flag=0;
if((timep->ti_sec+=DELAY_SEC)>59)
     {timep->ti_sec=timep->ti_sec-60;
     sec_flag=1;
     }
flag=0;
do{
     gettime(timen);

     if(sec_flag==1 && timen->ti_sec >= timep->ti_sec
                    && timen->ti_min !=timep->ti_min)  flag=1;
        else if(sec_flag==0 && timen->ti_sec >= timep->ti_sec
                            && timen->ti_min == timep->ti_min)  flag=1;
        else flag=0;
        }while(flag==0);
if(min_flag==1 && timen->ti_min >= times->ti_min
          && timen->ti_hour != times->ti_hour)    return(1);
else if(min_flag==0 && timen->ti_min >= times->ti_min
          && timen->ti_hour == times->ti_hour)    return(1);
else return(0);

}      /*end of checktime      */


/*       This routine get the system output data and stores in file for
         analysis         */
int kohen_calc(int call_no)
{
int i,j,k;
char ch;
```

```
char str[25];
float temp,x;
times=malloc(sizeof(struct time));      /* reserves memory */
timep=malloc(sizeof(struct time));
timen=malloc(sizeof(struct time));
clrscr();
sprintf(str,"\\process\\kohen%1d.dat",call_no);
          /* generates file name        */
fp=fopen(str,"wt");                      /* open file    */
printf(\
   "Do you want to start? If yes press any key.To quit press q.");
        /* checks for further data acqusition permission */
ch=getch();
if(ch=='q'){fclose(fp);return(0);}

gettime(times);
fprintf(fp,"\nStart time %d:  %d:  %d... \n",\
          times->ti_hour,times->ti_min,times->ti_sec);
          /* record the start time of operation */
min_flag=0;
                         /* Setting for delay */
if((times->ti_min+=DELAY_MIN)>59)
     {times->ti_min=times->ti_min-60;
     min_flag=1;
     }
printf("\nStart time %d:  %d:  %d... ",\
               times->ti_hour,times->ti_min,times->ti_sec);

for(;;){
     temp=filter(2);
     gettime(timep);printf("\a");
     fprintf(fp,"Time = %d:%d:%d.%d  ",\
     timep->ti_hour,timep->ti_min,timep->ti_sec,timep->ti_hund);
     fprintf(fp,"\tTemperature %f\n",temp);
     printf("\n%d:  %d:  %d... %f",\
     timen->ti_hour,timen->ti_min,timen->ti_sec,temp);
     if(checktime()==1)
          {fclose(fp);
          return(1);
          }
     }

}       /* end of kohen_calc    */

/*
        This section was used for testing the developd routines
main()
{
kohen_calc();
}       end of main */
```

```
#include <math.h>
#include <stdio.h>
#include <graphics.h>
#include <dos.h>
#include <stdlib.h>
#include <a:\process\filter.h>

extern float channel(int channel_no);


float xreal[301],value[301],c[25],m[100];
float coeff_u[7],       coeff_c[7];
float amplitude,steady,setpoint,st_point;
     /*amplitude of relay,steady value
     of process, set point of process   */

float a,b,T,TD,TAO,KP;  /* constants of process  */
int order=4,N,count;

float K_PID,KC,TAO_I,TAO_D,e[10],A,B,cm[50],;
                /* constants for controller */
float KU,TU;    /* constants for autunner */

int pflag,prev_pflag=0;
float prev_pdata=0,pamplitude;


/*      To initialize graphics mode     */

void initial()
{int graphdriver,graphmode;
graphdriver=DETECT;
initgraph(&graphdriver,&graphmode,"\\bgi\\");

}       /*end of initial      */


/* To plot an array,value in y direction,maximum of 200 points */

void plotting(int axis,float array[],int p,float mult)
{
int i,x,y;
x=0;
y=(axis)-array[0] * mult;               /* mult times enlarged */
moveto(x,y);
for(i=0;i<p;i++){
        y=(axis)-array[i] * mult;       /* mult times enlarged */
        x+=2;                   /* increament in x axis by 5 */
```

```
        lineto(x,y);
        }

}       /* end of plotting routine */


/* initialize array of lenth dimension with d   */

void start(float array[],float d, int length)
{    int i;
for (i=0;i<length;i++)
        array[i]=d;

}       /*end of start  */


/*   Shifts data in array and last data is filled by data i.e put a
     new data at the end of the array and shift the previous data to
     immediate next position */

void putnew( float array[],int length,float data )
{
int i;
for(i=0;i<(length-1);i++)
array[i]=array[i+1];

array[length-1]=data;

}       /* end of putnew routine*/


/*      Sets key data of model of process       */

void model_data(void)
{
T=0.1;
TD=2.4;
TAO=21.5;
KP=16.0;
N=TD/T;

}       /*end of model_data     */


/*      Calculates model parameters     */

void model_param(void)
{
model_data();
a=exp(-T/TAO);
```

```
        lineto(x,y);
        }

}       /* end of plotting routine */


/* initialize array of lenth dimension with d   */

void start(float array[],float d, int length)
{    int i;
for (i=0;i<length;i++)
        array[i]=d;

}       /*end of start  */


/*   Shifts data in array and last data is filled by data i.e put a
     new data at the end of the array and shift the previous data to
     immediate next position */

void putnew( float array[],int length,float data )
{
int i;
for(i=0;i<(length-1);i++)
array[i]=array[i+1];

array[length-1]=data;

}       /* end of putnew routine*/


/*      Sets key data of model of process       */

void model_data(void)
{
T=0.1;
TD=2.4;
TAO=21.5;
KP=16.0;
N=TD/T;

}       /*end of model_data     */


/*      Calculates model parameters     */

void model_param(void)
{
model_data();
a=exp(-T/TAO);
```

```
b=(1-a)*KP;
printf("%f\n ",b);

}       /*end of model_param    */


/*      Model equation calculation and returns model output.    */

float model_calc()
{
float new;
new=b * m[0] + a * c[0];
return(new) ;

}       /*end of model_calc     */


/*      Set  the model output array     */

float model(float input)
{
float cx;
putnew(m,1+N,input);
cx=model_calc();
putnew(c,1,cx);
return(cx);

}       /*end of model  */


/*   This routine simulates the operation of relay i.e. when data
     exceed limit relay output changes  */

float relay(float data)
{
float ret;
int flag,prev_flag=0;
if(data > 0 )
        flag=1;
else    flag =-1;
if(prev_flag != flag)
        ret=amplitude * flag + st_point/KP;
prev_flag = flag;
return(ret);

}       /*end of relay  */


/*      This routine claculates the peak of model output        */
```

```
float peak(float pdata)
{

if((prev_pdata > pdata ) && pdata > 0 ) pflag=1;
else if((prev_pdata < pdata) || pdata <0) pflag=-1;
else pflag=0;
pamplitude=0;
if(pdata< 0) count=0;
if((prev_pflag==1 && pflag==-1)||(prev_pflag==0 && pflag==1) && count=

        {printf(" \n \t\tPEAK %f\n ",pamplitude=prev_pdata);
        count++;
        }
prev_pflag=pflag;
prev_pdata=pdata;

return(pamplitude);

}       /*end of peak   */


/*      Claculates the parameters of autotuner  */

calc(int k){
float xu,uu,d;
int i,peak_count,current,tu;
peak_count=0;
current=0;
for (i=10;i<k;i++)
        {
        xu=st_point-value[i-1];
        uu=relay(xu);
        xreal[i-1]=uu;
        value[i]=model(uu);     /* simulates the process        */
        if((d=peak(st_point-value[i])) != 0){
                peak_count++;
                printf(" KU= %f\n",KU=4*d/3.14/amplitude);
                printf(" TU= %f\n",TU=(i-current)* T);
                current=i;
                }

        }

}       /*end of calc   */


/*      This routine is of controller   */

control_calc(){
float mx,q;
q=0.9;
```

```
mx=q * cm[N] + (1-q) *cm[0] +(1-q) / b * (e[1] - a * e[0]);
        /* equation of controller    .    */
return(mx);

}       /*end of control_calc    */


/*      Parameters of analog PID controller     */

void analog_param(void)
{
printf( " \nKC= %f",KC=KU/2.0) ;
printf( "TAO_I %f",TAO_I =TU/2.0);
printf(" TAO_D %f\n",TAO_D=TU/8.0) ;

}       /*end of analog_param    */


/*      Parameters of digital PID controller    */

void digit_param(void)
{
printf(" \nK_PID %f",K_PID = KC + T * TAO_I);
printf(" A= %f",A= (T * KC + 2 * TAO_D) / (T * K_PID));
printf(" B= %f\n",B = TAO_D / ( T * K_PID));

}       /*end of digit_param     */


/*      Routine of initialisation of parameters of PID controller both

        and digital and also initialize model data array        */

void initparam(void)
{
int i;
analog_param();
digit_param();
for(i=0;i<=N;i++) cm[i]=0;

}       /*end of initparam       */


/*      Update the array of controller equation */

void control( float error)
{
float mx;
putnew(e,2,error);
printf(" MX %f ",mx=control_calc());
```

```
mx=q * cm[N] + (1-q) *cm[0] +(1-q) / b * (e[1] - a * e[0]);
        /* equation of controller        */
return(mx);

}       /*end of control_calc    */


/*      Parameters of analog PID controller     */

void analog_param(void)
{
printf( " \nKC= %f",KC=KU/2.0) ;
printf( "TAO_I %f",TAO_I =TU/2.0);
printf(" TAO_D %f\n",TAO_D=TU/8.0) ;

}       /*end of analog_param    */


/*      Parameters of digital PID controller    */

void digit_param(void)
{
printf(" \nK_PID %f",K_PID = KC + T * TAO_I);
printf(" A= %f",A= (T * KC + 2 * TAO_D) / (T * K_PID));
printf(" B= %f\n",B = TAO_D / ( T * K_PID));

}       /*end of digit_param     */


/*      Routine of initialisation of parameters of PID controller both

        and digital and also initialize model data array        */

void initparam(void)
{
int i;
analog_param();
digit_param();
for(i=0;i<=N;i++) cm[i]=0;

}       /*end of initparam       */


/*      Update the array of controller equation */

void control( float error)
{
float mx;
putnew(e,2,error);
printf(" MX %f ",mx=control_calc());
```

```
putnew(cm,N+1,mx);
return(mx);

}       /*end of control        */


/*      Output for step change in input of process model        */

void cont(int k){
int i;
float rawdata,excite;
rawdata=setpoint;
excite=setpoint/KP;
for (i=0;i<k;i++)
        {
        if(i==50)setpoint=2.0;
        xreal[i]=control(setpoint-rawdata);
        rawdata=model(excite+=xreal[i]);
        value[i]=rawdata;
        }

}       /*end of cont   */


/*       Main routine simulates the process with controller     */

main()
{
int i,j,k=300,r;

float z,d;
float par,uu,xu;
order=1;
par=100.0;
amplitude=0.10;
st_point=1.0;


model_param();
start(c,st_point,1);
start(m,st_point/KP,N+1);
relay(0.1);
for(i=0;i<10;i++){
        value[i]=1.0;
        xreal[i]=1.0;
        }

calc(k);
getch();
```

```
printf("AM I HERE");
initial();
plotting(200,value,k,par);
plotting(200,xreal,k,par/100);
getch();
closegraph();
initparam();
setpoint=1.0;
start(c,setpoint,1);
start(m,setpoint/KP,N+1);
cont(k);
initial();
plotting(200,value,k,par);
plotting(200,xreal,k,par/100);
getch();
closegraph();
}
```

```
/*   process.h includes all the external routines developed */
extern int atod(int channel);
extern int stepper(int step,int direction);
extern int sv_on(int valve_no);
extern int sv_off(int valve_no);
extern int heater(int fire_angle);
extern int stopheat(void);
extern void initmotor();
extern int getsolset();
extern int closemotor();
extern void _DELAY(void);
extern retrieve();
extern store(float *data);
extern fileintl();
extern fileclose();
extern void init(void);
extern void display(void);
/*
        This routines are called by display()
                void display_status(void);
                void display_alarm(void);
                void display_temp(void);
                void display_flow(void);
                void display_power(void);
*/
extern void setstatus(int status);
extern void resetstatus(int status);
extern void setalarm(int alarm_no);
extern void resetalarm(int alarm_no);
extern void settemppoint(int therm_no,float temerature);
extern void setflowpoint(float flow);
extern void setpowerpoint(float power);
extern void initfilter(void);
extern float filter(int channel_no);
extern float filter_avg(int channel_no);
```

```
                                        .MODEL      SMALL
                                        PAGE 59,90
                                        TITLE     ANALOG TO DIGITAL
0000                                .DATA
= 0278                      PORT            equ   0278h             ;port address
=                           CHANNEL_NO  equ       [bp+4]            ;
0000                                .code
0000                        delay     proc
0000  51                            push    cx
0001  B9 0008                                mov   cx,0008h
0004                        loop1:
0004  90                            nop
0005  E2 FD                                 loop    loop1
0007  59                            pop   cx
0008  C3                            ret
0009                        delay     endp
                                        public      _atod
                                    ;
                                    ;
                                    ;calling sequence of the module atod
                                    ;   int atod(int channel_no);
                                    ;channel no maximum 15
                                    ;
                                    ;
0009                        _atod     PROC
0009  55          atod1:        push        bp              ;saving basepointe
                                    r
000A  8B EC                                 mov     bp,sp           ;storing stackpoin
                                    ter to base pointer

000C  BA 027B                       first:      mov       dx,PORT+3             ;clear A/D reg
                                    ister
000F  B8 0000                               mov   ax,0
0012  EF                            out   dx,ax

0013  BA 0278                       second:       mov dx,PORT+0             ;select channel nu
                                    mber for A/D conversion
0016  8A 46 04                              mov   al,CHANNEL_NO
0019  EE                            out   dx,al

001A  B9 0006                               mov   cx,6
001D                        starthi:
001D  BA 027C                               mov   dx,PORT+4             ;start A/D con
                                    version for  high 6 bit
0020  E8 0000 R                            call    delay           ;and loop 7 times
                                    exactly
0023  EC                            in  al,dx
0024  E2 F7                                 loop    starthi

0026  B9 0006                               mov   cx,6
0029                        startlo:
0029  BA 027D                               mov   dx,PORT+5             ;start A/D con
                                    version for  low 6 bit
002C  E8 0000 R                            call    delay           ;and loop 7 times
                                    exactly
```

```
002F  EC                    in  al,dx
0030  E2 F7                             loop    startlo

0032  BA 027A                           mov   dx,PORT+2         ;reading highe
                            r nibble to ah
0035  E8 0000 R                         call    delay
0038  EC                    in  al,dx
0039  8A E0                            mov   ah,al
003B  80 E4 0F                          and   ah,00001111b

003E  BA 0279                           mov   dx,PORT+1         ;reading lower
                             byte  to al
0041  E8 0000 R                         call    delay
0044  EC                    in  al,dx

0045  5D                   pop   bp
0046  C3                   ret
0047                 _atod      endp
0047                            end
```

Segments and Groups:

Name	Length	Align	Combine	Class
DGROUP	GROUP			
_DATA	0000	WORD	PUBLIC	'DATA'
_TEXT	0047	WORD	PUBLIC	'CODE'

Symbols:

Name	Type	Value	Attr	
ATOD1	L NEAR	0009	_TEXT	
CHANNEL_NO	TEXT	[bp+4]		
DELAY	N PROC	0000	_TEXT	Length = 0009
FIRST	L NEAR	000C	_TEXT	
LOOP1	L NEAR	0004	_TEXT	
PORT	NUMBER	0278		
SECOND	L NEAR	0013	_TEXT	
STARTHI	L NEAR	001D	_TEXT	
STARTLO	L NEAR	0029	_TEXT	
@CODE	TEXT	_TEXT		
@CODESIZE	TEXT	0		
@DATASIZE	TEXT	0		
@FILENAME	TEXT	atod		
_ATOD	N PROC	0009	_TEXT	Global Length = 003E

64 Source Lines
64 Total Lines
23 Symbols

50990 + 425346 Bytes symbol space free

0 Warning Errors
0 Severe Errors

```
                        TITLE   HEATER CONTROL
                        PAGE 59,90
                        EXTRN   svc:byte  ;storage svc for data
                    of solenoid valve
                        extrn   strob_st:byte;for strobe status
                        .MODEL    small
0000                    .DATA
= 0378                  DATA_PORT equ   0378h           ;port address  for pri
                    nter data port
= 037A                  ADDRES_PORT   equ 037ah         ;port address for prin
                    ter hand shake out
= 0379                  STATUS_PORT   equ 0379h         ;port address for prin
                    ter hand shake in
=                       FIRE_ANGLE  equ     [bp+4]    ;
                        ;
                        ;  out put to adress hex 37A
                        ; |-----|-----|---|-----|-----|-----|-----|-------|
                        ; |7    | 6   | 5 |  4  | 3   | 2   | 1   | 0     |
                        ; |-----|-----|---|-----|-----|-----|-----|-------|
                        ; |  X  |  X  | X | intr| not | A1  | not | not   |
                        ; |     |     |   |enable| A2 |     | A0  | strobe|
                        ; |-----|-----|---|-----|-----|-----|-----|-------|
                        ; |  pin no.  |         | 17  | 16  | 14  | 1     |
                        ; |-----|-----|---|-----|-----|-----|-----|-------|
= 0000                  ADDRES_HC equ   000h          ;mask for heater controlle
                    r register enable
= 0002                  CLEAR     equ     002h      ;mask for diable all regis
                    ter
= 0001                  STROBE_HC equ   001h          ;mask for strobe heater co
                    ntroller register
= 0006                  ADDRES_SVC  equ     006h      ;mask for solenoid valve c
                    ontroller enable
= 0007                  STROBE_SVC  equ     007h      ;mask for strobe SVC
= 0008                  START_POWR  equ     00001000b

                        ;this macro used for strobe the SVC,HC or STEPPER
                        @strobe         MACRO   x
                                push    ax
                                cli
                                mov   dx,ADDRES_PORT          ;out the r
                    egister address
                                mov   al,ADDRES_&x
                                or  al,strob_st
                                out   dx,al
                                call    delay
                                mov   al,STROBE_&x      ;out strobe line h
                    igh
                                or  al,strob_st
                                out   dx,al
                                call    delay
                                mov   al,ADDRES_&x      ;out low strobe li
                    ne
                                or  al,strob_st
                                out   dx,al
                                call    delay
```

```
                                        mov  al,CLEAR                   ;reset all
                                address of register
                                        or  al,strob_st
                                        out   dx,al
                                        call    delay
                                        pop   ax
                                        endm

0000                                    .CODE

0000                            delay       proc
0000  51                                    push    cx
0001  B9 00FF                              mov   cx,0ffh
0004                            a:
0004  E2 FE                                 loop    a
0006  59                                    pop   cx
0007  C3                                   ret
0008                            delay       endp
                                ;
                                ;
                                ; calling sequence of the heater routine is
                                ;   void heater(int fire_angle)
                                ;fire_angle must lie between 10 and 240
                                ;this due to the unstable frequency generation
                                ;
                                        PUBLIC      _heater
0008                            _heater       PROC
0008  FA                                    cli
0009  55                                    push    bp
000A  8B EC                                 mov   bp,sp
000C  8A 46 04                              mov   al,FIRE_ANGLE   ;firing angle from m
                                ain
000F  BA 0378                               mov   dx,DATA_PORT
0012  EE                                    out   dx,al
                                            @strobe       HC                ;storing d
                                ata in heater control_card
0013  50                      1            push    ax
0014  FA                      1            cli
0015  BA 037A                  1            mov   dx,ADDRES_PORT            ;out the r
                                egister address
0018  B0 00                    1            mov   al,ADDRES_HC
001A  0A 06 0000 E             1            or  al,strob_st
001E  EE                       1            out   dx,al
001F  E8 0000 R                1            call    delay
0022  B0 01                    1            mov   al,STROBE_HC      ;out strobe line h
                                igh
0024  0A 06 0000 E             1            or  al,strob_st
0028  EE                       1            out   dx,al
0029  E8 0000 R                1            call    delay
002C  B0 00                    1            mov   al,ADDRES_HC      ;out low strobe li
                                ne
002E  0A 06 0000 E             1            or  al,strob_st
0032  EE                       1            out   dx,al
0033  E8 0000 R                1            call    delay
0036  B0 02                    1            mov   al,CLEAR                  ;reset all
```

```
                        address of register
0038  0A 06 0000 E  1              or  al,strob_st
003C  EE           1              out   dx,al
003D  E8 0000 R    1              call    delay
0040  58           1              pop   ax

0041  B9 FFFF                     mov   cx,0ffffh          ;delay for int
                        erfacing with uc & HC
0044                    stay_while:
0044  E2 FE                       loop    stay_while

0046  A0 0000 E                   mov   al,svc             ;retrieving data o
                        f svc
0049  0C 08                       or  al,START_POWR ;and set bit 5
004B  24 EF                       and   al,0efh
004D  BA 0378                     mov   dx,DATA_PORT  ;
0050  EE                         out   dx,al              ;storing this in s
                        vc reg. of
                                  @strobe       SVC   ;storage card

0051  50           1              push    ax
0052  FA           1              cli
0053  BA 037A      1              mov   dx,ADDRES_PORT           ;out the r
                        egister address
0056  B0 06         1              mov   al,ADDRES_SVC
0058  0A 06 0000 E  1              or  al,strob_st
005C  EE           1              out   dx,al
005D  E8 0000 R    1              call    delay
0060  B0 07         1              mov   al,STROBE_SVC      ;out strobe line
                        high
0062  0A 06 0000 E  1              or  al,strob_st
0066  EE           1              out   dx,al
0067  E8 0000 R    1              call    delay
006A  B0 06         1              mov   al,ADDRES_SVC      ;out low strobe l
                        ine
006C  0A 06 0000 E  1              or  al,strob_st
0070  EE           1              out   dx,al
0071  E8 0000 R    1              call    delay
0074  B0 02         1              mov   al,CLEAR                 ;reset all
                         address of register
0076  0A 06 0000 E  1              or  al,strob_st
007A  EE           1              out   dx,al
007B  E8 0000 R    1              call    delay
007E  58           1              pop   ax

007F  B9 00FF                     mov   cx,000ffh          ;delay for int
                        erfacing with uc & HC
0082                    while1:
0082  E2 FE                       loop    while1
0084  A0 0000 E                   mov   al,svc

0087  24 F7                       and   al,NOT START_POWR       ;reset bit 5

0089  A2 0000 E                   mov   svc,al             ;and saving this i
                        n svc
008C  24 EF                       and   al,0efh
```

```
008E  BA 0378                           mov   dx,DATA_PORT
0091  EE                               out   dx,al             ;storing this in s
                        vc reg. of
                                        @strobe       SVC   ;storage card
0092  50                    1           push    ax
0093  FA                    1           cli
0094  BA 037A               1           mov   dx,ADDRES_PORT            ;out the r
                        egister address
0097  B0 06                1           mov   al,ADDRES_SVC
0099  0A 06 0000 E          1           or  al,strob_st
009D  EE                    1           out   dx,al
009E  E8 0000 R             1           call    delay
00A1  B0 07                 1           mov   al,STROBE_SVC       ;out strobe line
                        high
00A3  0A 06 0000 E          1           or  al,strob_st
00A7  EE                    1           out   dx,al
00A8  E8 0000 R             1           call    delay
00AB  B0 06                 1           mov   al,ADDRES_SVC       ;out low strobe l
                        ine
00AD  0A 06 0000 E          1           or  al,strob_st
00B1  EE                    1           out   dx,al
00B2  E8 0000 R             1           call    delay
00B5  B0 02                 1           mov   al,CLEAR                  ;reset all
                         address of register
00B7  0A 06 0000 E          1           or  al,strob_st
00BB  EE                    1           out   dx,al
00BC  E8 0000 R             1           call    delay
00BF  5B                    1           pop   ax

00C0  5D                                pop   bp
00C1  C3                                ret
00C2                                _heater       endp                      ;heater procedure
                                    end

                                    ;
                                    ; To stop the heater, stopheat is prepared
                                    ; the calling sequence of the routine is
                                    ;   void stop_heat(void)
                                    ;
                                            public      _stopheat
00C2                                _stopheat proc
00C2  55                                    push    bp
00C3  8B EC                                 mov   bp,sp
00C5  A0 0000 E                             mov   al,svc
00C8  0C 08                                 or  al,START_POWR
00CA  A2 0000 E                             mov   svc,al
00CD  BA 0378                               mov   dx,DATA_PORT
00D0  EE                                    out   dx,al
                                            @strobe       SVC
00D1  50                    1               push    ax
00D2  FA                    1               cli
00D3  BA 037A               1               mov   dx,ADDRES_PORT            ;out the r
                         egister address
00D6  B0 06                 1               mov   al,ADDRES_SVC
00D8  0A 06 0000 E          1               or  al,strob_st
```

```
00DC  EE                1               out   dx,al
00DD  E8 0000 R         1               call    delay
00E0  B0 07             1               mov   al,STROBE_SVC     ;out strobe line
                                        high
00E2  0A 06 0000 E      1               or  al,strob_st
00E6  EE                1               out   dx,al
00E7  E8 0000 R         1               call    delay
00EA  B0 06             1               mov   al,ADDRES_SVC     ;out low strobe l
                                        ine
00EC  0A 06 0000 E      1               or  al,strob_st
00F0  EE                1               out   dx,al
00F1  E8 0000 R         1               call    delay
00F4  B0 02             1               mov   al,CLEAR                  ;reset all
                                         address of register
00F6  0A 06 0000 E      1               or  al,strob_st
00FA  EE                1               out   dx,al
00FB  E8 0000 R         1               call    delay
00FE  58                1               pop   ax
00FF  5D                                pop   bp
0100  C3                                ret
0101                                    _stopheat endp                  ;stop_heater proce
                                        dure end.
                                        ;
                                        ;
                                        ;This procedure returns the condition of the heater
                                        ;i.e. either it is on or not
                                        ;the calling sequence is
                                        ; int getheater(void);
                                        ;
                                        ;
                                                public      _getheater
0101                                    _getheater  proc
0101  A0 0000 E                                 mov   al,svc
0104  24 08                                     and   al,START_POWR
0106  75 05                                     jnz   heater_on
0108  2B C0                                     sub   ax,ax
010A  EB 04 90                                  jmp   return
010D  B8 0001                           heater_on:  mov       ax,1
0110  C3                                return:     ret
0111                                    _getheater  endp                    ;_getheater proced
                                        ure end
0111                                            end
```

Macros:

Name	Lines
@STROBE	20

Segments and Groups:

Name	Length	Align	Combine	Class
DGROUP	GROUP			
_DATA	0000	WORD	PUBLIC	'DATA'
_TEXT	0111	WORD	PUBLIC	'CODE'

Symbols:

Name	Type	Value	Attr	
A	L NEAR	0004	_TEXT	
ADDRES_HC	NUMBER	0000		
ADDRES_PORT	NUMBER	037A		
ADDRES_SVC	NUMBER	0006		
CLEAR	NUMBER	0002		
DATA_PORT	NUMBER	0378		
DELAY	N PROC	0000	_TEXT	Length = 0008
FIRE_ANGLE	TEXT	[bp+4]		
HEATER_ON	L NEAR	010D	_TEXT	
RETURN	L NEAR	0110	_TEXT	
START_POWR	NUMBER	0008		
STATUS_PORT	NUMBER	0379		
STAY_WHILE	L NEAR	0044	_TEXT	
STROBE_HC	NUMBER	0001		
STROBE_SVC	NUMBER	0007		
STROB_ST	V BYTE	0000	External	
SVC	V BYTE	0000	External	
WHILE1	L NEAR	0082	_TEXT	
@CODE	TEXT	_TEXT		
@CODESIZE	TEXT	0		
@DATASIZE	TEXT	0		
@FILENAME	TEXT	heater		
_GETHEATER	N PROC	0101	_TEXT	Global Length = 0010
_HEATER	N PROC	0008	_TEXT	Global Length = 00BA
_STOPHEAT	N PROC	00C2	_TEXT	Global Length = 003F

142 Source Lines
222 Total Lines
35 Symbols

50974 + 408978 Bytes symbol space free

0 Warning Errors
0 Severe Errors

```
                        TITLE   SOLENOIED VALVE CONTR
OL
                        PAGE 59,90
                        extrn   _getch:proc
                        public  svc
                        extrn   strob_st:byte
                        .MODEL  small
                        .DATA
0000                    
0000 ??         svc     db      (?)
= 0080          SV1     equ     080h
= 0040          SV2     equ     040h
= 0020          SV3     equ     020h
= 0378          DATA_PORT equ   0378h
= 037A          ADRS_PORT equ   037ah
= 0379          STATS_PORT equ  0379h
=               VALVE_NO  equ   [bp+4]
= 0006          ADDRES_SVC equ  006h
= 0007          STROBE_SVC equ  007h
= 0002          CLEAR   equ     002h

0000                    .CODE
                @strobe macro
                        cli
                        push    dx
                        mov     dx,ADRS_PORT    ;preparing strobe
on strobe line
                        mov     al,ADDRES_SVC
                        or      al,strob_st
                        out     dx,al
                        mov     al,STROBE_SVC
                        or      al,strob_st
                        out     dx,al
                        mov     al,ADDRES_SVC
                        or      al,strob_st
                        out     dx,al
                        mov     al,CLEAR
                        or      al,strob_st
                        out     dx,al
                        pop     dx
                        endm

0000            delay   proc
0000 FA                 cli
                ;       mov     dx,10
0001            aw1:
0001 B9 0FFF            mov     cx,00fffh               ;delaying
a few moment to stable
0004            while:                                  ;contact b
ounce of relay
0004 E2 FE              loop    while
0006 4A                 dec     dx
0007 75 F8              jnz     aw
0009            delay   endp

                ;
```

```
                                ;
                                ;
                                ;
0009                            filter      proc
0009  FA                                        cli
000A  BA 0379                                   mov   dx,STATS_PORT         ;reading the s
                                tatus of the
000D  EC                                        in  al,dx                   ;relay
000E  80 FC 01                                  cmp   ah,001h
0011  75 0C                                     jne   other1
0013  25 0008                                   and   ax,00008h
0016  D0 E8                                     shr   al,1
0018  D0 E8                                     shr   al,1
001A  D0 E8                                     shr   al,1
001C  EB 26 90                                  jmp   return
001F  80 FC 02                  other1:         cmp ah,02h
0022  75 0C                                     jne   other2
0024  25 0010                                   and   ax,00010h
0027  D0 E8                                     shr   al,1
0029  D0 E8                                     shr   al,1
002B  D0 E8                                     shr   al,1
002D  EB 15 90                                  jmp   return
0030  80 FC 03                  other2:         cmp ah,03h
0033  75 0C                                     jne   other3
0035  25 0020                                   and   ax,00020h
0038  D0 E8                                     shr   al,1
003A  D0 E8                                     shr   al,1
003C  D0 E8                                     shr   al,1
003E  EB 04 90                                  jmp   return
0041                            other3:
0041  25 00FF                                   and   ax,000ffh
0044                            return:
0044  C3                                        ret
0045                            filter      endp

                                                PUBLIC      _getsolset
0045                            _getsolset  proc
0045  B4 01                                    mov   ah,01
0047  E8 0009 R                                 call    filter
004A  8B D8                                     mov   bx,ax
004C  B4 02                                     mov   ah,02
004E  E8 0009 R                                 call    filter
0051  0B D8                                     or  bx,ax
0053  B4 03                                     mov   ah,03
0055  E8 0009 R                                 call    filter
0058  0B D8                                     or    bx,ax
005A  8B C3                                     mov   ax,bx
005C  C3                                        ret
005D                            _getsolset  endp
                                ;
                                ;
                                ;To set a solenoid valve on this routine is used
                                ;The calling sequence is
                                ;   int       sv_on(int valve_no);
```

```
                                PUBLIC      _sv_on
                                PROC
005D                    _sv_on
                        ;   cli
005D  55                        push    bp
005E  8B EC                     mov   bp,sp
0060  8B 46 04                  mov   ax,VALVE_NO          ;retrieve valv
                        e no. from main
0063  3D 0001                   cmp   ax,01h               ;if valve
                        no. 1 on valve 1
0066  74 10                     je  valve1
0068  3D 0002                   cmp   ax,02h               ;if valve
                        no. 2 on valve 2
006B  74 13                     je  valve2
006D  3D 0003                   cmp   ax,03h               ;if valve
                        no. 3 on valve 3
0070  74 16                     je  valve3
0072  B8 FFFF                   mov   ax,0ffffh            ;if none o
                        f this return 0
0075  5D                        pop   bp
0076  FB                       sti
0077  C3                       ret
0078  A0 0000 R         valve1:      mov al,svc            ;retrieve
                        svc and set bit 7
007B  0C 80                     or  al,SV1
007D  EB 0E 90                  jmp   go_on
0080  A0 0000 R          valve2:      mov al,svc           ;retrieve
                        svc and set bit 6
0083  0C 40                     or  al,SV2
0085  EB 06 90                  jmp   go_on
0088  A0 0000 R          valve3:      mov al,svc           ;retrieve
                        svc and set bit 5
008B  0C 20                     or  al,SV3
008D  24 EF               go_on:     and      al,0efh
008F  A2 0000 R                 mov   svc,al               ;storing t
                        his at svc
0092  BA 0378                   mov   dx,DATA_PORT         ;send the al on da
                        ta bus of
0095  EE                        out   dx,al                ;storage c
                        ard
                                @strobe                    ;store it
                        at svc reg. of
1                               cli
1                               push    dx
1                               mov   dx,ADRS_PORT         ;preparing strobe
                         on strobe line
1                               mov   al,ADDRES_SVC
1                               or  al,strob_st
1                               out   dx,al
1                               mov     al,STROBE_SVC
1                               or  al,strob_st
1                               out   dx,al
1                               mov   al,ADDRES_SVC
1                               or  al,strob_st
1                               out   dx,al
1                               mov   al,CLEAR
0096  FA
0097  52
0098  BA 037A
009B  B0 06
009D  0A 06 0000 E
00A1  EE
00A2  B0 07
00A4  0A 06 0000 E
00A8  EE
00A9  B0 06
00AB  0A 06 0000 E
00AF  EE
00B0  B0 02
```

```
00B2  0A 06 0000 E      |              or  al,strob_st
00B6  EE                1              out   dx,al
00B7  5A                1              pop   dx
                                                                    ;s
torage card
00B8  BA 0001                          mov   dx,1
00BB  E8 0000 R                        call      delay
00BE  8A 66 04                         mov   ah,VALVE_NO
00C1  E8 0009 R                        call    filter               ;to set va
lve flag
00C4  5D                               pop   bp
00C5  FB                               sti
00C6  C3                               ret
00C7                    _sv_on      endp

                        ;
                        ; The calling sequence of the routine sv_off is
                        ;   int sv_off(int valve_no);
                        ;
                        ;
                                    PUBLIC      _sv_off
00C7                    _sv_off   PROC
00C7  FA                               cli
00C8  55                               push    bp
00C9  8B EC                            mov   bp,sp
00CB  8B 46 04                         mov   ax,VALVE_NO   ;retriev valve no.
00CE  3D 0001                          cmp   ax,01h            ;is it valve no. 1
                        then go to valve01
00D1  74 10                            je  valve01
00D3  3D 0002                          cmp   ax,02h            ;is it valve no. 2
                        then go to valve02
00D6  74 13                            je  valve02
00D8  3D 0003                          cmp   ax,03h            ;is it valve no. 3
                        then go to valve03
00DB  74 16                            je  valve03
00DD  B8 FFFF                          mov   ax,0ffffh           ;none then retu
                        rn
00E0  5D                               pop   bp
00E1  FB                               sti
00E2  C3                               ret
00E3                    valve01:
00E3  A0 0000 R                        mov   al,svc
00E6  24 7F                            and   al, NOT SV1
00E8  EB 0E 90                         jmp   go_off
00EB                    valve02:
00EB  A0 0000 R                        mov   al,svc
00EE  24 BF                            and   al, not SV2
00F0  EB 06 90                         jmp   go_off
00F3                    valve03:
00F3  A0 0000 R                        mov   al,svc
00F6  24 DF                            and   al,not SV3
00F8  24 EF             go_off:       and al,0efh
00FA  A2 0000 R                        mov   svc,al                ;storing d
                        ata at svc
00FD  BA 0378                          mov   dx,DATA_PORT      ;transfering data
```

```
                                to svc reg.
0100  EE                                out   dx,al                     ;register
                                card
                                @strobe
0101  FA                      1         cli
0102  52                      1         push    dx
0103  BA 037A                1         mov   dx,ADRS_PORT      ;preparing strobe
                                on strobe line
0106  B0 06                   1         mov   al,ADDRES_SVC
0108  0A 06 0000 E            1         or  al,strob_st
010C  EE                      1         out   dx,al
010D  B0 07                   1         mov     al,STROBE_SVC
010F  0A 06 0000 E            1         or  al,strob_st
0113  EE                      1         out   dx,al
0114  B0 06                   1         mov   al,ADDRES_SVC
0116  0A 06 0000 E            1         or  al,strob_st
011A  EE                      1         out   dx,al
011B  B0 02                   1         mov   al,CLEAR
011D  0A 06 0000 E            1         or  al,strob_st
0121  EE                      1         out   dx,al
0122  5A                      1         pop   dx
0123  BA 0001                            mov   dx,1
0126  E8 0000 R                         call    delay
0129  8A 66 04                          mov   ah,VALVE_NO
012C  E8 0009 R                         call    filter
012F  5D                                pop   bp
0130  FB                                sti
0131  C3                                ret
0132                            _sv_off endp
0132                                    end
```

Macros:

Name	Lines
@STROBE	16

Segments and Groups:

Name	Length	Align	Combine	Class
DGROUP	GROUP			
_DATA	0001	WORD	PUBLIC	'DATA'
_TEXT	0132	WORD	PUBLIC	'CODE'

Symbols:

Name	Type	Value	Attr	
ADDRES_SVC	NUMBER	0006		
ADRS_PORT	NUMBER	037A		
AW	L NEAR	0001	_TEXT	
CLEAR	NUMBER	0002		
DATA_PORT	NUMBER	0378		
DELAY	N PROC	0000	_TEXT	Length = 0009
FILTER	N PROC	0009	_TEXT	Length = 003C
GO_OFF	L NEAR	00F8	_TEXT	
GO_ON	L NEAR	008D	_TEXT	
OTHER1	L NEAR	001F	_TEXT	
OTHER2	L NEAR	0030	_TEXT	
OTHER3	L NEAR	0041	_TEXT	
RETURN	L NEAR	0044	_TEXT	
STATS_PORT	NUMBER	0379		
STROBE_SVC	NUMBER	0007		
STROB_ST	V BYTE	0000		External
SV1	NUMBER	0080		
SV2	NUMBER	0040		
SV3	NUMBER	0020		
SVC	L BYTE	0000	_DATA	Global
VALVE01	L NEAR	00E3	_TEXT	
VALVE02	L NEAR	00EB	_TEXT	
VALVE03	L NEAR	00F3	_TEXT	
VALVE1	L NEAR	0078	_TEXT	
VALVE2	L NEAR	0080	_TEXT	
VALVE3	L NEAR	0088	_TEXT	
VALVE_NO	TEXT	[bp+4]		
WHILE	L NEAR	0004	_TEXT	

@CODE	TEXT _TEXT			
@CODESIZE	TEXT 0			
@DATASIZE	TEXT 0			
@FILENAME	TEXT solenoid			
_GETCH	L NEAR 0000		External	
_GETSOLSET	N PROC 0045	_TEXT	Global	Length = 0018
_SV_OFF	N PROC 00C7	_TEXT	Global	Length = 006B
_SV_ON	N PROC 005D	_TEXT	Global	Length = 006A

190 Source Lines
222 Total Lines
46 Symbols

50958 + 408994 Bytes symbol space free

0 Warning Errors
0 Severe Errors