PID Controller Of Servo System In Real Time Linux Environment

A Thesis

Submitted to the Department of Computer Science and Engineering

of

BRAC University

by

Marzia Alam (ID: 05310034)

C.M. Iftekhar Hussain(ID: 06310049)

MD. Moniruzzaman(ID: 06310056)

Samina Chowdhury (ID: 06320025)

In Partial Fulfillment of the

Requirements for the Degree

of

Electronics And Communication Engineering

April 2008

BRAC University, Dhaka, Bangladesh

DECLARATION

We hereby declare that this thesis is based on the results found by ourselves . Materials of work found by other researcher are mentioned by reference. This thesis, neither in whole nor in part, has been previously submitted for any degree.

Marzia Alam

Signature of
Supervisor

Signature of
Authors

ACKNOWLEDGMENTS

We would like to take this opportunity to express our gratitude to the many people who have provided help and encouragement over the time leading up to and during the progress of this work presented in this theses .

First and foremost, We would like to express our most sincere thanks to our supervisor Dr. Azad Akm. We are indebted to him not only for the support and guidance that he generously offered us throughout this research, but also for the invaluable impact that he left on our approach to control engineering and different technical problems.

We are grateful to every faculties who offered us continuous support while studying and preparing for the dissertation. We also wish to thank all the LTOs for their advice and support given to us over the semester. Their encouraging attitude and their views on systems and PID control research have been of great influence. This research has been performed in the frame of the program, Electronics & Telecommunications Engineering under the department of Computer Science & Engineering. I appreciate not only the financial support, but also the attempts by the department to improve the attitudes towards gradual education and research.

BRAC University, 2008

ABSTRACT

This work presents an implementation of a digital PID (Proportional+ Integral +Derivative) controller algorithm in Real-Time Linux environment. PID controllers are well known and have a wide range of applications in automatic control of systems, such as servomotors and temperature control etc. PID values are dependent on the sampling rate at which system output is read and the new value for system input generated. In order to achieve the desired performance the system output (axis position) is feed-back to the PID controller which generates the system input (motor voltage), in a closed loop. The Real-Time system is the accurate system which gives lowest latency. Through this real-time the temperature control unit is controlled. The PID controller computes the error of the temperature control circuit between the desired system output (reference) and the actual system output. Then, the system input, which is the PID controller output, is generated as weighted sum of the error, its integral and derivative . The weighting coefficients are known as controller gains and referred as K_p, K_i, and K_d. The values of the controller gains determine the performance of the closed loop system and even its stability. The data acquisition card AX5411 receives the data from the computer to control the temperature controller and retrieve the data and send it to computer again from the output as a feed back element simultaneously.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER I

1. Introduction

Most common control of physical systems with a digital computer are aircraft autopilots, mass-transit vehicles, oil refineries, paper-making machines, and countless electromechanical servomechanisms. PID values are dependent on the sampling rate at which system output is read and the new value for system input generated. Therefore it is highly desirable to have an accurate sampling rate. For typical mechanical systems, the adequate sampling rate is in the range of milliseconds. Thus, well implemented PID controllers are usually hard real time systems. Further more many advanced digital control applications are being stimulated by microprocessor technology including control of various aspects of automobiles and household appliances. ncreased flexibility of the control programs and the decision-making or logic capability of digital systems are among the advantages of digital logic for control, which can be combined with the dynamic control function to meet other system requirements. The digital control, which we would like to demonstrate, is a closed-loop (feedback) system. PC gives the necessary command to the hardware and also receives the feedback for accurate measurement.

In chapter II the algorithm of the PID controller is described where, it is shown how different parameters of proportional integral and derivative influence the controller in terms of tuning. PID controller response curve will confirm the different action provided by the theoretical equation given by the PID algorithm. In chapter III leads to the description of the real-time operating system RT-Linux. In this chapter the features and application of real-time and non real-time system is taken into the account. The implemented temperature controller in Linux environment is described chapter IV. The elaborate explanation of the features and temperature controller itself along with the lab result is given on that chapter.

CHAPTER II

PID CONTROLLER

2.1 PID Algorithm

The PID controller calculation (algorithm) involves three separate parameters; the Proportional, the Integral and Derivative values. The Proportional value determines the reaction to the current error, the Integral determines the reaction based on the sum of recent errors and the Derivative determines the reaction to the rate at which the error has been changing. The weighted sum of these three actions is used to adjust the process via a control element such as the position of a control valve or the power supply of a heating element.

By "tuning" the three constants in the PID controller algorithm the PID can provide control action designed for specific process requirements. The response of the controller can be described in terms of the responsiveness of the controller to an error, the degree to which the controller overshoots the set point and the degree of system oscillation. Note that the use of the PID algorithm for control does not guarantee optimal control of the system or system stability.

Some applications may require using only one or two modes to provide the appropriate system control. This is achieved by setting the gain of undesired control outputs to zero. A PID controller will be called a PI, PD, P or I controller in the absence of the respective control actions. PI controllers are particularly common, since derivative action is very sensitive to measurement noise, and the absence of an integral value may prevent the system from reaching its target value due to the control action.

2.2 PROPORTINAL CONTROLLER

The first element of PID control to be developed is Proportional control. The action may be either direct or reverse. In a direct acting control loop an increase in the process measurement causes an increase in the output to the final control element.

The proportional only equation is:

output = gain x error + bias

The bias is sometimes known as the manual reset. Some control systems (such as Foxboro products, use proportional band rather than gain. The proportional band and the gain are related by:

$$\text{Gain} = \frac{100\%}{\text{Proportional Band}}$$

$$\text{Proportional Band} = \frac{100\%}{\text{Gain}}$$

Gain is the ratio of the change in the output to the change in the input.

$$\text{Gain} = \frac{\text{Output change}}{\text{Input change}}$$

A discrete implementation of proportional control is identical to continuous. If we consider the continuous signal of p proportional control $u(t) = K_p\, e(t)$.Where Kp is the proportional gain and e(t) is error of the input signal. In frequency domain it can be written as

$$D(s) = K_p\, E(s) \qquad (1)$$

E(s) is considered as error signal.

The continuous signal can be converted discrete by performing *Z* transform. And in this case the signal *s* which is indeed a error of the controller will be *s= 1-z-1/T*

$$s=> \frac{z-1}{Tz} \tag{2}$$

If it is considered as a discrete.the expression will be as follows:

$$D(s)=K_p \tag{3}$$

Although the proportional controller involves two parameters input signal or error and gain. But mostly the gain plays the roll on tuning. If gain is increased the raise time of the out put signal regarding the set point increases. In case of proportional controller the gain is mostly considered as the prime parameter. Because this gain control is involved to reduce error in the system. So the transfer function as well as the discrete equation of proportional controller is as follows:

$$D(z)= K_p \tag{4}$$

A high proportional gain results in a large change in the output for a given change in the error. If the proportional gain is too high, the system can become unstable. In contrast, a small gain results in a small output response to a large input error, and a less responsive (or sensitive) controller. If the proportional gain is too low, the control action may be too small when responding to system disturbances.In the absence of disturbances, pure proportional control will not settle at its target value, but will retain a steady state error that is a function of the proportional gain and the process gain. The proprotinal gain effect of is illastrated in fig 2.1. Where it is clear that the more gain K_P gives less rise time to the system. And it also reduce the overshoot.

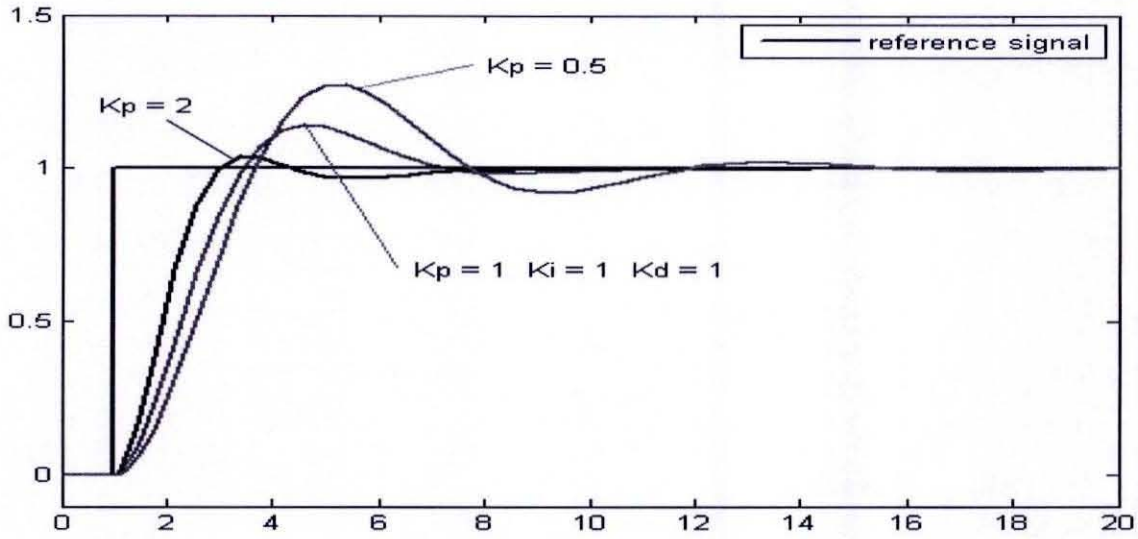

Figure 2.1: Change of response for varying Kp

2.3 PROPORTINAL INTEGRAL CONTROLLER

Input to the controller is the error from set point. If the error remains non zero for long time then the integrator will integrate all the errors and it may produce a very large value which is call integral wind up. To avoid integral wind up a limit on the value can be placed.

In mathematical term we can express the integral controller as follows;

$$u(t) = \frac{K_p}{T_i \int e(t)dt} \quad (5)$$

If we convert above equation in frequency domain. That is:

$$D(s) = \frac{K_p}{T_i s} \quad (6)$$

Where K_p is the proportional gain, T_i is integral time and s is signal.

This signal is to be gone through a discrete process and in this case if we consider u(t) is the present signal than u(t-1)is the previous one. And all the previous signal should be sum up, which gives an expression is $u(k)=u(k-1) + (K_pT/T_i)\ e(k)$.

Equation *(4)* can be obtained as:

$$D(z)= \frac{K_pTz}{T_i(z-1)} \qquad (7)$$

The above equation will give a discrete value of PI control. Consider the block diagram of figure 2.2.

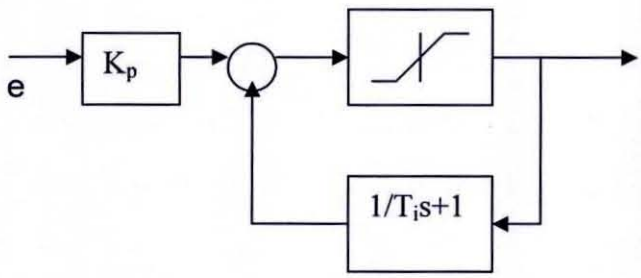

Figure 2.2: A block Diagram of a PI controller

If the integral control is connected with the proportional control as a feed back as the figure 2.2 shows. Than the transfer function of the PI control should be $1/T_i(s+1)$.

Because as in many design the compensation is a sum of proportional and integral control (PI control) which gives us: $D(s)= K_p(1+ 1/T_is)$.

$$D(s)= \frac{K_p(T_i s+1)}{T_i s} \quad (8)$$

And from the block diagram If we reduce into the system we will find $T_i sK_p/1+T_i s$

Here Proportional gain is not considered as feed back element. In integral control it is expressed as $H(s)=1/Ts+1$ as feed back. And the discrete equivalent of the feedback is $H(z)= 1/T_i(z/z-e-T/T_i)$

The integral term (when added to the proportional term) accelerates the movement of the process towards setpoint and eliminates the residual steady-state error that occurs with a proportional only controller. However, since the integral term is responding to accumulated errors from the past, it can cause the present value to overshoot the setpoint value (cross over the setpoint and then create a deviation in the other direction). Just as for continuous system, the primary reason for integral control is to reduce or eliminate steady-state errors, but this typically occurs at the cost of reduced stability and increased overshoot of the system.The integral controller effect is illustrated in fig 2.3.The integral gain 2 will makes the system more unstable where K_I which is infect K_P/T_I is equal to 1 gives much relative stability with a very little oscillation.

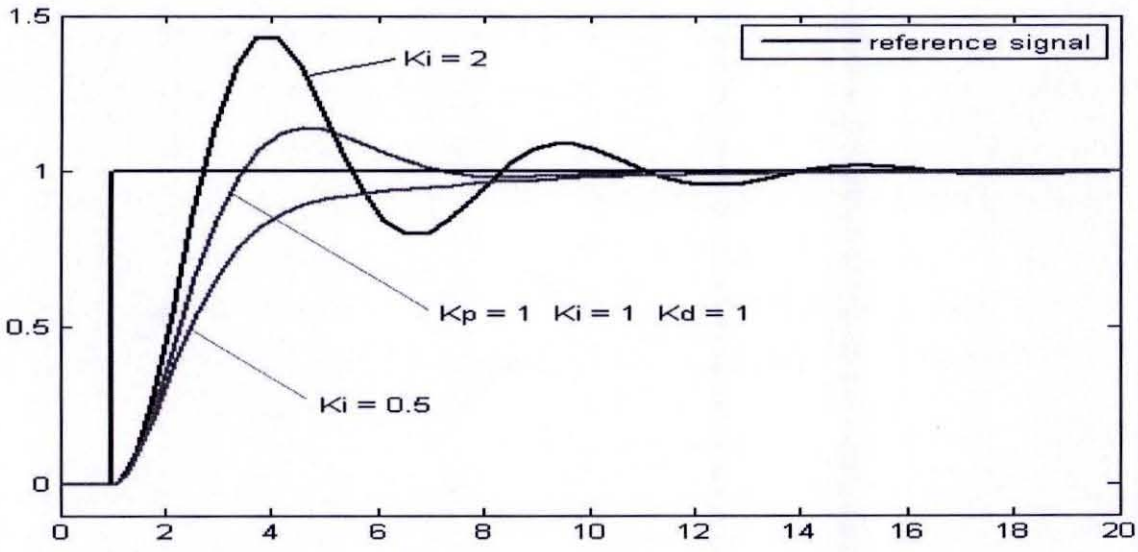

Figure 2.3 : Change of response for varying K_i

2.4 PROPORTINAL DERIVATIVE CONTROLLER

In the proportional and derivative controller the transfer function of a differentiator is $T_d s$, but this is undesirable because it magnifies any noise which may be introduced by measurement for example, quantization effects in the A/D converter. consequently, a differentiator is approximated by $T_d s/as+1$,as it is shown in figure 2.4, which is a differentiator and a low-pass filter. Here a is considered very small. Most case it is close to *0.1*.

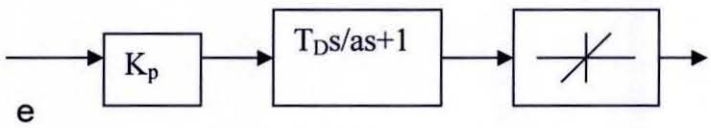

Figure 2.4: A block Diagram of a PD controller

For continuous system ,derivative or the rate control has the form of $u(t)= K_p T_D e(t)$. Which is in frequency domain $D(s)= K_p T_D s$. Where T_D is called the derivative time. Differentiation can be approximated in the discrete domain as the first diffrence, thet is.

$$u(k) = \frac{K_p T_d((e(k) - e(k-1))}{T} \quad (9)$$

If in the equation above the z transform is performed than it gives

$$D(z)= \frac{K_p T_d (z-1)}{Tz} \quad (10)$$

Or we can replace s as *(z-1)/Tz*. According to the equation *(2)*.

Like PI controller, in many design, the compensation is sum of proportional and derivative control or PD control. In this case we have.

$$D(z) = \frac{K_p(1+T_d(Z-1))}{Tz} \tag{11}$$

The derivative term slows the rate of change of the controller output and this effect is most noticeable close to the controller setpoint. Hence, derivative control is used to reduce the magnitude of the overshoot produced by the integral component and improve the combined controller-process stability. However, differentiation of a signal amplifies noise and thus this term in the controller is highly sensitive to noise in the error term, and can cause a process to become unstable if the noise and the derivative gain are sufficiently large. If the K_d that is K_pT_i is set into 1 it gives the better effect by reducing the overshoot at maximum.

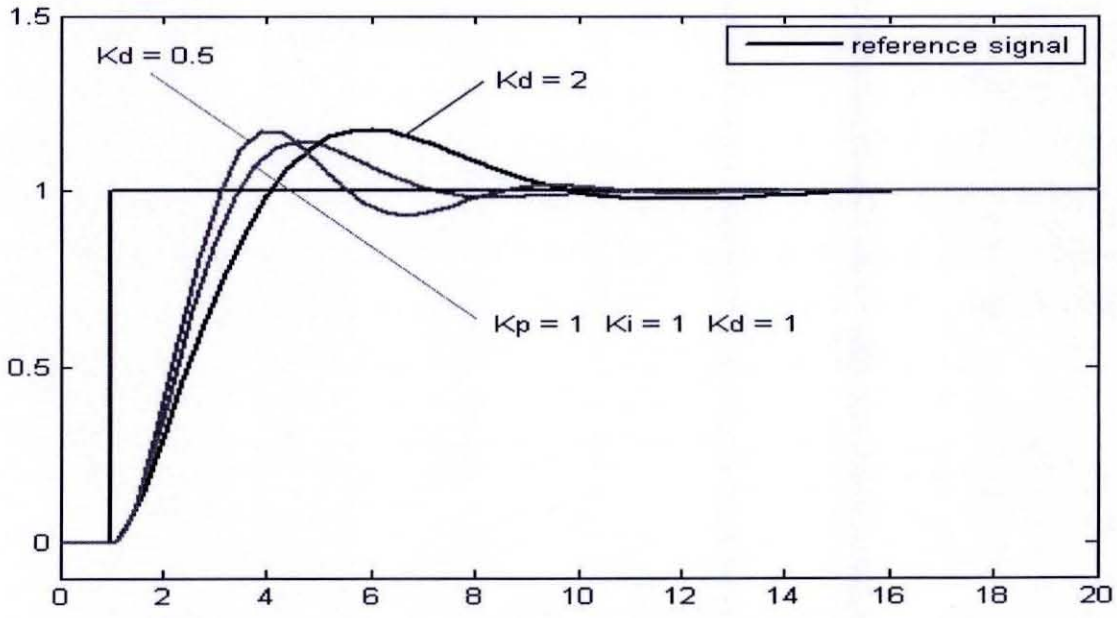

Figure 2.5: Change of response for varying K_d

2.5 PROPORTINAL INTEGRAL DERIVATIVE(PID) CONTROLLER

Combining all the above yields the PID controller.

$$D(s)= \frac{K_p(1+K_p)}{T_{is}+K_pT_ds} \tag{12}$$

In discrete form The expression will give the form of D(z) shown in equation 13.

$$D(z)= K_p(1+\frac{Tz}{T_i(z-1)}+T_d\frac{(z-1)}{Tz}) \tag{13}$$

In Figure 2.6 the block diagram of PID controller is shown. The entire out put of PID controller can be sent into a wind- up protection for better out put result.

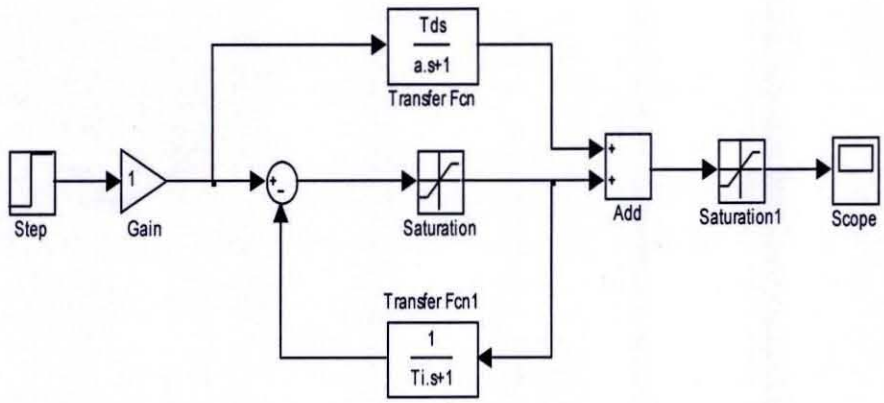

Figure 2.6: A block Diagram of a PID controller

K_p: Proportional Gain - Larger K_p typically means faster response since the larger the error, the larger the Proportional term compensation. An excessively large proportional gain will lead to process instability and oscillation.

K_i: Integral Gain - Larger K_i implies steady state errors are eliminated quicker. The trade-off is larger overshoot: any negative error integrated during transient response must be integrated away by positive error before we reach steady state.

K_d: Derivative Gain - Larger K_d decreases overshoot, but slows down transient response and may lead to instability due to signal noise amplification in the differentiation of the error.

If the PID controller parameters (the gains of the proportional, integral and derivative terms) are chosen incorrectly, the controlled process input can be unstable, i.e. its output diverges, with or without oscillasion, and is limited only by saturation or mechanical breakage. Tuning a control loop is the adjustment of its control parameters (gain/proportional band, integral gain/reset, derivative gain/rate) to the optimum values for the desired control response.

The optimum behavior on a process change or set point change varies depending on the application. Some processes must not allow an overshoot of the process variable beyond the set point if, for example, this would be unsafe. Other processes must minimize the energy expended in reaching a new set point. Generally, stability of response (the reverse of instability) is required and the process must not oscillate for any combination of process conditions and set points. Some processes have a degree of non-linearity and so parameters that work well at full-load conditions don't work when the process is starting up from no-load.

There are several methods for tuning a PID loop. The most effective methods generally involve the development of some form of process model, then choosing

P, I, and D based on the dynamic model parameters. Manual tuning methods can be relatively inefficient.

2.6 Ziegler–Nichols method

Another tuning method is formally known as the Ziegler–Nichols method, introduced by John G. Ziegler and Nathaniel B. Nichols. This method will produces the initial value of K_p, K_i, K_d on which the system observes the maximum desired out put As in the method, the *I* and *D* gains are first set to zero. The "P" gain is increased until it reaches the "critical gain" K_c at which the output of the loop starts to oscillate. K_c and the oscillation period P_c are used to set the gains as shown:

Table 2.1

Ziegler- Nicole's Chart

Ziegler–Nichols method			
Control Type	K_p	K_i	K_d
P	$0.5 \cdot K_c$	-	-
PI	$0.45 \cdot K_c$	$1.2K_p / P_c$	-
PID	$0.6 \cdot K_c$	$2K_p / P_c$	$K_p P_c / 8$

2.7 Runge-Kutta Algorithm

Runge-Kutta methods are useful for numerically solving certain types of ordinary differential equations. Deriving high-order Runge-Kutta methods is no easy task, however. There are several reasons for this. The first difficulty is in finding the so-called order conditions. These are nonlinear equations in the coefficients for the method that must be satisfied to make the error in the method of order $O(h^n)$ for some integer n where h is the step size. The second difficulty is in solving these equations. Besides being nonlinear, there is generally no unique solution, and many heuristics and simplifying assumptions are usually made. Finally, there is the problem of combinatorial explosion. For a twelfth-order method there are 7813 order conditions! In our case we used 3rd order Runge-Kutta iteration.

This package performs the first task: finding the order conditions that must be satisfied. The result is expressed in terms of unknown coefficients a_{ij}, b_j, and c_i. The s-stage Runge-Kutta method to advance from x to $x+h$ is then

$$Y(x+h) = y(x) + h\sum_{j=1}^{s} b_j f\left(Y_j(x+h)\right) \tag{13}$$

where

$$Y_i(x+h) = y(x) + h\sum_{j=1}^{s} a_{ij} f\left(Y_j(x+h)\right), \quad i = 1, 2, \ldots, s \tag{14}$$

Sums of the elements in the rows of the matrix $[a_{ij}]$ occur repeatedly in the conditions imposed on a_{ij} and b_j. In recognition of this and as a notational convenience it is usual to introduce the coefficients c_i and the definition

$$c_i = \sum_{j=1}^{s} a_{ij}, \quad i = 1, 2, \ldots, s \tag{15}$$

This definition is referred to as the row-sum condition and is the first in a sequence of row-simplifying conditions.

If a_{ij}=0 for all $i<=j$ the method is *explicit*; that is, each of the $Y_i(x+h)$ is defined in terms of previously computed values. If the matrix $[a_{ij}]$ is not strictly lower triangular, the method is *implicit* and requires the solution of a (generally nonlinear) system of equations for each timestep. A diagonally implicit method has a_{ij}=0 for all $i<j$.

There are several ways to express the order conditions. If the number of stages *s* is specified as a positive integer, the order conditions are expressed in terms of sums of explicit terms. If the number of stages is specified as a symbol, the order conditions will involve symbolic sums. If the number of stages is not specified at all, the order conditions will be expressed in stage-independent tensor notation. In addition to the matrix a and the vectors b and c, this notation involves the vector e, which is composed of all ones. This notation has two distinct advantages: it is independent of the number of stages *s* and it is independent of the particular Runge-Kutta method.

CHAPTER III.

REAL TIME OPERATING SYSTEM

3.1 REALTIME SYSTEM

A real time system is an information system, whose correction depends on moment in time when logic output occur rather than the logic output of the algorithm. The output must be reached within a specified time interval.It is not sufficient that the resulting output is correct; Thus, a real time system is not necessarily fast, but must be accurate in time. The design of the real time system goes under multiple stage. At the first stage the task to be performed and the temporal restriction that must be satisfied are identified. At the next stage the code is written and finally the run time of each task is measured and schedualbility test is done to ensure that the tasks will not miss their deadline while the system is running. Real time is divided into two areas: hard real time and soft real time. The hard real time application fails if their operating system timing requirements are not met. On the other hand soft real time applications tolerate large latencies in what they have requested from the operating system. The real time system is implemented with a combination of Linux, RT-Linux, data acquisition cards, source code and standard PC.

3.2 Linux and RT Linux

RT- Linux is an operating system in which a small real time kernel coexist with Posix-like Linux kernel shows in Figure 3.1.The intention is to make use of the sophisticated services and highly optimized average case behavior of a standard time shared computer system while still permitting real time functions to operate in a predictable and low latency environment. The design philosophy behind RT Linux was to minimize the changes made to Linux itself, providing only the essentials necessary for implementing real time applications . Instead of modifying the kernel of Linux to make it predictable, what it does is to build directly over the processor a small kernel, independent of the Linux kernel with a scheduler.

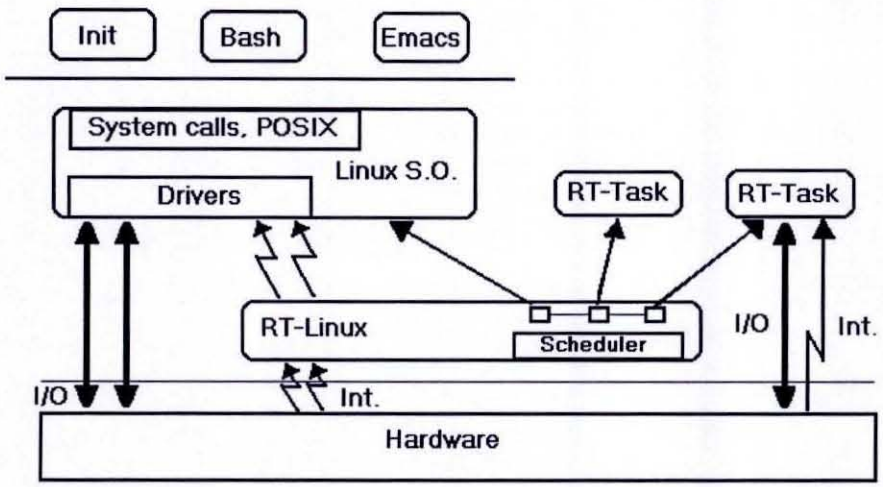

Figure 3.1: A Linux based Real time Operating System(RT Linux)

The Linux kernel runs on top of this kernel sharing the processor with other RT tasks. Linux then shares the CPU with other tasks, and runs only when no other RT task are running. The default scheduler that comes with RT Linux is a preemptive, fixed priority scheduler and considers the Linux task as the task with the lowest priority. If the real time tasks consume all the processor time, then the Linux task will not receive time from the CPU and it may give the impression that

the system is halted. A room temperature control using PID controller in real time Linux environment would be an example, where a sensor will sense the temperature and it will be fed back to the PID controller which will maintain a certain level of temperature and not let the system to exceed the desired temperature in any condition.

3.3 Real time implementation of servo system

A Real Time System is an information system whose correction does not only depend on the logical output of the algorithms but also on the moment in time when these output occurred. Thus, a real time system is not necessarily fast, but must be accurate in time. The design of a real-time system goes through several phases: First the tasks to be performed and the temporal restrictions that must be satisfied are identified, Secondly the code is written and Finally the run-time of each task is measured and a schedulability test is done to ensure that the task will not miss its deadline while the system is running. Real-time is divided into two areas: hard and soft real-time. We will concentrate our discussion to hard real-time systems.

There is a wide range of real-time operating systems available in the market to support any specific real-time performance required by the control applications. The real-time performance is limited to the low-level interaction with the hardware. lack of synchronization and random delays appears in a natural way in Industrial distributed computing and parallel processing, where shared communication is used with a lot of sensors and actuators. In the servo experiment, random delays are generated by means of real-time thread program instead of using real industrial multi-processor systems. The real-time systems is implemented with a combination of Linux, RT-Linux, data acquisition card, servo systems, availability

of source code and a standard 120 MHz Pentium based PC, with a hard disk, SVGA video interface and a 128Kb RAM.

3.4 Hardware/Software Data Acquisition Structure

The servo experiment under RT-Linux environment included a 120 MHz Pentium laboratory PC with the feedback modules and ax5411 data acquisition card. We have loaded Knoppix 2.78 and the basic system came up and used Vxscope for plotting the systems response. RT-Linux comes as a set of different files from the source version 2.2.18. From the software point of view, a real-time thread program is developed. It caught an interrupt from a clock, latched the interrupts, read the clock, waited the right amount of time for the encoder latch to settle and read the position. With the position known, it called the PID routine to determine the output voltage to drive the motor. The clock interrupt was arbitrarily set to10msec. Among the two important real-time issues, one is the ability to latch the encoders in the interrupt service routine and the other one is the ability to compute the response within a reasonable period of time(10 milliseconds) so that no interrupt is missed. Real-time data acquisition framework of multi-rate sampled-data system is presented in RT-Linux environment. The RT-Linux facilities for task handling are basic. There is rt_task_init(), which creates and starts a task. The stack size and priority can be specified.Linux itself is run as a real-time task with the lowest priority. The task is set up to run at periodic intervals by rt_task_make_periodic(). The rt_task_wait() facility blocks the calling task. The Task are run using a simple preemptive scheduler.

The primary means of communication between the real-time tasks and the Linux processes is the FIFO. The rtf_create() facility creates a FIFO of a desired size. Data is enqueued onto the FIFO by rtf_put(), returning an error if

the FIFO is full. Similarly, rtf_get() dequeues data from the FIFO, returning an error if the FIFO is empty. The most obvious use for this FIFO scheme is data streaming. In a data acquisition application, for example, a real-time task could be set up using rt_task_init() and rt_task_ make_periodic() to acquire samples from an I/O board at fixed intervals. This task would send its data to a Linux process using rtf_put() . The Linux process would be in a loop, reading data from the FIFO and perhaps writing the data to disk, sending it over a network, or displaying it in an X-window. The FIFO would serve as a buffer, so the Linux process could operate without real-time constraints.

A simple real-time kernel with a priority scheduling scheme was implemented for the data acquisition card with different sampling and hold rate. Tasks are assigned to each module during the initialization step. The flow of data to and from the data acquisition card is implemented with real-time tasks through real-time FIFO. A preemptive scheme was implemented using interrupt-based techniques to handle three real-time tasks. The inter-task communication among the three real time task is done with shared memory. The RT-kernel receives a fixed set of tasks at the time of initialization and each of the tasks has a priority level assigned in the preemptive scheme. The real-time task communicates with non real-time Linux and the data acquisition card and the RT-FIFOs avoid message losses. The non real-time GUI program using GTK (GIMP Toolkit) in the Linux environment provides the high level interfaces between the user and the experiment. The set of tasks (e.g., data logging, display and GUI etc.) assigned to it is non time-critical. In the data acquisition scheme, the real-time tasks delivering to Linux a low rate of results and final data through shared memory and circular buffer. Thus, Linux is slightly loaded saving CPU resources for non time-critical tasks (display, data logging etc.).

3.5 Real-time Controller

A synchronization between controller and plant and loss of information due to the random delays are a potential cause of system instability. One possible solution to this problem involves using multi-rate technique and also RT-Linux operating system is needed to perform control task and time management of this kind of systems. In this regard, a multi-rate sampled-data controller can be implemented with different frequencies in the sampler and hold. In fact, a slow frequency is applied to the sampler (A/D) to provide the controller with the necessary information to take its decisions and employ a fast frequency to the hold (D/A) to apply control actions. Thus, in the real-time tasks, the priority of the fast frequency to the 'dtoa' must be higher than the slow frequency to the sampler. In selecting these frequencies RT-Linux resolution 0.01 seconds must be considered so that the fast frequency to be high enough to achieve required control specifications and slow frequency to be low enough to avoid the loss of information due to the random delays. Multi-rate controller generates higher frequency discrete control signal in its output and the lower frequency feedback signal in its input. A controller program is referred to the appendix A.

CHAPTER IV.

TEMPERATURE CONTROLLER

4.1 Introduction to Temperature control

Temperature control is a process in which the temperature of an object is measured and the passage of heat energy into or out of the object is adjusted to achieve a desired temperature.

A thermostat is a simple example for a closed control loop: It constantly measures the current temperature and controls the heater's valve setting to increase or decrease the room temperature according the user-defined setting. A simple method switches the heater or cooler either completely on, or completely off, and an overshoot and undershoot of the controlled temperature must be expected. A more expensive method varies the amount of heat or cooling provided by the heater or cooler depending on the difference between the required temperature (the "set point") and the actual temperature. This minimizes over/undershoot.

To accurately control process temperature without extensive operator involvement, a temperature control system relies upon a controller, which accepts a temperature sensor such as a thermocouple or thermostat as input. It compares the actual temperature to the desired control temperature, or set point, and provides an output to a control element. The controller is one part of the entire control system, and the whole system should be analyzed in selecting the proper controller. The following items should be considered when selecting a controller:

1. Type of input sensor (thermocouple, thermostat) and temperature range
2. Type of output required (electromechanical relay, analog output)

3. Control algorithm needed (on/off, proportional, PID)
4. Number and type of outputs (heat, cool, limit)

There are three basic types of controllers: on-off, proportional and PID. Depending upon the system to be controlled, the operator will be able to use one type or another to control the process.

4.1.1 On/Off Control

An on-off controller is the simplest form of temperature control device. The output from the device is either on or off, with no middle state. An on-off controller will switch the output only when the temperature crosses the setpoint. For heating control, the output is on when the temperature is below the setpoint, and off above setpoint. Since the temperature crosses the setpoint to change the output state, the process temperature will be cycling continually, going from below setpoint to above, and back below. In cases where this cycling occurs rapidly, and to prevent damage to contactors and valves, an on-off differential, or "hysteresis," is added to the controller operations. This differential requires that the temperature exceed setpoint by a certain amount before the output will turn off or on again. On-off differential prevents the output from "chattering" or making fast, continual switches if the cycling above and below the setpoint occurs very rapidly. On-off control is usually used where a precise control is not necessary, in systems which cannot handle having the energy turned on and off frequently, where the mass of the system is so great that temperatures change extremely slowly, or for a temperature alarm. One special type of on-off control used for alarm is a limit controller. This controller uses a latching relay, which must be manually reset, and is used to shut down a process when a certain temperature isreached.

4.1.2 Proportional Control

Proportional controls are designed to eliminate the cycling associated with on-off control. A proportional controller decreases the average power supplied to the heater as the temperature approaches setpoint. This has the effect of slowing down the heater so that it will not overshoot the setpoint, but will approach the setpoint and maintain a stable temperature. This proportioning action can be accomplished by turning the output on and off for short time intervals. This "time proportioning" varies the ratio of “on” time to "off" time to control the temperature. The proportioning action occurs within a “proportional band” around the setpoint temperature. Outside this band, the controller functions as an on-off unit, with the output either fully on (below the band) or fully off (above the band). However, within the band, the output is turned on and off in the ratio of the measurement difference from the setpoint. At the setpoint (the midpoint of the proportional band), the output on:off ratio is 1:1; that is, the on-time and off-time are equal. if the temperature is further from the setpoint, the on- and off-times vary in proportion to the temperature difference. If the temperature is below setpoint, the output will be on longer; if the temperature is too high, the output will be off longer.

4.1.3 PID Control

The third controller type provides proportional with integral and derivative control, or PID. This controller combines proportional control with two additional adjustments, which helps the unit automatically compensate for changes in the system. These adjustments, integral and derivative, are expressed in time-based units; they are also referred to by their reciprocals, RESET and RATE, respectively. The proportional, integral and derivative terms must be individually

adjusted or "tuned" to a particular system using trial and error. It provides the most accurate and stable control of the three controller types, and is best used in systems which have a relatively small mass, those which react quickly to changes in the energy added to the process. It is recommended in systems where the load changes often and the controller is expected to compensate automatically due to frequent changes in setpoint, the amount of energy available, or the mass to be controlled.

4.2 Method of Tuning PID in Temperature Controller

This procedure is based on the assumption that a critically damped system is optimal and the fact that stability and noise must be traded for response time. Please bear in mind that the second step may involve large temperature oscillations and so the procedure would not be suitable if these could be dangerous or cause damage, for example in a chemical processing plant.

John Shaw's (Ziegler-Nichols Based) Method

1. Adjusting the set-point value, T_s, to a typical value for the envisaged use of the system and turn off the derivative and integral actions by setting their levels to zero. Select a safe value for the maximum power M and set the proportional gain to minimum.

2. Progressively increase the gain until suddenly decreasing or increasing T_s by about 5% induces oscillations that are just self-sustaining.

3. The gain at this stage will be set to the ultimate gain G_u the period of the oscillations is known as the ultimate period t_u. Note the values of each quantity.

4. Set the controller parameters as follows:

    - P-Control: $P=0.50*G_u$, $I=0$, $D=0$.
    - PI-Control: $P=0.45*G_u$, $I=1.2/t_u$, $D=0$.
    - PID-Control: $P=0.60*G_u$, $I=2/t_u$, $D=t_u/8$.

5. Check the overall performance of system is satisfactory under the conditions it will be used.

This procedure was adapted slightly from Jhon Show's, description of the Ziegler-Nichols Closed Loop method. It should yield a system that is slightly under damped; if a less "aggressive" response is desired try reducing P to half the values listed. As was the case with the CDHW method the second step may involve large temperature oscillations and so the procedure would not be suitable if these could be dangerous or cause damage, for example in a nuclear reactor. The Ziegler-Nichols method was developed for the traditional series, or interacting design of controller.

4.3 Implementing Temperature Controller

The goal of this project is to give input from the computer and computer will control entire system. The input given from the computer is the set point of the system. In the initial case system will be on, and it will be on until system come up to set point. When the system reaches up to the set point the system will be fixed. A 220V, 60W bulb is used in this project as temperature control unit, relay as an actuator. The thermostat is working as a feedback. ADC, amplifier,

Whetstone bridges are combined working as a signal processing unit; GAL is working as a controller where PLD program is installed.

Initially, the bulb will be switched on from the 220V power supply. This bulb is connected to the relay as well. A thermostat is connected in such a way that this thermostat is sensing the temperature of the bulb and converts this temperature to variable resistance. This variable resistance is converted into variable voltage by using Whetstone bridge. The output voltage of the Whetstone bridge is very small. So two amplifiers is used to amplify this voltage. This amplified voltage is connected to the analog to digital converter (ADC) which converts this analog voltage to 8bit digital output. This digital output is combining with the computer input in the gal chip and the output of the gal chip is given as an input of the relay.

Through the parallel port, input is given from the computer. This input will come to the GAL. Initially the input from computer will not be same with hardware. So this will give the output 1 from GAL which will make the relay on, and bulb will be on. When the computer input and bulb temperature will be same then the bulb will be off as the output of the GAL is 0 which will make relay off.

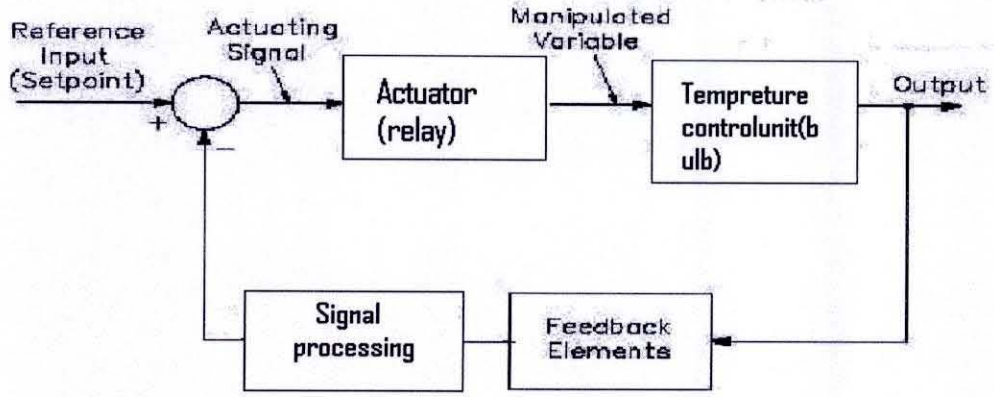

Figure 4.1: Block Diagram of Temperature Controller

4.3.1 ADC0804LCN A/D Converter

The A/D converter under study is a successive approximation type 8-bit converter. It runs from a 5-volt dc supply. The chip is microprocessor compatible and has a conversion time of 100 microseconds. The half of the reference voltage is 2.5V.So when thermocouple give2.5V then ADC will give maximum output which is 128. The following diagram shows the basic configuration of analog to digital converter.

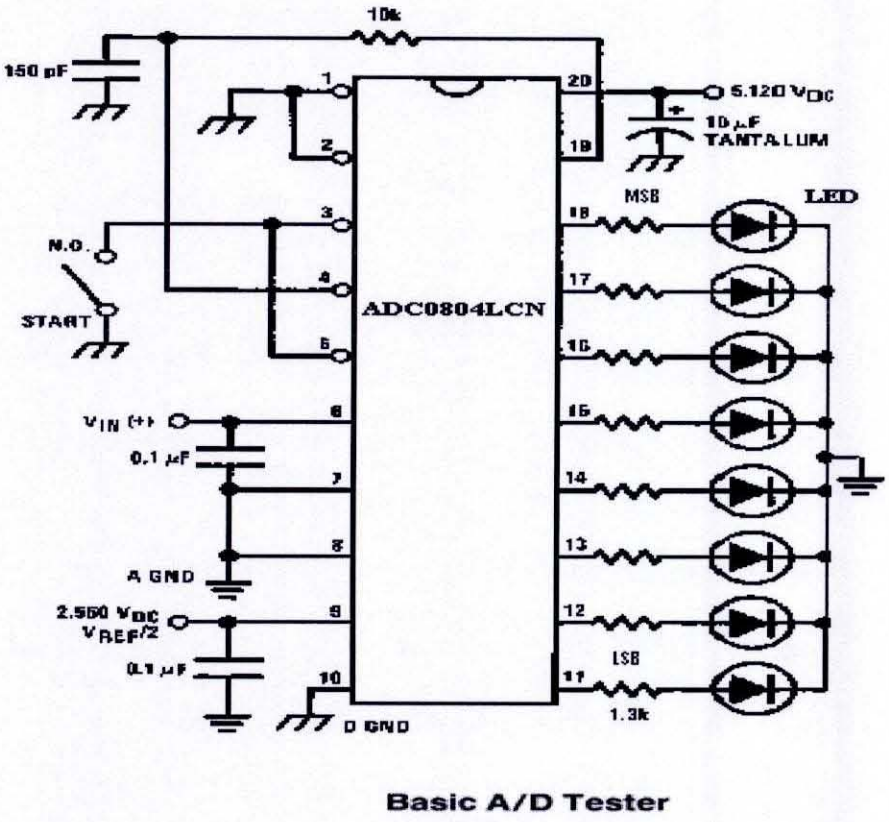

Figure 4.2: Block Diagram of A/D converter

4.3.2 Operational Amplifier

The Operational amplifier or OPAMP is a versatile analog Integrated Circuit (IC) that is capable of producing a very high gain. The property of infinite impedance and infinite gain of an operational amplifier results in a situation of zero voltage between the two input terminals. The effect is known as a virtual ground. Due to this effect, the op-amp can be used to perform some mathematical operations.

In this project amplifier is used because the output voltage or Wheatstone bridge is very small. (Less than 1). So 2 amplifiers are used, one is to amplify the output, another is for the sign changer. Sign changer is used as the input is given to inverting input of amplifier.

The equation of inverting output voltage is: Vout = - (Rf/Ri) Vin.
5V is used as biasing voltage and input resistance of amplifier is 2kohm, and feedback resistance is 6kohm.

4.3.3 Relay:

A relay is an electrical switch that opens and closes under the control of another electrical circuit. In the original form, the switch is operated by an electromagnet to open or close one or many sets of contacts. In this project relay is working as a actuator.

Normally-open (NO) contacts connect the circuit when the relay is activated; the circuit is disconnected when the relay is inactive. It is also called a Form A contact or "make" contact. Normally-closed (NC) contacts disconnect the circuit when the relay is activated; the circuit is connected when the relay is inactive. It is also called a Form B contact or "break" contact. Change-over, or double-throw, contacts control two circuits: one normally-open contact and one normally-closed contact with a common terminal. It is also called a Form C contact or "transfer" contact. If this type of contact utilizes a "make before break" functionality, then it is called a Form D contact.

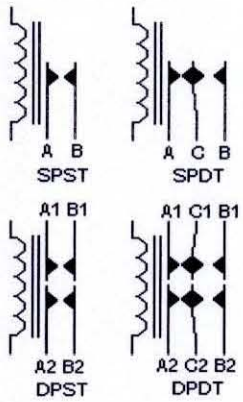

Figure 4.3: Relay

This is used to control high -voltage circuit with a low-voltage signal. Relay is connected through a control circuit. This control circuit consists of a transisitor, diode, resistance, GND and Vcc connection. Relay has got 5 pins.2 pins are connected to the control circuit's Vcc (1^{st} pin) and collector of the transistor (2^{nd} pin). Middle pin (3^{rd} pin) is common terminal of rest of the 2 pin's (4^{th} & 5^{th} pin). Among the rest of the 2 pins, one is initially short (5^{th} is short with 3^{rd} pin) with the middle pin, and another is initially open with the middle pin (4^{th} is open with 3^{rd} pin). When transistor base input is 0 then no current will flow through as no path is established between ground and Vcc.When transistor base is 1 then the transistor will be short. So current will flow through the internal inductor of the relay. As current start flowing, it will create magnetic field and this magnetic field will make the 2 initially open pins (3^{rd} & 4^{th} pin) of the relay short. so the relay will be on. The middle pin of the relay is connected to the positive 220V supply. So the short terminal will be providing 220V.and this 4^{th} pin is connected to the bulb's one terminal. The other terminal of the bulb is connected to negative 220V supply. As the bulb is getting 220V supply, so it become on. This is the process where relay is working as a switch of the bulb.

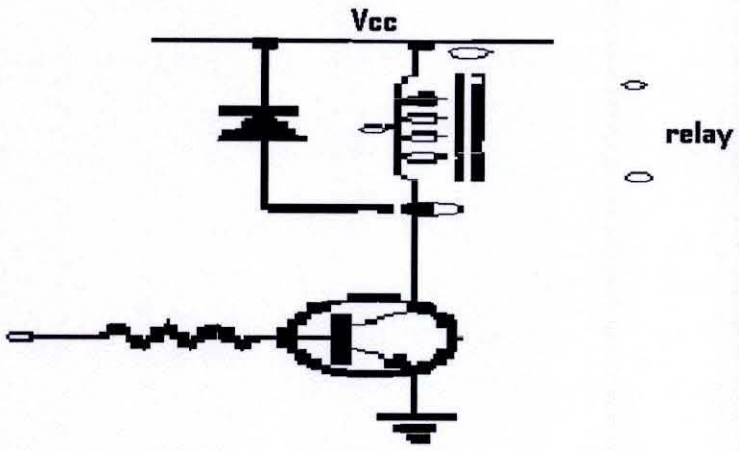

Figure 4.4 : Driving a Relay

4.3.4 Thermostat

A thermostat is a type of resistor with resistance varying according to its temperature. The word is a combination of thermal and resistor. Thermostats are widely used as inrush current limiters, temperature sensors, self resetting over current protectors, and self regulating heating elements.

Assuming, as a first-order approximation, that the relationship between resistance and temperature is linear, then:

$$\Delta R = k\Delta T$$

where

ΔR = change in resistance
ΔT = change in temperature
k = first-order temperature coefficient of resistance

This is the feedback element used in this project. According to the temperature it gives variable resistance.

4.3.5 Whetstone Bridge:

In the circuit at right, R_x is the unknown resistance to be measured; R_1, R_2 and R3 are resistors of known resistance and the resistance of R_2 is adjustable. If the ratio of the two resistances in the known leg (R_2 / R_1) is equal to the ratio of the two in the unknown leg (R_x / R_3), then the voltage between the two midpoints (B and D) will be zero and no current will flow through the galvanometer V_g. R_2 is varied his condition is reached.The current direction indicates whether R_2 is too high or too low.

Detecting zero current can be done to extremely high accuracy (see galvanometer).Therefore, if R_1, R_2 and R_3 are known to high precision, then R_x can be measured to high precision.Very small changes in R_x disrupt the balance and are readily detected.At the point of balance, the ratio of R_2 / R_1 = R_x / R_3.Therefore, Alternatively, if R_1, R_2, and R_3 are known, but R_2 is not adjustable, the voltage or current flow through the meter can be used to calculate the value of R_x, using Kirchhoff's current_ laws (also known as Kirchhoff's rules).

If all four resistor values and the supply voltage (V_s) are known, the voltage across the bridge (V) can be found by working out the voltage from each potential divider. And subtracting one from the other.

V=[(Rx/Rx+R3)-(R1/R1+R2)]Vs (16)

In this project it is used to convert the thermocouple resistance to variable voltage. 6V supply is used. the resistance of the bridge is 91ohm.

This can be simplified to:

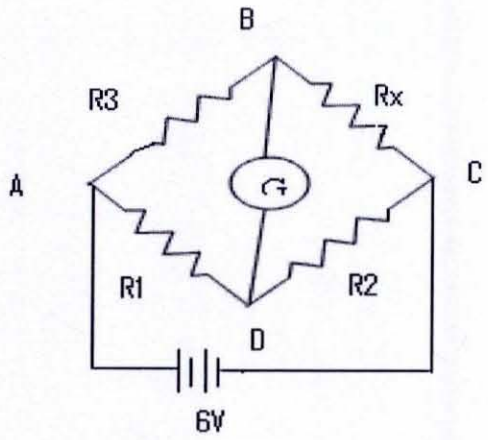

Figure 4.5 : Whetstone Bridge

4.4 Programmable Logic Device

A programmable logic device or PLD is an electronic component used to build reconfigurable digital circuit. Unlike a logic gate, which has a fixed function, a PLD has an undefined function at the time of manufacture. Before the PLD can be used in a circuit it must be programmed.

The generic array logic device, or GAL is device has the logical properties that can be erased and reprogrammed. The GAL is very useful in the prototyping stage of a design, when any bugs in the logic can be corrected by reprogramming. GALs are programmed and reprogrammed by using the in-circuit programming technique on supporting chips.

In this project, GAL has got 8 inputs, one output.4 inputs are given from computer and 4 is given from ADC(4 MSB).this 8 inputs are going to 4 AND gates each with 2 inputs. These 4 AND gates give 4 outputs which will go to another 4 AND gates each with 4 inputs. These 4 AND gates will again give 4 outputs which will go to 4 input OR gate and will have one output. This OR gate output will go to an inverter. So the whole output will be inverted for this inverter initially when input of the GAL is 0, output of GAL will be 1 which will make the relay on all time. The Boolean equation of the GAL is:

$$Y' = AB'C'D' + A'BC'D' + A'B'CD' + A'B'C' \quad (17)$$

$$Y = (A+B'+C'+D')(A'+B+C'+D')(A'+B'+C+D')(A'+B'+C'+D) \quad (18)$$

When the bulb is given any input, it will be heated up. So the corresponding digital output of the temperature is given to GAL and computer input is also given to GAL. Suppose computer input is 30 deg or 64bit. Then initially the bulb will be on as the compute input and hardware input is not same. So gal output will be 1 and relay is on which will make the bulb on. The bulb will be heated up until the ADC give 30 deg's corresponding digital output. When bulb's temperature will be 30 deg then the bulb will start fluctuating. It will be happen because the set point is 30 deg, when the bulb temperature is 30 deg it will be give output 1 from GAL's OR gate and will be inverted and give 0 output which will make the relay off. So bulb will be off. Again when bulb's temperature will become less than 30 deg(suppose 29 deg) then again inverted output of GAL will be 2 and this will make the relay on so bulb will be on. This process will go on until we change the computer input. So GAL is working as the controller of this temperature control unit. A Program of Programmable Logic Device ig referred to the chapter appendix 5.C.

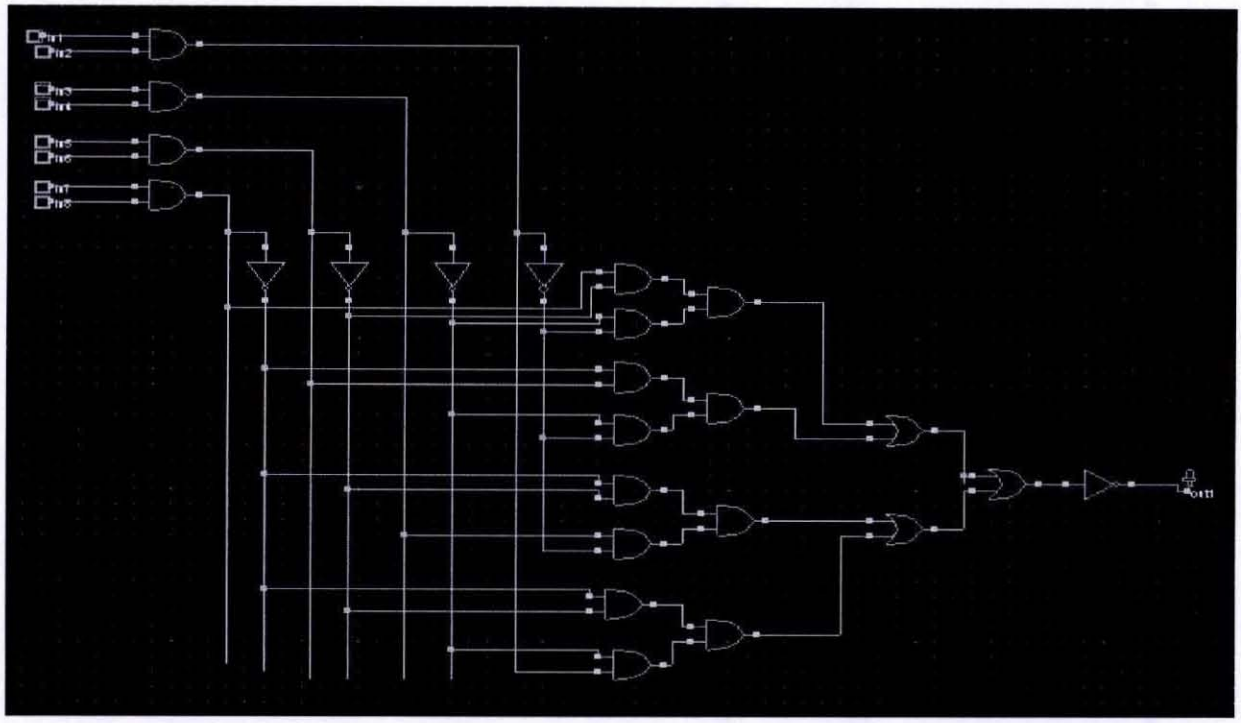

Figure 4.6: Inter logic diagram of GAL chip

4.5.1 AX5411 Data Acquisition Card

AX5411H is a multifunction analog/digital input/output board and is functianlly compatible with AX5411. It may also be plugged into one of the available slots in the IBM□ PC/AT or compatibles. Analog input characteristic of the AX5411H is designed to allow you to sample data at high throughput, the combination of hardware auto-scanning multiplexer, high-speed sample/hold and A/D converter allow input sampling speeds up to 60 KHz. DMA transfer allows you to transfer large amounts of data to memory at such high rate. With programmable gains of 1, 2, 4, 8 and 16, and full scale range of ±5V and ±10V, user can define a particular range for each input corresponding to the signal level connected to that channel. This feature will give optimum resolution to each channel's measurement.

In addition to the data acquisition channels of AX5411H, the board contains

two independent analog voltage output channels. Each channel has its own 12-bit D/A converter. These two channels can be individually set to output voltage within the range of 0 to 5V or 0 to 10V. The AX5411H also provides a 24 channel digital input port and a 24 channel digital output port. Both ports are TTL compatible. The converted data may be collected through the software command, an interrupt service routine or DMA channels. A complete utility .AS59099 DAC Driver CD, containing driver routines and example programs, is furnished with the board to minimize user's efforts on application software development. These subroutine libraries are available to control AX5411H functions from user written programs.

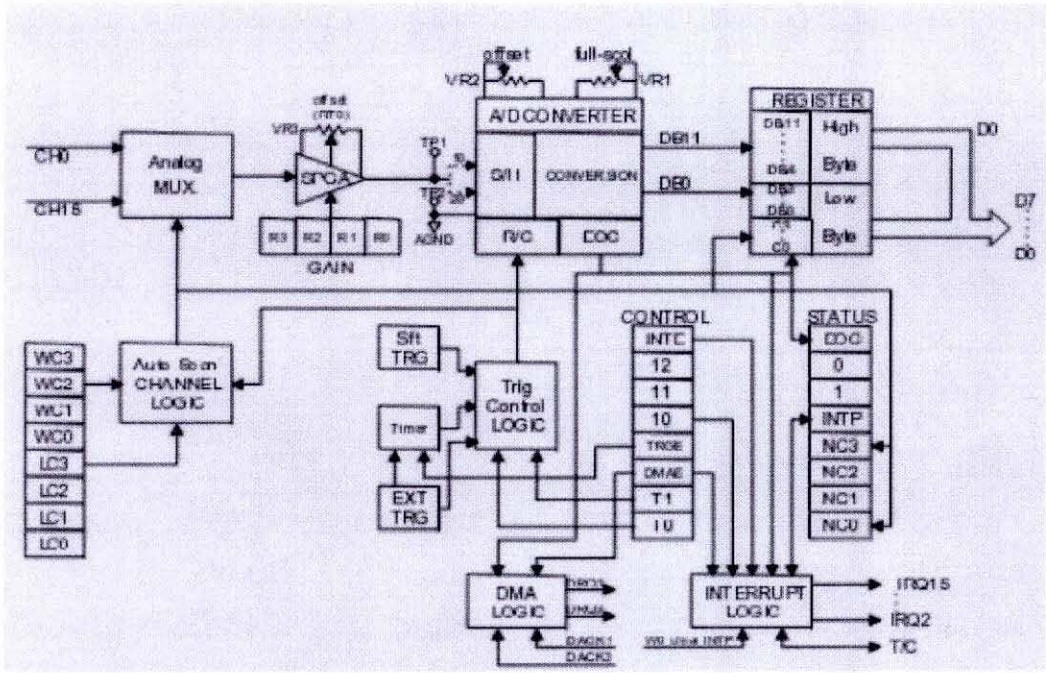

Figure:4.7 Internal Block diagram of AX5411

4.5.2 Data Acquisition Principles

Factory automation (FA) and laboratory automation(LA) have become a truth tactic owing to the revolution in computer. The advent of the personal computer (PC) makes this task more prevalent and versatile because of the low

cost and flexibility. Typically a computerized data acquisition and control system is constructed as the structure provided in chapter five D.

4.5.3 Analog Input System

The basic function of analog input system (A/D system) is to convert the analog input signal to corresponding digital format that the computer can be read. In order to benefit A/D system's stability and obtain good performance, several other additional parts such as multiplexed, amplifier, sample and hold are required. The following figure exhibits many input signals come into the A/D system.

A/D converter has several kinds been developed for different application needs. The most widely used A/D converters are integrating and successive approximation. Integrating type has the advantage of high noise rejection but its speed is lower. Most data acquisition system needs at least 12-bit resolution to recognize the measured signal. More high resolution a/d will elapse more time and normally the price is higher. To select suitable specification to meet your need is a good sense.

Multiplexed is a device containing a group of switches to select exact channel signal go through A/D converter. Because the amplifier and A/D are shared, the cost will be lower, but the channel's acquisition speed will be reduced too.

Sample and hold (S/H) is designed for A/D converter to keep track of input signals. When a high speed A/D converter is used, such as successive approximation, it requires a no changing input signal during the conversion stage. The S/H circuit can keep the input signal with a constant level until the a/d conversion is completed.

Often the limiting factor in the application of the S/H is the uncertainty in the time the actual sample is taken - i.e. The "aperture jitter" or Taj. The aperture jitter causes an amplitude uncertainty for any input where the voltage is changing. The

approximate voltage error due to aperture jitter depends on the slew rate of the signal at the sample point.

4.5.4 Lab Result of Temperature Controller

In our lab we have tested this temperature control unit (bulb) temperature and we have got this the corresponding voltages. It has a linear relationship between this temperature and voltage.

Table 4.1
Lab result of temperature controller

Temperature(C)	Voltage(V)
27	0.1
28	0.13
30	0.2
31	0.23
32	0.27
33	0.3
34	0.33
35	0.36
36	0.4
37	0.41
38	0.42
39	0.44

Here the voltage is very small. So this low voltage is amplified and it has been amplified six times. So 27 deg corresponding amplified voltage is 0.33V, 30 deg amplified voltage is 0.64V, 33 deg amplified voltage is 1.23V and 39 deg amplified voltage is 2.6V. The graphical representation is as follows.

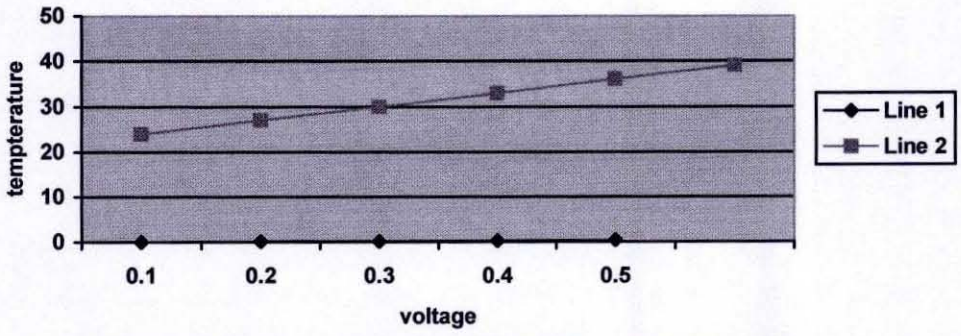

Figure 4.8: Linear graph of Temperature controller

CHAPTER V.

5.1 Conclusion

PID controllers have a wide range of applications in automatic control of systems such as servomotors and temperature control. This controller combines proportional control with two additional adjustments, which helps the unit automatically compensate for changes in the system. These adjustments, integral and derivative, are expressed in time-based units. For achieving the desired performance the system output (axis position) is feed-back to the PID controller which generates the system input (motor voltage), in a closed loop. The controller computes the error between the desired system output (reference) and the actual system output. By measuring these two values the entire system will give accurate output.In real time, first the tasks to be performed and the temporal restrictions that must be satisfied are identified, Secondly the code is written and Finally the run-time of each task is measured and a schedulability test is done to ensure that the task will not miss its deadline while the system is running.

The PID controller calculation (algorithm) involves three separate parameters; the Proportional, the Integral and Derivative values. The Proportional value determines the reaction to the current error, the Integral determines the reaction based on the sum of recent errors and the Derivative determines the reaction to the rate at which the error has been changing.

Temperature control is a process in which the temperature of an object is measured and the passage of heat energy into or out of the object is adjusted to achieve a desired temperature. It provides an output to a control element. The controller is one part of the entire control system, and the whole system should be analyzed in selecting the proper controller.

AX5411H is a multifunction analog/digital input/output board and is functionally compatible with AX5411. It may also be plugged into one of the available slots in the IBM□ PC/AT or compatibles. Analog input characteristic of the AX5411H is designed to allow you to sample data at high throughput. In addition to the data acquisition channels of AX5411H, the board contains two independent analog voltage output channels. Each channel has its own 12-bit D/A converter. These two channels can be individually set to output voltage within the range of 0 to 5V or 0 to 10V. Because unavailability of AX5411 within the time constraint we construct ADC and Logical control for our thesis.

In our lab we have tested this temperature control unit (bulb) temperature and we have got this the corresponding voltages. It has a linear relationship between this temperature and voltage.

REFERENCES

[1] SHARP, "Data Sheet: GP2Y0A02YK Long Distance Measuring " Sensor,http://sharp-world.com/products/device/lineup/data/pdf/datasheet.

[2] "Modern Control Engineering Fourth Edition", Katsuhiko Ogatha.

[3] FOX Electronics, "Data Sheet: 5.0 TTL Clock Oscillator," 06 Jan. 2004, http://www.foxonline.com/PDFS/f1100e.pdf.

[4] Custom Computer Services, "C Compiler Reference Manual" Version 4, Nov. 2006, http://www.ccsinfo.com/downloads/ccs_c_manual.pdf.

[5] Microchip, "PIC 16F87XA Data Sheet: 28/40/44- Pin Enhanced Flash Micro controllers," Version DS39582B, 2003, http://ww1.microchip.com/downloads/en/DeviceDoc/39582b.pdf.

[6] Samsung, "LCD-82 Data Sheet: UC-20102-GNARS 1 line 20 character," University of Illinois at Urbana-Champaign ECE workshop.

[7] Advance A/D and D/A Conversion Techniques and Their Applications, 27-28 July 1999 Conference Publication no. 466 (c) IEE 1999

[8] Taylor,H.R,1997,"Data acquisition for sensor systems", Chapman and Hall

[9] OPTEK, "Reflective Object Sensor Datasheet: OPB710, OPB710F, IPB730, OPB730F," Issue A.3, Apr. 2006,

[10] "Digital Control of dynamic systems", Gne. F. Franklin,S.david Powel, Michel L. Workman.

[11] Real-time implementation of multirate sampling system in RT-Linux Environment

[12] Real-time Performance anlysis of Multi-rate controller.

[13] AX5411 General Purpose DA&C Board, User's Manual.

[14] A digital PID controller uning RT-Linux,R.S Guerra, H.J Gonclalves , W. F Lages

[15] PID Controller Implementation in Numerical Simulation Program Applied to the Power static converter. Denz Cruz Martins, Pedro Armando da Silva, Rubens.

[16] A genetic algorithm approach to designing finite-precision PID controller structure. J . F Whidborne.

[17] A simple procedure for improving performance of PID controllers.

[18] Rt-Linux.http://www.rtlinuxgpl.org/mailing_list/rtl.w5archive/0105

[19] An Integral Design Approch in PID controller. Jen-Yang Chen.

[20] Real-Time Linux features and applications, http://www.linux.com/

APPENDICES

A. PID control code.

```
#include <rtl.h>
#include <pthread.h>
#include <linux/ioport.h>
#include <sys/io.h>
#include "mbuff.h"
#include "io.h"
#include "cdsm.h"


/* rt thread code */
void *control_code(void *arg);

/* controller calculation function */
void control_calc(void);

/* rt thread for simulating the process */
pthread_t control_thread;
extern pthread_t dtoa_thread;

/* control loop data structure - shared memory */
volatile cont_loop *cont;


/* this is run when the module is inserted */
int init_module(void)
{
        /* declare the shared memory to be tagged "pid_control". Each task that
wants to access */
        /* this memory must do so using this label */
        cont = (volatile cont_loop*) mbuff_alloc("pid_control",sizeof(cont_loop));
        if (cont == NULL) {
                rtl_printf("mbuff_alloc failed\n");
                return -1;
        }

        CDSM_init();

        /* initialise controller */
        init_controller(1);

        /* create thread. they start to run immediately after this call */
```

```
        /* set up with default attributes - NOT real-time yet */
        pthread_create(&control_thread,NULL,control_code,NULL);


        return 0;
}

/* called when the module is removed from the kernel address space */
void cleanup_module(void)
{
        /* delete the thread */
        pthread_delete_np(control_thread);

      /* reset control loop */
      reset_controller();

        CDSM_done();

        /* deallocate the module from shared memory */
        /* this does not necessarily free up the memory space (other tasks may
still be using it) */
        mbuff_free("pid_control",(void*)cont);
}



/* sample thread code */
void *control_code(void *arg)
{
        struct sched_param p;

        /* set attributes of thread */
        p.sched_priority = 1;
        pthread_setschedparam(pthread_self(),SCHED_FIFO,&p);

     /* enable floating point operations */
     pthread_setfp_np(pthread_self(),1);


        /* infinite loop */
        while(1) {

                /* suspend this thread */
                pthread_suspend_np(pthread_self());

                /* only if on */
```

```
            if (cont->state == 1) {

              // calculate the control signal (PID controller)
              control_calc();

            }

          //rtl_printf("debug: control thread\n");

          /* wakeup dtoa thread */
          pthread_wakeup_np(dtoa_thread);

        }
        return 0;
}


/* the engine of the controller */
void control_calc(void)
{
        float a,b,N,T,k,ti,td;
        float u1,u2,u3,u4;
        float r,y,u,e;

        /* pick off the top setpoint and output from the buffer (-1,1) */
        r = cont->r[cont->first];
        y = cont->y[cont->first];

        /* constants */
        k = cont->k; ti = k*cont->ti; td = cont->td;
        N = cont->N; T = cont->dt;
        a = td / (N*T+td); b = ti / (ti+T);

        /* the algorithm starts here - according to the block diagram in the notes */
        e = r - y;

        u3 = b*cont->u3old + (1-b)*cont->u1old;
        u1 = k*e + u3;
        if (u1 > 1.0) u1 = 1.0;
        if (u1 < -1.0) u1 = -1.0;

        u4 = a*cont->u4old + (1-a)*e;
        u2 = N*k*(e - u4);

        u = u1 + u2;
```

```
        //rtl_printf("control : %4.3f %4.3f %4.3f %4.3f %4.3f %4.3f %4.3f
%4.3f\n",r,y,e,u,u1,u2,u3,k);

        /* update old signal values for next time around */
        cont->u1old = u1; cont->u3old = u3; cont->u4old = u4;

        /* signal limiter (-1,1) */
        if (u > 1.0) u = 1.0;
        if (u < -1.0) u = -1.0;

        /* set the control value */
        cont->u[cont->first] = u;

        CDSM_set(0,(int)(1000*r));
        CDSM_set(1,(int)(1000*y));
        CDSM_set(2,(int)(1000*u));

}
```

B. PID Graphical user Interface Program

```
#include <stdio.h>
#include <gtk/gtk.h>
#include <gtk/gtkhscale.h>
#include <gtk/gtkvscale.h>
#include <math.h>
#include "io.h"
#include "mbuff.h"


/* controller data structure - shared memory */
volatile int *value1;
volatile cont_loop *cont;



/* callback functions */

/* setpoint change */
```

```
void setpt_change( GtkAdjustment *adj, char *flag );
/* proportional gain change */
void prop_change( GtkAdjustment *adj, char *flag);
/* integral time change */
void integral_change( GtkAdjustment *adj, char *flag);
/* derivative time change */
void deriv_change( GtkAdjustment *adj, char *flag);
/* change state */off( GtkWidget *widget, GtkLabel *label);
/* quit */
gint delete_event( GtkWidget *widget, GdkEvent *event, gpointer data );

/* initialisation functions */
int init_everything(void);
int gtk_setup(void);
float valuefr;

int dec=255;



/* the main program */
int main( int argc, char *argv[] )
{
        /* set up everything first */
        init_everything();
        gtk_setup();

        /* then `sit' in gtk main */
        gtk_main();

      /* reset control loop */
      reset_controller();

        /* deallocate shared memory */
        mbuff_free("pid_control",(void *)cont);
        return 0;
}



/* Setpoint Change Callback */
void setpt_change( GtkAdjustment *adj, char *flag )


{
        cont->setpoint = adj->value;
```

```
        printf("%f \n",cont->setpoint);
        valuefr = (dec/5*cont->setpoint);
        value1 = abs(valuefr);
        printf(" %d \n",value1);

}


/* Prop. Gain Callback */
void prop_change( GtkAdjustment *adj, char *flag)
{
        cont->k = adj->value;
        printf("%f \n",cont->k);
        valuefr = (dec/5*cont->k);
        value1 = abs(valuefr);
        printf("%d \n",value1);


}


/* Integral Time Callback */
void integral_change( GtkAdjustment *adj, char *flag)
{
        cont->ti = adj->value;
        printf("%f \n",cont->ti);
        valuefr = (dec/5*cont->ti);
        value1 = abs(valuefr);
        printf("%d \n",value1);


}

/* Deriv. Gain Callback */
void deriv_change( GtkAdjustment *adj, char *flag)
{
        cont->td = adj->value;
        printf("%f \n",cont->td);
        valuefr = (dec/5*cont->td);
        value1 = abs(valuefr);
        printf("%d \n",value1);

}


/* On/Off Callback */
```

```
void state_onoff( GtkWidget *widget, GtkLabel *label)
{
      // if not on, then turn on
        if (cont->state == 0) {
                cont->state = 1;
                gtk_label_set_text(label, "ON");
        }
      // if on, then turn off and reset loop
        else {
                cont->state = 0;
                reset_controller();
                gtk_label_set_text(label, "OFF");
        }
}


/* This callback quits the program */
int delete_event( GtkWidget *widget, GdkEvent  *event, gpointer   data )
{
        cont->state = 0;
        cont->quit = 1;
        gtk_main_quit ();
        return(FALSE);
}


/* initialise everything */
int init_everything(void)
{
cont = value1;
        /* setup shared memory */
        value1 = (volatile int*) mbuff_alloc("lab1",1024);
        cont = (volatile cont_loop*) mbuff_alloc("pid_control",sizeof(cont_loop));
        if (cont == NULL) {
                printf("Shared Memory Allocation Failed!\n");
                return -1;
        }

        /* initialise control loop */
        init_controller(1);
        set_pid_params(0,100,0);

        /* initialise gtk */
        gtk_init(0,0);
        return 0;
}
```

```
int gtk_setup(void)
{
        GtkWidget *window;
        GtkWidget *setpt_scale, *prop_scale, *deriv_scale, *integral_scale;
        GtkWidget *table;
        GtkWidget *button;
        GtkObject *setpt, *prop, *deriv, *integral;
        GtkWidget *label, *clabel;

        /* setup shared memory */
        value1 = (volatile cont_loop*) mbuff_alloc("pid_control",sizeof(cont_loop));
        cont = (volatile cont_loop*) mbuff_alloc("pid_control",sizeof(cont_loop));
        if (cont == NULL) {
                printf("Shared Memory Allocation Failed!\n");
                return -1;
        }

        /* Create a new window */
        window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
        gtk_widget_set_usize (window,450,350);

        /* Set the window title */
        gtk_window_set_title (GTK_WINDOW (window), "Lab4 - Closed Loop PID
Control");

        /* Set a handler for delete_event that immediately exits GTK. */
        gtk_signal_connect (GTK_OBJECT (window), "delete_event",
GTK_SIGNAL_FUNC (delete_event), NULL);

        /* Sets the border width of the window. */
        gtk_container_set_border_width (GTK_CONTAINER (window), 20);

        /* Create a 5x4 table */
        table = gtk_table_new (2, 2, TRUE);
        gtk_table_set_row_spacings (GTK_TABLE (table), 55);
        gtk_table_set_col_spacings (GTK_TABLE (table), 65);

        /* Put the table in the main window */
        gtk_container_add (GTK_CONTAINER (window), table);

        clabel = gtk_label_new("OFF");
        gtk_label_set_justify(GTK_LABEL(clabel),GTK_JUSTIFY_LEFT);
        gtk_table_attach_defaults (GTK_TABLE(table), clabel, 2, 4, 2, 4);
        gtk_widget_show (clabel);
```

```
        /* create adjustments and scale widget */
        setpt = gtk_adjustment_new(0,0,5,0.1,0.1,0);
        prop = gtk_adjustment_new(0,0,5,0.1,0.1,0);
        deriv = gtk_adjustment_new(0,0,5,0.1,0.1,0);
        integral = gtk_adjustment_new(0,0,5,0.1,0.1,0);

        setpt_scale = gtk_hscale_new(GTK_ADJUSTMENT(setpt));
        prop_scale = gtk_hscale_new(GTK_ADJUSTMENT(prop));
        deriv_scale = gtk_hscale_new(GTK_ADJUSTMENT(deriv));
        integral_scale = gtk_hscale_new(GTK_ADJUSTMENT(integral));

        gtk_scale_set_digits(GTK_SCALE(setpt_scale),2);
        gtk_signal_connect( GTK_OBJECT(setpt), "value_changed",
GTK_SIGNAL_FUNC(setpt_change), NULL);
        gtk_scale_set_digits(GTK_SCALE(prop_scale),2);
        gtk_signal_connect( GTK_OBJECT(prop), "value_changed",
GTK_SIGNAL_FUNC(prop_change), NULL);
        gtk_scale_set_digits(GTK_SCALE(deriv_scale),2);
        gtk_signal_connect( GTK_OBJECT(deriv), "value_changed",
GTK_SIGNAL_FUNC(deriv_change), NULL);
        gtk_scale_set_digits(GTK_SCALE(integral_scale),2);
        gtk_signal_connect( GTK_OBJECT(integral), "value_changed",
GTK_SIGNAL_FUNC(integral_change), NULL);

        gtk_range_set_update_policy(GTK_RANGE(setpt_scale),
GTK_UPDATE_DISCONTINUOUS);
        gtk_range_set_update_policy(GTK_RANGE(prop_scale),
GTK_UPDATE_DISCONTINUOUS);
        gtk_range_set_update_policy(GTK_RANGE(deriv_scale),
GTK_UPDATE_DISCONTINUOUS);
        gtk_range_set_update_policy(GTK_RANGE(integral_scale),
GTK_UPDATE_DISCONTINUOUS);

        /* Create label for setpt */
        label = gtk_label_new("Setpoint");
        gtk_label_set_justify(GTK_LABEL(label),GTK_JUSTIFY_LEFT);
        gtk_table_attach_defaults (GTK_TABLE(table), label, 1, 3, 3, 4);
        gtk_widget_show (label);

        /* label for prop gain */
        label = gtk_label_new("P");
        gtk_label_set_justify(GTK_LABEL(label),GTK_JUSTIFY_LEFT);
        gtk_table_attach_defaults (GTK_TABLE(table), label, 0, 4, 0, 1);
        gtk_widget_show (label);
```

```
        /* label for deriv gain */
        label = gtk_label_new("D");
        gtk_label_set_justify(GTK_LABEL(label),GTK_JUSTIFY_LEFT);
        gtk_table_attach_defaults (GTK_TABLE(table), label, 0, 4, 2, 3);
        gtk_widget_show (label);

        /* label for int time */
        label = gtk_label_new("I");
        gtk_label_set_justify(GTK_LABEL(label),GTK_JUSTIFY_LEFT);
        gtk_table_attach_defaults (GTK_TABLE(table), label, 0, 4, 1, 2);
        gtk_widget_show (label);

/* attach things to table */
        gtk_table_attach_defaults (GTK_TABLE(table), setpt_scale, 0, 2, 0, 1);
        gtk_widget_show (setpt_scale);
        gtk_table_attach_defaults (GTK_TABLE(table), prop_scale, 0, 2, 1, 2);
        gtk_widget_show (prop_scale);
        gtk_table_attach_defaults (GTK_TABLE(table), deriv_scale, 0, 2, 3, 4);
        gtk_widget_show (deriv_scale);
        gtk_table_attach_defaults (GTK_TABLE(table), integral_scale, 0, 2, 2, 3);
        gtk_widget_show (integral_scale);

       /* Create start-stop buttons */
        button = gtk_button_new_with_label ("ON/OFF");
        gtk_signal_connect (GTK_OBJECT (button), "clicked",
GTK_SIGNAL_FUNC (state_onoff), GTK_LABEL(clabel));
        gtk_table_attach_defaults (GTK_TABLE(table), button, 2, 3, 1, 3);
        gtk_widget_show (button);

        /* Create "Quit" button */
        button = gtk_button_new_with_label ("Quit");
        gtk_signal_connect (GTK_OBJECT (button), "clicked",
GTK_SIGNAL_FUNC (delete_event), NULL);
        gtk_table_attach_defaults (GTK_TABLE(table), button, 1, 3, 4, 5);
        gtk_widget_show (button);


        /* show table and window */
        mbuff_free("lab1",(void*)value1);
        gtk_widget_show (table);
        gtk_widget_show (window);

        return 0;
}
```

C. PLD program

```
TITLE   GATE01
PATTERN A
REVISION 1.0
COMPANY MIDAS
AUTHOR   TOMMY
DATE    2006-04-15
CHIP    GATE PAL16V8

; PIN Declarations

PIN 2          A   ; INPUT
PIN 3    B    ; INPUT
PIN 4    C    ; INPUT
PIN 5    D    ; INPUT
PIN 6          E    ; INPUT
PIN 7    F    ; INPUT
PIN 8    G    ; INPUT
PIN 9    H    ; INPUT


PIN 19   Y0     ; OUTPUT
PIN 18   Y1     ; OUTPUT
PIN 17   Y2     ; OUTPUT
PIN 16         Y3     ; OUTPUT
PIN 15   Y4     ; OUTPUT
;
; Boolean Equation
EQUATIONS

Y1 =  (A * /B * /C * /D) *  E
```

Y2 = (/A * B * /C * /D) * F

Y3 = (/A * /B * C * /D) * G

Y4 = (/A * /B * /C * D) * H

Y0 = /(Y1 + Y2 + Y3 + Y4)

;

D. Specifications of AX5411

A/D Subsystem

Number of inputs: 16 single-ended

Resolution:12 bits

Sampling Rate: 60KHz max.

A/D Conversion Time:15us max.

Channel Acquisition Time:5us max.

System Accuracy: ±0.03% FSR Input Ranges: 10V, 5V, 2.5V, 1.25V, 0.625V, 0.3125V, All ranges software selectable

Output Coding: Offset binary

Maximum Input Without Damage

Power On:30V

Power Off:45V

Input Impedance

Off Channel:100 megohms, 10pF

On Channel: >10 megohms, 50pF

Nonlinearity:!1 LSB

Differential Nonlinearity:1 LSB

Inherent Quantizing Error:1 LSB

Zero Drift:

Bipolar: 17ppm of FSR/C

Gain Drift:30ppm of FSR/C

Monotonicity: Monotonic 0-70C

D/A Subsystem

Bias Current:100nA

Number of channels:2

Output Ranges:0 to 5V, 0 to 10V

Input data coding: Straight binary

Current Output, Voltage range: +5mA max.

Protection: Short circuit to Common for voltage ranges,
current outputs are short circuit, and reverse polarity protected

E. AX5411 Device driver

```
// ax5411.c
//      Implementation of various das 16 functions
//
#include "ax5411.h"

// init()
//      Initialise the AX5411 card
//
void init(void)
{
        // get permission to use I/O device (in non-RT)
        // only compile if used in non-real-time
#ifndef __RTL__
        ioperm(BASE, 16, 1);
#endif __RTL__

        /* reset control and status registers */
        outb(0, CONTROL);
        outb(0, STATUS);
```

```
}

// ax5411()
//      This function is used to either read a A/D value (should only do
//      this when you received an interrupt) or write a D/A value to a
specified
//      channel (There are 2 write and 16 read channels we can use for
the card)
//      The format of the function is as follows:
//       inout: specify whether the operation is a read or a write
//              'a' = write
//              'm' = read
//       channel: specify which channel to read or write
//            (use channel 0 for both read and write)
//       value: what value to write to the DAS 16 card (only significant

//              if you are performing a write operation)
//      The function returns a value which is only significant if you are
//      performing a read operation, or you try to write an invalid value to
//      to DAS 16 card
int ax5411(char inout, int chan, int value)
{
      int ch;
        int ilo, ihi;
        int dataL, dataH;

        // User wants to do a write operation
        //
        if (inout == 'a') {
            // check make sure the value we are writing to the DAS 16
            // card is valid
            //
            if ( (value > 4095) || (value < 0)) { return (value); }


// Split the value into a lower 4 bits, and higher 8 bits
                dataL = (value << 4) & 0x00F0;
                dataH = (value >> 4) & 0x00FF;

                // write our lower 4 bits to the D/A register
                outb( dataL, (BASE+4+chan) );

                // write our higher 8 bits to the D/A register
                outb( dataH, (BASE+5+chan) );
        }
        // User wants to perform a read operation
```

```
        //
        else if ( (inout == 'm') && (value == 0) ) {
              // mask out the higher 4 bits
              ch = chan & 0x000F;

                ch = ch + (ch << 4);

                // Select our channel by writing our value to the MUX
                //  ==> we start and finish on the same channel
                //
                outb( ch, (BASE+2) );

                // clear to A/D register 1st
                outb( 0, BASE);

                // wait until the A/D conversion is complete (shouldn't
                // be necessary
                //
                while (inb(BASE+8) & 0x80) {};

                // read our least significant 4 bits
                ilo = (inb(BASE) >> 4) & 0x000F;

                // read our most significant 8 bits
                ihi = (inb(BASE+1) << 4) & 0x0FF0;

                // combine our results and return the value
                value = (ihi | ilo);
        }
        return (value);
}


// ax5411.h
//     header files and defintions of various functions
//

#include <sys/io.h>

#define  BASE        0x320 /* base address of ax5411 */
#define  STATUS       BASE+8 /* status for ax5411  */
#define  CONTROL      BASE+9 /* control for ax5411 */


////////////////////////////////////////////////////////////////////////
// Here are the function defined in the file
```

```
//


// init(void)
//        function to initialise the AX5411 card
//     call within Linux task - NOT RT-Linux
void init(void);

// ax5411()
//     read or write some values to the das16 card
int ax5411(char inout, int channel, int value);
```