I hereby declare that all the material incorporated into this thesis is my own original unaided work except where specific reference is made by name or in the form of a numbered reference. The work contained herein has not been submitted for a degree at any other university.

Signed : _____

M. L Walker

Abstract

As South Africa's electricity consumption increases, Eskom is promoting Demand Side Management (DSM) to aid control of both the electricity consumption and its more effective usage, thereby delaying the need to construct new power stations, which pose large economic and environmental problems. Eskom has investigated various DSM strategies, such as load shifting, co-generation, alternative fuels and energy efficient processes, and has targeted the areas of load shifting and energy efficiency as prime areas for energy savings as they are relatively inexpensive and easy to implement. Pumps and fans form a large part of the industrial load. By improving the power usage of these devices with the use of variable speed drives, large energy savings may be achieved.

To enable the energy usage of industrial loads to be evaluated, a test bed system which enables a variable speed drive to be loaded with a configurable load, was constructed. The test bed system forms a tool for evaluating and demonstrating the energy savings that are possible, by replacing fixed speed drives with variable speed drives when controlling the flow rate of pumps and fans. Results from the test bed system show that by using variable speed operation of pumps and fans, some energy savings are achievable when compared to existing methods of flow control. The achievable energy savings are dependent on the system properties and the duty cycle of the pump or fan system.

Acknowledgments

The Work presented in this thesis was carried out under the supervision of Mr G. Diana of the Department of Electrical Engineering, University of Natal, Durban. I wish to thank Mr G. Diana for his encouragement and constant support throughout my studies.

I wish to thank the following:

My family for encouragement during my studies

The postgraduate students of the department of Electrical and Electronic Engineering and especially Bruce Van Blerk, Cedric Worthmann, Adam Stylo, Ryan Uys and Magash Pillay for providing encouragement and a stimulating work environment.

The technical support staff of the Department of Electrical Engineering, University of Natal for aid in the construction of the test bed system, especially Mr G Loubser, Mr AG Munnik, Mr A Roos, Mr A Stengel

Dr DA Hoch for his encouragement during the write up of the thesis

The Foundation for Research and Development (FRD), Eskom TESP and the University of Natal who provided financial support.

Siemens for the donation of the DC motor used in the test bed system

ARMSCOR for the donation of the REFU drive used in the test bed system

TABLE OF CONTENTS

CHAPTER 3 CONFIGURABLE LOAD SIMULATIONS AND LOAD CURVE CALCULATIONS

CHAPTER 4 TEST BED CONSTRUCTION AND INTERFACING

List of Symbols and Abbreviations

Commonly used symbols and abbreviations used in this thesis are listed below

Abbreviations

A/D	Analog to Digital Converter
AC	Alternating Current
ASIC	Application Specific Integrated Circuit
D/A	Digital to Analog Converter
DC	Direct Current
DLL	Dynamic Linker Library
DSM	Demand Side Management
DSP	Digital Signal Processor
EMF	Electromotive force
EMI	Electromagnetic Interference
FOC	Field Oriented Control
IED	Integrated Electricity Policy
IGBT	Insulated Gate Bipolar Transistor
PC	Personal Computer
PWM	Pulse Width Modulation
RIDE	Real Time Integrated Development Environment
RFI	Radio Frequency Interference
VSD	Variable Speed Drive
VSI	Voltage Source Inverter

Symbols

A_d	Cross Sectional area of Duct
A_p	Cross Sectional area of Pipe
B	Viscous friction
D,d	Impeller Diameter of centrifugal pump or fan
dphi	Incremental angle for PWM ASIC
e_a	Instantaneous back EMF of DC motor
E_a	Back EMF of DC motor
E_p	Potential Energy
E_k	Kinetic Energy
e_n	Current error signal
e_{n-1}	Previous error signal
F_{loss}	Friction loss of liquid flow in a pipe
f_{2n}	Slip frequency
f_{nm}	Nominal supply frequency
g	Constant of gravity acceleration
h	Height
H	Head of a pump
H_d	Static discharge head
H_f	Friction head
H_s	Static suction head
H_T	Total system head
H_v	Velocity head
i_a	Instantaneous armature current
I_a	Armature current
I_{amax}	Maximum armature current of DC motor
I_{avg}	Average current
I_{max}	Maximum current from REFU drive
i_a	Instantaneous A phase stator current

i_b	Instantaneous B phase stator current
i_c	Instantaneous C phase stator current
i_{ds}	Instantaneous d-axis stator current
i_{qs}	Instantaneous q-axis stator current
i_{dr}	Instantaneous d-axis rotor current
i_{qr}	Instantaneous q-axis rotor current
I_r	Current flowing from three phase rectifier
i_s	Instantaneous stator current
I_s	Stator current
I_{sf}	Full load stator current
I_{ds}	d-axis component of stator current (RMS)
I_{qs}	q-axis component of stator current (RMS)
J	Inertia
K	Constant
K_{fd}	Pressure loss constant for a fan
K_{fq}	Flow rate constant for a fan
K_i	Integral gain of a PI controller
K_m	DC motor armature constant
K_p	Proportional gain of a PI controller
L_a	DC motor armature inductance
L_l	Equivalent length of pipe
L_m	Stator and rotor mutual inductances
L_{22}	Rotor self inductances
L_{11}	Stator self inductances
L_{tb}	Total Losses in the test bed system
n	Rotational speed
n_f	Fan rotational speed
n_p	Pump rotational speed
n_{nm}	Nominal speed of induction motor
O_{max}	Maximum output of PI controller

O_n	Current output of PI controller
O_{n-1}	Previous output of PI controller
P	Power (in W)
P_f	Fan Pressure
phi	Internal phase angle of PWM ASIC
P_a	Power generated in DC motor armature
P_{dc}	Power generated by DC drive and returned to DC link
P_{mech}	Mechanical Power
P_s	Power supplied by 3 phase mains
P_{out_vsd}	Output power from VSD (from the induction motor)
P_{vsi}	Power supplied to VSI
Q	Capacity or flow rate of a pump or fan per unit time
R	Equivalent resistance of a fan system
R_a	Armature resistance
rho	Displacement angle in αβ reference frame
R_s	Stator resistance
R_r	Rotor resistance
s	Slip
S_f	Scaling factor for REFU drive
T	Torque
T	Switching period
T_E	DC motor electrical torque produced
T_{em}	Shaft torque produced
T_i	Integrator time constant of a PI controller
T_L	Load torque
T_M	Total torque able to accelerate armature
T_{IL}	Load torque generated by DC motor (load placed on VSD)
T_o	Time upper switch remains on in PWM ASIC
T_s	Sampling period
T_u	Time lower switch remains on in PWM ASIC

U	Magnitude of voltage required from PWM ASIC
U_a	Magnitude of α voltage
U_b	Magnitude of β voltage
U_n	Pole voltage required for Phase n (where n=1,2,3)
U_{pf}	Magnetizing current set point voltage for REFU drive
U_{pd}	Torque limit set point voltage for REFU drive
U_{ps}	Slip frequency set point for REFU drive
V_a	DC motor armature voltage
V_{air}	Velocity of Air
V_{amax}	Maximum armature voltage of DC motor
v_{ds}	d-axis stator voltage
v_{qs}	q-axis stator voltage
v_{qr}	q-axis rotor voltage
v_{dr}	d-axis rotor voltage
V_b	DC motor brush volt drop
V_l	Liquid velocity
V_{link}	DC link voltage of VSD
W_l	Mass of liquid lifted
ΔP	Pressure loss in fan system
φ	DC motor flux
λ_{ds}	d-axis stator flux linkage
λ_{qs}	q-axis stator flux linkage
λ_{dr}	d-axis rotor flux linkage
λ_{qr}	q-axis rotor flux linkage
ω_o	Synchronous angular frequency
ω_r	Angular rotational speed

Chapter 1

Introduction

1.1 General

Population and industrial growth in South Africa over the past few years, has resulted in an increase in electricity consumption [11], as shown in Fig. 1.1, with a consequent reduction of Eskom's (South Africa's only energy utility) spare generation capacity [1]. The high cost and environmental impact of constructing further power stations has forced Eskom to set out plans in the form of an Integrated Electricity Plan [12], with the aim of reducing electricity consumption via the implementation of Demand Side Management strategies such as load shifting, co-generation, alternative fuels and energy efficient processes [2]. In light of this, the Motion Control Research Group at University of Natal is conducting research on behalf of Eskom, focussing on the application of Variable Speed Drives (VSDs) to assist energy savings in industrial applications.

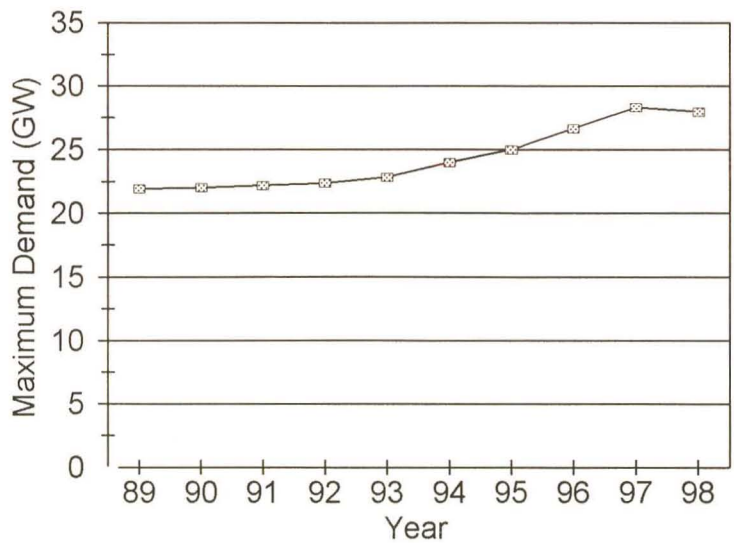


Fig. 1.1 Maximum demand from Eskom's supply 1989 to 1998

Centrifugal pumps and fans are widely used in industry with constant speed electric motor drives [34]. Flow control of centrifugal pumps and fans driven by constant speed electric drives has traditionally been accomplished by methods involving a considerable expenditure of energy (throttling) as a trade-off for having flow control [34]. It has been widely publicized that some energy savings may be achieved by replacing fixed speed drives in pump or fan systems with VSDs which can vary the rotational speed of the pump or fan, thereby controlling the flow rate [13, 34, 37, 44, 40]. Although VSDs have been widely used in pump and fan systems, the primary reason for using the VSD was to improve the performance and operation of the pump or fan system [6, 37, 41, 51]. VSD's have not traditionally been used to achieve energy savings due to their high installation costs and the expertise required to commission the VSD. The development of new switching devices and their increased use over the past few years [10] has resulted in a reduction in the cost of VSDs (mainly AC VSDs). Due to escalating energy costs and decreasing cost of VSDs, the use of VSDs in pump or fan systems in order to reduce the energy consumption of the system can now be considered as a primary reason for installation [5].

The energy savings that a VSD can provide depends primarily on the physical properties and size of the system as well as the associated duty cycle of the process. Both the process and mechanical aspects of the pump or fan system have to be considered when evaluating the possible energy savings [34]. It is important that an energy savings evaluation be performed on every VSD application under consideration prior to purchasing and installing the VSD to ensure an acceptable return of investment [34]. The possible energy savings in specific systems has shown that the return of investment on the purchased VSD can be as little as three years [13, 37]. In spite of the achievable energy savings when using VSDs and their short payback period there is a reluctance to apply them due to reliability problems, lack of trained service personnel, spares holding and susceptibility to quality of supply phenomenon [37]. Down time caused by a faulted VSD (whether due to a fault in the drive or a disturbance on the mains) results in a loss of production which can often exceed the energy savings provided by the VSD thereby increasing the return of investment period [37]. By improving the energy efficiency of centrifugal pump and fan systems with the use of VSDs, this saved energy can be used by other Eskom customers (electricity users) without Eskom having to construct further power stations.

In this study a 40kW test bed system was developed to enable the power usage of a VSD driving a simulated pump or fan load to be determined. As it is important to consider every VSD application to ensure a suitable return on capital expenditure [34], the test bed system can be used to simulate a pump or fan system from which the achievable energy savings may be calculated without retrofitting an installation.

The test bed system comprises three main components namely the VSD (AC VSD), the configurable load and the test bed controller, as shown in Fig. 1.2. The AC VSD used in the test bed system is a commercially available unit manufactured by REFU [33]. The industrial load is simulated by the configurable load, which is a DC drive whose output torque is controlled by regulating the armature current. The power generated by the configurable load while loading the VSD is fed back to the DC link of the VSD (as will be discussed in Chapter 2), ensuring that the test bed system is efficient and draws only the power required to replace the losses in the test bed system from the mains supply. The test bed controller controls the configurable load such that the desired industrial load characteristic is simulated while the power usage of the VSD is measured.

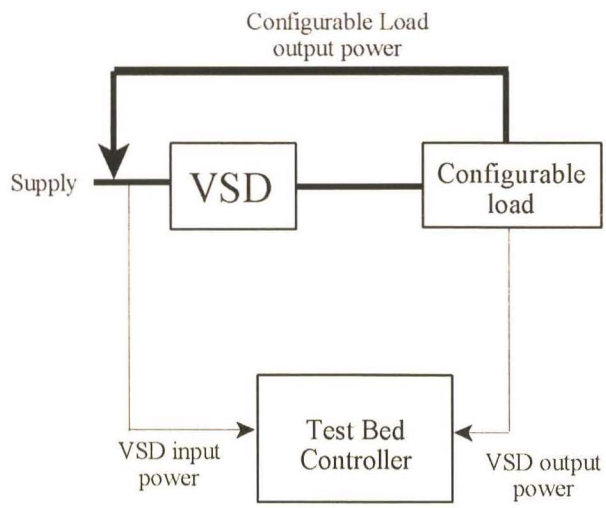


Fig. 1.2 Block diagram of test bed system

The test bed system enables pump and fan systems to be simulated, thereby demonstrating the power usage of the VSD from which energy savings may be determined based on the duty cycle of the process. Although there are many types, sizes and characteristics of pumps and fans, this study is limited to centrifugal pumps [26] and centrifugal fans [31] as they are most commonly used. The test bed system is used to demonstrate the difference in power usage that can be achieved by replacing conventional flow control methods in a centrifugal pump and centrifugal fan systems with that of variable speed operation. Two pump systems and one fan system are used to demonstrate the achievable energy savings based on simulated results from the test bed system. These results show that it is important to consider the entire system as not all pump or fan systems will provide sufficient energy savings to guarantee a suitable return of investment.

The project was divided into three phases: the first phase involved determining and understanding the characteristics and operations of centrifugal pumps and fans and the operation of the VSD used in the test bed system; the second phase involved the design, construction and commissioning of the test bed system, based on the information gained from the first phase; the third phase involved simulating pump and fan loads on the test bed system in order to determine the energy savings achievable when using variable speed to control the flow rate in pump and fan systems.

1.2 Thesis Outline

The thesis is structured as follows :

Chapter 2 presents the relevant information required to understand the operation of pumps and fans. The operation of a Field Oriented Controlled (FOC) Voltage Source Inverter (VSI) drive is presented as this theory is required when commissioning the AC VSD used in the test bed system. The theory of a separately excited DC motor is presented such that a model of the DC motor can be determined. The model of the DC motor is required for the design and simulation of a suitable current controller which will enable the DC drive to act as a configurable load. The

strategy used to determine the efficiency of the VSD in the test bed system is also discussed.

Chapter 3 presents two theoretical pump and one theoretical fan test systems. These theoretical systems are used to determine the load that will be placed on the VSD in the test bed system while driving the respective pump or fan system at fixed or variable speeds. A digital armature current controller is developed and simulated which enables the DC drive to operate as a configurable load thereby simulating the pump or fan load.

Chapter 4 presents the design and construction of the test bed system. This includes the design of the DC drive power electronic converter and interfacing, the measurement, filtering and interfacing of the test bed system variables required for the control and efficiency measurement of the VSD. The design and construction of a PWM/tachometer card, which together with a DSP card can control AC and DC drives, is presented.

Chapter 5 describes the implementation of the test bed controller. It provides an overview of the software development system used to implement the test bed controller. The operation of the test bed controller is described in two stages, the first being the means in which the controller controls the configurable load in order to simulate the pump of fan system. The second stage is the means in which the controller measures the power drawn by the VSD in the test bed system.

Chapter 6 describes the commissioning and testing of the test bed system. The operation of the DC drive and armature current controller is compared to the simulation results of Chapter 3 to determine if the DC motor and controller are operating correctly. The method used to setup and evaluate the test bed controller is discussed.

Chapter 7 presents the practical results captured from the test bed system. The torque speed curves for the theoretical pump and fan systems, discussed in Chapter 3, are simulated on the test bed system and the results of the simulations run on the test bed system are presented. Energy savings are then calculated based on the results gained from the test bed system.

Chapter 8 draws conclusions about the energy savings that can be gained when using VSDs to control the output of pumps and fans, and provides suggestions for further work.

1.3 Contributions

The work presented in this thesis makes the following contributions:

(1) The Test bed developed to complete this work, may be used to investigate the operation of VSDs under varying load conditions and fault conditions experienced in industry.

(2) The work performed, practically demonstrates that energy savings may be achieved by using VSDs to control the flow rate of pumps and fans.

1.4 Research Publications

Certain material presented in this thesis has been presented at an international conference ([48]) and at two national conferences ([49],[50]).

Chapter 2

Theory

2.1 Introduction

Fixed speed induction motors are commonly used to drive pumps and fans due to their low cost, high reliability and low maintenance. With fixed speed pumps or fans the flow rate (output of the pump or fan) is commonly regulated using throttling valves in the case of pumps or dampers in the case of fans [34]. The flow rate of pumps and fans may also be controlled by varying the speed of the pump or fan [34], in which case some energy savings are possible. The primary purpose for developing a test bed system is to be able to demonstrate the energy savings that are achievable by controlling the flow rate of pumps or fans by varying their speed.

The evaluation of the energy usage of pumps or fans being run at fixed or variable speeds is problematic as the assessment requires the use of a pump or fan and an existing system in which the pump or fan is operated. The test bed system overcomes this because it may be used to simulate a pump or fan running at fixed speed and the same pump or fan being operated at variable speed without the need for making changes to a physical system.

A simplified block diagram of the test bed system to be developed is shown in Fig. 2.1. The VSD in the test bed system consists of two components namely an induction motor and a Field Oriented Controlled (FOC) Voltage Source Inverter (VSI), both of which are commercially available components. To measure the energy usage of the VSD in the test bed system while driving a pump or fan, a simulated pump or fan characteristic needs to be presented to the VSD in the form of a configurable load. The configurable load consists of a DC motor, H-Bridge inverter and controller which controls the torque produced by the DC motor and measures the power usage of the VSD.

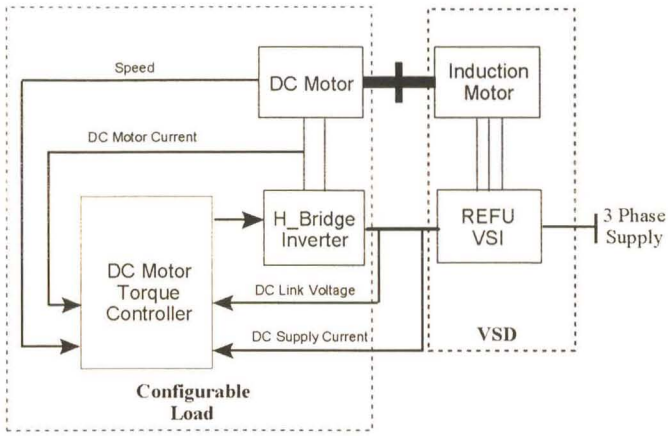


Fig. 2.1 Simplified block diagram of the test bed system

In order to be able to simulate a pump or fan one must first gain an understanding of their operating principles. In the next section the operational characteristics of centrifugal pumps and fans under both fixed and variable speed operation are presented to enable their characteristic curves to be determined for variable flow rates. In order to implement a configurable load the operating principles of the DC motor employed must first be understood, from which a suitable control method for the DC motor may be implemented. The principle of FOC is discussed, paying particular attention to the various parameters that are required for commissioning a FOC drive. Finally the method used to determine the power usage and efficiency of the VSD is discussed.

2.2 Pumps

The operational principle of a pump and how its operation is affected by the system in which it operates has to be understood prior to determining the characteristic curves required to simulate a pump while being driven at fixed or variable speeds.

The primary function of any pump is to move liquid from one point to another. The type of pump used depends on the flow rate required, the distance and height over which the liquid needs to be moved, the viscosity of the liquid and its abrasiveness. This study only considers the centrifugal pump, as it is the most commonly used pump in industry [26].

The operation of a centrifugal pump in a specific system is determined by both the pump and system characteristics. The pump characteristics are supplied by the pump manufacturer in the form of pump curves. The system characteristics are dependent on the type of piping used to transport the liquid, the height the liquid is to be lifted and the flow rate of the liquid. The pump and system curves are overlaid and the intersection of the two curves determines the operating point of the pump in the specific system.

2.2.1 Centrifugal Pump Operation

Excluding the prime mover, a centrifugal pump consists of two main components, namely a rotating and a stationary element. A centrifugal pump operates as follows: the liquid is forced into the rotating vanes at their centre, as shown in Fig. 2.2 and Fig. 2.3, by either atmospheric pressure or by the suction caused by the liquid exiting the pump. These vanes constitute an impeller which discharges the liquid at its periphery at a higher velocity which is then converted to a pressure with the aid of a volute or a diffuser as shown in Fig. 2.2 and Fig. 2.3 respectively. Centrifugal pumps may therefore be classified as either a volute pump, which converts the velocity to a pressure by means of a volute as shown in Fig. 2.2, or a diffuser pump, which converts the velocity to a pressure using a diffuser as shown in Fig. 2.3. The volute type pump is typical of large industrial pumps whereas diffuser type pumps are more typical of smaller pumps.

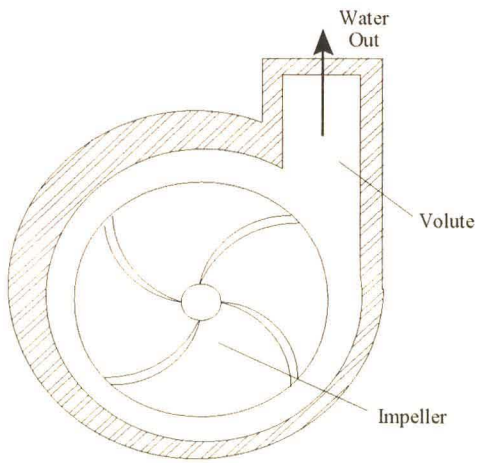


Fig. 2.2 Volute type pump [26].

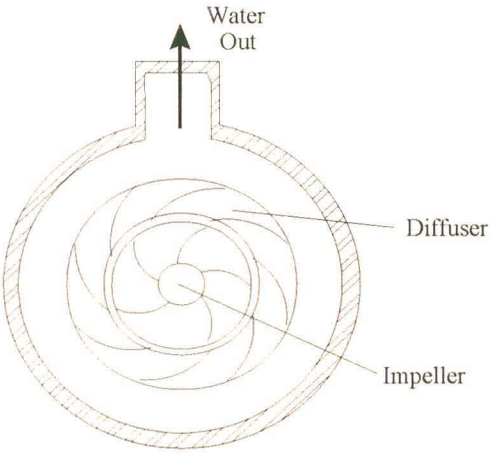


Fig. 2.3 Diffuser type pump [26].

Centrifugal pumps may be designed to meet a wide variety of needs. Therefore, each pump type has different characteristics. The most important characteristics of a pump are:

Capacity Q : is the volume of liquid (m^3/h or m^3/s) delivered by the pump per unit time (described as the flow rate).

Head H : (in metres) represents the net work done by the pump in lifting a unit of liquid from the input of the pump to the discharge point. The system head comprises three main components, the *static or elevation* head which is the vertical height over which the liquid is raised from the inlet or suction point to the outlet or discharge point and remains fixed for any system. The *friction* head is the resistance the liquid encounters in flowing through the piping and fittings and is dependent on the flow rate. The *velocity* head which represents the energy contained in the moving liquid.

Power P : is the input or shaft power required by the pump to move the liquid from the inlet point to the exit point and is related to the capacity, head and the specific gravity of the liquid being moved in the system.

Efficiency : of the pump is the net power used in moving the liquid divided by the power input to the pump shaft.

The capacity, head, power and efficiency of a centrifugal pump are affected by the impeller type, diameter, construction and housing and the impeller speed. The type and size of the impeller, construction and housing are fixed and the operating speed of the pump is normally selected according to the type of impeller used and corresponds to the point where the pump is most efficient. The direction of rotation of a centrifugal pump is important as the suction and discharge points are affected by this, therefore they are designed to turn only in one direction especially in the case of the diffuser type pumps.

The achievable flow rate for a specific pump, working against a specific head may be determined from its characteristic curve, which is normally available from the pump manufacturer. Fig. 2.4 shows the pump curve for a centrifugal pump with a 210mm diameter impeller manufactured by *Rapid Allwailer Pump and Engineering Co.*, and is used throughout the remainder of this study. This pump was chosen for two main reasons. The first is that the power that it draws from the prime mover is 32kW (this may differ slightly for different systems) when the pump is delivering the maximum flow rate at full speed, which is close to the full load power of the test bed system. The second is that it is supplied to industry by a local supplier [3].

The pump curve shown in Fig. 2.4 shows the achievable flow rate for a specified head for various speeds. Lines of constant power and efficiency are also shown. The lines of constant power allow the user to determine the power consumed by the pump at a particular operating point, and the efficiency by the line of constant efficiency.

The physical laws governing the operation of centrifugal pumps are known as the affinity laws [19], which relate the pump head, capacity, speed and power at one operating point to that of another as defined by Eqs 2.1 to 2.3. Subscripts 1 and 2 represent the two operating points. The affinity laws allow the characteristic curve at one operating point to be translated to that of another operating point, which is particularly useful when determining pump characteristic curves for speeds other than that displayed on the pump characteristic curve.

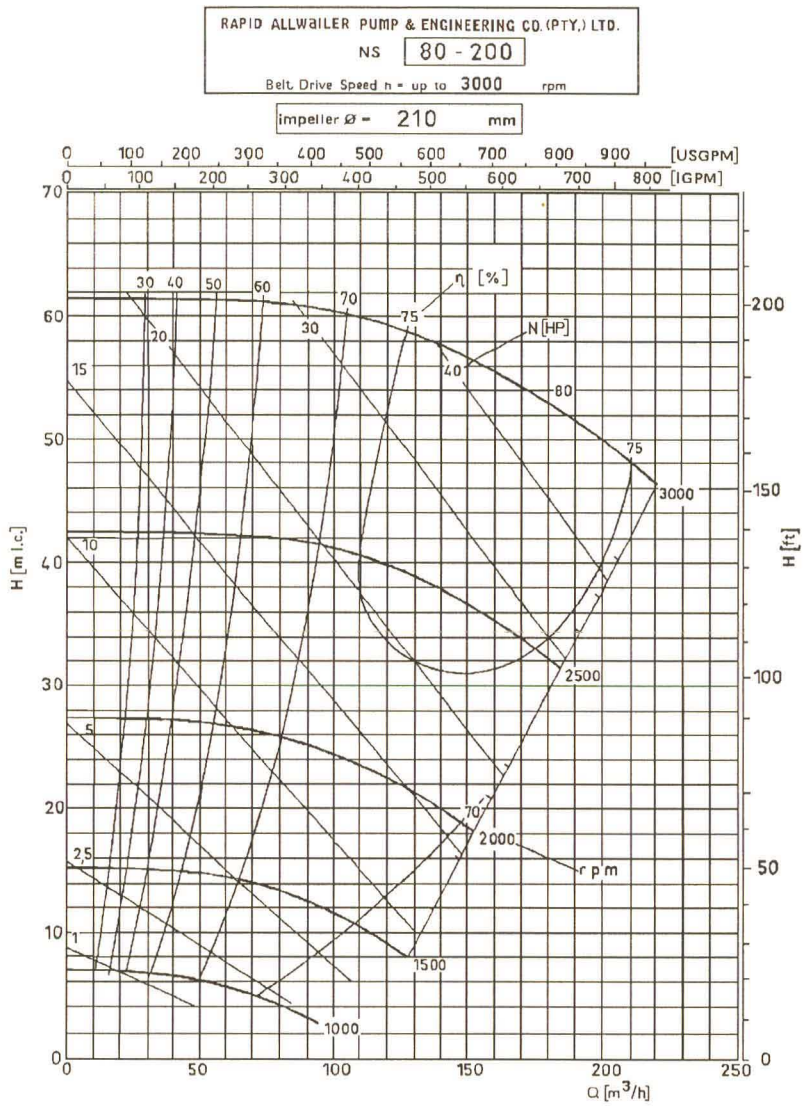


Fig. 2.4 Pump characteristic curve showing pump efficiency and power as the head and flow rate vary [3].

$$Q_2 = Q_1 \frac{n_2}{n_1} \tag{2.1}$$

$$H_2 = H_1 \left(\frac{n_2}{n_1} \right)^2 = H_1 \left(\frac{Q_2}{Q_1} \right)^2 \tag{2.2}$$

$$P_2 = P_1\left(\frac{n_2}{n_1}\right)^3 \tag{2.3}$$

where n is the pump speed

Q is the pump flow rate

H is the pump head

P is the pump input power.

2.2.2 System Characteristic Curve Calculations

The head a pump experiences when moving liquid, at a specified flow rate, in a specific system is known as the system characteristic and may be calculated from the physical properties of the system (pipe diameters, number of pipe fittings, elevation head, inlet and outlet construction). The system characteristic curve is obtained by calculating the system head for various flow rates.

The total system head comprises the total suction head and the total discharge head. The suction head and discharge head each have three main components: the static head which is the vertical distance from the source of the liquid to the pump's inlet for the suction lift (H_s) and from the outlet of the pump to the discharge point for the static discharge head (H_d), as shown in Fig. 2.5; the velocity head which represents the energy contained in the moving liquid; the friction head which represents the force required to overcome the interaction of the liquid with the walls of the piping at a specific flow rate [22].

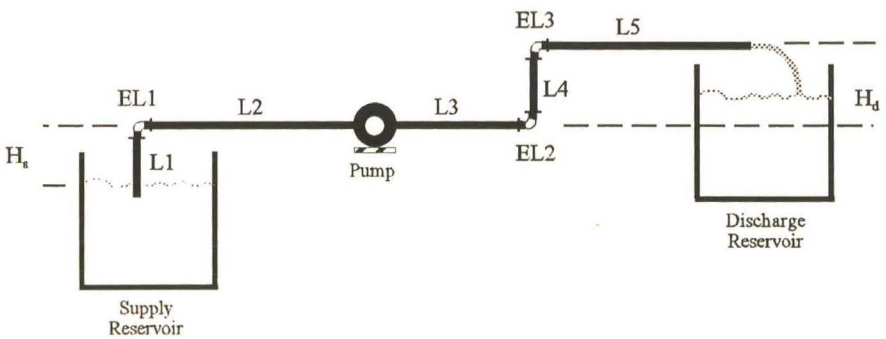


Fig. 2.5 Pump system showing system parameters required to calculate the system head.

The total system head (H_T) can be calculated using Eq. 2.4.

$$H_T = H_s + H_d + H_f + H_v \tag{2.4}$$

where H_T is the total system head.

H_s is the suction static head

H_d is the discharge static head

H_f is the friction head due to the liquid flowing in the pipes

H_v is the velocity head.

The friction head (H_f) is calculated using Eq. 2.5.

$$H_f = L_l . F_{loss} \tag{2.5}$$

where L_l is the equivalent length of pipe

F_{loss} is the frictional loss for a specific flow rate in a specific pipe, and is found experimentally and given in tables [22].

The equivalent length of pipe (L_l) represents the length of straight pipe through which the liquid would have to flow to produce the same frictional force as the system places on the liquid. The equivalent length of straight pipe is found by measuring the total length of the pipe in the system (L1+L2+L3+L4 from Fig. 2.5) then adding an additional length (EL1+EL2+EL3 from Fig. 2.5), which represents the length of pipe which would produce the same effect when the liquid encounters a pipe fitting (such as flanges and elbows). The equivalent lengths of pipe for the various types of pipe fittings may be determined experimentally, the results of which are given in tables (Table C.1 Appendix C) [22].

The velocity head (H_v) is found using Eq. 2.6, but is often omitted as it is negligible in respect to the static ($H_s + H_d$) and friction head (H_f) of the system.

$$H_v = \left(\frac{V_l^2}{2g} \right) \tag{2.6}$$

where V_l is the liquid velocity in the pipe

g is the force of gravity.

Since both the friction (H_f) and velocity (H_v) heads are dependent on the flow rate in the system, the total system head has to be calculated for each flow rate to gain the system characteristic curve. The system characteristic curve represents the net work the pump has to perform to produce a specific flow rate.

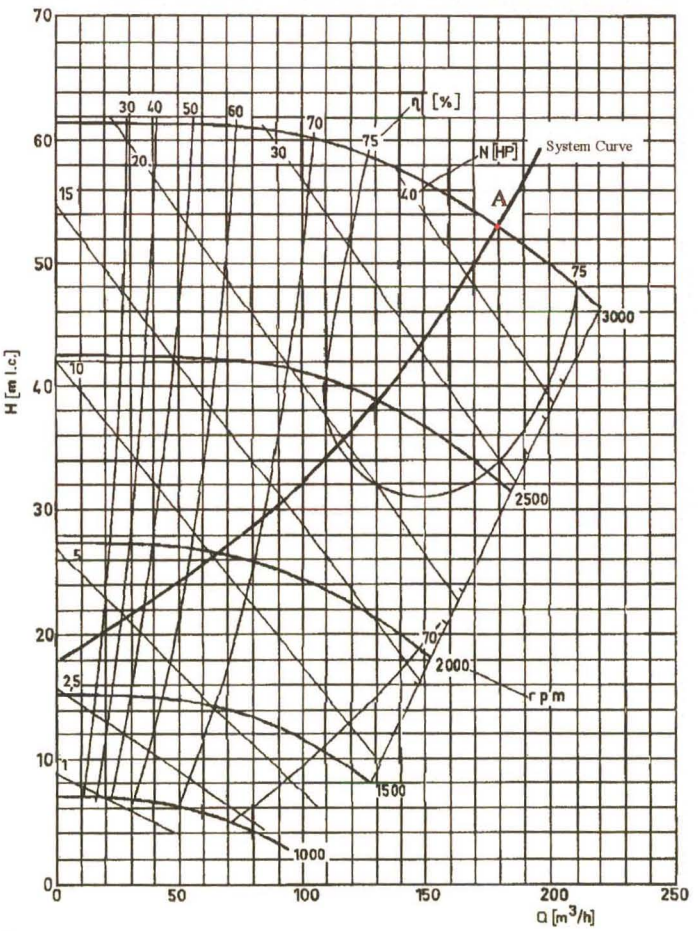


Fig. 2.6 Pump and system characteristic curve

Once the system characteristic curve has been determined, it is overlaid on the pump curve to select or evaluate a pump in a system as shown in Fig. 2.6. The point where the pump and system characteristic curves intersect is the operating point of the pump in the system and indicates the expected flow rate, efficiency and power usage of the pump. Fig. 2.6 shows a typical pump and system characteristic curve and indicates that the pump running at 3000 rpm will operate at point A where the flow rate is 180 m^3/h and the pump efficiency is 80%.

2.2.3 Capacity regulation of centrifugal pumps

Capacity regulation of centrifugal pumps is often achieved by changing the system head, pump speed or both. Capacity regulation of pumps is an important factor when pumps are used in automated production systems where a variable flow rate is required. There are many different forms of capacity regulation which can be applied to centrifugal pumps of which the most common forms are discussed below. The two most common forms of capacity regulation namely discharge throttling and speed regulation [34, 41] are discussed, as well as other methods of capacity regulation for the sake of completeness.

- **Discharge throttling**

  This is one of the cheapest and most commonly used methods of flow control, which is achieved by placing a valve in the discharge line [26]. Fig. 2.7 shows a pump curve and a system head curve where the throttling valve is completely open (operating point A) and thus not impeding the flow of liquid. Closing the throttling valve (operating point B) modifies the system resistance by introducing an additional head as shown by the dotted line in Fig. 2.7. The static head of the system remains constant as this is determined by the height that the liquid is to be lifted. The new flow rate of the pump system now occurs at the intersection of the pump curve and modified system curve (dotted Line). The required input power to the pump will drop off slightly due to the decreased flow rate. The pump will generally operate in a region of decreased efficiency as it works against a larger head.

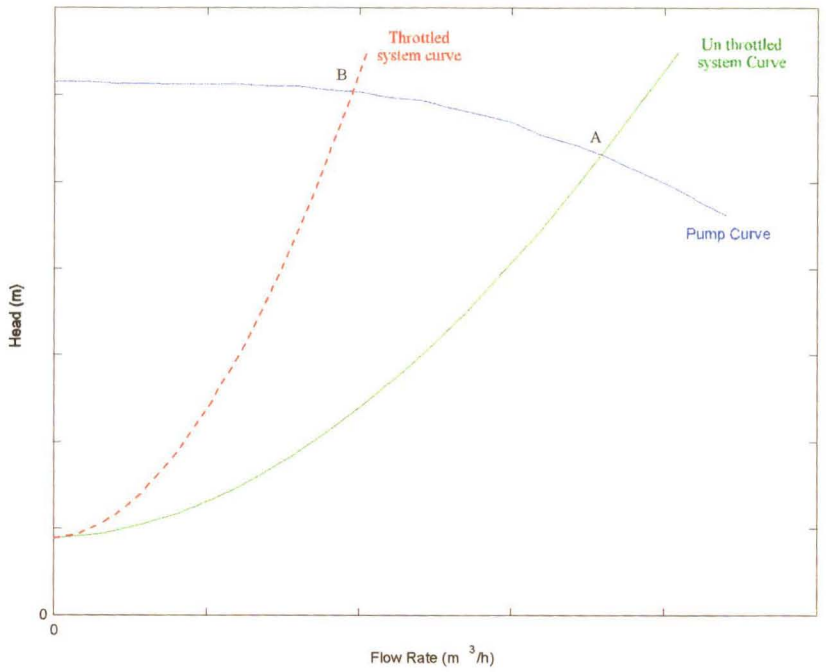


Fig. 2.7 Effect of throttling valve on system curve and flow rate

- **Speed Regulation**

  This form of capacity regulation has not been used very frequently in the past, due to the high cost and complexity of the equipment required to vary the speed of the pump as is the case of DC drives, slip recovery drives (Kramer drive) and eddy current drives [40]. The advent of AC induction motor inverter drives and their diminishing costs has seen this form of capacity regulation become more feasible. When using this form of capacity regulation, it is possible to minimize power requirements, and eliminate overheating (losses in the system) during capacity regulation.

  Varying the speed of a pump leads to an infinite number of pump head versus flow rate curves being available for controlling the flow rate of a pump. When the pump speed is reduced, the pump characteristic curve shrinks in accordance with the affinity laws (Eq. 2.1 and 2.2) as shown in Fig. 2.8. The flow rate achieved as the speed is reduced follows the system curve (in

a direction from point A to B). The system curve remains unchanged as it is dependent on the physical properties of the pump system and the flow rate. It must be noted that below a specified speed the pump ceases to pump liquid as the pump cannot produce the head required to overcome the static head in the system, as shown in Fig. 2.8. It should be noted that the affinity laws apply to the pump characteristic curve and not to the system as a whole, as the affinity laws state that at zero speed zero flow rate will occur, but Fig. 2.8 shows that this is not the case. The actual head and flow requirements are determined by the system curve and the pump curve for a specified speed [34].

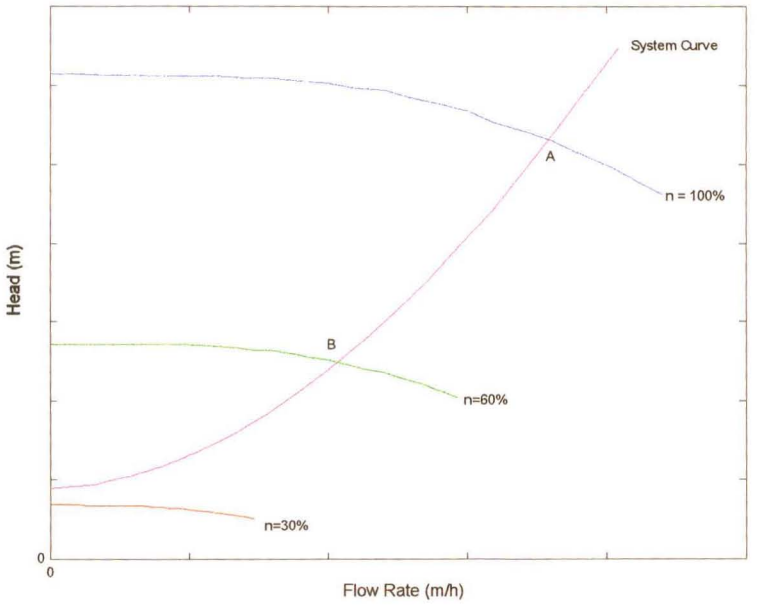


Fig. 2.8 Effect of speed on pump curve and flow rate

In accordance with the affinity laws (Eq. 2.3) the power consumed by the pump varies with the cube of the speed. If Fig. 2.8 is considered it can be seen that zero flow rate occurs at a point where the pump is running at 30% of its full speed. Thus, the power drawn by the pump at zero flow rate will not be zero due to the static head in the system. The possible energy savings achievable when using speed regulation is thus primarily dependent on the system [34].

- **On off control**

  This is the simplest and cheapest form of flow control used and is generally used where small or fairly steady flow rates are required. However, as the flow rates become larger or more intermittent, the starting and stopping of pumps causes uneven flow and disturbances. This method also places stress on all the related components during the starting and stopping periods. When induction motors are used, large currents are drawn during the start up period, which can result in a high power consumption and premature motor failure.

- **Suction throttling**

  If sufficient net positive suction head is available, the flow rate can be controlled by throttling the suction line. The impellers employed when using suction throttling, have to be considered carefully to avoid cavitation.

- **Bypass regulation**

  In this process some or all of the pump's capacity is diverted by the use of a bypass line from the discharge line of the pump to the suction line or other suitable point. The bypass line may contain suitable control valves. In some cases considerable power savings can be made if excess capacity is bypassed instead of using discharge throttling.

- **Regulation by adjustable vanes**

  Adjustable guide vanes placed ahead of the impeller have been found effective in regulating the output of specific pumps. The vanes produce a positive prewhirl which reduces the head capacity and efficiency of the pump. Adjustable outlet diffusion vanes have been used with good success on several large storage pumps for hydroelectric developments. In general these methods are expensive and complicated and have limited application in practice [47].

- **Air Admission**

  When admitting air into the pump suction, the capacity of some pumps can be regulated, with some savings over discharge throttling. Having air in the pumped liquid is often found to be undesirable and there is always the concern of the pump losing prime (without water in the

pump the impeller rotates but is unable to generate suction).

2.3 Fans

The operational principle of a fan and how the fan's operation is affected by the system in which it operates has to be understood prior to determining the characteristic curves required to simulate a fan being driven at fixed and variable speeds. This section considers the operating characteristics of centrifugal fans and how the system affects their operation.

The majority of fans found in industry are centrifugal fans [31], and so in this study only this type of fan is investigated. The purpose of a fan is to move air continuously at moderate pressures. When working with fans it is normally assumed that air is incompressible, due to the moderate to low pressures involved with the movement of the air [31].

The operation of a centrifugal fan, in a specific system, is determined by both the fan and system characteristics. The fan characteristics are supplied by the fan manufacturer in the form of fan characteristic curves and are found as a result of tests performed on the fan under various conditions. The system characteristics are dependent on the type of ducting used to transport the air. The fan and system characteristic curves are overlaid and the intersection of the two characteristic curves determines the operating point of the fan.

2.3.1 Centrifugal Fan Operation

A centrifugal fan consists of an impeller running in a spiral casting as shown in Fig. 2.9. The air enters the impeller in an axial direction and is discharged at the periphery. The amount of work done by the fan on the air is dependent on the angle of the fan blades (pitch) with respect to the rotation of the impeller.

The pressure resulting from the fan moving the air is expressed in terms of a pressure rise through the unit as opposed to a ratio between the inlet and outlet pressures. The various pressures

resulting from the air movement are as follows:

Total pressure: is the difference between the total pressures at the fan outlet and inlet.

Velocity pressure: is the pressure due to the movement of the air in the outlet of the fan and is found by dividing the air flow rate Q by the area of the fan discharge orifice.

Static pressure: is the fan total pressure less the fan velocity pressure.

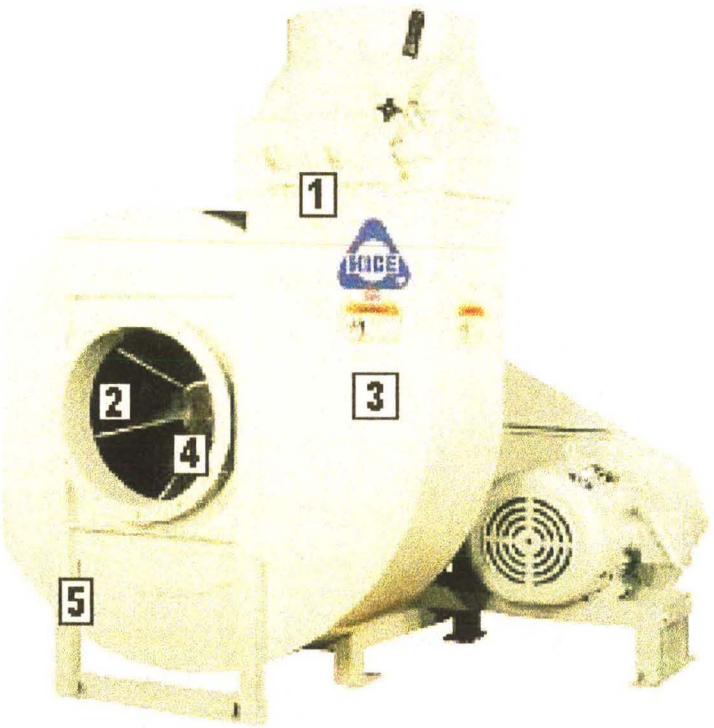


Fig. 2.9 Centrifugal fan showing, air outlet (1), impeller (2), spiral shaped casting (3), air inlet (4), framework (5) [8].

The performance of a fan, in terms of pressure, flow rate and power absorbed, depends on a number of factors [31]. The most important are:

- the design, type and size of the fan;
- the point of operation on the flow/pressure characteristic curve of the fan;
- the speed of rotation;
- the condition of the gas or air passing through the fan.

The achievable flow rates produced by fans working against specified pressures are displayed on charts known as fan characteristic curves. These fan characteristic curves are supplied by fan manufacturers and are based on tests carried out under ideal conditions. The characteristic curve for a *Donkin* fan with backward curved blades is shown in Fig. 2.10. This fan is used throughout the remainder of this study, in determining the energy usage of a VSD while driving a fan load. This fan was chosen as it can be operated at speeds up to 2000rpm where it will draw 36kW from the prime mover. The speed range and power consumption of the fan thus falls into the operating speed range of the test bed. This fan is also supplied to industry by a local fan manufacturer [8].

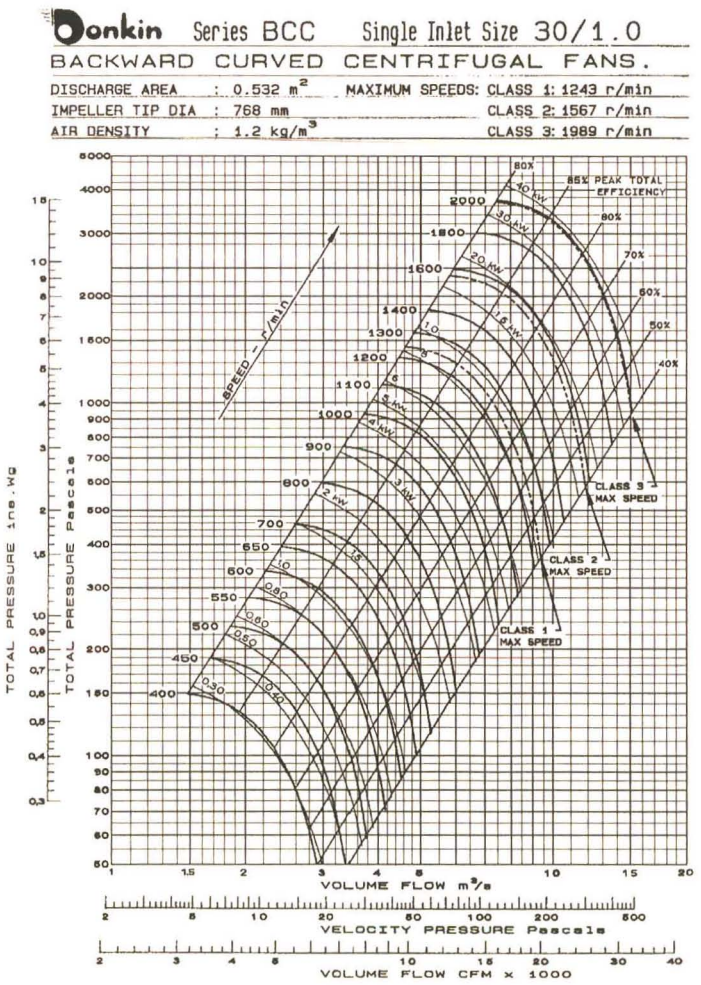


Fig 2.10 Fan characteristic curve showing efficiency and power as the flow rate and pressure drop vary [8].

The fan characteristic curve, shown in Fig. 2.10, shows the flow rate achievable for a specified pressure loss at specified fan speeds. Lines of constant power are also shown and allow the user to determine the power consumed by the fan at a particular operating point. The efficiency of the fan is shown in the form of constant efficiency lines.

The flow versus pressure relationship cannot be expressed in a simplified form as in the case of the affinity laws for centrifugal pumps. This makes the computation of various operating points difficult although, by considering any single point on the fan characteristic curve, it is possible to derive some simple relationship known as the fan law. The fan laws are given in Eqs 2.7 to 2.9, where Eq. 2.7 represents the flow rate and Eq. 2.8 represents the fan pressure as the fan speed is varied [31]. Once the constants in Eqs 2.7 and 2.8 have been found for an operating point at a specified speed, an operating point at a different speed can then be determined.

$$Q = K_{fq} d^3 n \tag{2.7}$$

$$P_f = K_{fp} d^2 n^2 \rho \tag{2.8}$$

where Q is the fan flow rate

n is the fan speed

d is the impeller diameter

P_f is the fan pressure

ρ is the air density

K_{fp} is the fan pressure loss coefficient

K_{fq} is the fan flow rate coefficient.

A third law relating the fan power to the flow rate and fan pressure is given by Eq. 2.9 and can be used to calculate the power consumed at a different speed according to the power consumed at a known operating speed.

$$P = pQ = K_{fp} d^5 n^3 \rho \tag{2.9}$$

Since the fan laws are valid for any particular point on the fan characteristic curve, similar laws will be valid for every other point of operation, with the only difference being the numerical value of the coefficients. It must be noted that these laws only apply to the fan characteristic curve and not to the fan system as a whole.

2.3.2 System Characteristic Curve Calculations

The performance of a fan in a system is determined by the system parameters as is the case with pumps. In a fan system the pressure loss varies linearly with the flow rate. An electrical resistance analogy (Ohm's Law) can be used in calculating the pressure loss of the system as the flow rate changes as shown in Eq. 2.10 [31]. Thus as the flow rate increases the pressure drop increases linearly, just as the voltage dropped across a resistance increases as the current flow increases.

$$R = \frac{\Delta P}{Q^2} \tag{2.10}$$

where R is the equivalent resistance of the fan system

ΔP is the pressure loss of the system

Q is the volume flow rate of air in the system.

When the equivalent resistance of the fan system is known the pressure loss in the system can be calculated for each flow rate. The system pressure loss is plotted against the flow rate to give the system characteristic curve. The system characteristic curve is overlaid on the fan characteristic curve when evaluating the performance or operation of a fan in a specific system, as shown in Fig. 2.11. The point of intersection between the fan and system curves determines the operating point of the fan in the system. Thus for the system and fan characteristic curves shown in Fig. 2.11 when the fan is running at 1600rpm the achievable flow rate is $8m^3/s$.

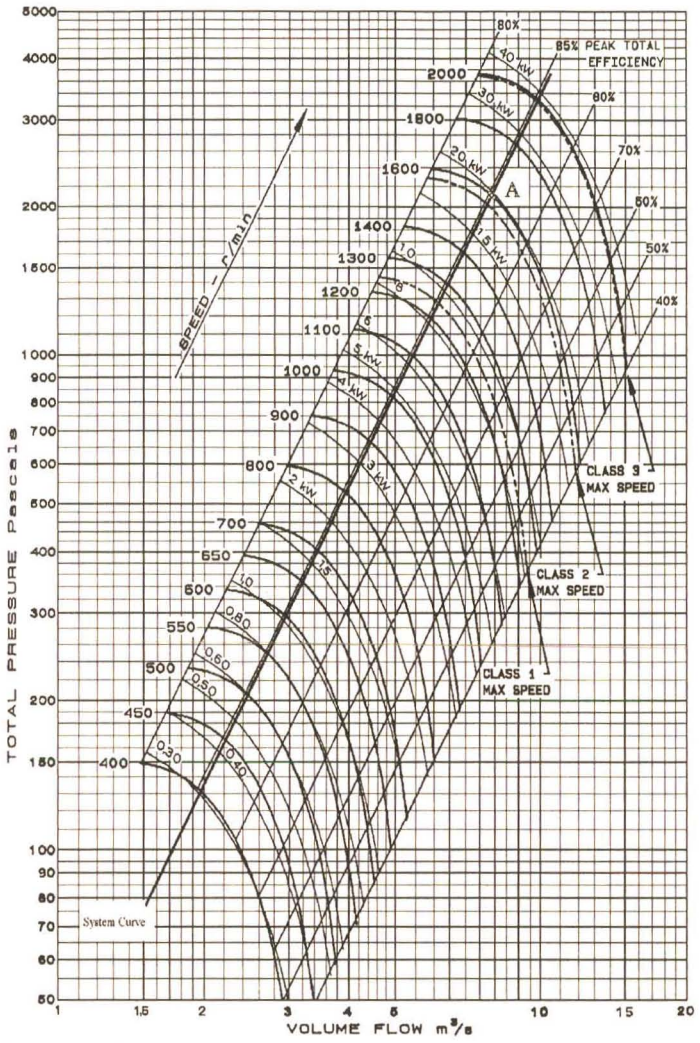


Fig. 2.11 Fan and system characteristic

2.3.3 Flow Control of Centrifugal Fans

In many industrial applications, it is necessary to alter the flow rate of a fan according to the system needs, thus flow control of fans is an important factor. There are a number of methods for controlling the flow rate of centrifugal fans. The methods in which dampers and the fan speed is used to control the flow rate are considered in more depth; the remaining methods are described for the sake of completeness.

- **Throttling by means of a Damper**

The simplest form of throttling is by means of an adjustable damper. The effect of the damper is to increase the system resistance. Closing the damper increases the system resistance resulting in a change in the system curve, as shown by the dotted red line in Fig. 2.12. The flow rate now occurs at point B.

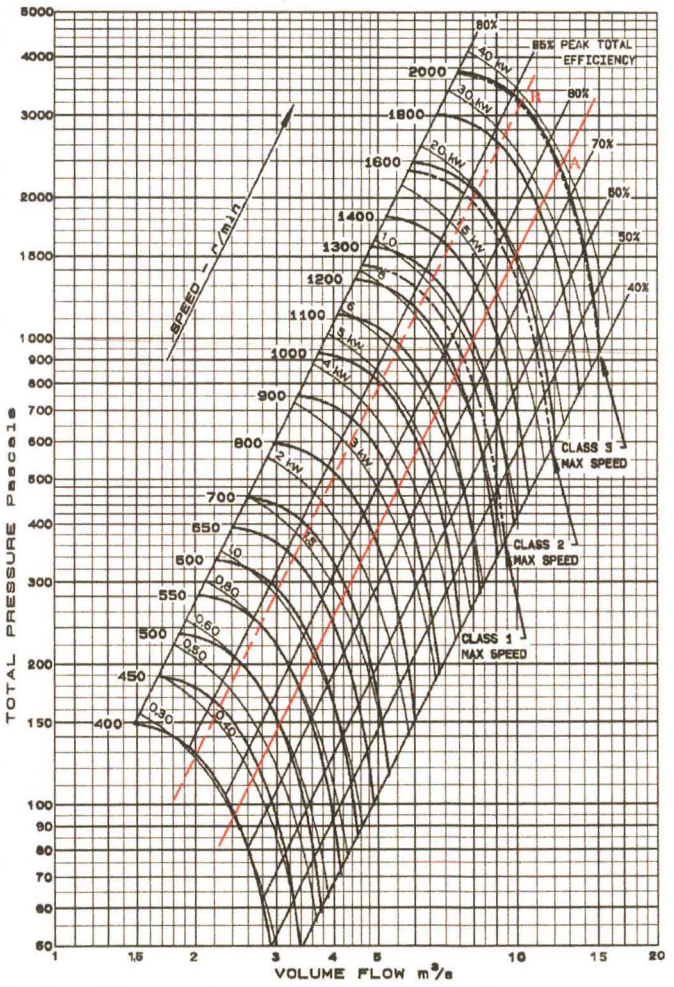


Fig. 2.12 Effect of damper setting on system characteristic curve and flow rate

The installation costs of this form of flow control are low in comparison with other forms of flow control [31]. The power drawn by the fan is almost constant irrespective of the flow rate, as shown by the power curves in Fig. 2.12 which run parallel to the fan curves. Therefore the running costs of a system employing dampers to control the flow rates are high due to the inefficiencies involved, especially where a wide range of flow rates are required [31, 34].

- **Speed Regulation**

  This is the most efficient and controllable means of flow control in fans [31]. Previous methods of fan speed control such as DC motor drives or variable slip drives incorporating wound rotor induction motors and variable rotor resistance (which are inefficient) have been expensive. AC VSDs provide a more efficient method of controlling the fan speed and thus the flow rate. The effect of a change in speed on the fan characteristic curve is shown in Fig. 2.13. When the fan speed is reduced the fan characteristic curve reduces in accordance with Eqs 2.7 and 2.8. The system curve remains unchanged as the system remains fixed as the fan speed is varied. The operating point of a fan in a system occurs at the intersection of the fan and system characteristic curves (point A), thus when the fan speed is changed the new operating point in the system occurs at the intersection of the new fan and system characteristic curves (point B) as shown in Fig. 2.13.

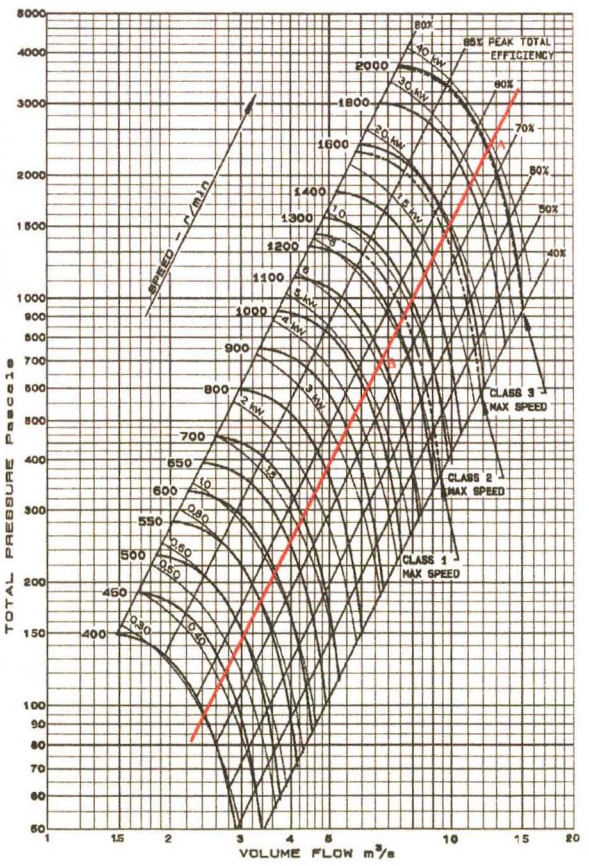


Fig. 2.13 Effect of changing fan speed on system operating point

As the system curve and the lines of constant efficiency are parallel the efficiency of a fan remains constant, irrespective of the fan speed. This can be of importance if the fan is selected to operate at its maximum efficiency in the system. The power consumed by the fan varies as the fan speed is altered and changes in accordance with Eq. 2.9. Thus as the fan speed is lowered to reduce the flow rate the power consumed by the fan decreases.

- **Flow Control using Adjustable Vanes**

  This method uses adjustable vanes placed at the inlet of the fan, to circulate the input air in the same direction as the impeller motion, this has the effect of reducing the work done on the air by the impeller. Thus, for each vane setting the fan has a different characteristic, which results in the fan having a different operating point resulting in an alternate flow rate.

2.4 DC Motor Theory

The load placed on a AC VSD while driving a pump or fan is determined by the point at which the pump or fan characteristic curve and the system characteristic curves intersect, as described in the previous sections. For the power usage of the VSD in the test bed system to be measured while driving a pump or fan, the pump or fan may be simulated with the use of a configurable load. A separately excited DC motor was chosen to form the configurable load for the test bed system for three main reasons. The first is the ability of the DC motor to produce a constant torque throughout the desired speed range, the second is the ease in which the DC motor torque can be controlled and the third is the ease with which the losses in the DC motor can be calculated.

A good model for a system is essential to ensure that simulated results are meaningful. The greater the accuracy of the model the better the predicted results will be to the real system. This section presents the theory required to setup a simple yet accurate model of a DC motor.

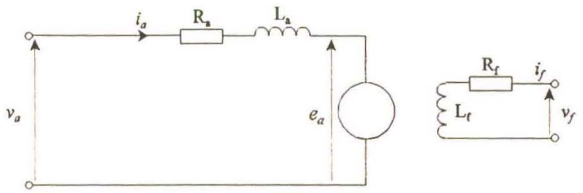


Fig. 2.14 Equivalent circuit diagram of a separately excited DC motor

The equivalent circuit diagram for a separately excited DC motor is shown in Fig 2.14. Eq. 2.11 is developed by summing the voltages around the armature circuit.

$$v_a = e_a + i_a R_a + L_a \frac{di_a}{dt} \tag{2.11}$$

where v_a is the applied armature voltage (V)

e_a the back EMF of the armature given by Eq. 2.12 (V)

i_a is the resultant armature current (A)

R_a is the armature resistance (Ω)

L_a is the armature inductance (H).

The back EMF of the DC motor armature is given by Eq. 2.12.

$$e_a = K\phi\omega_r \tag{2.12}$$

where K is the DC motor constant

φ is the flux inside the DC motor (wb)

ω_r is the rotation speed of the armature (rad.s).

If the field flux is held constant and the effects of armature reaction are neglected then Kφ can be expressed as K_m resulting in Eq. 2.13

$$e_a = K_m\omega_r \tag{2.13}$$

The electrical torque produced by the DC motor (T_E) is given in Eq. 2.14, showing that the total torque produced by the machine is proportional to the armature current, provided K_m remains constant.

$$T_E = K_m i_a \tag{2.14}$$

The mechanical torque (T_M) available to accelerate the machine up to an equilibrium speed, is the difference between the electrical torque (T_E) and the load torque (T_L), where the load torque could be a constant value or a complex function varying with the motor speed.

$$T_M = T_E - T_L \tag{2.15}$$

The dynamics of the mechanical system of the DC motor can be described using Eq. 2.16

$$T_M = J\frac{d\omega_r}{dt} + B\omega_r \tag{2.16}$$

where J is the inertia of the armature ($kg.m^2$)

B is the coefficient of viscous friction for the DC motor (Nm/rad/sec).

The block diagram shown in Fig. 2.15 is obtained by transforming Eqs 2.11 to 2.16 into the s domain using the Laplace transform [27].

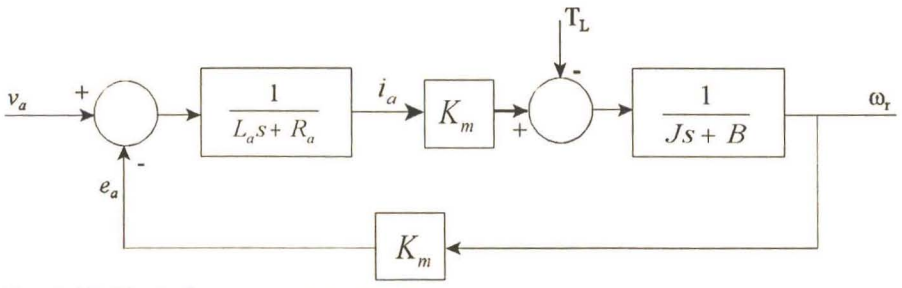


Fig. 2.15 Block diagram model of DC motor

The transfer function relating the armature current to the applied armature voltage is given by Eq. 2.17.

$$P(s) = \frac{i_a(s)}{v_a(s)} = \frac{\frac{1}{L_a J}(sJ + B)}{s^2 + s\left(\frac{R_a}{L_a} + \frac{B}{J}\right) + \left(\frac{K_m^2 + R_a B}{L_a J}\right)} \tag{2.17}$$

This model and transfer function is used in Chapter 3 for the design and simulation of an armature current controller.

2.5 Principles of Field Oriented Control

The AC VSD used in the test bed system is an FOC VSD. A clear understanding of field oriented control is required when commissioning the AC VSD. Industry has primarily relied on DC motors for motion control applications for two reasons, the first is the ease in which DC motors are controlled and the low cost of SCR based DC drives and the second is the reluctance of people to adopt AC drive technology [28]. Induction motors are smaller, cheaper and require less maintenance than DC motors of the same size, but have not been used extensively in motion control applications as it is difficult to predict and control the induction motor's performance. FOC enables the induction motor to exhibit similar characteristics to that of a separately excited DC motor [7, 20]. The objective of FOC is to establish and maintain an explicit angular relationship between the stator current and the rotor flux that is established in an induction motor [7, 25]. This angular relationship may be achieved by regulating the slip of the induction motor to a value which causes the rotor flux vector to align with the d-axis component of the stator current vector, as shown in Fig. 2.16 [7]. The magnetizing flux inside the induction motor is then produced by the d-axis stator current vector and the q-axis component of the stator current vector is used to regulate the torque produced by the induction motor.

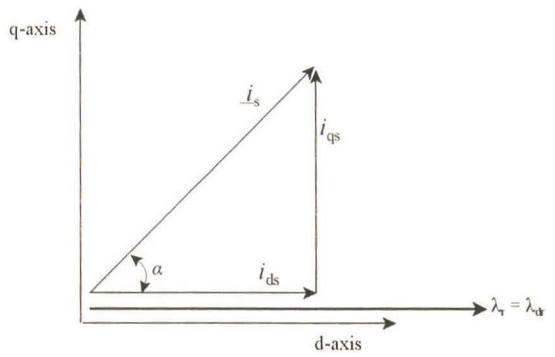


Fig. 2.16 Conditions for FOC

The block diagram of an induction motor operating under FOC is shown in Fig. 2.17 (the derivation of which is given in Appendix A). In Fig. 2.17 the d-axis component of the stator current sets up the magnetizing flux inside the induction motor and is held constant. The q-axis current component of the stator current is used to control the torque produced by the induction motor.

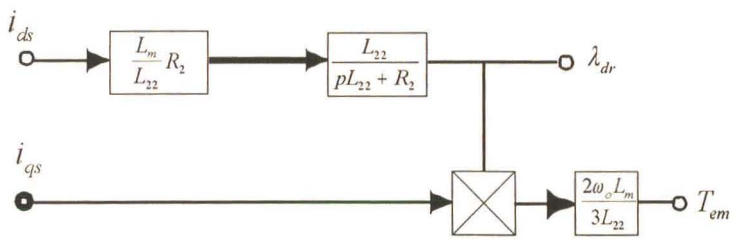


Fig. 2.17 Induction motor under FOC with stator current control.

The torque produced by the induction motor is described by Eq. 2.18. If Eq. 2.18 is compared to that of Eq.2.14 it can be seen that the induction motor will produce torque in a similar manner to that of the separately excited DC motor.

$$T_{em} = \lambda_{dr} i_{qs} K \tag{2.18}$$

where T_{em} is the motor shaft torque produced

λ_{dr} is the d-axis stator flux linkage (set by i_{dr})

i_{qs} is the q-axis stator current

K is a constant.

In order for an FOC drive to operate correctly the d-axis and q-axis current set points have to be known as well as the machine parameters. Since it is often difficult to determine the machine parameters accurately (especially if test equipment is unavailable) a parameter known as the slip frequency is used in commissioning the drive. The method used to determine these parameters are given below.

2.5.1 Determining I_{ds} and I_{qs} Parameters

In order to control the induction motor correctly, the correct d and q-axis stator current reference values have to be determined to commission the FOC drive. The d-axis stator current sets up the magnetic field in the induction motor and should equal the magnetic field found in the induction motor while running under normal operating conditions. The magnetizing current or d-axis current of the motor is found by running the motor at no load. When the induction motor is running at no load, the slip is almost zero, resulting in the rotor flux and the rotor d-axis flux lining up, as shown in Fig. 2.16, thus the motor operates under the same conditions as under FOC. The currents flowing into the machine only setup the flux (I_{qs}=0), thus $|I_{ds}| = |I_a| = |I_b|=|I_c|$ for a balanced three phase machine.

The q-axis current is used to control the torque produced inside the machine when it is running under FOC control. The q-axis current has to be set, such that the machine current will be limited to avoid overloading the induction motor if a large load torque is applied. The full load current of the machine (I_{sf}) is given on the name plate of the machine and since the magnetizing current can be measured, the maximum q-axis current can be calculated using Eq 2.19.

$$I_{qs} = \sqrt{I_{sf}^2 - I_{ds}^2} \qquad (2.19)$$

where I_{sf} is the induction motor full load current

I_{ds} is the d axis stator current

I_{qs} is the maximum q-axis stator current.

The method described above is used in Chapter 6 and Appendix H to calculate the required d and q-axis currents. These parameters are used in setting up and commissioning the FOC drive in the test bed system.

2.5.2 Determining the Slip Gain Parameter

One of the conditions that has to be met for an induction motor to be controlled under FOC is given in Eq. 2.20 and Eq. A.15 in Appendix A. Eq. 2.20 states that the slip frequency ($s\omega$) is a function of the motor parameters and the q-axis stator current reference value. The motor parameters remain almost constant, therefore if the slip frequency is known for a particular q-axis stator current then the motor parameters can be calculated (the slip at full load is given on the motor name plate). The slip frequency is used to calculate a parameter known as the slip gain which is setup on the FOC VSD used in the test bed system.

$$s\omega = \frac{L_m R_2 i^*_{qs}}{L_{22}\lambda_{dr}} \tag{2.20}$$

where s is the induction motor slip

ω is the IM rotation speed

L_m is the stator and rotor mutual inductance

R_2 is the rotor resistance

i^*_{qs} is the q-axis current reference

L_{22} is the rotor self inductance

λ_{dr} is the q-axis rotor flux linkage.

The slip frequency calculations are used in Chapter 6 and Appendix H when determining the slip gain for the AC VSD in the test bed system.

2.6 Power Usage Determination of the VSD in the Testbed System

In order for the power usage of the AC VSD in the test bed system to be measured while driving a simulated pump or fan load the power flow in the test bed system has to be measured. It is desirable to measure as few variables as possible, as it reduces the number of possible measurement errors. Fig. 2.18 shows a block diagram of the test bed system and the power flow in the system. The power flow in the test bed system has to be in steady state in order to determine the power usage of the AC VSD while driving the simulated pump or fan load. This can be achieved by ensuring that the torque reference supplied to the configurable load is not changed rapidly thereby reducing any large system disturbances.

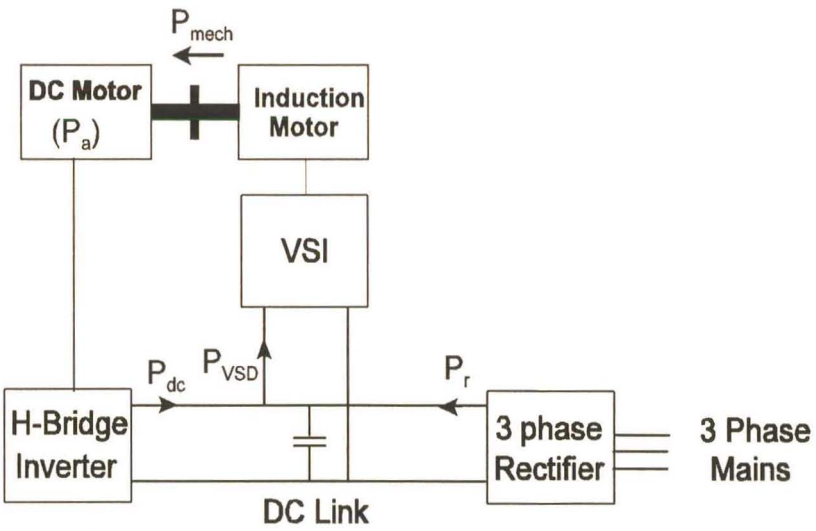


Fig. 2.18 Block diagram representing power flow in test bed system

In Fig. 2.18, the FOC VSI draws electrical power from the DC link (P_{VSD}) and supplies electrical power to the induction motor, which converts the electrical power into mechanical power (P_{mech}) in the form of torque and speed. The output torque of the induction motor is then converted back into electrical energy by the DC drive (DC motor an H-bridge inverter), which generates power back into the DC link (P_{dc}) thereby loading the VSD (IM and VSI). The AC VSD in the test bed

system contains a speed loop which will attempt to maintain the output speed of the IM at its set speed (as set by a reference to the speed control loop) as the mechanical load changes. Due to the speed loop of the AC VSD controlling the output torque produced by the IM, the IM will balance the torque produced by the DC drive, thereby maintaining a constant speed. The power flowing into the DC link of the test bed system ($P_r + P_{dc}$) has to equal the power flowing out of the DC link (P_{VSD}) in order for the voltage across the DC link capacitors to remain constant during steady state. As the AC VSD and the DC drive in the test bed system both have mechanical and electrical losses the power flowing from the three phase rectifier (P_r) replaces any mechanical and electrical losses in the test bed system thereby maintaining a constant DC link voltage. The test bed system is very efficient as it only draws the power lost (P_r) from the supply.

The power flowing to the AC VSD can be calculated from a knowledge that the power entering the DC link must equal that leaving, ensuring that the DC link voltage remains constant under steady state conditions, as shown by Eq. 2.21.

$$P_{VSD} = P_r + P_{dc} \tag{2.21}$$

The power flowing into the DC link from the three phase mains (P_r) is given by Eq. 2.22 and equals the total system losses, as explained above.

$$P_r = I_r . V_{link} \tag{2.22}$$

where I_r is the DC current flowing from the three phase rectifier of the AC VSD

V_{link} is the DC link voltage of the AC VSD.

The electrical power generated internally by the DC motor (P_a) is not a function of the DC link voltage, but is a function of the back EMF of the DC motor (E_a) and the armature current (I_a), as given by Eq. 2.23 ($E_a = e_a$ under steady state, as E_a is only defined during steady state).

$$P_a = I_a . E_a \tag{2.23}$$

The value of E_a can be calculated using Eq. 2.13, as the flux in the DC motor remains constant and the speed of the armature is known. The armature current ($I_a = i_a$) is controlled and is therefore known. The electrical power generated by the DC motor (P_a) is not equal to the power that flows back into the DC link due to the losses in the armature circuit. The power returned to the DC link (P_{dc}) is therefore equal to the electrical power generated by the DC motor armature less the electrical losses as shown in Eq. 2.24.

$$P_{dc} = P_a - (I_a^2 R_a + I_a V_b) \tag{2.24}$$

where V_b is the volt drop across the brushes in the DC motor

The output power of the induction motor is equal to the output power to the DC motor plus the electrical and mechanical losses of the DC motor. Thus, the output power of the VSD can be calculated using Eq. 2.25.

$$P_{mech} = P_{dc} + I_a^2 R_a + I_a V_b + B\omega_r^2 \tag{2.25}$$

Eq. 2.25 can be simplified by substituting Eq. 2.24 into Eq. 2.25 yielding Eq. 2.26.

$$P_{mech} = E_a I_a + B\omega_r^2 \tag{2.26}$$

The efficiency of the AC VSD can now be calculated using Eq. 2.27.

$$Efficiency(\%) = \frac{P_{mech}}{P_{VSD}} 100 \tag{2.27}$$

By substituting Eqs 2.26 and 2.21 into Eq. 2.27 the full expression for the VSD efficiency is determined and shown in Eq. 2.28.

$$Efficiency(\%) = \frac{E_a I_a + B\omega_r{}^2}{E_a I_a - (I_a^2 R_a + I_a V_b) + V_{link} I_r} \tag{2.28}$$

From the above equation it can be concluded that by measuring the DC motor armature current (I_a), the supply current from the three phase rectifier (I_r), the DC link voltage (V_{link}) and the rotational speed of the machines in the test bed (ω_r), the power usage and efficiency of the AC VSD can be calculated.

The load torque (T_{IL}) imposed on the induction motor under steady state conditions is given by Eq. 2.29.

$$T_{IL} = K_m I_a + B\omega_r \tag{2.29}$$

Thus the required armature current for a specified load at a particular speed is calculated using Eq. 2.30.

$$I_a = \frac{T_{IL} - B\omega_r}{K_m} \tag{2.30}$$

The DC motor used in the DC drive in this study has a viscous friction torque of 9.21Nm when running at 2000rpm. This was determined from tests performed on the DC motor given in Appendix B .This load torque decreases as the speed of the DC motor is reduced. Pumps and fans rarely run at extremely low flow rates and the power drawn by the pumps and fans never reaches zero [34]. Due to the type of control used in the DC drive, as discussed in Chapter 3, the armature current flowing in the DC motor has a high frequency ripple present, which results in the induction motor being subjected to a high frequency oscillatory torque with a peak of 10.16Nm. As the

viscous friction torque and the ripple torque produced by the DC motor are small, the viscous friction may be neglected and Eq. 2.30 can be further simplified yielding Eq. 2.31.

$$I_a = \frac{T_{IL}}{K_m} \tag{2.31}$$

The theory presented in this section is used in Chapter 5 to implement the test bed controller software which calculates the power flow in the test bed system. Although the test bed system is only required to measure the power usage of the AC VSD while driving a simulated load the AC VSDs efficiency is also calculated. The VSD efficiency is not presented in this thesis but is implemented to provided a system which may also be used to investigate how the efficiency of AC VSDs can be improved.

2.7 Conclusion

The pump and fan characteristic curves described in this chapter are used to determine torque speed curves for a number of pump and fan examples in Chapter 3. The DC motor theory is also used in Chapter 3 to perform simulations to determine the operating points of the configurable load. The theory on the FOC drive is used in Chapter 6 where the commissioning of the VSD is considered.

The power usage calculations for the VSD plays an important role in deciding which parameters should be measured when setting up the test bed system controller. This together with the implementation of the DC drive controller is discussed in Chapter 5.

Chapter 3

Configurable Load Simulations and Load Curve Calculations

3.1 Introduction

To demonstrate the power usage of different methods of flow control for pump and fan systems a test bed system was constructed. Fixed and variable speed pump and fan systems may be simulated on the test bed system and is achieved by controlling the speed set point of the AC VSD, which is determined by the flow rate and type of flow control being simulated. While the speed of the VSD is controlled it is loaded with a simulated pump or fan load. The load is simulated by the DC drive which is designed and controlled as a configurable load. The power drawn by the AC VSD in the test bed system is measured while driving the configurable load simulating a pump or fan.

For the DC motor to load the VSD with a simulated pump or fan load it must produce a constant set point torque (torque required to simulate a pump or fan) under steady state conditions. The torque of a DC motor may be controlled by varying the DC motor's armature current, therefore a current controller had to be designed to control the armature current in such a manner that the DC motor would represent the desired pump or fan load characteristic. The DC motor simulations are used to determine the controller gains as well as to investigate any system limitations and changes required. This chapter uses the DC motor model presented in Chapter 2, to simulate the operation of the DC motor and current controller when used in the test bed system.

Pump and fan characteristic curves have to be used to determine the torque that the configurable load must impose on the AC VSD to simulate a pump or fan. As the power usage of the VSDs installed in a system is dependent on the characteristics of the pump system, two test systems are

presented. The first pump system results in large power savings at low flow rates whereas the second system does not. Both pump systems are presented in which flow control is firstly achieved using discharge throttling and secondly by speed regulation. The theory presented on pumps in Chapter 2 and the two pump systems presented are used to determine the load placed on the AC VSD while driving the pump at either fixed speed (flow control using discharge throttling) or variable speed (flow control using regulation). A test case for a fan operating in a system, firstly using dampers (fixed speed operation) and secondly using speed regulation to control the flow rate is presented in order to determine the load placed on the AC VSD while driving the fan under different forms of flow control.

3.2 DC Motor Simulations

The model of the DC motor developed in Chapter 2 is used to design and simulate an armature current controller to control the torque produced by the DC motor in accordance with Eq. 2.14. The DC motor simulations are important, as they give an indication of the system performance, limitations and stability. The DC motor simulations were performed in four parts: the first part was to determine the parameters of the DC motor; the second part was to investigate the operation of the DC motor under PWM control; the third part was the design and simulation of a PI current controller in the continuous domain; the fourth part used the controller gains from part three, to implement and simulate a discrete PI controller and to evaluate its performance when controlling the armature current of the DC motor.

3.2.1 DC Motor Model Parameters

Accurate parameters for the DC motor are required for both controller design and to enable the losses in the DC motor to be accurately estimated, such that the power usage of the AC VSD can be calculated. The DC motor used in the test bed system is a 40kW DC motor manufactured by Siemens, and has a rated armature current of 100A at a rated armature voltage of 400V. The DC motor is separately excited with a fixed field current, thereby maintaining a constant magnetizing flux and machine constant (K_m). The armature circuit parameters, as well as the inertia of the

machine were supplied by Siemens [42]. The motor constant K_m was determined from an open circuit generator test with a fixed field current, where the armature voltage was measured at various motor speeds and used to calculate K_m [38]. The coupling between the induction motor and the DC motor was removed and the mechanical losses due to the bearings and windage losses were measured by performing no load motor tests [38] at various speeds whereafter the electrical losses were subtracted leaving the mechanical losses of the machine. From the mechanical losses the coefficient of viscous friction was calculated. The test results for the open circuit generator and no load motor tests appear in Appendix B, and the resultant parameters are given in Table 3.1.

Table 3.1 Parameters of 40kW Siemens DC Motor

R_a	0.11Ω
L_a	2mH
J	1.1 Kgm^2
B	0.044 N.m/rad/sec
K_m	2.08 V/rad/sec

The block diagram of the DC motor model in Fig. 2.10, can be modified as shown in Fig. 3.1.

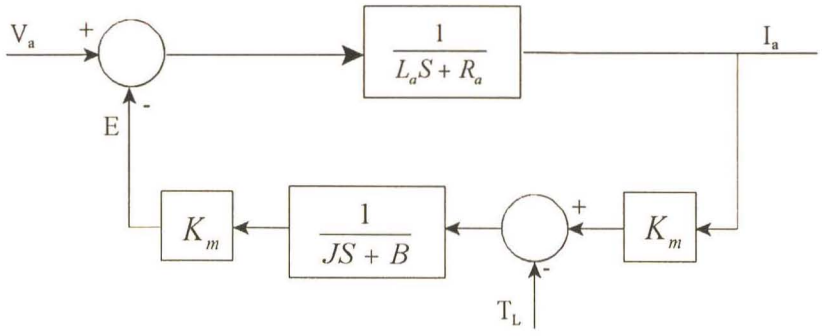


Fig. 3.1 Modified DC motor model block diagram

From Fig. 3.1, a transfer function relating the DC motor's armature current and armature voltage may be derived, for use in the design and simulation of the DC motor and armature current controller.

3.2.2 PWM Control of the DC motor

By varying the armature voltage of the DC motor the armature current can be controlled to achieve any desired level. A PWM-controlled H-Bridge inverter, as shown in Fig. 3.2 (the fly back diodes required across the switches have not been shown), was used to control the armature current to the DC motor. The use of the PWM-controlled H-Bridge inverter allows for the implementation of a precise and rapid armature current controller which is not limited by the mains supply as in conventional SCR DC motor drives.

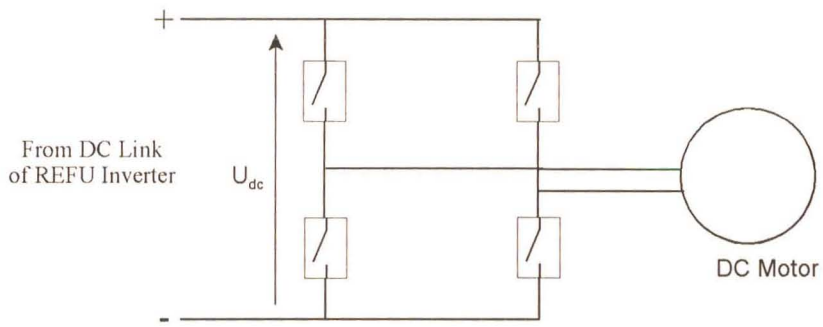


Fig. 3.2 Connection of DC motor to PWM controlled H-Bridge

The torque that the DC drive produces is determined by the magnitude of the armature current flowing in the DC motor according to Eq. 2.14. When the DC drive operates in the motoring region (driving a load) the rotation of the motor and current flowing into the motor is positive. For the DC drive to load the VSD in the test bed system the DC motor has to operated in the regenerative region, this occurs when the motor is turning in a positive direction but the armature current is negative, thus the motor produces a negative torque. The power regenerated by the DC drive has to be dissipated in some load. Due to the size of the DC drive (40kw) it is impractical to dissipate this energy in a resistive load. The regenerated power cannot be returned to the three phase mains supply using the H-Bridge configuration without the use of a controlled rectifier. The AC VSD in the test bed system is a FOC controlled VSI and has a DC link to which the H-bridge inverter may be connected so that the power generated by the DC drive in loading the VSD

may be returned and used by the VSD. The VSD therefore forms the load into which the DC drive regenerates power. As the DC link of the VSD is connected to the mains supply by a diode bridge power is circulated in the test bed system and the only power drawn from the mains supply is that required to cover the losses in the system.

The PWM controlled inverter was modelled in Simulink. Simulink is a tool for modelling, analysing and simulating physical and mathematical systems in both the continuous and discrete time systems [43]. The PWM controlled inverter was modelled such that any limitations of the PWM inverter topology could be considered. The armature inductance and resistance of the DC motor act as a filter, filtering the PWM pulses from the H-Bride inverter. If the switching frequency of the H-Bridge inverter is too low the armature inductance will not provide sufficient filtering action resulting in a large ripple in the armature current and subsequently large torque oscillation. The switching frequency of the power devices (IGBTs) is limited to around 5kHz mainly due to switching losses. The PWM ASIC used to control the switching of the devices in the H-Bridge inverter produces discrete switching frequencies based on its oscillator frequency. The nearest frequency that the PWM ASIC can produce to 5kHz is 4.88kHz. The switching frequency of the inverter was therefore simulated at 4.88kHz. The simulations performed in this section will determine whether more inductance is required in the armature circuit to ensure sufficient filtering of the armature current.

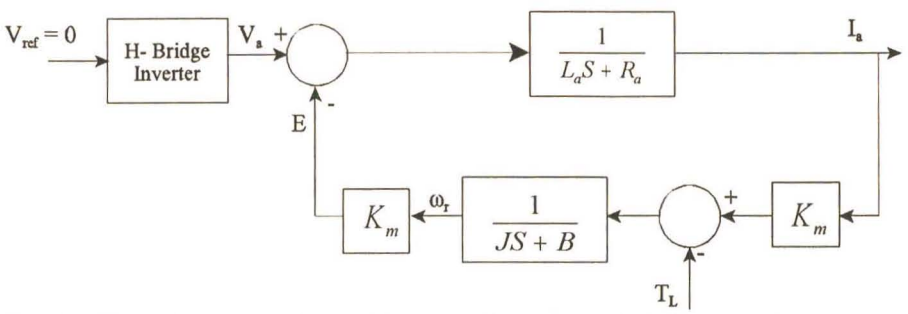


Fig. 3.3 Block diagram of the model used to investigate the DC motor ripple current.

The ripple in the armature current can be determined by setting the output of the PWM controlled H-Bridge inverter to zero ($V_{ref} = 0$) as shown in Fig. 3.3. The H-Bridge inverter then produces a 50% duty cycle square wave with an amplitude equal to that of DC link voltage (570V) as

shown in Fig. 3.4. The average voltage supplied to the DC motor by the inverter is zero as the positive half cycle of the square wave cancels the negative half cycle.

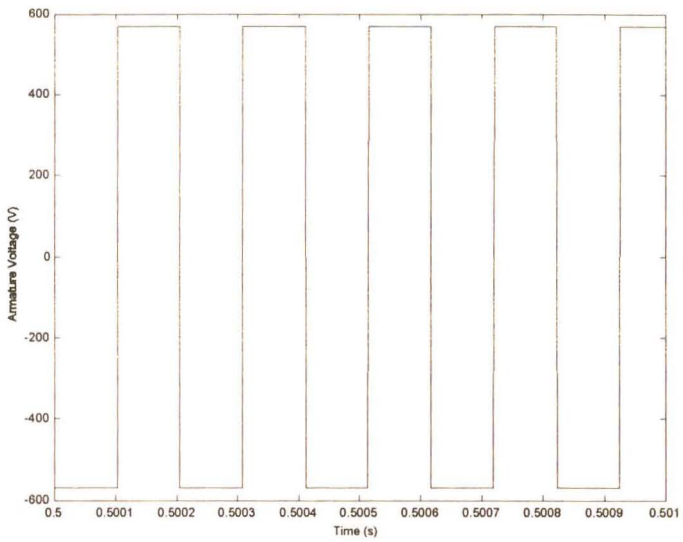


Fig. 3.4 PWM inverter output voltage

Fig. 3.4 shows the simulated armature voltage that was applied to the DC motor model shown in Fig. 3.3. Fig. 3.5 and Fig. 3.6 show the corresponding armature current and speed responses gained from the simulation.

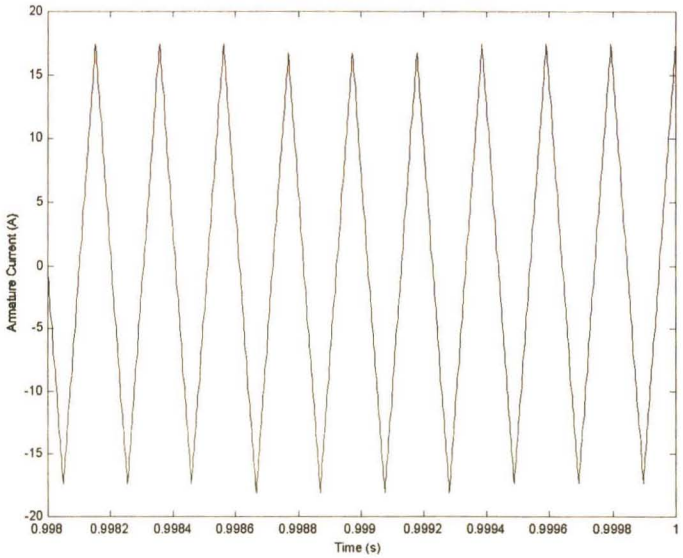


Fig. 3.5 Armature current ripple due to PWM switching waveform

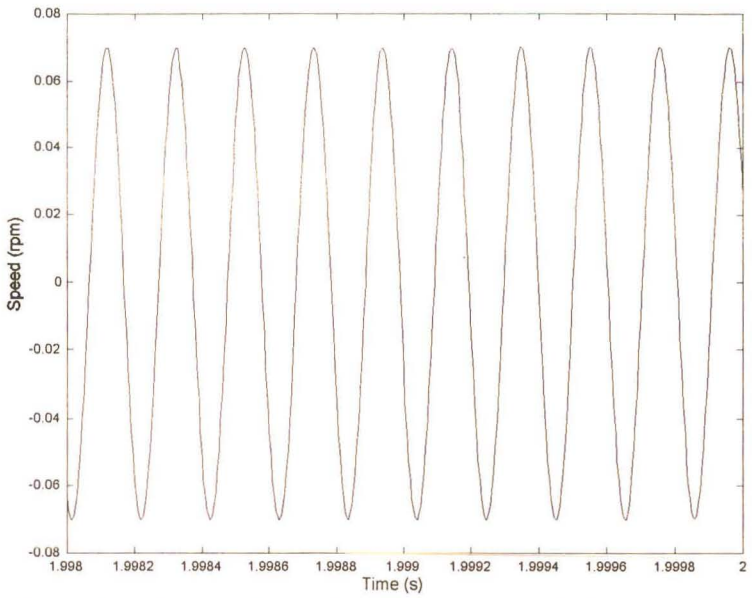


Fig. 3.6 Armature speed variation due to armature ripple current

Fig. 3.5 shows that the ripple present in the armature current has a peak value of 17A, which is 17% of the full load current which corresponds to 17% (35.36 N.m) of the full load torque that the DC motor can produce. The effect of the ripple current on the speed can be seen in Fig. 3.6. As the task of the DC motor is to load the VSD in the test bed system with a load that can simulate a pump or fan, it was felt that the ripple torque produced by the ripple current on the armature was too high. Since the IGBTs of the H-bridge inverter are switching near their maximum frequency (5kHz), extra inductance is needed in the armature circuit to reduce this ripple current. Two inductors of 10mH with a resistance of 0.6Ω and a current rating of 60A were available, and were placed in parallel resulting in an equivalent inductor of 5mH with a series resistance of 0.3Ω and a current rating of 120A. The ripple on the armature current with the inductors in series with the armature is shown in Fig 3.7, and has a peak value of 5A resulting in the DC motor producing a ripple torque of 5% (10.16N.m) of the full load torque. This ripple torque is close to the frictional torque (measured experimentally) experienced by the DC motor when running at full speed and is therefore acceptable for use in the test bed system. The speed variation due to the ripple current was reduced from 0.07rpm to 0.02rpm. Although the ripple current and therefore the ripple torque produced by the DC motor is not zero the inductors

inserted into the armature circuit greatly reduce the ripple torque which is experienced by the induction motor in the VSD. The addition of the inductors allows the DC motor to more accurately simulate a pump or fan load.

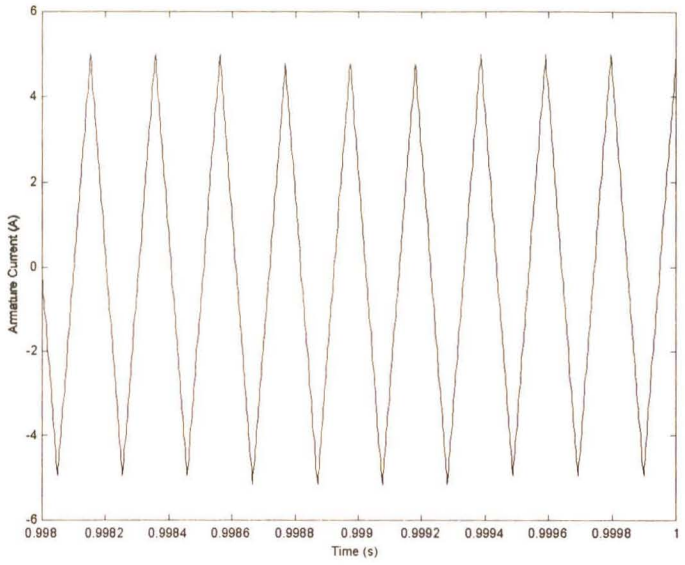


Fig. 3.7 Armature current ripple with in line armature inductors

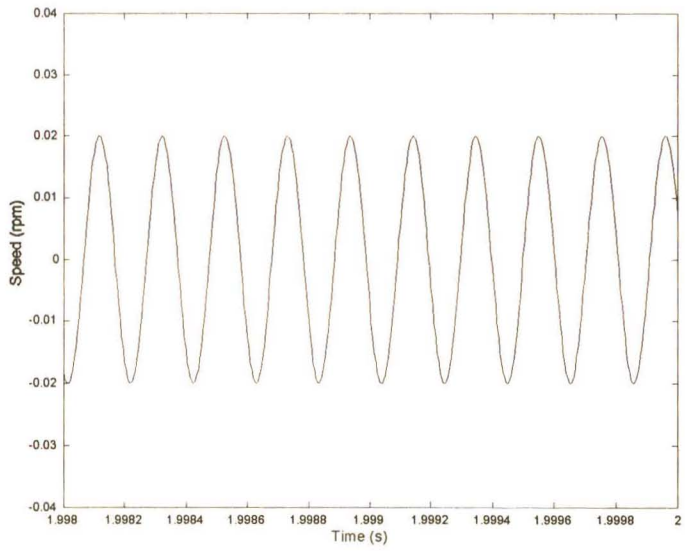


Fig. 3.8 Armature speed variation due to armature ripple current with in line inductors

3.2.3 DC Motor Controller Simulations

For the DC motor to act as a configurable load a current controller is required to control the armature current, thereby controlling the torque produced by the DC motor. Pumps and fans follow set steady state torque speed curves, provided there is no change in the physical pump or fan system (the static head remains constant and the specific gravity of the liquid does not change). For a pump or fan load to be accurately simulated it is important that the armature current controller (torque controller) has zero steady state error to ensure that the correct torque is applied to the induction motor in the VSD. The steady state error of a system is the difference between the actual value of the variable being controlled (armature current) and its desired value (desired armature current) once in steady state. Steady state can be defined as the manner in which the output of the system behaves as time approaches infinity [29]. A PI current controller is used in the system as it exhibits infinite gain at zero frequency, this characteristic of increased gain at low frequencies forces the steady state error of the system to zero. The block diagram of the DC motor and armature current controller is shown in Fig. 3.9. Due to the inverter having a high switching frequency and the DC motor with added armature inductors having a sufficiently high filtering effect the inverter can be considered as a controlled voltage source. The inverter is therefore neglected from the simulation as shown in Fig. 3.9.

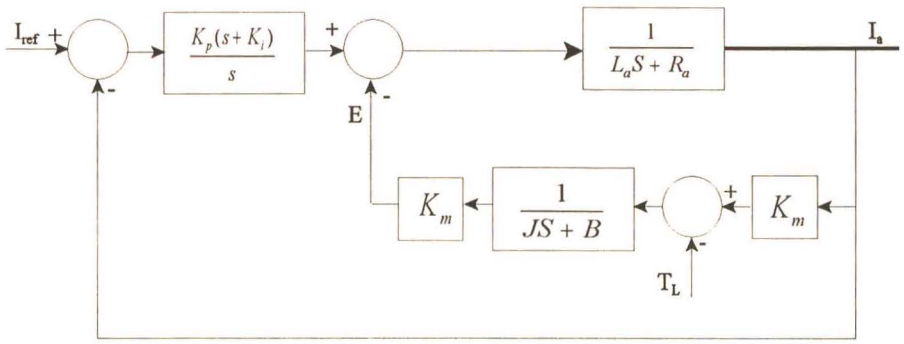


Fig. 3.9 DC motor and armature current controller block diagram

The response of the system shown in Fig. 3.9 is limited by three fixed factors namely the DC link voltage, the PWM switching frequency of the inverter and the dynamic response of the DC motor.

By correct design of the PI controller (selecting K_p and K_i) it is possible to alter the transient response of the system when a step input is applied. The desired system response under armature current control would be to have a fast settling time and exhibit no overshoot. The PI controller parameters (K_p and K_i) required to give the system response described above were selected as shown below using the pole placement method [29] in the continuous domain. The motor's current loop transfer function relating the armature voltage and current, is given by Eq. 3.1. Eq. 3.1 is gained by substituting the motor parameters given in Table 3.1 and the armature inductance values into Eq. 2.17 (R_a = 0.11+ 0.3Ω and L_a = 2+5mH).

$$P(s) = \frac{I_a(s)}{V_a(s)} = \frac{142.86(s+0.04)}{(s+46.46)(s+12.14)} \tag{3.1}$$

From Eq. 3.1 it can be seen that all the motor system poles and zeros are real and lie in the left-hand plane indicating that the open loop operation of the DC motor is stable. A PI controller has a transfer function G(s) [29], as defined in Eq. 3.2

$$G(s) = K_p\left[1+\frac{1}{T_i s}\right] = \frac{K_p(s+K_i)}{s} \tag{3.2}$$

where K_p is the proportional gain

$K_i = 1/T_i$ and is the integral gain.

The PI controller possesses a zero at $s = -1/T_i$ and a pole at $s = 0$. The pole at $s = 0$ forces the steady state error of the system to zero. The PI controller parameters (K_p and K_i) may be determined using the open loop system response. The PI controller gains determined using the open loop response are then applied to the closed loop system to achieve the desired system response. The proportional gain K_p is determined as follows, the output of the controlling device (PWM inverter) should not exceed the maximum limit (determined by the DC link voltage) when

the error signal is at its maximum which occurs when zero armature current is flowing and 100% (100A) armature current is demanded. K_p can be found using Eq. 3.3.

$$K_p \leq \frac{V_{link}}{I_{a\max}} \tag{3.3}$$

where V_{link} is the DC link voltage

I_{amax} is the maximum armature current which may be demanded (maximum DC motor rating).

Assuming that the DC link voltage never falls below 550V (V_{link} = 550V) and the maximum rated armature current of the DC motor is 100A (I_{amax} = 100A) the limit of K_p is 5.5. The controller $G(s) = K_p(s+K_i)/s$ is inserted into the open loop system resulting in the open loop transfer function given in Eq. 3.4. The closed loop transfer function is given in Eq. 3.5. Eq. 3.4 is used to plot a number of root loci to determine a location for the zero such that the desired system response is achieved. The response of the system is determined by the position of the closed loop poles [29] and Eq. 3.5 is used to evaluate and check the simulation results obtained.

$$T(s) = \frac{785.73(s+0.04)(s+K_i)}{s(s+46.46)(s+12.14)} \tag{3.4}$$

$$T(s) = \frac{785.73(s+0.04)(s+K_i)}{s^3 + 844.38s^2 + (785.73(0.04+K_i)+564.0244)s + 31.4292(K_i)} \tag{3.5}$$

The method of pole zero cancellation is used in the design of the PI controller, and a root locus is used to evaluate the effect of the pole (K_i) on the system. The value of K_i can be chosen to cancel two zeros in the system, the first is at s = -12.14 and the second is at s = -46.46. The effect of placing the pole at s = -12.14 (K_i = 12.14) and varying the gain from zero to infinity is shown by the root locus in Fig. 3.10.

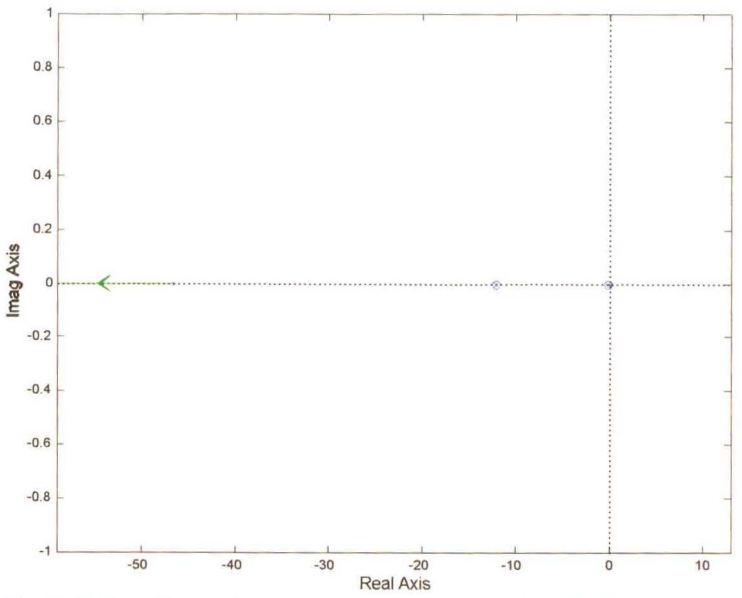


Fig. 3.10 Root Locus for open loop system with $K_i = 12.14$

From Fig. 3.10 it can be seen that the system is stable for all values of K_p as it is moved from zero to infinity. As K_p is increased the root locus moves away from the imaginary axis. The response of the system to a 100A step input when $K_i = 12.14$ and $K_p = 5.5$ is shown in Fig. 3.11. From Fig. 3.11 the system also appears to exhibits a steady state error which is due to the controller responding slower than the increase in the back EMF of the DC motor armature, the controller therefore applies insufficient voltage to the DC motor to achieve a current flow of 100A. For a value of $K_p = 5.5$ and $K_i = 12.14$ the system has closed loop zeros at s = -0.04 and s = -12.14 and closed loop poles at s = -0.0394, s = - 12.14 and s = -832.15. The poles at s = - 0.0394 and s = -12.14 cancel with the zeros at s = -0.04 and s = - 12.14 respectively, this results in a dominant pole being present at s = -832.15. This dominant pole forces the system to act as a first order system with a time constant of (1/832.15) 1.2mS, the high value of this pole has the effect of reducing the overall gain of the system (782.73/832.15). The effect of the pole can clearly be seen in Fig. 3.11 where the time constant of the system may be calculated as 1.2mS and the effect of the reduced gain can be seen in that the final value of the system does not reach 100A.

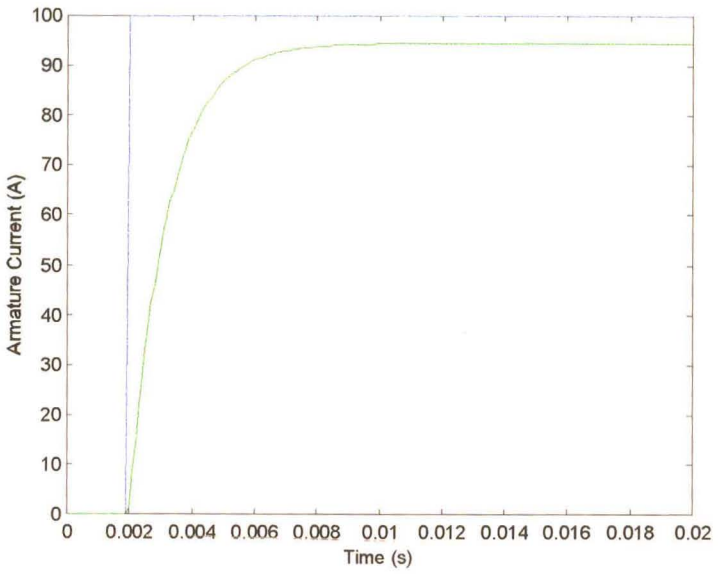


Fig. 3.11 System response to a step input of 100A (K_p = 5.5, K_i = 12.14)

The effect of placing the pole at s = -46.46 (K_i = 46.46) and varying the gain from zero to infinity is shown in Fig. 3.12. From Fig. 3.12 it can be seen that the system will remain stable for all values of K_p although K_p will affect the response of the system. The response of the system to a 100A step input when K_i = 46 and K_p = 5.5 is shown in Fig. 3.13. For K_p = 5.5 and K_i = 46 the system has closed loop zeros at s = -0.04 and s = -46 and closed loop poles at s = -0.0394, s = - 45.97 and s = -798.31. The poles at s = - 0.0394 and s = -45.97 cancel with the zeros at s = -0.04 and s = - 46.46 respectively, this results in a dominant pole being present at s = -798.31. This dominant pole forces the system to act as a first order system with a time constant of (1/798.31) 1.25mS as can be seen from Fig. 3.13, the effect of the pole on the overall system gain (785.73/798.31) can also be seen as the system armature current reaches the step input value of 100A. Although the response shown in Fig. 3.13 is slower than that shown in Fig. 3.11 the response of Fig. 3.13 is preferred as the system is capable of reaching and maintaining the reference supplied to the controller (100A step).

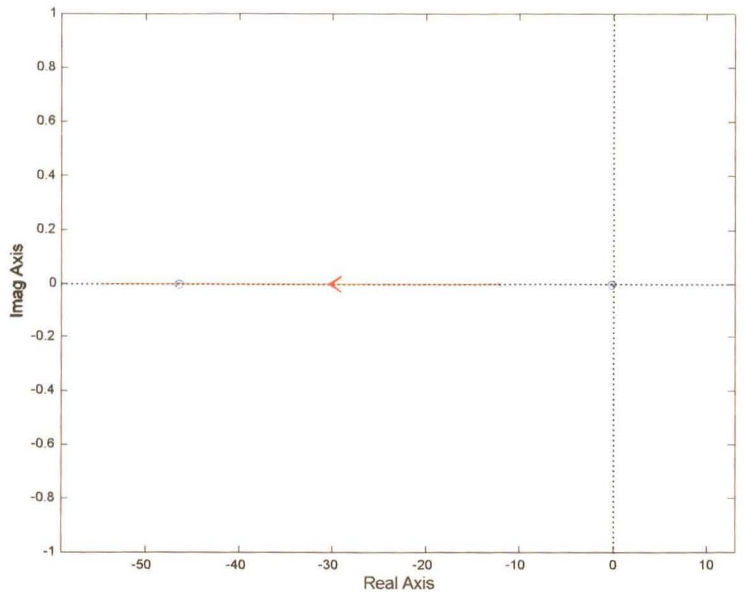


Fig. 3.12 Root locus of open loop system with $K_i = 46.46$

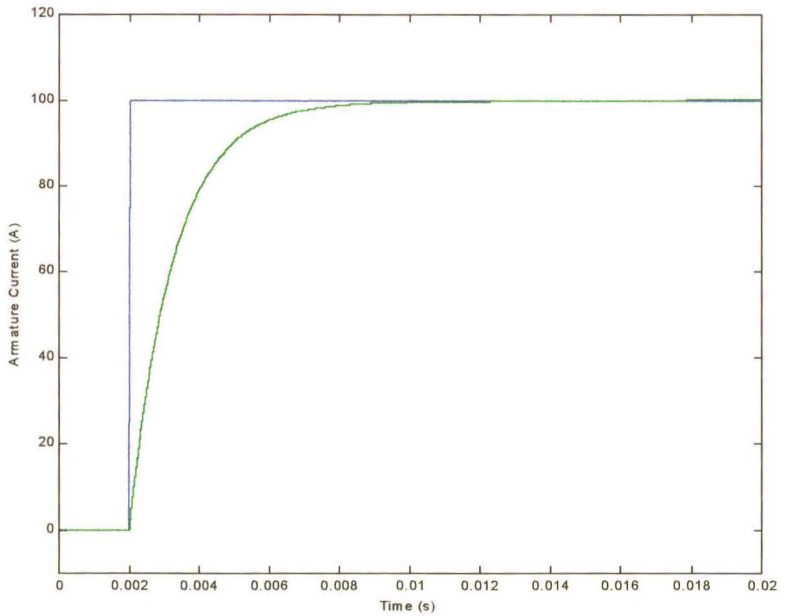


Fig. 3.13 System response to a step input of 100A ($K_p = 5.5$, $K_i = 46$)

The effect of choosing $K_i = 55$ as K_p is varied from zero to infinity is shown in Fig. 3.14. From

Fig. 3.14 it can be seen that the system could be chosen to be under damped if K_p was chosen incorrectly (system could exhibit overshoot). The response of the system to a 100A step input when $K_i = 55$ and $K_p = 5.5$ is shown in Fig. 3.15. For $K_p = 5.5$ and $K_i = 55$ the system has closed loop zeros at s = -0.04 and s = -55 and closed loop poles at s = -0.0394, s = - 55.49 and s = -788.79. The poles at s = - 0.0394 and s = -55.49 cancel with the zeros at s = -0.04 and s = - 55 respectively, this results in a dominant pole being present at s = -788.79. This dominant closed loop pole will force the system to act as a first order system (lies on the real axis) with a time constant of (1/788.79) 1.26mS, which may be calculated from Fig. 3.15. From Fig.3.15 it can be seen that the system is beginning to exhibit a small amount of overshoot which indicates that the system is becoming second order and if the gain in the system had to be increased the system would become second order dominant.

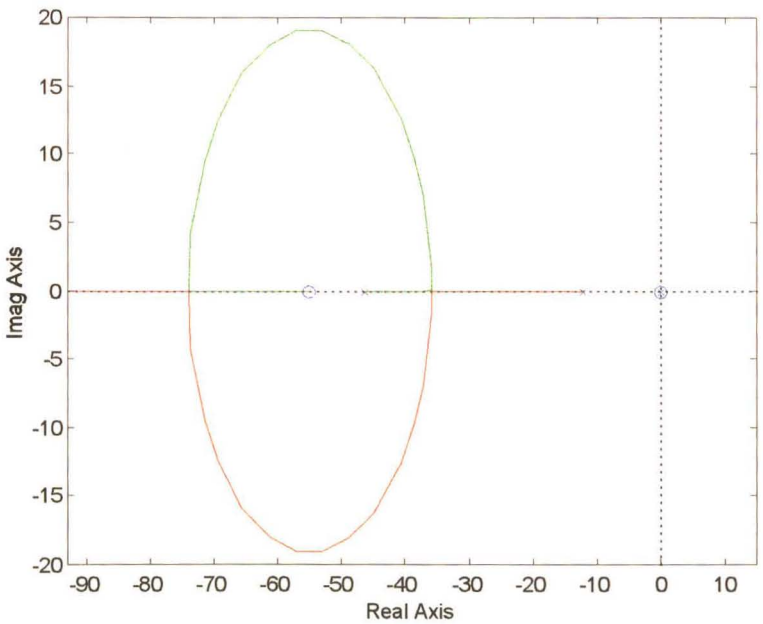


Fig. 3.14 Root locus for open loop system with $K_i = 55$

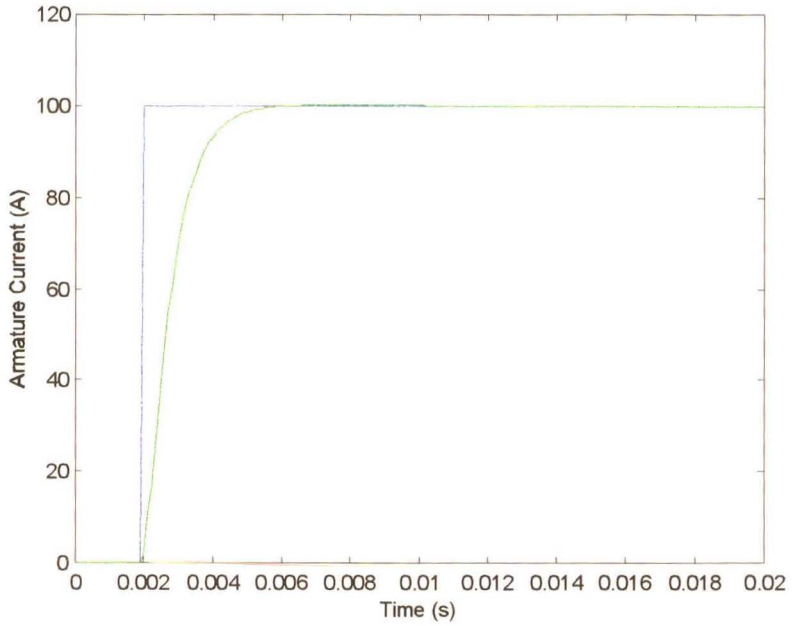


Fig. 3.15 System response to a step input of 100A (K_p = 5.5, K_i = 55)

From the root loci and the system responses presented it can be seen that for a value of K_i less than 46.46 the system is first-order dominant (will not exhibit overshoot). When K_i is increased above 46.46 the system may become second-order dominant is the system gain increases. The proportional and integral gains for the test bed system are thus chosen as K_p = 5.5 and K_i = 46, although this does not result in a system with the highest response to a step input it does result in a system that will not produce any overshoot even if the gain in the system changes due to a varying DC link voltage.

A PI armature current controller was designed in the continuous domain using the well-established method of pole placement. If the discrete controller samples the system sufficiently fast the gains determined for the continuous PI controller can be used in the discrete controller, this relates to the Nyquist theorem and standard engineering practice where it is stated that the discrete PI controller should sample the system at least ten times during the response of the system in order for the effects of the discrete controller to be ignored [30]. Due to the discrete PI controller

having a sampling time (102μS) much higher than the settling time of the system (the discrete PI controller will sample the system 61 times during the settling time of the system) the PI controller gains determined in the continuous domain can be used for the discrete PI controller. A discrete PI controller using the gains determined in the continuous domain was simulated with its associated components (A/D converters, sample and hold circuits) in order to determine if it could provide the desired system response. A discrete PI controller was implemented as a digital controller was used in the test bed system. The response of the discrete PI controller to a 100A step is shown in Fig. 3.16. The sampling rate of the PI controller is set equal to twice the switching frequency of the PWM inverter (T_s= 102 μs), as in the practical system the PI controller is run in accordance with an interrupt generated by the PWM ASIC, which occurs twice in one switching cycle.

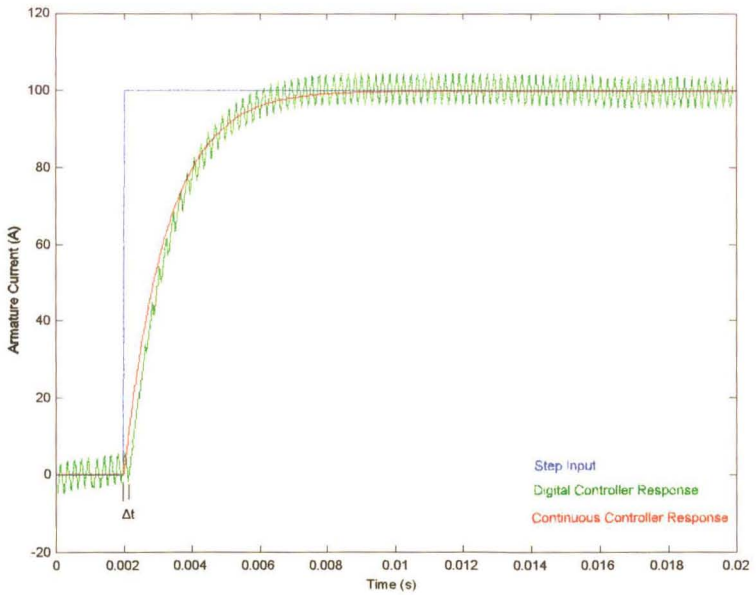


Fig. 3.16 Armature current response to step input of 100A for continuous and discrete time controllers (K_p = 5.5, K_i = 46, T_s = 102μs)

From Fig. 3.16 it can be seen that there is a two sample period time lag (Δt = 204 μs) between the response of the analog and digital controller. This time delay is due to the nature of the discrete PI controller and the operation of the PWM ASIC in the PWM inverter. The two sample period delay can be explained with the aid of Fig. 3.17.

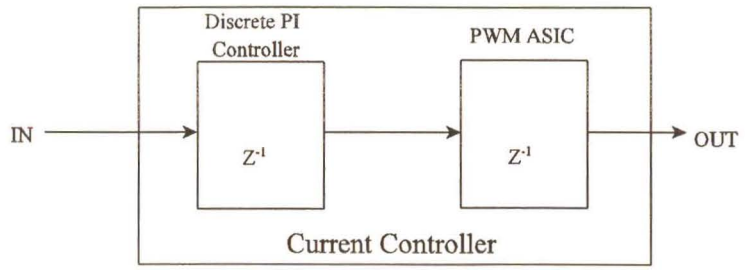


Fig. 3.17 Block diagram showing origin of two period sample delay in armature current controller

The first sample delay is due to the discrete PI controller, which is inherent in all discrete PI controllers. The second sample delay is due to the PWM ASIC used in the PWM inverter which outputs the corresponding switching waveform on the following switching cycle [16]. When there is a change to the set point value input to the armature current controller the discrete PI controller takes one sample period (102 μs) to respond, this response is then fed to the PWM ASIC which takes a further sample period to output the corresponding switching waveform. Therefore, a two period sample delay exists between the period when the set point changes to when the control variable responds.

The simulations performed on the armature current controller and PWM inverter show that it is capable of controlling the armature current of the 40kW DC motor to be used in the test bed system. The armature current controller also ensures that there is very little error between the set point value and the actual armature current which is required when simulating a pump or fan load on the test bed system. The reference supplied to the armature current controller will be determined by the type of load being simulated. The pump and fan loads that will be simulated on the test bed system are determined from the pump or fan characteristics and the system characteristics.

3.3 Pump Load Curve Calculations

The loads that industrial equipment, such as pumps and fans, place on fixed and variable speed drive systems have to be determined so that the armature current reference for the DC motor can be set. The performance of a pump differs according to the system that the pump is expected to operate in. To highlight this two pump systems are considered in this study. The first pump system shown in Fig. 3.18 (System A) has a low static head but a high frictional head. The second pump system shown in Fig. 3.19 (System B) has a high static head and a low frictional head. The load these two pump systems place on the AC VSD in the test bed system can be calculated when using either discharge throttling or speed regulation. The two pump systems are used to evaluate the possible energy savings when using variable speed to control the flow rate as opposed to discharge throttling (fixed speed).

Pump system A as shown in Fig. 3.18 comprises of the following sections. The first is a vertical lift of 3m from the water level of the supply reservoir to the centre line of the pump's inlet. It is assumed that the supply reservoir water level remains constant so that the static head in the system remains constant. An elbow is used to change the pipe direction after which there is a 6m length of horizontal pipe to the pump. The pump discharges into a 100m of pipe connected to an elbow. The liquid is then lifted by a further 6m from the centre line of the outlet of the pump, to the centre line of a 100m pipe transporting the liquid to the discharge reservoir.

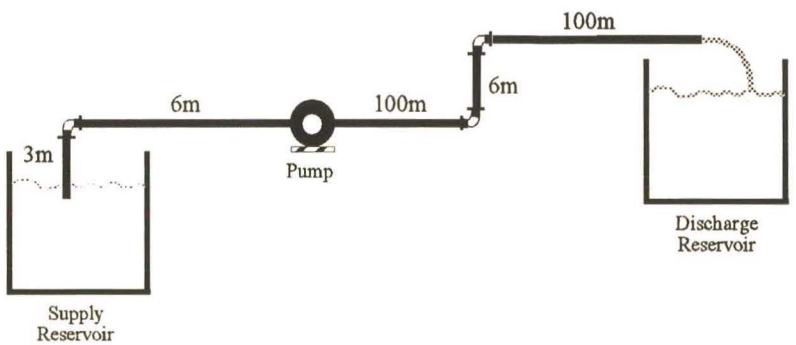


Fig. 3.18 Pump System A used for determining pump performance

Pump system B shown in Fig. 3.19 comprises of the following sections. The first is a vertical lift of 2m from the water level of the supply reservoir to the centre line of the pump's inlet. Again it is assumed that the supply reservoir water level remains constant so that the static head in the system remains constant. An elbow is used to change the pipe direction after which there is 1m of horizontal pipe to the pump. The pump discharges into a 1m length of pipe connected to an elbow. The liquid is then lifted by a further 40m from the centre line of the outlet of the pump to the centre line of a 1,5m pipe transporting the liquid to the discharge reservoir.

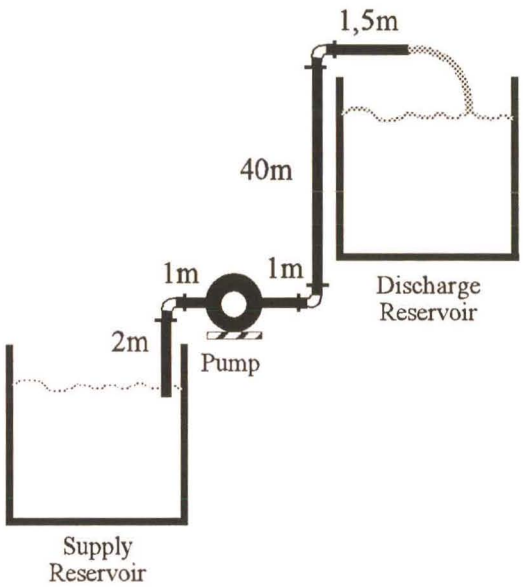


Fig. 3.19 Pump system B used for determining pump performance

The pipes and elbows in pump system A and B all have a 127mm inside diameter (5 in. pipe) and are made of iron. Due to the non linearity of the system and the non linear characteristics of the pump all performance calculations are done graphically using the pump and system characteristic curves. Both pump systems A and B have been chosen to deliver the same flow rate of liquid when running at full speed without any throttling. The two pump systems demonstrate the importance of considering each system prior to installing an AC VSD in order to achieve energy savings. This also demonstrates the usefulness of the test bed system which enables pump and fan systems to be simulated in order to determined the power usage of the AC VSD when performing an energy savings evaluation.

3.3.1 Pump System Characteristic Curves

The method used to calculate the system head was discussed in Chapter 2, where it was noted that the system characteristic curve is determined by the properties of the components in the system. The component properties are measured experimentally and are displayed in tables [22], [see Appendix C].

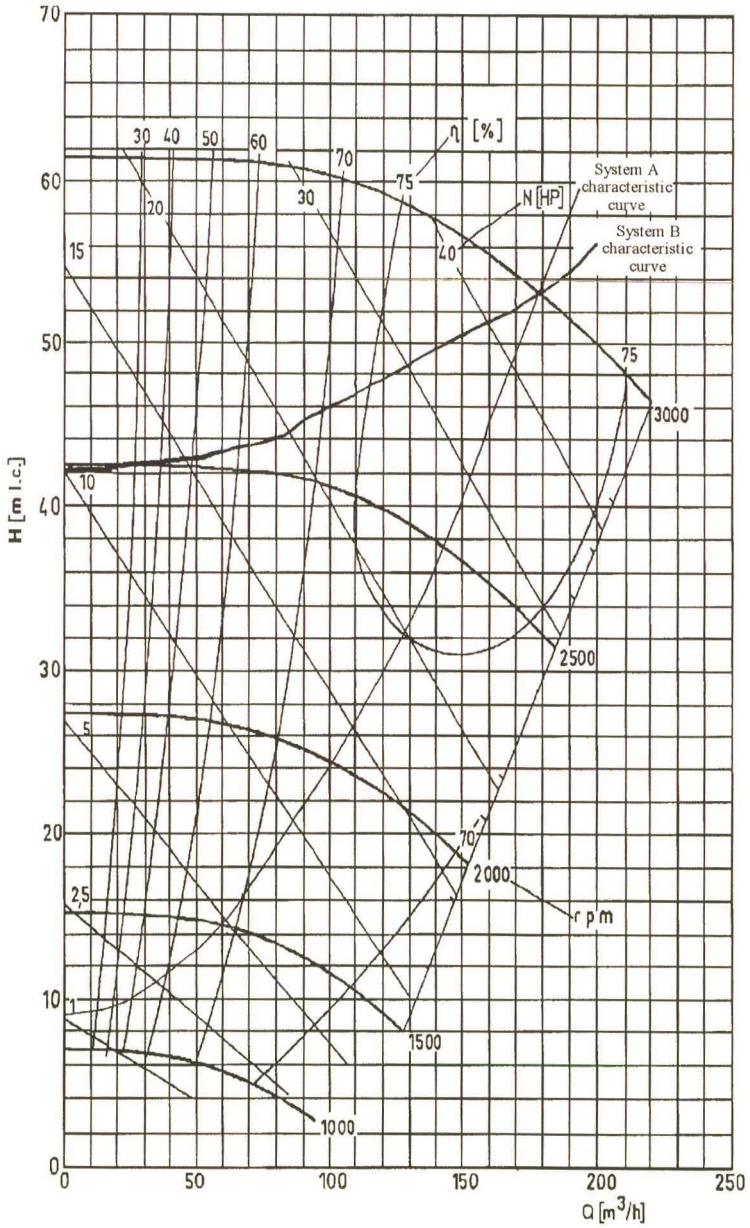


Fig. 3.20 Pump and system characteristic curves for pump systems A and B.

The system head has to be calculated for each flow rate in the system to gain the system characteristic curve. The system head calculations for pump systems A and B are given in Appendix C. The two system characteristic curves and the pump characteristic curve are given in Fig. 3.20 where it can be seen that the pump will produce the same flow rate when operated in both systems when running at 3000rpm ($181m^3/h$).

3.3.2 Flow Control Using Discharge Throttling For Pump Systems A and B

In order to use the test bed system to simulate a pump system which employs discharge throttling to control the flow rate the torque that the pump would place on the motor for various flow rates has to be determined.

As discussed in Chapter 2, discharge throttling is a cheap method of controlling the flow rate of a pump. The main disadvantage of this form of flow control is the inefficiency of the system at medium and low flow rates. To use discharge throttling on the pump systems shown in Figs 3.18 and 3.19, a throttling valve would have to be placed in the discharge pipe between the pump and the discharge reservoir.

When the throttling valve is partially closed the liquid experiences a force when it meets the valve, which causes an effective increase in the friction head of the system. The increased system head results in the system characteristic curve intersecting the pump characteristic curve at a lower flow rate, as shown in Fig. 3.21. Fig. 3.21 shows the system curve for three positions of the throttling valve for pump system A: position Q1 is the flow rate when the valve is completely open (un throttled), Q2 and Q3 are the flow rates when the valve is being closed. Fig. 3.22 shows the system curve for the same three positions of the valve in pump system B. The system characteristic curves for Q2 and Q3 in Figs 3.21 and 3.22 are determined by adding an extra frictional head (which represents the extra force the valve places on the liquid) to the system head, after which the total system head is calculated. As the intersection between the pump and system characteristic curve moves along the pump characteristic curve the system curve does not have to be calculated for each position of the valve.

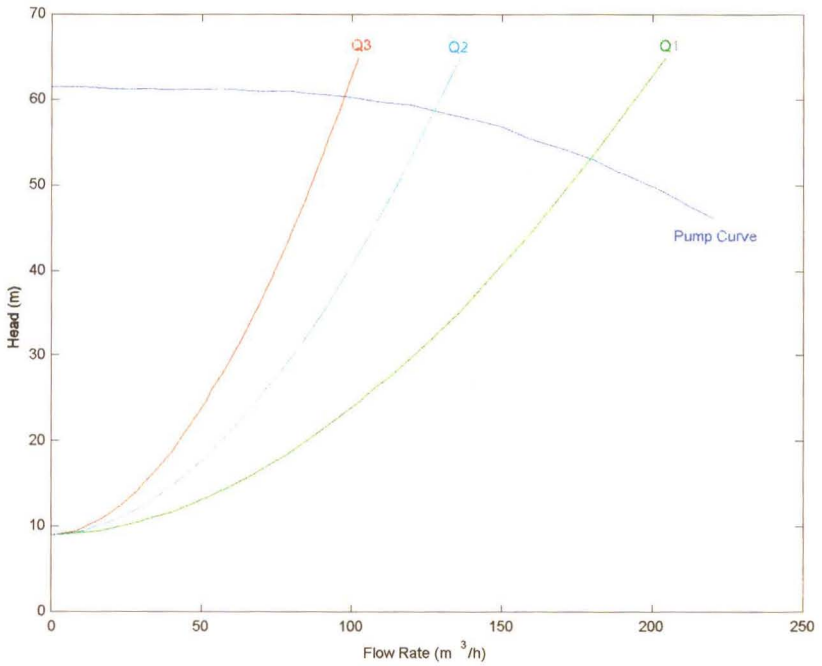


Fig. 3.21 Effect of closing the throttling valve on system A characteristic curve with different throttle valve settings

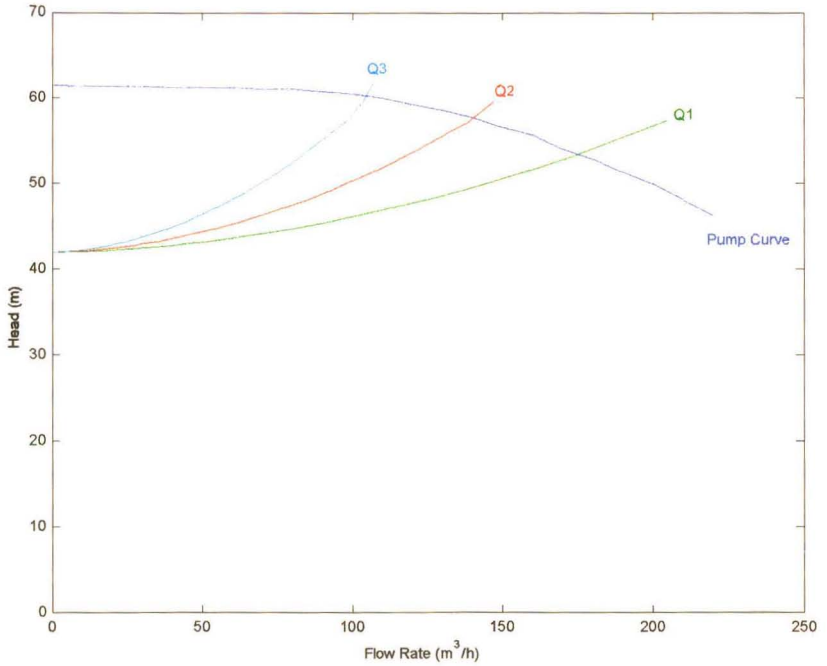


Fig. 3.22 Effect of closing throttling valve on system B characteristic curve with different throttle valve settings.

Figs 3.21 and 3.22 show that as the throttling valve is closed the intersection of the system characteristic curve and the pump characteristic curve follows the pump curve. Therefore the pump cannot distinguish whether it is operating in pump system A or B as the two system curves produce the same total system heads for each flow rate that the pump produces. The load torque produced by the pump will therefore be identical for both pump systems A and B for any given flow rate. The power required by the pump for both system A and B versus flow rate is shown in Fig. 3.23. Fig. 3.23 is determined by reading the power requirements for the pump running at 3000 rpm for the various flow rates from Fig. 3.20.

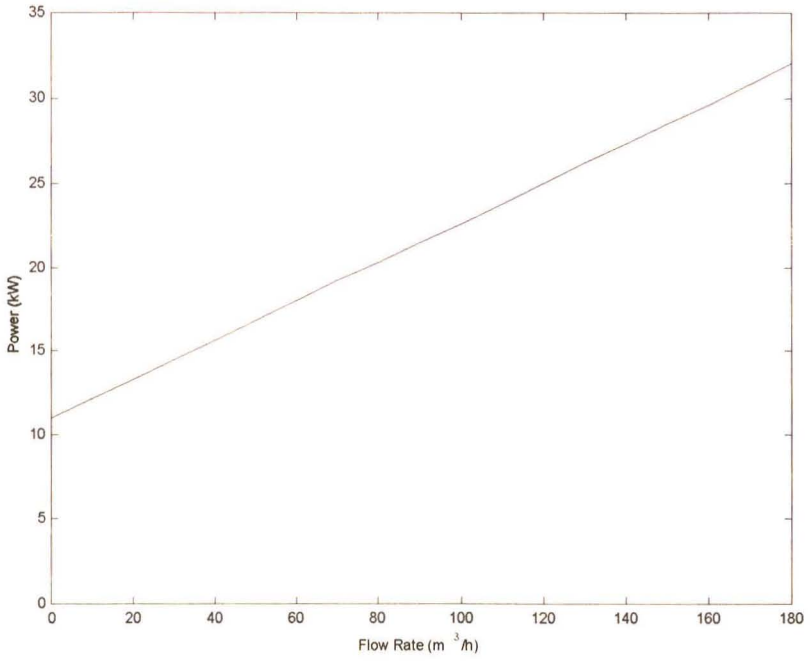


Fig. 3.23 Input power vs flow rate in pump system A and B when using discharge throttling to control the volume flow rate

When using discharge throttling the speed of the motor remains almost constant (very small change in the slip), and therefore the torque placed on the motor while driving the pump varies almost linearly with the flow rate as shown in Fig. 3.24. The load curve shown in Fig. 3.24 is used in Chapter 5, when implementing the controller to simulate the load placed on the VSD running at fixed speed and using discharge throttling to control the flow rate.

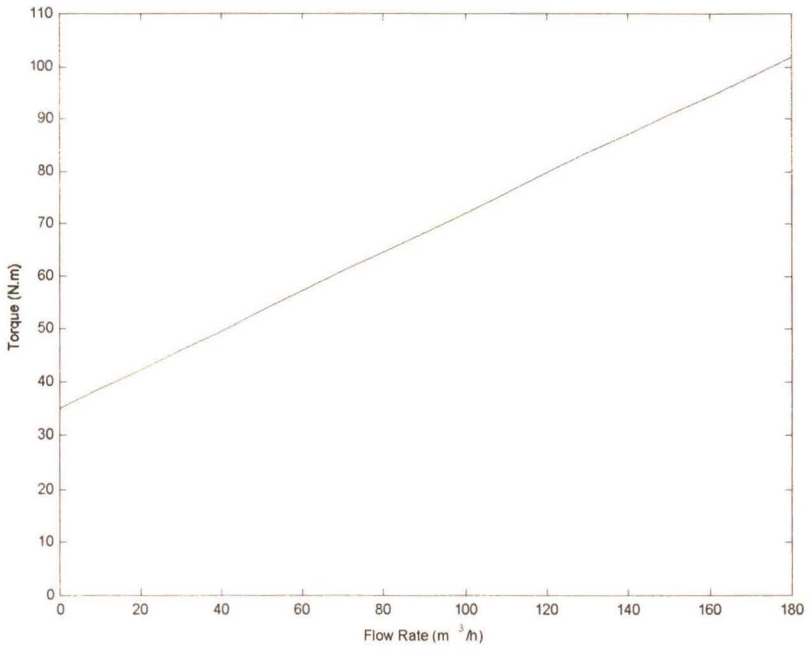


Fig. 3.24 Pump shaft torque vs flow rate in pump system A and B using discharge throttling to control the volume flow rate

3.3.3 Flow Control Using Speed Regulation for Pump System A

To enable a comparison to be made between the power usage of a pump system using discharge throttling and speed regulation on the test bed system, the configurable load must be able to load the AC VSD with the appropriate load characteristic produced by the pump when variable speed operation is being used.

The effect of varying the speed of a pump was discussed in Chapter 2, where it was stated that as the pump speed is reduced its characteristic curve changes in accordance with the affinity laws. Using the affinity laws for pumps, presented in Chapter 2 (Eqs 2.1 and 2.2), the pump characteristic curves for various pump speeds can be calculated from the original pump curve. The pump characteristic curves for a number of speeds have been plotted to determine the intersection between the system and pump characteristic curves as shown in Fig 3.25. As the speed is reduced the intersection of the pump characteristic curve at the reduced speed and the system

characteristic curve (which remains constant) results in a reduced flow rate. The relationship between the flow rate and the speed of the pump is non linear due to the non linear characteristics of the pump and system characteristic curves.

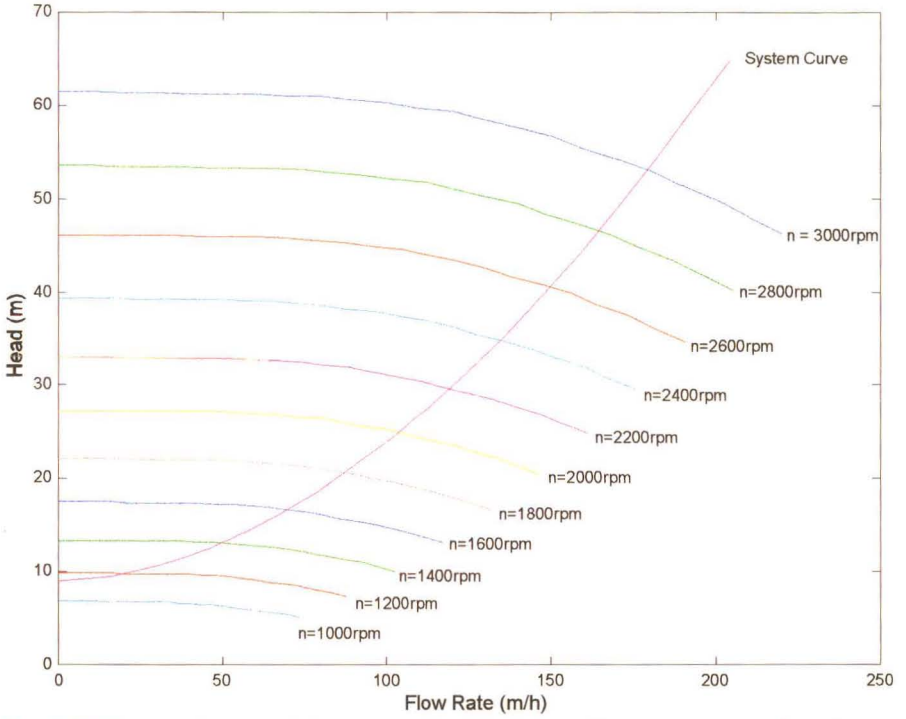


Fig. 3.25 System characteristic curve and pump curves for pump system A with pump speed varying from 1000 to 3000rpm

From Fig. 3.25 the speed required to produce a specific flow rate can be found and is displayed in Fig. 3.26. The estimated speed required to produce a specified flow rate is also shown and is calculated using Eq. 3.6, and is found using curve fitting techniques [14]. It is important to note that below a specific speed the pump is unable to produce a head capable of overcoming the static head of the system and at this point the pump stops pumping the liquid.

$$n_p = \frac{(3000-1100)}{(180)^{1.4}} Q^{1.4} \tag{3.6}$$

where n_p is the pump speed in rpm

Q is the flow rate.

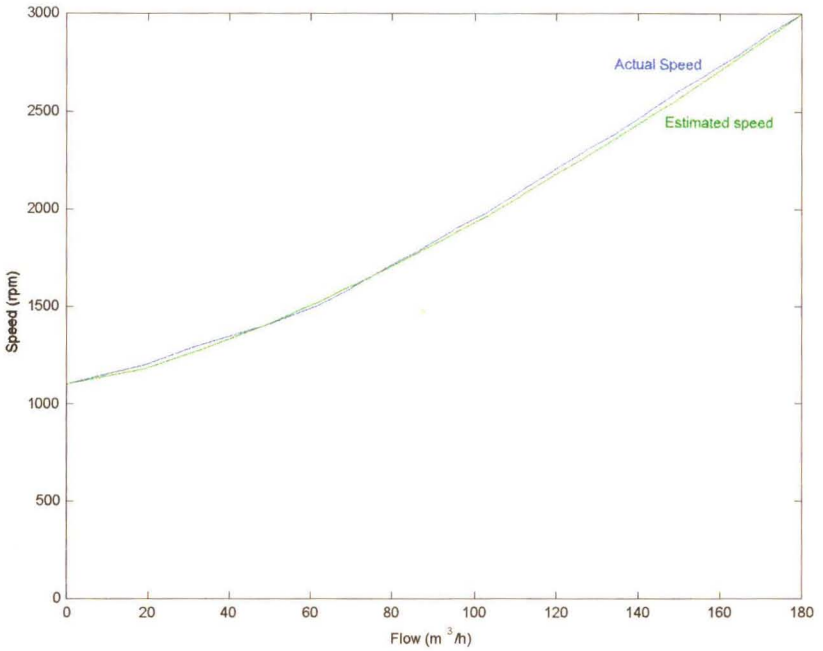


Fig. 3.26. Pump speed vs flow rate for pump system A

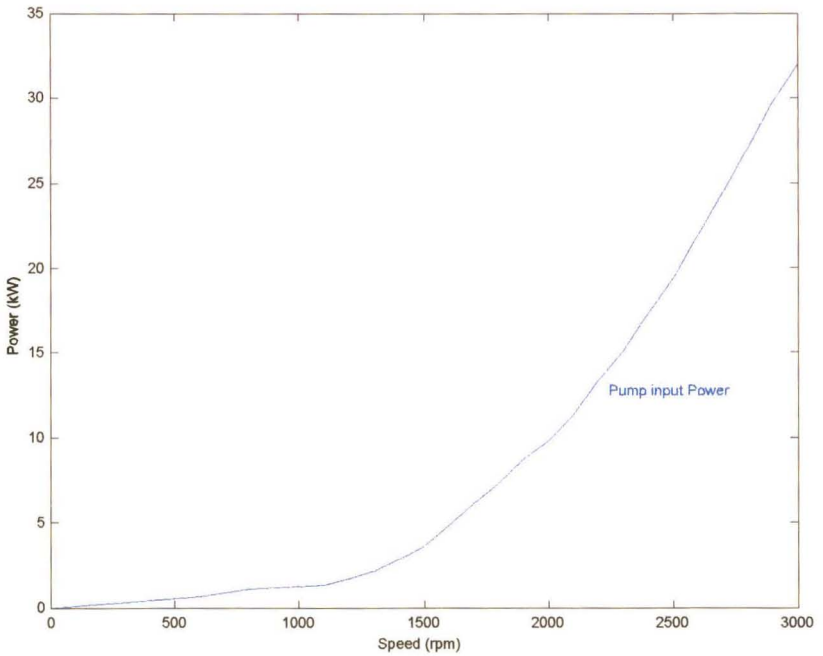


Fig. 3.27 Pump input power vs pump speed for pump system A

Fig. 3.27 shows the variation of the pump input power versus speed, which was found by reading the power off the pump characteristic curve at the specified flow rate and system head. The pump ceases to function below 1100 rpm, as shown in Fig. 3.26, but still requires power to be turned. This power is dissipated as heat inside the pump resulting in possible damage if operated in this state for prolonged periods of time. The load torque placed on the induction motor of the AC VSD when using speed regulation can be calculated from Fig. 3.27, by dividing the power by the speed in rad/sec, the results of which are shown in Fig. 3.28. The torque produced by the pump can be estimated using Eq. 3.7 which was found using Lagrangian polynomials [14] and will be used in the test bed controller to enable it to replicate Fig. 3.28 when controlling the DC motor to load the AC VSD.

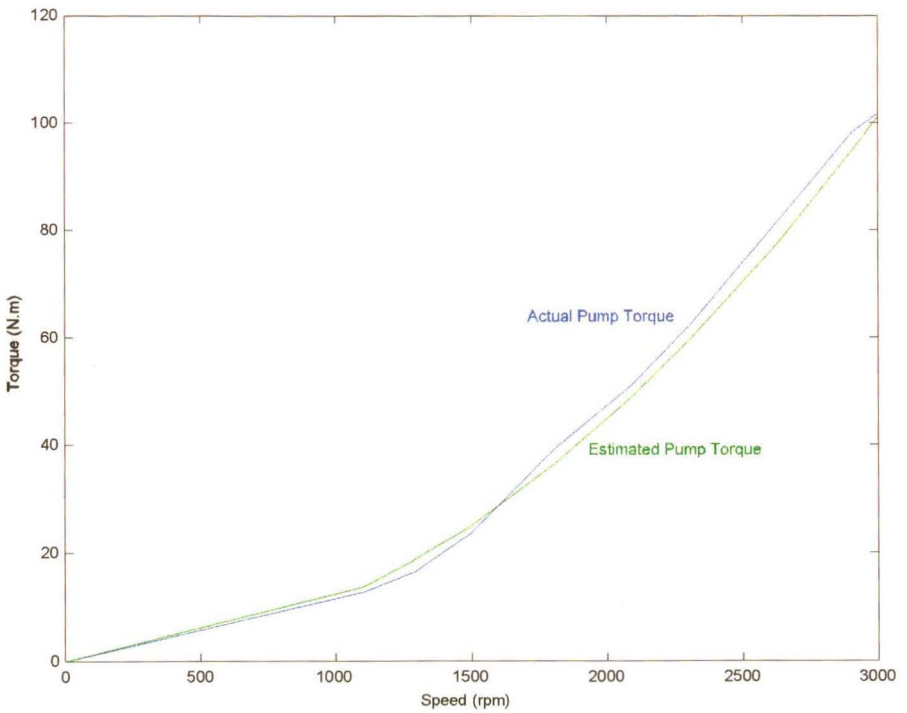


Fig. 3.28 Pump shaft torque vs speed for pump system A

$$T = \frac{101}{3000^2} n_p^{\,2} \tag{3.7}$$

where T is the load torque produced by the pump

n_p is the pump speed in rpm.

3.3.4 Flow Control Using Speed Regulation for Pump System B

To evaluate the effect that a system has on the power usage of the pump and therefore the possible energy savings when speed regulation is used in place of discharge throttling, the power and energy usage of system B will be compared to that of system A in Chapter 7. The load that the pump would place on the VSD in pump system B has to be known prior to performing any pump simulations on the test bed system. As with pump system A the load placed on the VSD by system B is determined from the pump and system characteristic curves. Fig. 3.29 shows the pump and system characteristic curves for system B. From Fig. 3.29 it can be seen that the pumping of liquid stops at a much higher speed when compared with system A, due to the high static head present in system B. Fig. 3.29 allows a speed vs flow rate curve to be plotted as shown in Fig. 3.30. The speed required to produce a specific flow rate in system B is given by Eq. 3.8 which was found using Lagrangian polynomials [14].

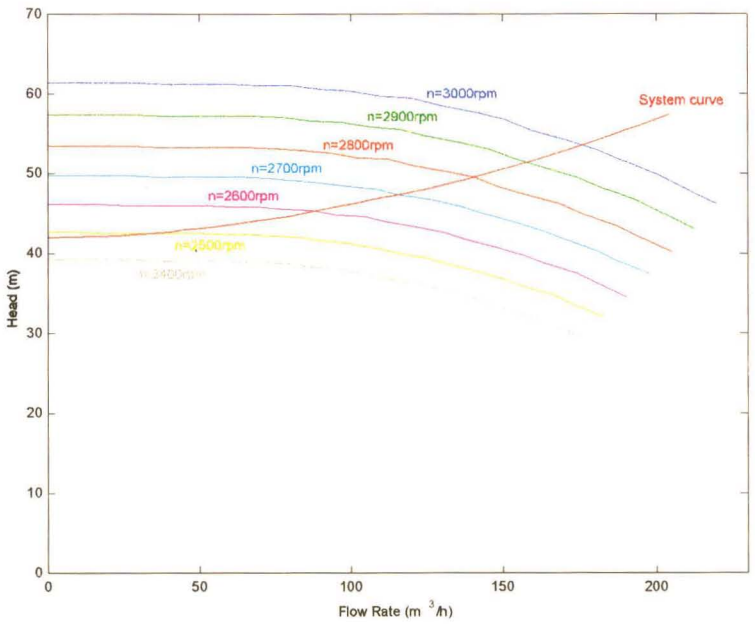


Fig. 3.29 System characteristic curve and pump characteristic curve for pump system B

$$n_p = \frac{(3000 - 2480))}{(180)^2} Q^2 + 2480 \tag{3.8}$$

where n_p is the pump speed in rpm

Q is the volume flow rate.

From the pump characteristic curves the power required to pump the liquid in system B for the various speeds can be determined as shown in Fig. 3.31. A deflection in the curve in Fig. 3.31 can be seen at the point where the pump ceases to pump any liquid as it cannot produce the required head to overcome the large static head present in system B. The load torque that the AC VSD is subjected to while driving a pump in system B in which speed regulation is used to control the flow rate can be calculated from Fig. 3.31 by dividing the power by the speed, the results of which are shown in Fig. 3.32. Fig. 3.32 also shows a deviation in the curve at the point where the pump ceases to pump any liquid.

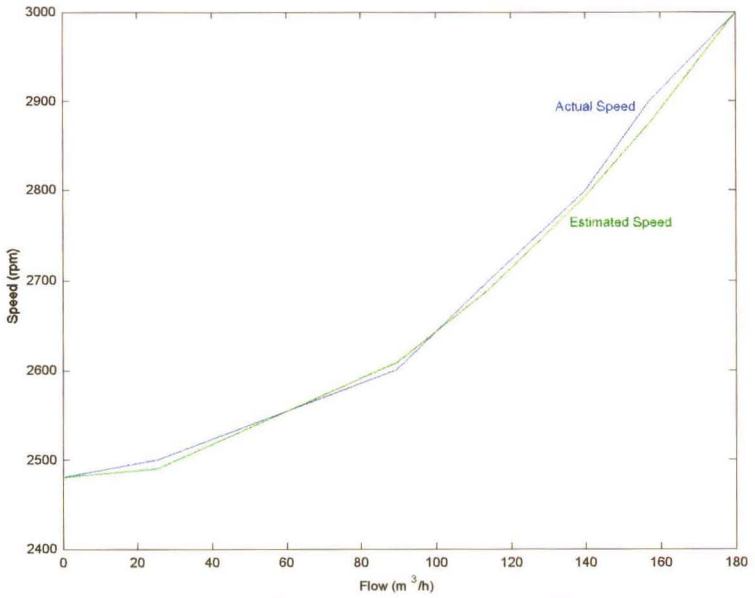


Fig. 3.30 Pump speed vs flow rate for pump system B

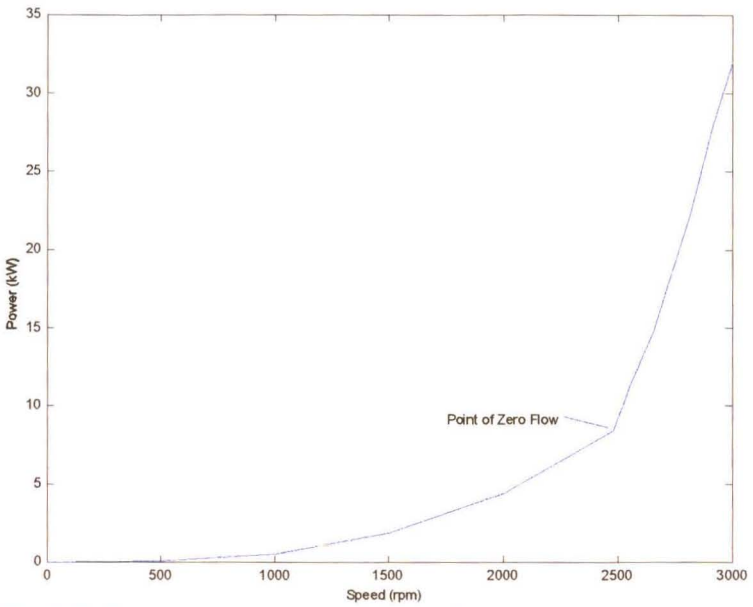


Fig. 3.31 Pump input power vs pump speed for pump system B

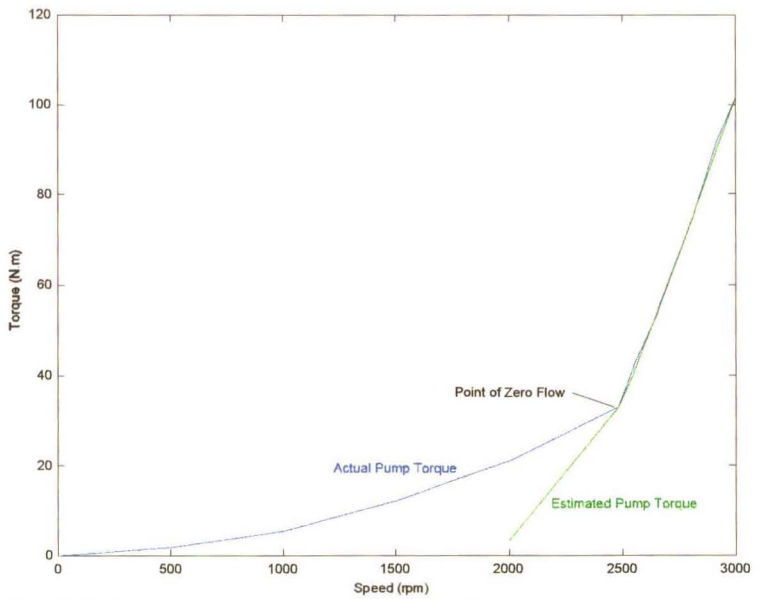


Fig. 3.32 Pump shaft torque vs pump speed for pump system B

The pump torque required at a specific speed can be estimated using Eq. 3.9 which was determined using Lagrangian polynomials [14]. The estimated curve shown in Fig. 3.32 only

covers the region where the liquid is being pumped. Below 2480 rpm the flow rate is zero and points below this speed are of no interest as the pump should never be operated in this state.

$$T = -630.511 \times 10^{-10} n_p^3 + 543.429 \times 10^{-6} n_p^2 - 142.07 \times 10^{-2} n_p + 1175.467 \qquad (3.9)$$

where T is the load torque produced by the pump

n_p is the pump speed in rpm.

The torque speed and torque flow rate curves presented for a pump operating in system A and B in which the flow rate is controlled using speed regulation and discharge throttling are used in Chapter 5 when implementing the test bed control software.

3.4 Fan Load Curve Calculations

To enable a comparison between the power usage of a fan system using dampers to be compared to the same fan system using speed regulation to control the flow rate, the torque speed curves for both systems have to be determined. These torque speed curves are used to load the AC VSD when determining the power usage of the two methods of flow control.

As in the case of a pump the performance of a fan in a system is determined by the fan and system characteristics. Two types of flow control methods: damper control; and speed regulation are presented. As the operation of a fan in a system depends on both the fan and system characteristics a test system needed to be proposed. In Chapter 2 the method for calculating the system curve for a fan system was discussed and is used to calculate the system curve for a test system. Only one fan system is presented as fans are selected to operate in a specific system. If the system changes and the resistance of the fan is operating against changes then the fan should be changed to one which is more suited to the new system.

The proposed fan system has the following characteristics. A pressure loss of 2400 pascal when the flow rate is $12m^3/s$. This results in an equivalent resistance of the ducting being equal to 16.6 pascals/m^5/s^2. The system curve can now be calculated from Eq. 2.11 and is plotted on the fan curve, as shown in Fig. 3.33. The cross sectional area of the ducting is equal to the cross sectional area of the fan's outlet ($0.532m^2$), such that no velocity changes occur in the ducting system as this affects the flow rate due to turbulence.

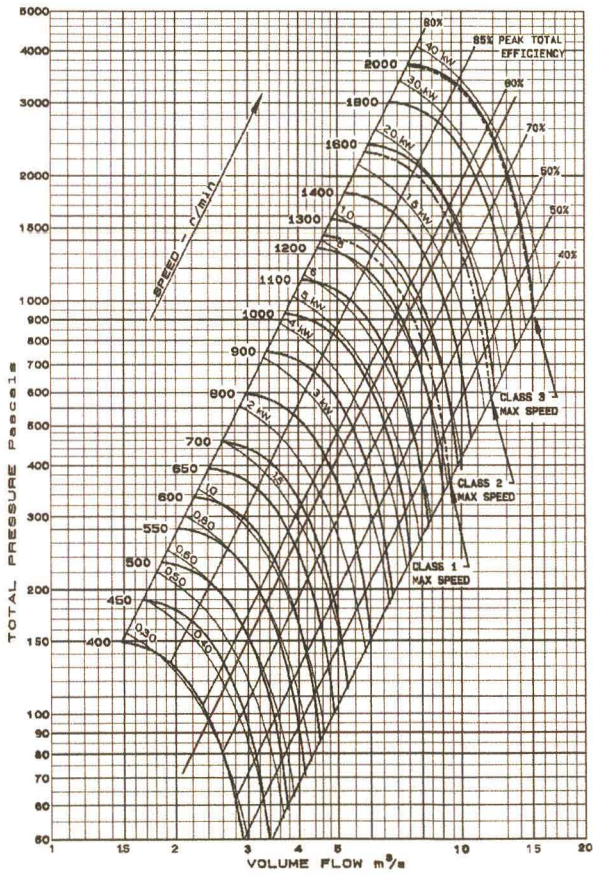


Fig. 3.33 Fan and system curve

3.4.1 Flow Control Using Dampers

In order to simulate a fan system where dampers are used to control the flow rate on the test bed system, the torque flow rate curves have to be known. The effect of closing the damper is an increase in the system resistance, resulting in the total pressure for a specific flow rate increasing. The increased pressure created by closing the damper results in a change in the system

characteristic curve and a corresponding operating point. Since the fan characteristic curve remains unchanged (constant speed) the operating point moves along the fan curve to the point of lower flow as shown in Fig. 3.33. The power consumed by the fan remains constant and thus the torque produced by the motor driving the fan remains constant as shown in Fig. 3.33. The main point to consider when using a fan running at fixed speed is that the power drawn by the fan remains independent of the damper settings. Due to this, fan system using dampers become very inefficient at low flow rates. It is therefore uneconomical to operate a fan at fixed speed in a system which requires large variations in flow rate. The fan will place 95Nm of torque on the IM in the test bed system when running at 1500rpm.

3.4.2 Flow Control Using Speed Regulation

For a comparison to be made between damper control and speed regulation the load that a VSD experiences while driving a fan using regulation to control the flow rate has to be determined. This method of flow control is achieved by reducing the fan speed; when the fan speed is reduced the fan characteristic curve changes, while the system characteristic curve remains unchanged, the operating point for the fan in the system occurs at the intersection of the new fan characteristic curve and the system curve, as discussed in Chapter 2.

The fan speed required to produce a specified flow rate can be gained from Fig. 3.33 and is shown in Fig. 3.34. This curve can be approximated by a straight line given by Eq. 3.10.

$$n_f = \frac{2000}{12} Q \tag{3.10}$$

where n_f is the required fan speed in rpm

Q is the desired volume flow rate in m^3/h.

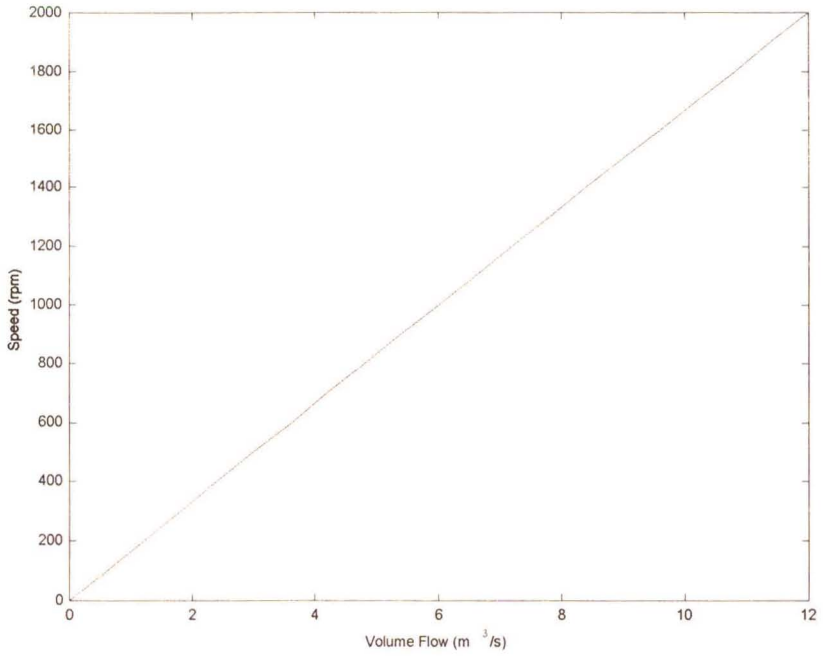


Fig. 3.34 Speed require to produce a specified flow rate for fan system

The power consumed by the fan, as the speed changes, is shown in Fig. 3.35. From Fig. 3.35 and Eq. 2.5 the torque speed curve shown in Fig. 3.36 can be calculated, furthermore the torque speed curve can be estimated by Eq. 3.11.

$$T = \frac{176}{(2000)^2} n_f^{\,2} \tag{3.11}$$

where T is the Torque required to drive the fan

n_f is the fan speed in rpm.

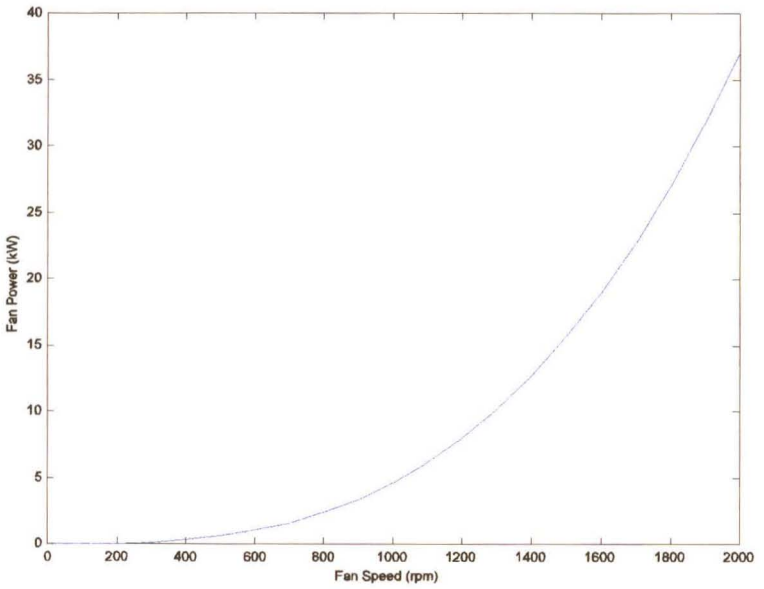


Fig. 3.35 Required fan power vs fan speed for fan system

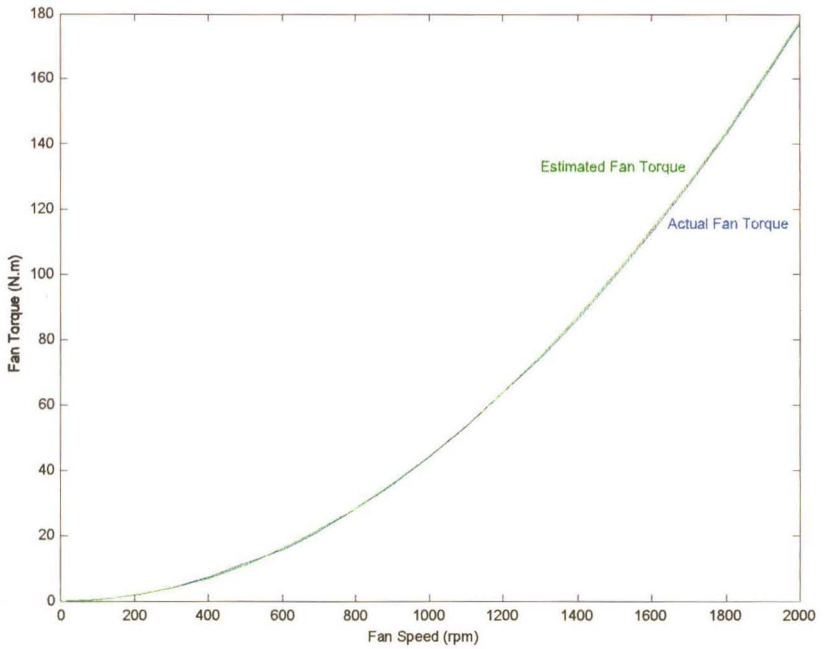


Fig. 3.36 Fan shaft torque vs fan speed for fan system

The test case for a fan in which dampers and speed regulation are used to control the flow rate is used in Chapter 5 when implementing the test bed controller.

3.5 Conclusion

The theory for a separately excited DC motor discussed in Chapter 2 was used in setting up a model of the 40kW DC motor used in the test bed system. The effect of applying PWM switching waveforms onto the armature of the DC motor was investigated and used to select inductors to reduce the ripple current. A PI armature current controller was designed in the frequency domain, whereafter it was simulated and the response was found to be suitable for the test bed system (had no overshoot and responded rapidly). The PI controller was then transformed into the discrete time domain and simulated with the PWM inverter and A/D converters and was found to be suitable for use in the test bed system. The PI controller gains designed in this chapter are used in Chapter 5 when implementing the test bed system software and the simulated results are compared to the practical results attained in Chapter 6.

In order to simulate a pump or fan on the test bed system its torque/speed or torque/flow rate curves had to be determined. Two pump systems were presented in which the flow rate was controlled by discharge throttling or speed regulation. From the two pump systems the torque speed curves were determined for the pump operating under discharge throttling. The operation of the pump when speed regulation is used to control the flow rate was also investigated upon which a curve relating the flow rate to the speed and a curve relating the torque to the speed was determined. A fan system was presented where the flow rate was controlled using both dampers and speed regulation. For the fan system where dampers were used to control the flow rate it was found that the fan produced the same torque irrespective of the flow rate. A fan system in which speed regulation is used to control the flow rate was then presented where a curve relating the flow rate to the speed and a curve relating the torque to the speed was presented. The pump and fan load torque curves are used in Chapter 5 when implementing the test bed system software.

Chapter 4

Test Bed Construction and Interfacing

4.1 Introduction

In Chapter 2 and 3 the characteristics of pumps and fans and their respective load characteristics were described. These load characteristics are required to control the DC motor such that it loads the AC VSD when evaluating the power usage of the AC VSD when driving a pump or fan. The AC VSD in the test bed system shown in Fig. 4.1 is loaded by the DC motor, which is controlled as a configurable load. Before any pump or fan simulations could be performed, the test bed system hardware had to be constructed. A block diagram of the test bed system is shown in Fig. 4.1, and consists of nine main components. The AC VSD comprises of two components: the REFU drive and the induction motor. The remaining components, the DC motor, H-Bridge inverter, PC, PC32 card, PC32 Card Analog interface, PC32 card digital interface and PWM/tachometer card constitute the configurable load.

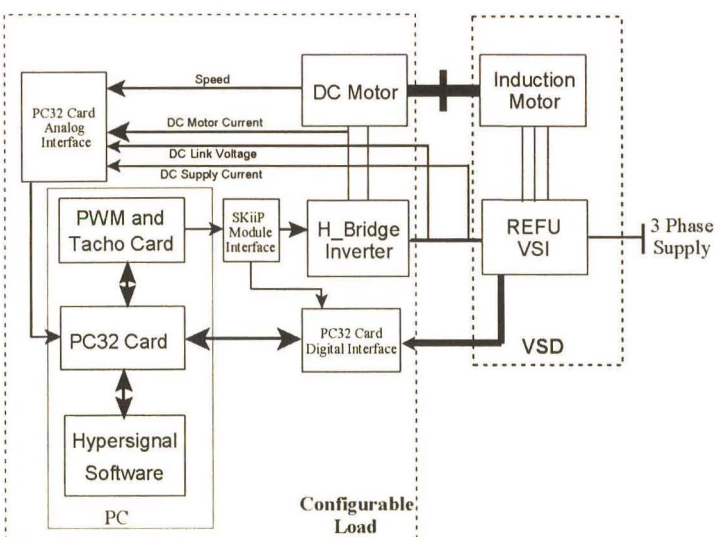


Fig. 4.1 Block diagram of test bed system

A photograph of the test bed system, excluding the PC and signal conditioning circuitry, is shown in Fig. 4.2. The induction motor, with associated resolver, is shown on the right, and the DC motor with associated analogue tachometer is shown on the left. The REFU drive is shown in the right hand side of the cabinet, on the left of Fig. 4.2. The DC drive consisting of a SKIIP module (H-bridge inverter) and SKIIP module interface is housed in the left hand side of the cabinet next to the REFU drive.

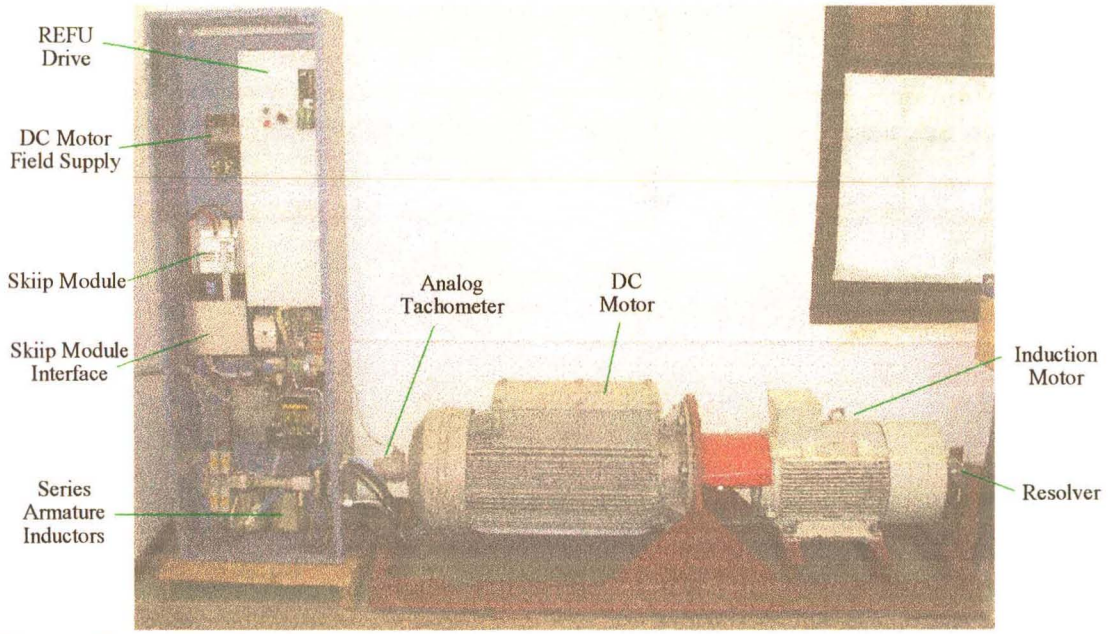


Fig. 4.2 Photograph of test bed system.

This chapter describes the individual components that constitute the test bed system, as shown in Fig. 4.1. The construction of the framework on which the DC motor and induction motor are mounted is described, whereafter the installation of the REFU drive is described. The operational principles of the PC32 card are described as this determines the requirements for the PWM/tachometer card. The design and operation of the PWM/tachometer card are described, whereafter the operation of the DC drive, in which the PWM/tachometer card is used, is discussed. For the analogue signals in the test bed system to be measured by the PC32 card an analogue interface circuit had to be constructed to ensure that the signals from the various analogue transducers were terminated and filtered correctly. A digital interface for the PC32 card

was constructed such that the PC32 card could monitor both the REFU drive and H-Bridge inverter for any faults. The earthing strategy employed in the test bed system is discussed, as a well earthed system ensures that the electrical noise induced in the signal cables is kept as low as possible.

4.2 Motor Framework

The framework which holds the DC motor and induction motor had to be constructed such that both the motors were supported in a position that would allow their shafts to be directly coupled. The DC motor used on the test bed system is flange mounted. Although this method of mounting is not ideal, it was used because this motor was available and fulfilled the requirements for the configurable load. A foot-mounted induction motor was purchased. The frame work for the test bed system is made from 50 by 100mm channel iron. The DC motor is held in place by a vertical plate which is supported by four ribs, as shown in Fig. 4.2. These ribs ensure that the force from the weight of the DC motor is spread over the test bed framework. The induction motor is mounted on two pieces of channel iron placed across the framework. The channel iron on which the induction motor was mounted was machined at the point where the induction motor is mounted so that the DC motor and induction motor shafts line up. The DC and induction motor shafts are directly coupled by a stiff coupling. A metal cover which covers the rotating components (coupling on the DC and induction motor shafts) was constructed and fixed to the induction motor case and the plate holding the DC motor.

4.3 FOC Drive Installation

The AC VSD consists of a commercially-available, 40kVA, FOC inverter manufactured by REFU, which produces sinusoidal current waveforms and achieves speed feed back by means of a resolver [33]. The REFU drive supplies power to a 30kW, four-pole, Siemens motor, as shown in Fig. 4.2. The resolver used has the same number of poles as the induction motor to ensure that the correct signals ($\sin\theta$ and $\cos\theta$) are generated. The supply cables from the inverter to the induction motor are run in Copex tubing, which has a metal jacket for the reduction of EMI noise.

The output phase-rotation of the FOC inverter had to be noted, to ensure that the induction motor rotated in the correct direction, ensuring that the correct sign of the feed back signal was received by the REFU drive speed control loop. Both the local and remote stop and start buttons, were wired to enable the user to control the REFU drive from either the panel, or the remote interface, as shown in Figs 4.2 and 4.18 respectively. A wiring diagram of the external connections to the FOC inverter is given in Appendix E.

The speed reference for the drive can be derived from a ten turn potentiometer mounted on the drive, or via a remote speed set point generated by the test bed controller. A selector switch allows the user to determine which set point is used in the system (as shown in Appendix E Fig.E.2). The setup and commissioning of the REFU drive is discussed in Chapter 6.

4.4 PC32 DSP Card

The PC32 card is the digital control platform on which the test bed controller software is implemented. The PC32 card is a low cost DSP card available from Innovative Integration . The PC32 card occupies an ISA slot on the host PC's motherboard and is based upon the Texas Instruments (TI) TMS320C32 digital signal processor [18], which is capable of a sustained performance of up to 30 million instructions per second (MIPS) [45]. A photograph of the PC32 card is shown in Fig. 4.3 and the block diagram of the PC32 card is shown in Fig. 4.4.

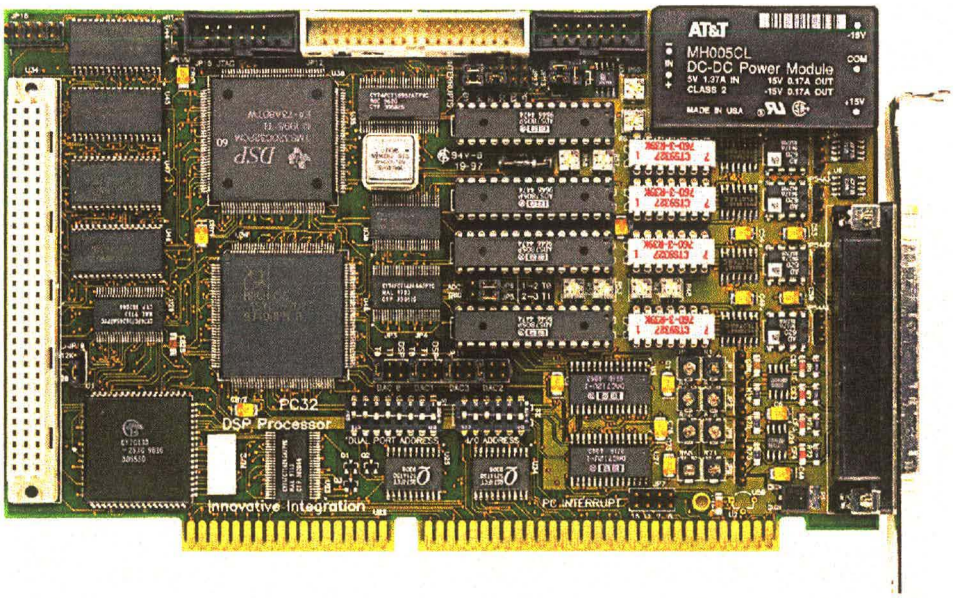


Fig. 4.3 PC32 card

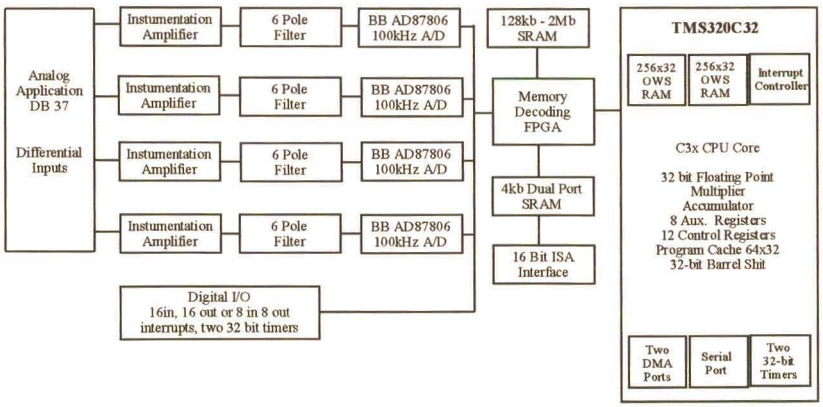


Fig. 4.4 PC32 card block diagram [18].

The main components of the PC32 card, as shown in Fig. 4.4 [18], are:

- **ISA Bus Interface**

  The bus interface provides the host PC access and control of the PC32 card's peripherals and memory resources via the TMS320C32 processor.

- **PC32 Memory**

  There are a number of different memory types available on the PC32 Card for various functions. The card supports up to 512 kWords of program SRAM, up to 256 kWords of data memory and 1k of 32 bit wide dual port RAM, which can be accessed by the PC32 Card and the PC (non simultaneously), and is used for inter-processor data transfers.

- **Digital I/O port**

  This is a 16 bit wide parallel I/O port, which is TTL compatible and capable of 64mA sink and 48mA drive. The digital I/O port can be configured as 16 outputs, 16 inputs or as a combination of 8 inputs and 8 outputs.

- **Analog to Digital Converters**

  The PC32 Card has four sixteen bit analog to digital converters, which can convert an analog signal in 10μs. The incoming signal is internally sampled by the A/D converter prior to a conversion. The default input range is ±10V and passes though a 6 pole anti-alias filter with a cut off frequency of 50kHz. The start conversion of the A/D converters may be triggered by either writing to the A/D converter, or externally by TTL signals on the Analog I/O connector or by one of the two timers in the TMS320C32 processor [18]. The start conversion signals are logically ORed with the memory mapped address, this enables memory mapped start conversions and timer triggered conversions to be mixed. The A/D converters have an end of conversion signal, which signals the TMS320C32 processor, via one of the interrupt pins, that the A/D converter can be read.

- **Digital to Analog Converters**

  The PC32 Card has four independent sixteen bit D/A converters capable of updating at 200kHz with a voltage swing of ±10V. The D/A converters are double buffered, therefore the value written to the D/A converter address is only output when an update output signal is received. The update output signal can be generated by a memory write to the required location or by a trigger signal from one of the timers on the TMS320C32 processor.

- **DECODES0 and DECODES1 address Lines**

  Two areas within the I/0 address range of the TMS320C32 processor are further decoded to enable users to add interface or daughter cards to the PC32 Card. These two decoded areas are referred to as DECODES0 and DECODES1, whose memory locations are logically 16 bits wide. The DECODES0 and DECODES1 lines and the TMS320C32 control lines are also available via a 96 pin expansion header shown in Fig. 4.3.

- **PC32 Card Interrupts**

  There are four interrupts available on the PC32 card, EI0 to EI3. The interrupts have been configured as shown in Table 4.1

Table 4.1 Interrupt Sources for PC32 Card

Interrupt	Source
EI0	Spare - used for PWM/Tachometer Card
EI1	AD converter pair 0 and 1 complete
EI2	PC interrupt 1 to DSP
EI3	C interrupt 0 to DSP

The PC32 card interrupts are used extensively when implementing the test bed controller software which is run synchronously with the interrupt generated by the PWM/Tachometer card.

4.5 PWM/Tachometer Card

The PC32 card used in the test bed system provides the host processor and the analog I/O required for system control, but does not have any means to control the switching of the power electronic devices in the H-Bridge inverter described in Chapter 3. The PWM/tachometer card, shown in Fig. 4.5, was developed to enable the PC32 card to control the switching of the power electronic devices through the use of a PWM control ASIC, and to gain speed or position information from an encoder via a tachometer control ASIC.

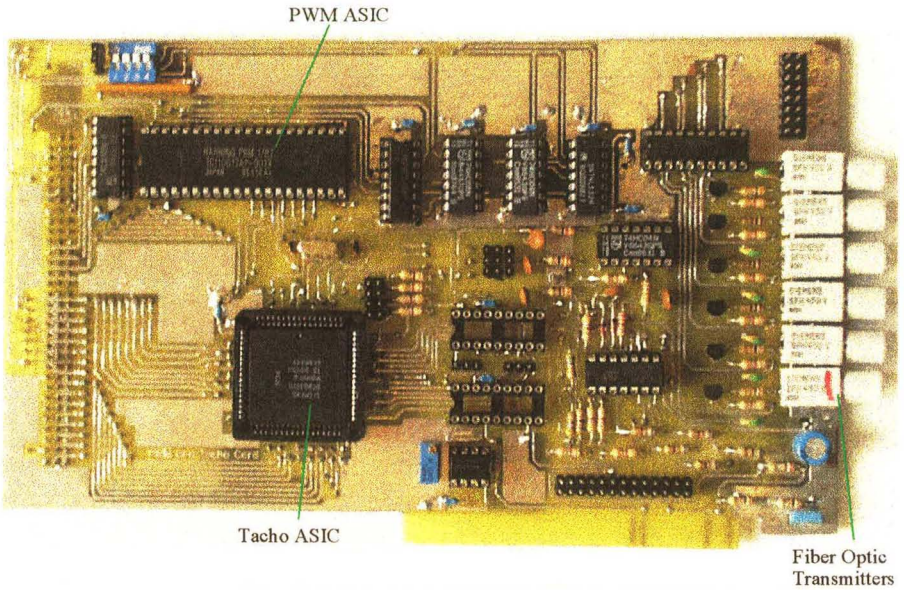


Fig. 4.5 PWM/tachometer card.

The PC32 card and PWM/tachometer card occupy two ISA slots as shown in Fig. 4.6. Power for PWM/tachometer card is taken from the ISA slot, ensuring that both the PC32 card and PWM/tachometer card have a common supply and ground.

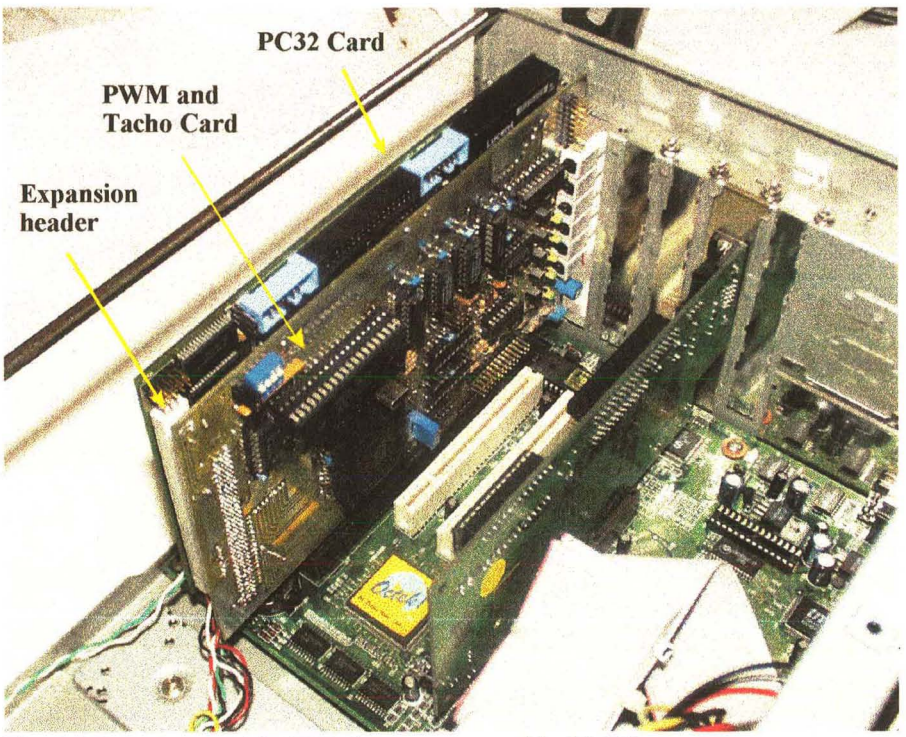


Fig. 4.6 Orientation of the PWM/tachometer card inside PC.

The main components of the PWM/tachometer card are shown in the block diagram in Fig. 4.7. The expansion header on the PWM/tachometer card allows the card to plug into the PC32 Card as shown in Fig. 4.6. The address decoding circuitry enables the PC32 card to communicate and control the various peripherals on the PWM/tachometer card. The fibre optic transmitters provide a convenient and safe connection between the PWM/tachometer card and the power electronics. The tachometer interface enables the PWM/tachometer card to be configured such that different tachometer configurations can be implemented.

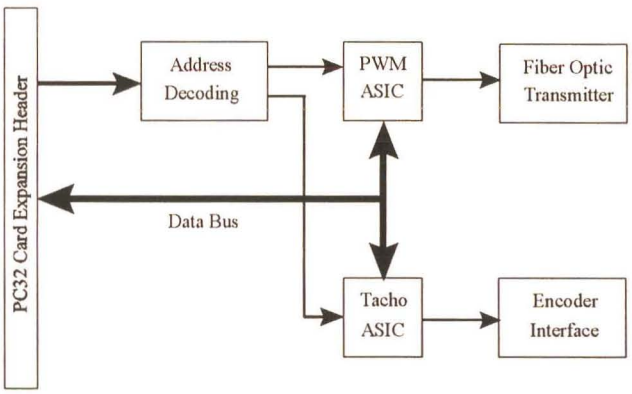


Fig. 4.7 Block diagram of the PWM/tachometer card

An overview of the operational principles of the PWM ASIC is provided, whereafter the address decoding circuitry and fibre optic interface are described, which enables an understanding of the operation of the PWM/tachometer card to be gained. The tachometer ASIC and encoder interface are presented in Appendix D, as these components are not used in the implementation of the test bed system, they were added to the PWM/tachometer card to ensure that it met the need of a complete motion control card for the Motion Control Research group.

4.5.1 PWM Controller ASIC

The PWM ASIC used on the PWM/tachometer card, is the PBM 1/87 manufactured by Hanning Elektro-Werke [16]. The PWM ASIC is a peripheral, which generates the switching signals for

a three phase inverter. This relieves the host processor from intensive processing and time critical calculations required when controlling the switching devices in power electronic drives.

The PWM ASIC is a three phase pulse width modulator, which is capable of generating the switching signals to produce a sinusoidal supply at the required voltage, phase and frequency, with presetable switching frequencies up to 20kHz. Due to these features the host processor only has to supply the PWM ASIC with the magnitude of the voltage, frequency and phase required. A block diagram of the PWM ASIC showing its primary components is shown in Fig. 4.8 [16].

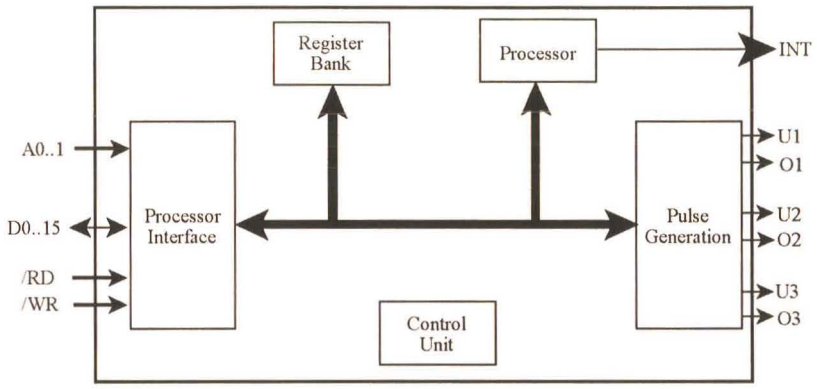


Fig. 4.8 Block diagram of PBM 1/87

The main components of the PWM ASIC are:

- **Processor Interface**

  The host processor interfaces to the PWM ASIC via a 16 bit data bus and a two bit address bus. Data is transferred to and from the PWM ASIC by writing to and reading from specified registers in the register bank, as described in Appendix D.

- **Pulse Generation**

  The switching pulses controlling the devices in an inverter are output from the pulse generation module. The output pulses are determined by the data written to the register bank by the host processor.

- **Processor**

  This controls the operation of the PWM ASIC and calculates the switching pulses that the pulse generation unit must output. The processor also produces an interrupt which indicates when data can be written to the resister banks.

The output pulses produced by the PWM ASIC are determined by the data supplied by the host processor. The method in which the PWM ASIC calculates the switching pulses is described below, as this is information is required to understand the procedure followed in enabling the PWM ASIC to control the DC drive.

The standard connection of the PWM ASIC to a three phase inverter supplying a three phase machine is shown in Fig. 4.9 (the fly back diodes present in an actual inverter are not shown for simplicity). The six output signals from the PWM ASIC are TTL compatible and there is a need for some form of interface between the PWM ASIC and the power electronic devices.

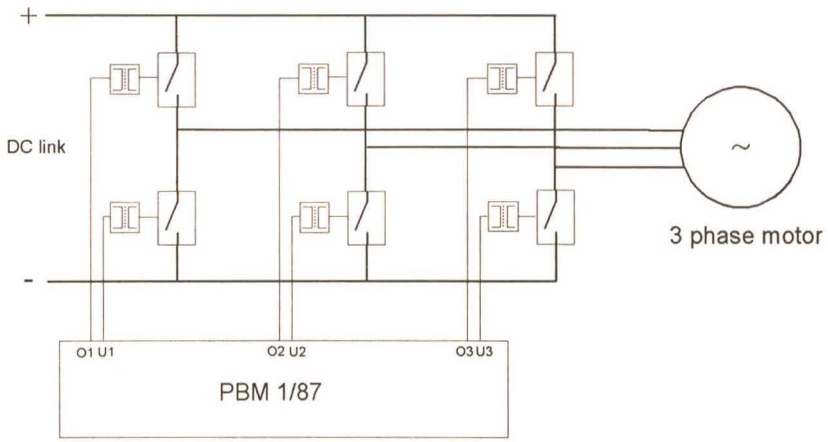


Fig 4.9 Connection of PBM 1/87 to three phase inverter

The PWM ASIC implements triangular modulation with switching pulses positioned symmetrically around a switching period. Fig 4.10 shows the modulation switching waveforms and the interrupt signal produced by the PWM ASIC.

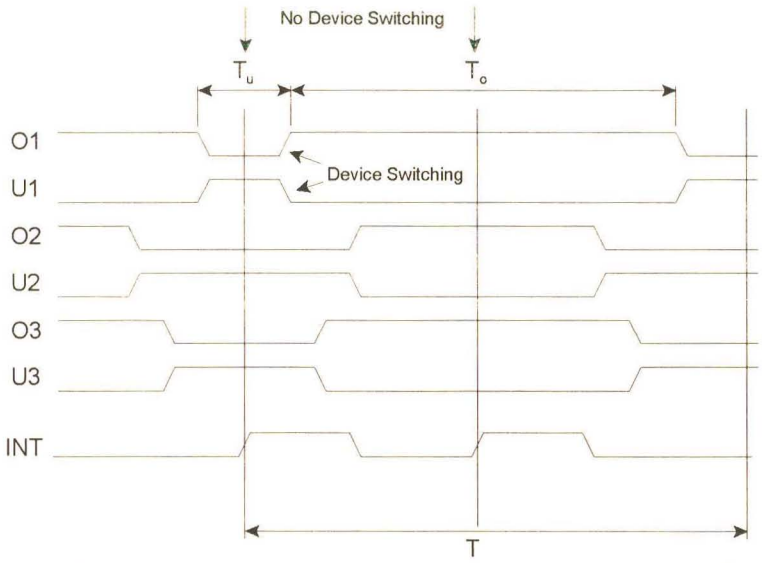


Fig. 4.10 Switching waveforms and interrupt signal from PWM ASIC

Fig 4.11 shows a representation of the voltage vectors which are supplied to the PWM ASIC.

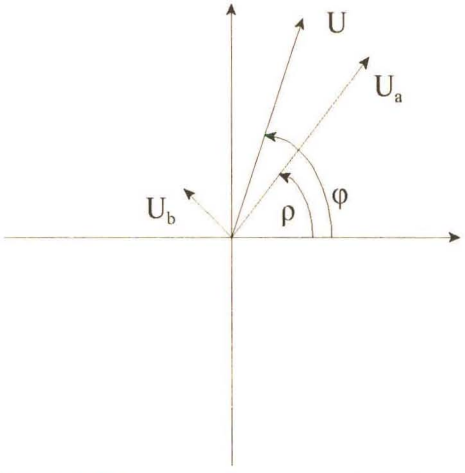


Fig. 4.11 Voltage vector representation

The voltages U_a, U_b and their associated phase angle ρ may be written to the PWM ASIC, in the following formats :

- α/β components where $U_a = U_\alpha$, $U_b = U_\beta$ and $\rho = 0$
- D/Q components where $U_a = U_d$, $U_b = U_q$ and $\rho = \varphi$
- Polar co-ordinates where $U_a = U$, $U_b = 0$ and $\rho = \varphi$

However, irrespective of the format in which the voltage vectors are written, they are internally transformed to α/β components.

U is the magnitude of the required output voltages and is calculated from Eq. 4.1.

$$U = \sqrt{(U_a)^2 + (U_b)^2} \tag{4.1}$$

where U_a and U_b are the magnitudes of the voltage vectors written to the PWM ASIC by the host processor.

The phase angle of the output voltage may be calculated using Eq. 4.2

$$\varphi = \rho + d\rho + \arctan\left(\frac{U_b}{U_a}\right) \tag{4.2}$$

where: ρ is the phase angle entered by the host processor

$d\rho$ is the incremental angle added each switching cycle to produce the output frequency

the arctan term is the phase angle between the voltage vectors U_a and U_b.

New values of U_a and U_b need to be supplied to the PWM ASIC after each processing cycle, otherwise the following conditions are applied to the next processing cycle.

$U_a = U$

$U_b = 0$

The normalized output voltage of three phase waveforms generated by the PWM ASIC with respect to the phase angle is shown in Fig. 4.12. The voltage waveforms generated by the PWM ASIC has a third harmonic injected into it which increases the efficiency of the inverter [16]. The

effect of the third harmonic on the output waveform can be seen from Fig. 4.12 were it deviates from a true sinusoidal waveform.

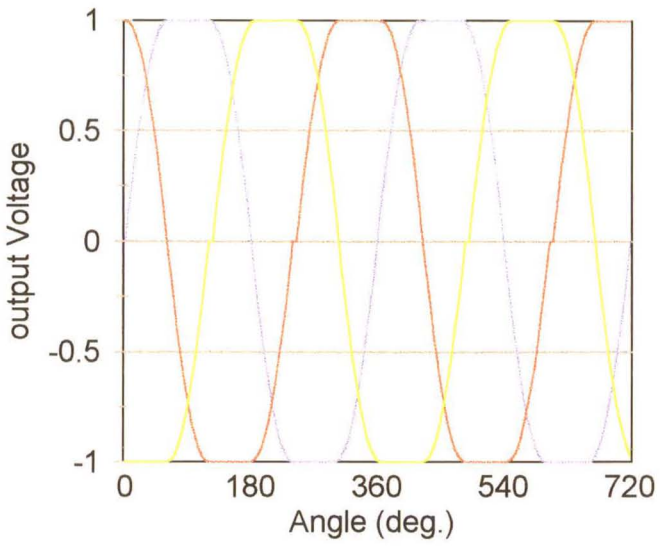


Fig. 4.12 Output voltage of the PWM ASIC as φ is varied [16].

Fig. 4.10 shows that the positive going edge of the interrupt produced by the processor unit, on the INT pin, of the PWM ASIC occurs at the centre of the switching pulses, and occurs twice per switching cycle or every half cycle. This interrupt indicates the start of the period in which data may be written to the PWM ASIC for use in the next calculation cycle. Fig. 4.10 also shows that no switching transitions take place at the rising edge of the interrupt pin so that if the sampling of the A/D converters are controlled by the INT signal the sampled values will be void of any switching noise. This technique is commonly known as synchronous sampling in AC drives. Once the A/D conversion is complete this data can be used by the controller to calculate the next output value to be sent to the PWM ASIC.

4.5.2 PWM/Tachometer Card Address Decoding

Address decoding is required on the PWM/tachometer card to enable the PC32 card to control the respective devices on the PWM/tachometer card. The address decoding on the PWM/tachometer card uses partially address decoded lines already provided on the PC32 card

in the form of the DECODES0 and DECODES1 lines. The DECODES0 and DECODES1 lines determine the base address for the PWM/tachometer card.

There may be a need to connect other interface cards or PWM/tachometer cards to the PC32 card, therefore further address decoding is performed on the PWM/tachometer card to enable more than one card to be connected to the same DECODES0 or DECODES1 line. This further address decoding gives the card an offset address which is selectable by the use of a dip switch and makes provision for eight cards to be connected to a single decodes line (DECODES0 or DECODES1). Both ASICs are operated in 16 bit bus mode to get maximum performance from both the TMS320C32 processor and the ASICs. Appendix D gives the relevant information required to calculate the address of the peripherals on the PWM/tachometer card.

4.5.3 PWM/Tachometer Card Fibre Optic Interface

Fibre optic links are unaffected by EMI and RFI noise and therefore provide a more reliable link than wire links in most industrial applications and are being used more frequently. The largest drawback of using fibre optic cables is the cost of the transmitters, receivers and fibre optic cable itself. A plastic fibre optic link was used with the PWM/tachometer card. This link provides isolation between the PWM/tachometer card and the power electronic drivers, thereby preventing any possible problems with ground loops. Noise cannot be introduced into the fibre optic cables as with copper cables, and plastic fibre optic is easy to work with as it does not need to be polished after it has been cut, and can be cut with a clean sharp knife. The largest drawback with this type of fibre optic cable is that it is limited to a maximum length of fifty metres due to the internal losses in the fibre.

The fibre optic transmitters are mounted on the PWM/tachometer card and are controlled by the six output signals from the PWM ASIC, via a driver transistor. The fibre optic link also enables the PWM/tachometer card to interface with power electronic control circuits which operate at different voltage levels.

4.6 DC Drive

The DC drive power electronics consist of two main components: the SKIIP module and the SKIIP module interface as shown in Fig. 4.2. The power electronic devices and their drivers are contained in the SKIIP module, which is connected to the DC link of the REFU drive. Connecting the DC drive to the DC link of the REFU drive enables the DC Drive to generate power back into the DC link when loading the induction motor in the test bed system, as shown in Fig. 4.1. The SKIIP module interface supplies power to the SKIIP module drivers and the switching signals which are derived from the PWM/tachometer card and fed to the SKIIP module interface via fibre optic cables. The PWM ASIC on the PWM/tachometer card is designed to control the switching devices in a three phase AC drive, but by manipulating the data supplied to the control registers of the PWM ASIC, it can be used to control a DC drive. The method used in ensuring the PWM ASIC controls the DC drive is discussed, whereafter the SKIIP module and SKIIP module interface are discussed.

4.6.1 Using the PWM ASIC for a DC Drive

Although the PWM ASIC is used primarily for controlling three phase inverters it may be used to control an H-Bridge, as shown in Fig. 4.13.

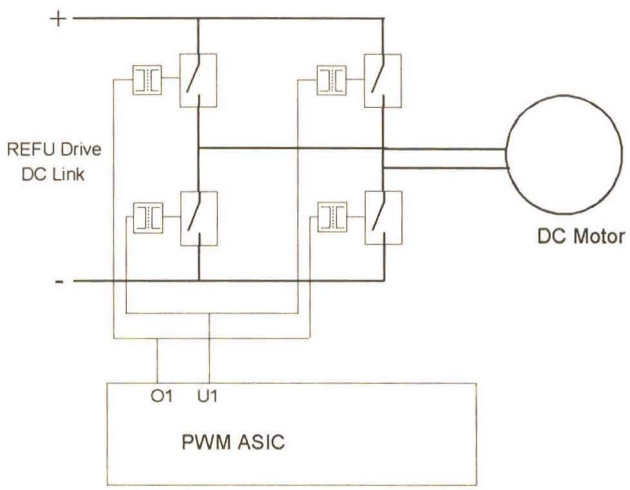


Fig. 4.13 PWM ASIC Control of H-Bridge Inverter

If the output signals O1 and U1 are used to control the switching devices in the H-bridge inverter (as in the case of the DC drive in the test bed system) as shown in Fig. 4.13, then the voltage applied to the DC motor would be a sine wave of magnitude U, where Eq. 4.1 determines the magnitude of U and Eq. 4.2 determines the frequency. If $d\rho$ (the incremental angle) was set to zero then the output of the inverter would be a DC voltage determined by the angle φ as given in Eq. 4.2 and the magnitude of the voltage vector U. The output voltage produced by the PWM ASIC as φ is varied is shown in Fig. 4.12 (assuming U remains fixed). To ensure that the DC voltage produced by the PWM ASIC is equal to U the angle φ must be set to 90° when using switching signals O1 and U1. Since the angle φ is also determined by the inputs U_a and U_b, the input variables to the PWM ASIC may be set as follows: $U_b = 0$, $\rho = 90°$, $d\rho = 0$ and $U = U_a$. This ensures that the phase angle remains fixed at 90° and the output voltage of the inverter is controlled by U_a which is fed from the PI armature current controller (see Appendix D.1 for parameter calculations). The output voltage of the H-Bridge inverter can be varied between positive DC link voltage and negative DC link voltage, by varying U_a between positive one and negative one respectively.

4.6.2 SKIIP Module Functions and Operation

The SKIIP module used in the test bed system is the SKIIP 232 GH 120-210-CTV manufactured by Semikron. The power electronic devices and drives are supplied in a single module mounted on a heat sink, as shown in Fig 4.14. The SKIIP module consists of two main components: the power electronic devices and the device drivers, as shown in Fig. 4.15 [39].

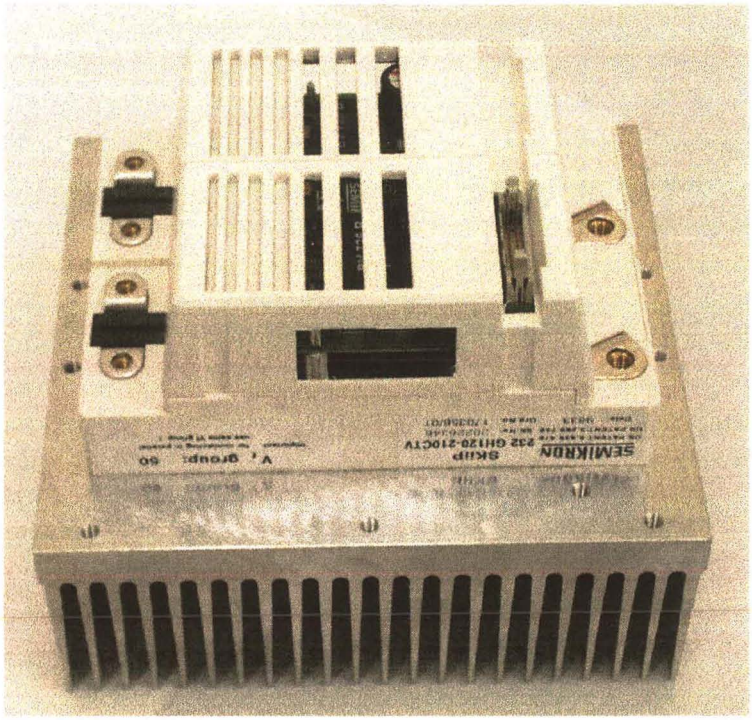


Fig. 4.14 Photograph of the SKIIP module

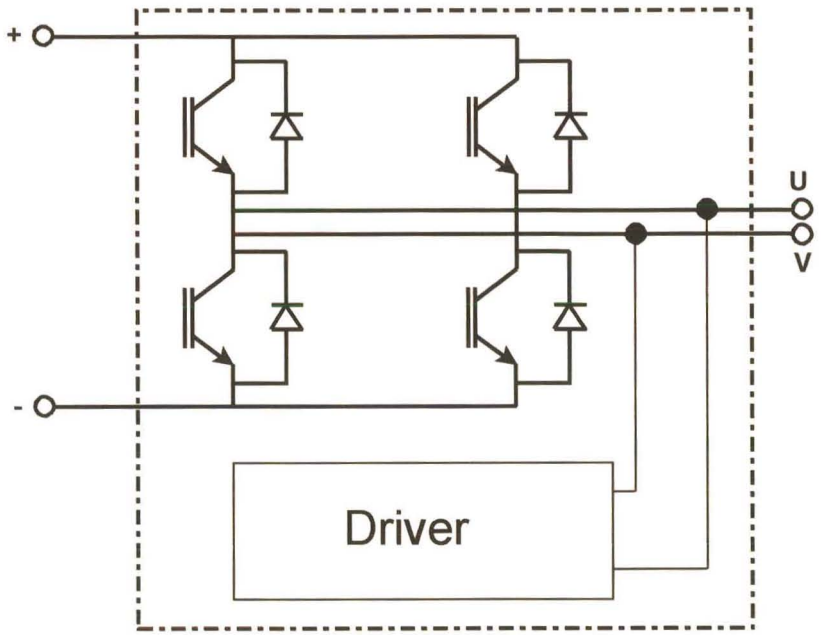


Fig. 4.15 Block diagram of SKIIP module

The SKIIP module contains four IGBTs in an H-Bridge configuration with fly-back diodes across each device, as shown in Fig. 4.15. The power electronic devices are housed in a single package, reducing stray inductance and thereby alleviating the need for snubbers. The driver circuitry is used to control the switching of the power electronic devices. This circuitry is mounted above the power electronic package to ensure that the control lines to the IGBTs remain as short as possible. A block diagram of the SKIIP module driver for one leg is shown in Fig. 4.16.

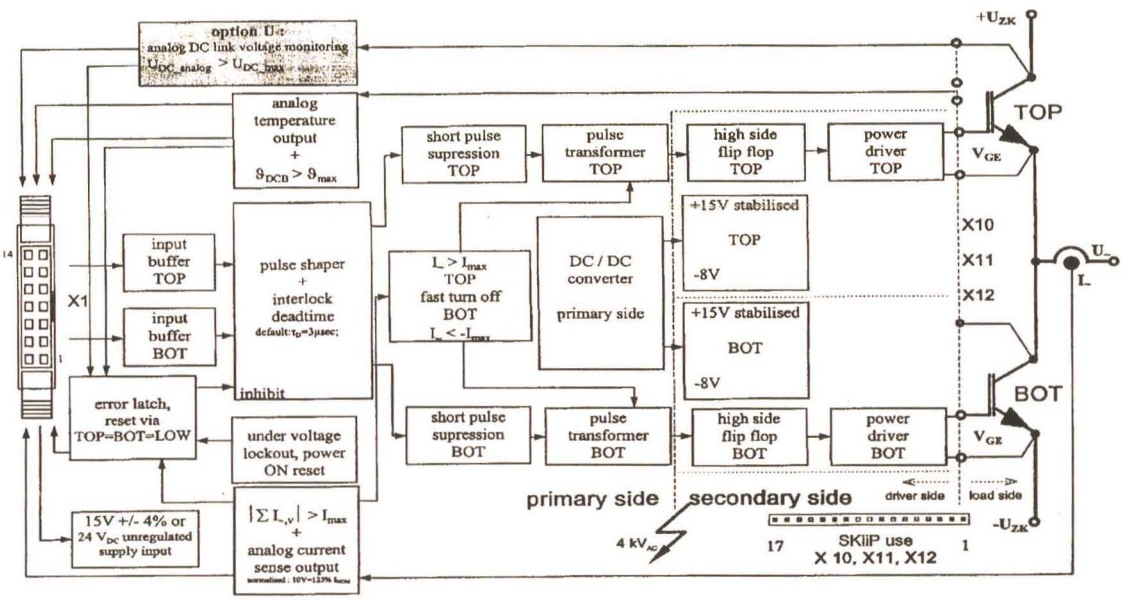


Fig. 4.16 SKIIP module driver block diagram [39].

From the block diagram, shown in Fig. 4.16 [39], it can be seen that the SKIIP module offers the following functions and safety features:

- interlock to prevent both the top and bottom devices being turned on simultaneously;
- dead time generation;
- short pulse suppression;
- power supply under voltage detection;
- isolation between the high-voltage and low-voltage sides of the driver module.

The SKIIP module driver can be supplied by either an unregulated 24V or a regulated 15V supply. The SKIIP module has the following safety features, which when violated set an error latch and disables the switching of the IGBTs:

- over current protection;
- over voltage protection;
- heat sink over temperature protection.

4.6.3 SKIIP Module Interface

For the PWM/tachometer card to control the power electronic devices in the SKIIP module, there is a need for some interface circuitry, which is provided by the SKIIP module interface, as shown in Fig. 4.17.

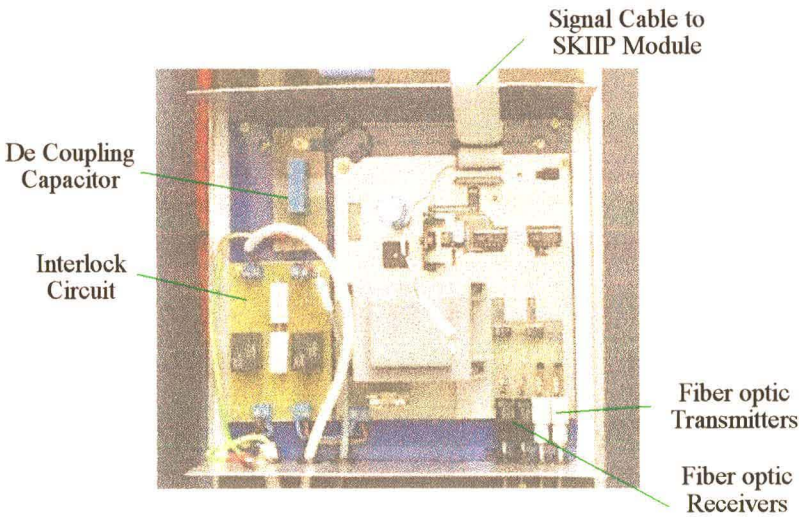


Fig. 4.17 SKIIP module interface circuit

From Fig. 4.17 it can be seen that the SKIIP module interface consists a number of components, namely :

- **Power Supply**

  The SKIIP module Interface has a power supply on board, which supplies power to both the SKIIP module and the fibre optic transmitters and receivers.

- **Fibre Optic Receivers**

  The switching signals generated by the PWM ASIC, on the PWM/tachometer card are transmitted via plastic fibre optic cables. The fibre optic receiver converts these signals to electrical signals which are then fed to the SKIIP module. There are two fibre optic receivers on board, one for the top IGBT in first leg and the bottom IGBT in the second leg, while the other fibre optic receiver controls the bottom IGBT in the first leg and the top IGBT in the second leg.

- **Fibre Optic Transmitters**

  To ensure that there is complete isolation between the SKIIP module and the controller (PC, PC32 card and PWM/tachometer card), fibre optic transmitters are used to transmit error signals back to the PC32 card. The fibre optic transmitters are used to transmit the SKIIP module group fault error and the over temperature error.

- **Interlock Circuit**

  The interlock circuit for the SKIIP module interface ensures that the SKIIP module is only supplied with power if both the signal interface circuitry for the PC32 Card and the field supply for the DC motor are on. This ensures that the DC motor will not be operated while either the field supply is faulty or the interface circuitry to the PC32 card does not have power.

4.7 PC32 Card Analogue Interface

The analogue signals from the various transducers in the test bed system produce signals with differing requirements. All the signals from the transducers in the test bed system are fed to the PC32 card analogue interface. The PC32 card analogue interface amplifies, scales and filters all the incoming analogue signals before they are passed to the PC32 card. A photograph of the PC32 card analogue interface is shown in Fig. 4.18 and its associated block diagram is shown in Fig. 4.19. The functions of the main components of the analogue interface will be discussed, whereafter the measurement transducers used in the test bed system are discussed.

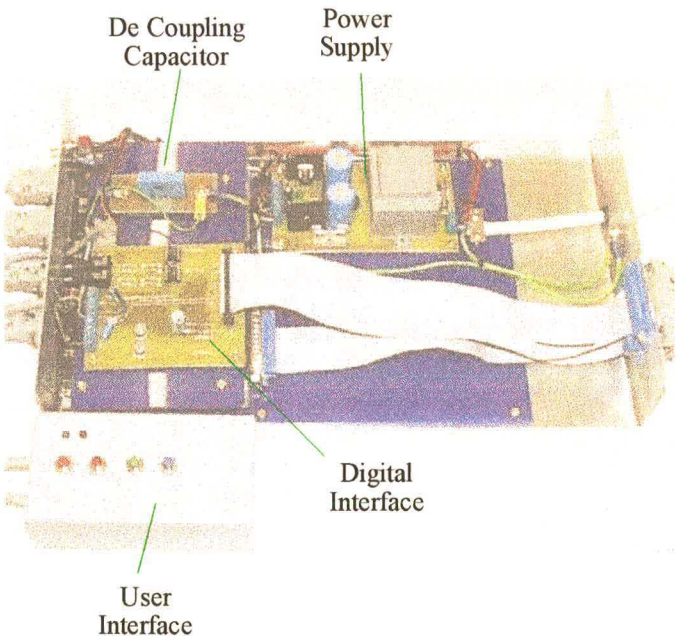


Fig. 4.18 PC32 card interface unit

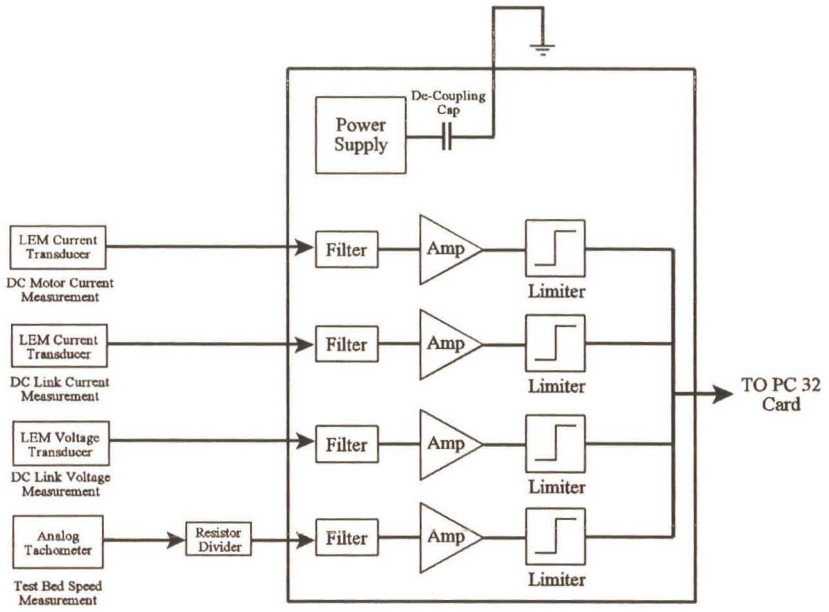


Fig. 4.19 Block diagram of PC32 Card analogue interface.

4.7.1 Analogue Interface Circuitry

The analogue interface circuitry is required to ensure that the signals generated by the measurement transducers are in a range capable of being measured by the PC32 card. The various components of the analogue interface circuitry shown in Fig. 4.19 are :

- **Power Supply**

  The power supply for the PC32 card analogue interface circuitry has two functions: the first is to supply power to the current and voltage transducers; and the second is to supply power to the analogue interface circuitry which filters and amplifies the analogue signals produced by the transducers in the test bed system.

- **Filter**

  All the signals fed to the analogue interface are passed through a first-order filter with a cut off frequency of 5kHz, which reduces any unwanted switching noise. The filter's cut off frequency is sufficiently high that no information is lost, but ensures that the switching noise of both the DC drive and AC VSD are attenuated.

- **Amplifier**

  The amplifier is required because the analogue signals produced by the various transducers have magnitudes less than 10V. By amplifying the various signals the full input range of the A/D converters can be used, which increases the resolution and therefore the accuracy of the measured values.

- **Limiter**

  The limiter limits the output of the amplifier to 9.8V. This is achieved by placing two 9.1V zener diodes back to back at the output of the amplifier (see Appendix E, Fig. E.5). The zener diodes provide over voltage protection for the PC32 card's analogue inputs.

The analogue interface channels were commissioned individually by comparing the values returned from the analogue interface with those measured using the appropriate measuring instruments. This allowed the amplification factors of the amplifications stages on the analogue interface board to be set. Fine tuning of the analogue signals is done in software where any offset errors and small scaling errors can be overcome.

4.7.2 Measurement Transducers

Four analogue transducers are used to measure the system variables required to control the configurable load, and to measure the power usage and efficiency of the AC VSD. The variables which are to be measured, as discussed in Chapter 2, are the DC motor armature current, DC current flowing from the three phase rectifier, the DC link voltage and the speed signal from the

analogue tachometer. The three types of transducers required to measure the above variables are discussed below.

- **Current Transducers**

  Two current transducers are used in the system: one is used to measure the DC motor current while the second is used to measure the current flowing into the REFU drive DC link. The current transducers used are LEM LA205-S which are Hall effect devices capable of measuring up to 200A RMS and produce a current output which is proportional to the current being measured. The LEM LA205-S sensors have an overall accuracy of 0.8% of the full load current being measured and a linearity better than 0.1% [23]. The output current from the LEM sensor is terminated in a burden resistor as shown in Fig. 4.20. The voltage dropped across the burden resistor is proportional to the measured current (current flowing in the current carrying conductor). The burden resistor has an accuracy of 1% (1% accuracy resistor used). To ensure that the highest accuracy was attained the values measured by the software in the test bed system were compared against those measured using test equipment at various current values. The measured values attained by the software were scaled to give the correct values. The calculations for the burden resistors are given in Appendix E.

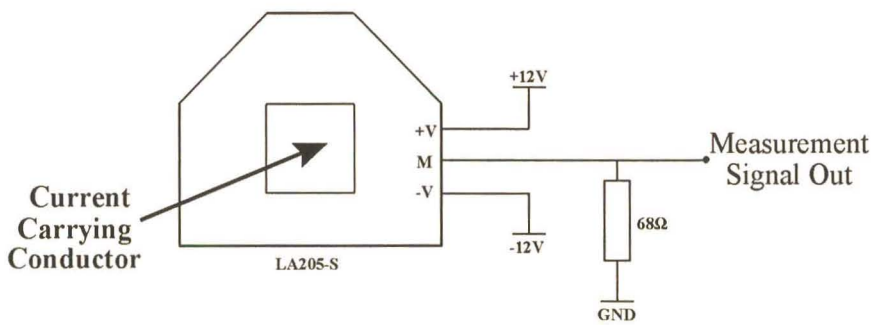


Fig. 4.20 Connection Diagram for LEM Current Sensor

- **Voltage Transducer**

A voltage transducer is used to measure the REFU drive DC link voltage. The voltage transducer used is the LEM LV25-P, which relies on the Hall effect to measure the current flowing through the primary of the device. The device measures voltage by converting the voltage to a current with the aid of a resistor placed in series with the device, as shown in Fig. 4.21. The current flowing through the resistor and LV25-P is proportional to the applied voltage. The output of the LV25-P is fed to a burden resistor which produces a voltage proportional to the applied voltage. The LV25-P has an accuracy of 0.8% and a linearity better than 0.2% [24]. The largest error is due to the resistors used in the system, the resistor used to convert the voltage signal to a current signal has a 5% tolerance and the burden resistor has a tolerance of 1%. Due to the resistors used in the setup of the LV25-P having a large effect on the accuracy of the measured voltage, the value measured by the software package was compared and adjusted in accordance with that of a digital voltmeter. The software package was calibrated at the DC link operating voltage to further reduce any errors that could be caused due to non-linear errors. The calculations for the burden resistors are given in Appendix E.

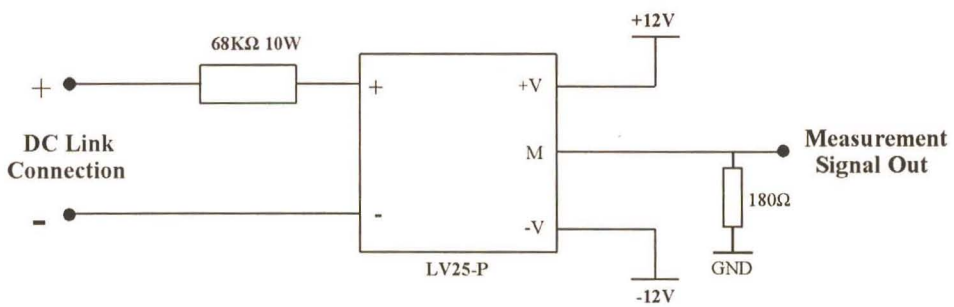


Fig. 4.21 LEM Voltage Sensor Connection Diagram

- **Speed Transducer**

The rotational speed of the test bed system is measured using an analogue tachometer, manufactured by RADIO-ENERGIE and is a type Reo 444-RC1/CA. The tachometer has

an accuracy of 1% over the entire speed range [35]. A resistor divider is used to scale the tachometer voltage so that it can be applied to the input of the PC32 card analogue interface. The calculation for the resistor divider is given in Appendix E. As with the other transducers used in the test bed system the values measured by the software package were compared and adjusted in accordance with the speed measured by a hand held digital tachometer at various speeds.

4.8 PC32 Card Digital Interface

Various components in the test bed system produce digital fault signals. It is necessary for the test bed controller to monitor these fault signals so that the appropriate action can be taken should an error occur. The error signals are applied to the input of the digital I/O port of the PC32 card. As the error signals have varying signal levels there is a need for an interface circuit to convert these signals to the TTL level required by the PC32 card. The block diagram of the PC32 card digital interface along with the associated components that generate the error signals, are shown in Fig. 4.22. An over temperature error and group fault signal are received from the SKIIP module interface. Error signals are also received from the REFU drive group fault signal and the user stop signal (load stop).

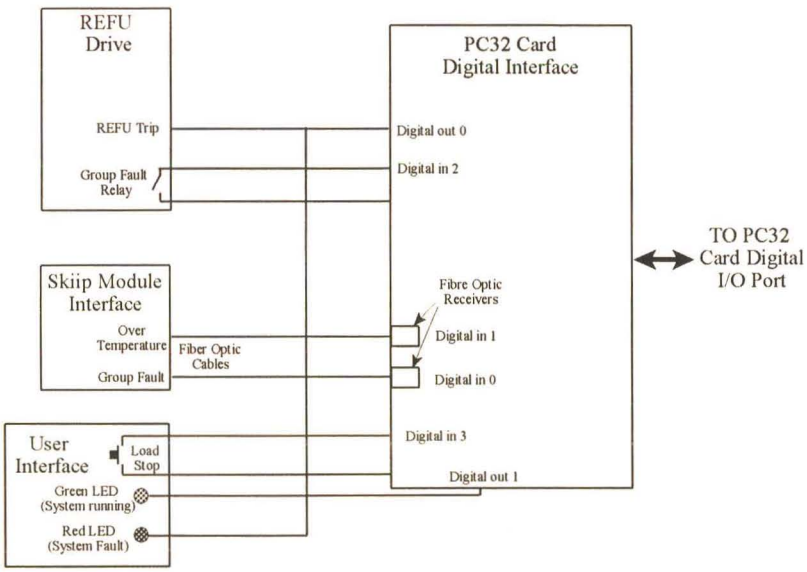


Fig. 4.22 Block diagram of the PC32 card digital interface

The error signals from the SKIIP module are transmitted to the PC32 digital interface via two fibre optic cables where they are converted to TTL signals. The fibre optic link ensures complete isolation between the SKIIP module interface and the PC. The REFU drive group fault and user stop signal are also converted to TTL signals. The TTL input signals from the PC32 card's digital interface are fed to the lower eight bits on the digital I/O port of the PC32 card.

The digital interface also contains two outputs in the form of open collector control lines, which are capable of sinking 800mA. The first output is capable of disabling the REFU drive by activating a relay placed in series with the stop button on the remote interface (see appendix E, Fig. E.2). This output also drives a LED, which indicates when an error has occurred. The second output is the system status LED, which indicates when the DC drive is running. The output signals from the PC32 card's digital interface are fed from the high eight bits on the digital I/O port of the PC32 card.

4.9 Earthing Strategy

A severe problem that can arise with equipment incorporating power electronic devices, is the generation of Electromagnetic Interference (EMI) due to large currents being switched [46]. The EMI generated by power electronic equipment manifests itself in two ways: the first is ground loops, which result in circulating currents flowing in the signal cable shielding and system framework that connect to separately earthed points; and secondly noise being induced into signal lines.

Ground loops may cause damage to components due to volt drops generated between ground points in the loop. This problem was addressed by ensuring that the test bed system was only earthed at one point and that the components in the test bed system were earthed employing a "tree" like earthing structure as shown in Fig. 4.23. It is, however, important to limit the resistance in the earth paths in order to limit the volt drops generated by any earth currents. All earth leads used in the test bed system were kept as short as possible and were made of fine multi-strand cores to limit the impedance to high frequency currents due to the skin effect.

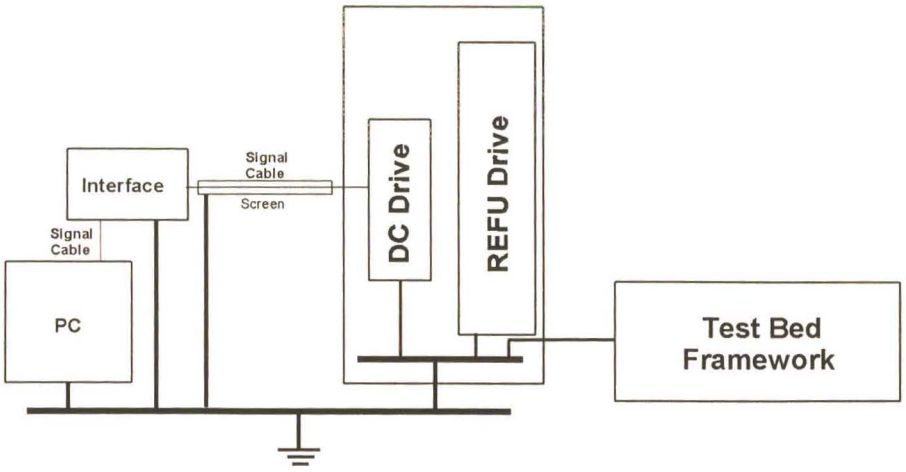


Fig. 4.23 Test bed system earthing strategy

The problems associated with EMI induced on the signal lines is addressed with the use of screened cables. The screened cables are only earthed on the signal return side (PC side) to avoid ground loops. The effects of EMI are further reduced by enclosing all electronic circuits in sheet metal enclosures, to ensure that noise is not induced in any of the electronic circuits which may result in either damage to the circuit or poor operation. The electronic circuit boards are designed with large ground planes and are decoupled to earth via a decoupling capacitor. The decoupling capacitor provides a low impedance path to earth for any EMI induced on the electronic circuit board.

4.10 Conclusion

A functional overview of the test bed hardware was presented, with an in depth discussion of the PC32 and PWM/tachometer cards which form the digital controller for the test bed system. The method by which the PWM ASIC is used to control the inverter in the DC drive was discussed, as this information is vital when setting up the software for the test bed system. The hardware and interface circuitry forms the core of the test bed system, yet remains transparent to the test bed controller, which is discussed in Chapter 5. The commissioning of the various components in the test bed system is discussed in Chapter 6.

Chapter 5

Test Bed Controller Software

5.1 Introduction

In Chapter 3 the loads that pumps and fans place on VSDs running at fixed and variable speed were discussed, as these load curves are required when evaluating the power usage of VSDs driving pumps and fans. For an AC VSD to be loaded with the pump and fan loads a test bed system consisting of two main sub-systems, namely the AC VSD and the configurable load, was constructed as discussed in Chapter 4. The configurable load comprises the DC motor, DC drive, PWM/tachometer card and PC32 card. The configurable load in the test bed system has to be controlled in such a manner that it can simulate the load of a pump or fan running at either fixed or variable speed. While the AC VSD is driving the pump or fan load its power usage and efficiency is measured. The power usage measurements are used to compare the power required to drive a pump or fan at fixed speed versus that of variable speed.

The PC32 card provides the processor on which the test bed controller is executed. The main functions of the test bed controller is to control the configurable load and to measure the power usage of the AC VSD. A software package which would run on the host PC and allow the test bed controller to be implemented, while providing a user interface for the test bed system was required. Hypersignal's RIDE 4.0 (Real Time Integrated Design Environment) [36] software package was found to be suitable.

This chapter begins by presenting a brief overview of Hypersignal's RIDE software functionality and operation. The requirements for the test bed controller are discussed, whereafter the operation of the test bed controller is discussed giving particular attention to the method in which

the test bed system controller block diagram was implemented in the RIDE software package. The way in which synchronous sampling was achieved in the test bed software is also discussed.

5.2 Hypersignal RIDE 4.0 Software

The RIDE software package was chosen, as it provided a flexible means of rapidly developing the test bed controller. This package also provided an interface between the PC32 card and the host PC, which enables information on the PC32 card to be passed to the host PC, which may be displayed, such that the power usage of the AC VSD in the test bed system can be evaluated by the test bed user.

RIDE is an application program capable of simulating and implementing real time algorithms in a graphical user interface environment, using a library of software building blocks [36]. A short description of the program is given to familiarize the reader with the operation and the functionality of the RIDE software.

5.2.1 Block Diagram Editor

A controller may be configured using the block diagram editor by placing blocks from the relevant libraries in a worksheet and connecting them together, which graphically and physically represents the data flow from one block to the next. The block diagram editor window, and an example controller is shown in Fig. 5.1. The controller shown in Fig. 5.1 is configured by selecting a number of function blocks from the available libraries and placing them in a blank worksheet. The blocks placed in the worksheet are then connected such that the controller performs a desired task. The operation of the controller can be altered quickly and efficiently, by deleting and inserting blocks and by altering their connections.

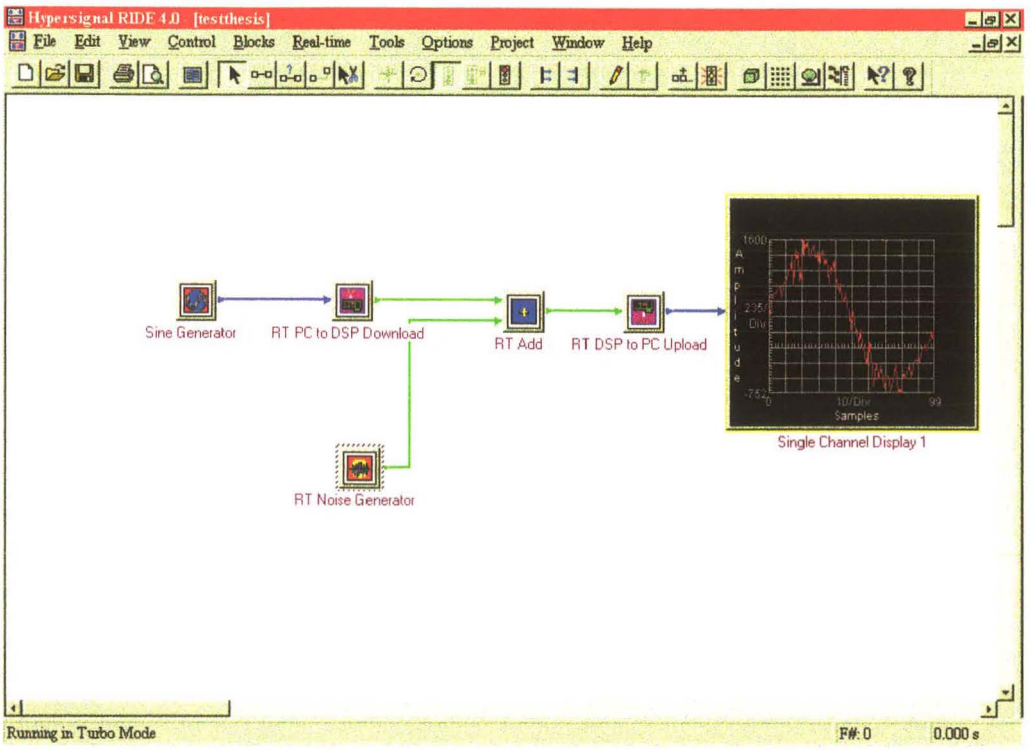


Fig. 5.1 Block diagram editor window with a controller example

Function blocks may be of two types, namely "simulation" blocks which execute on the PC, or "real time" blocks (RT) which execute on the DSP card (PC32 card). The outputs and inputs of simulation and real time blocks cannot be connected directly, as the two block types run on different platforms, and thus data cannot be passed from a simulation block to a real time block. RIDE makes provision for data transfers from real time blocks to simulation blocks with the aid of a *RT DSP to PC upload* block, and a *RT PC to DSP download* block for passing data from simulation blocks to real time blocks. Interconnections between real time blocks (green lines) and interconnections between simulation blocks (blue lines) are shown in different colours to avoid confusion between blocks running on the DSP card and PC respectively.

The ease in which a controller can be implemented can be shown by considering the example shown in Fig. 5.1: here a sine wave signal is generated by the simulation *Sine Generator* block

running on the PC. The data from this block is passed to the *RT PC to DSP Download* block, which enables data to be transferred from the PC to the DSP card. The sine wave data is then added with a signal generated by the *RT Noise Generator* block using a *RT Add block*. The output of the *RT Add* block is passed to a *RT DSP to PC Upload* block, which passes the data back to the PC where it is displayed on a *Single Channel Display*. The RIDE software also allows users to group blocks into sub-systems know as hierarchical blocks, which enables the user to place blocks that perform a specific function together, making the controller easier to visualise in the block diagram editor.

From Fig. 5.1 is has been shown that a controller can be implemented with no coding requirements (provided that the required function blocks are available) by simply connecting the relevant function blocks together in order to achieve the required output. Once the controller has been implemented in the block diagram editor it is ready to be executed. The process by which the controller is executed takes place in two stages: the controller is firstly compiled and linked; whereafter it is executed.

5.2.2 Linking

When the desired controller has been implemented in the block diagram editor it may be compiled and linked. This is achieved by pressing the compile button on the menu bar in the RIDE window. When the controller is compiled a data path is established, whereafter the final linking and execution takes place. The final linking and execution differs for the simulation and the real time blocks due to their structure.

- **Simulation Block Linking**

  Each simulation block consists of a Dynamic Linker Library (DLL), in which the block's function is embedded. The linker makes provisions for the transfer of data from one DLL to the next, as well as ensuring that the correct execution order takes place, so that the correct data is provided to each DLL.

- **Real time Block Linking**

  Real time blocks consist of two components, a RT block DLL which runs on the PC and a RT block DSP code object file which runs on the DSP processor on the PC32 card as shown in Fig. 5.2.

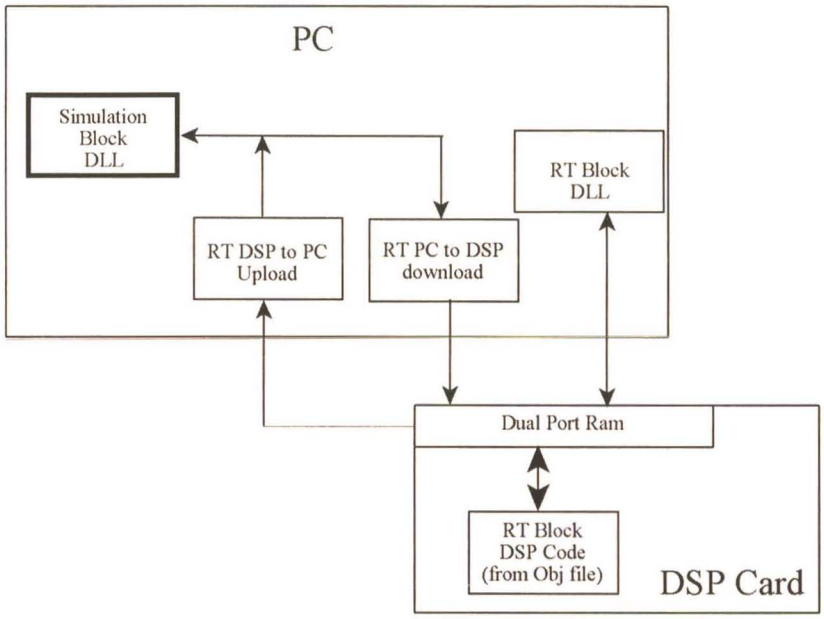


Fig. 5.2 Block diagram showing location of code for simulation and real time blocks

The RT block DLL running on the PC enables block parameters to be changed and transferred to the RT block DSP code via the dual port RAM without having to halt execution. When the controller or worksheet is compiled and linked, the linker has to determine whether the DSP code is executed by hardware or software interrupts or whether the block is to be executed by an internal timer. The linker makes provision for and controls any interrupt requirements, resulting in the operation of the interrupts being transparent to the user. Real time blocks can be connected to an interrupt by selecting the relevant interrupt from the block's parameter box in the block diagram editor as shown in Fig. 5.3. The function block in Fig. 5.3, would run whenever a signal is received on the external interrupt 1 pin of the DSP processor.

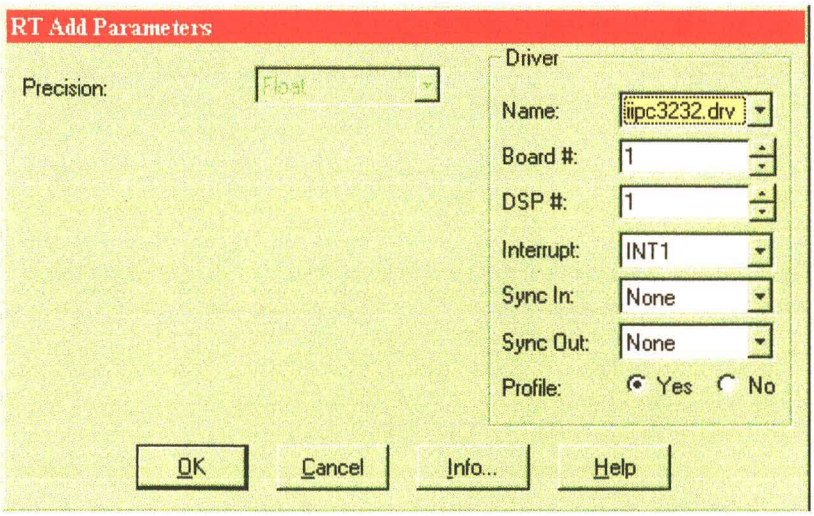


Fig. 5.3 Real time block options showing how block is linked to an interrupt

The compiling, linking and executing of a worksheet remains transparent to the user, making it easier and faster to implement and tune controllers as no code has to be modified.

5.2.3 Creating User Defined Real Time Block Functions

RIDE provides a utility known as the Block Wizard, which enables users to create user defined blocks. Due to the Block Wizard, user defined blocks can be implemented with very little knowledge of the specific details of the DSP card and operation of the RIDE software. When creating real time blocks, Block Wizard allows the user to define the number of inputs and outputs from the block and the number of parameters available within the block. With the aforementioned information Block Wizard generates the code for the DLL to handle data transfers and the initialization of the block when it is compiled by the RIDE application.

Block Wizard also produces the shell of the code that will run on the DSP card. This code is altered by the user, so that the block performs the necessary functions. This code is compiled into object code, which is used by the RIDE software when the controller, in the block diagram editor,

is compiled and linked. A detailed description of various user defined blocks written to enable the test bed controller to be implemented are given in Appendix G.

5.3 Test Bed Controller

For the AC VSD in the test bed system to be loaded with a simulated pump or fan load, as presented in Chapter 3, and for its power usage to be measured, as presented in Chapter 2, a real time controller is required. The controller can be broken down into two separate components, one which runs on the PC32 card to control the operation of the test bed system and measure the relevant variables, and a second which serves as a user interface displaying the measured variables and the power usage of the AC VSD in the test bed system. The power used by the test bed system at various flow rates can be used to determine the energy consumed for a given duty cycle. The test bed system does not calculate the energy usage of the pump or fan system as this adds extra software overhead which is undesirable in a real time control system. The energy usage calculation is based on the length of time that the AC VSD draws a specific power and it would be impractical to run the test bed system for a period equal to that of the duty cycle of the associated pump or fan system.

The test bed controller had to provide the following functions and features:

- to calculate the load torque with which the AC VSD has to be loaded (to simulate a pump or fan load);
- to control the DC motor, such that the desired pump or fan load torque is placed on the AC VSD;
- to check for system errors and to take appropriate action when encountered ;
- to measure the AC VSD's power usage;
- to display the pump or fan power vs the flow rate;
- to display the induction motor speed vs the load torque.

The test bed controller implemented in RIDE has been grouped into various hierarchical blocks to reduce the number of blocks that the user sees on the PC screen. This enables the relevant data to be displayed, without getting the user bogged down with the operation of the test bed controller. The user interface, shown in Fig. 5.4, consists of a hierarchical block which contains the test bed controller. The remaining blocks are required for user input and visualisation.

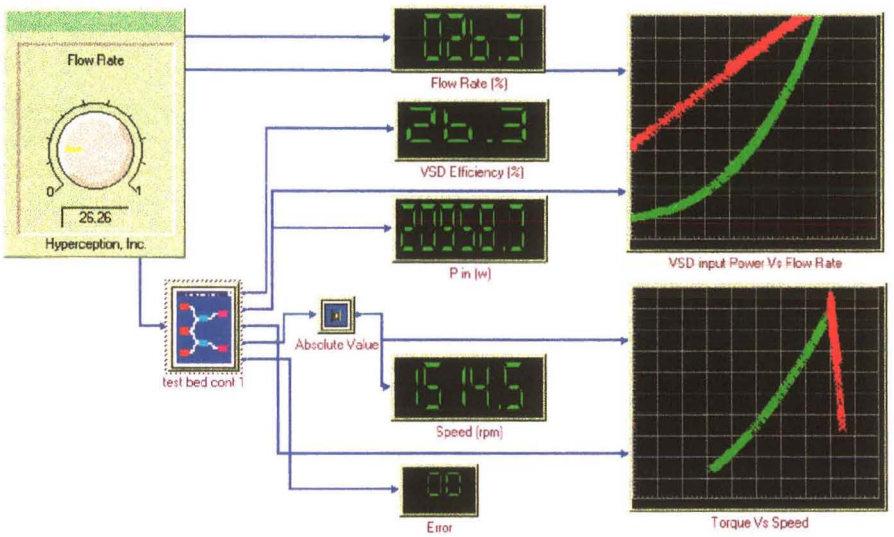


Fig. 5.4 Test Bed Controller Block Diagram

The user selects the flow rate using the flow rate control knob, this value is passed to the test bed controller block, as shown in Fig. 5.4, which controls the DC motor such that the AC VSD is loaded with the torque that either a pump or fan would place on the AC VSD for a desired flow rate. The output of the test bed controller block is the system variables as shown in Fig. 5.4 and are :

- the AC VSD efficiency;
- the AC VSD power usage;
- rotational speed of the motors in the test bed system;
- load torque placed on the AC VSD;
- system error number (system status).

To enable a comparison to be drawn between a pump or fan system in which the flow rate is controlled using either discharge throttling or dampers (fixed speed) and one in which the flow rate is controlled using speed regulation two graphs are plotted, as shown in Fig. 5.4. The first graph is the input power to the AC VSD versus flow rate, and the second graph, shows the torque applied to the AC VSD versus speed. These graphs enable the user to gain an understanding of the load placed on the AC VSD as the flow rate of the pump or fan is altered. Both graphs are set to have persistent storage, which ensures that the graphs are not cleared each time the controller is required to simulate different load conditions for either a pump or fan. The persistent storage of the graphs enables a comparison to be made between fixed and variable speed operation of pumps and fans. For the worksheet, shown in Fig. 5.4 the red points on the graphs indicate fixed speed operation for pump system A, and the green points represent variable speed operation for pump system A.

The main controlling block in Fig. 5.4 is the test bed controller block (test bed cont 1), this block comprises various user defined and hierarchical blocks. The internal structure of the test bed controller block is shown in Fig. 5.5.

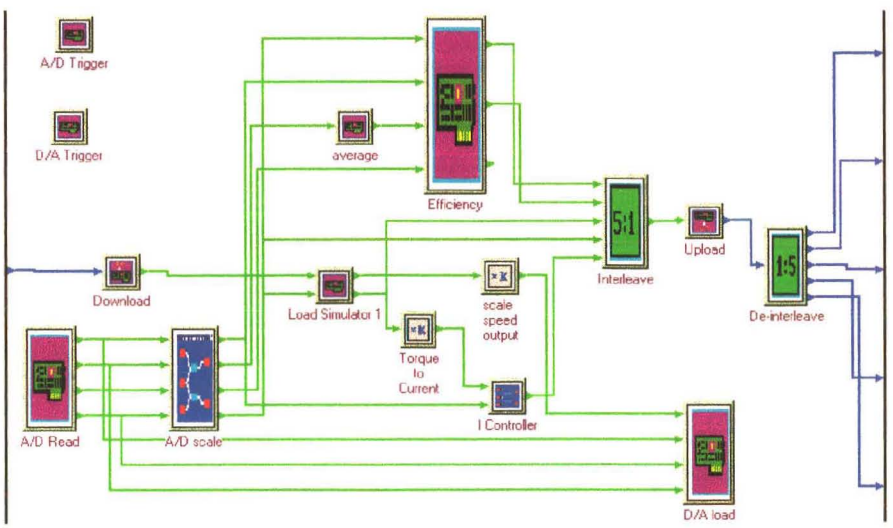


Fig. 5.5 Test bed controller hierarchical block diagram

An explanation of the test bed controller is best done in two sections: the first is the method in which the load placed on the AC VSD is calculated and used in the control of the DC motor; the

second section is the measurement of the AC VSD power usage. The blocks required to perform the two functions described above are displayed in Fig. 5.6 and 5.7, all hierarchical blocks have been broken down into their individual blocks. The blocks shown in Fig. 5.6 and Fig. 5.7 are not duplicated in the final system as shown in Fig. 5.5. The operation of the user defined blocks written to enable the test bed controller to be implemented are described in Appendix G.

5.3.1 Test Bed Controller DC Motor Control

The controller configuration shown in Fig. 5.6 is used to control the DC motor such that it presents the correct pump or fan characteristic to the AC VSD. The Load Simulator block is the block which simulates the desired pump or fan load. This simulated load is then used to calculate the appropriate current reference for the DC motor. The operation of the Load Simulator block is described whereafter the signal flow through the controller is described.

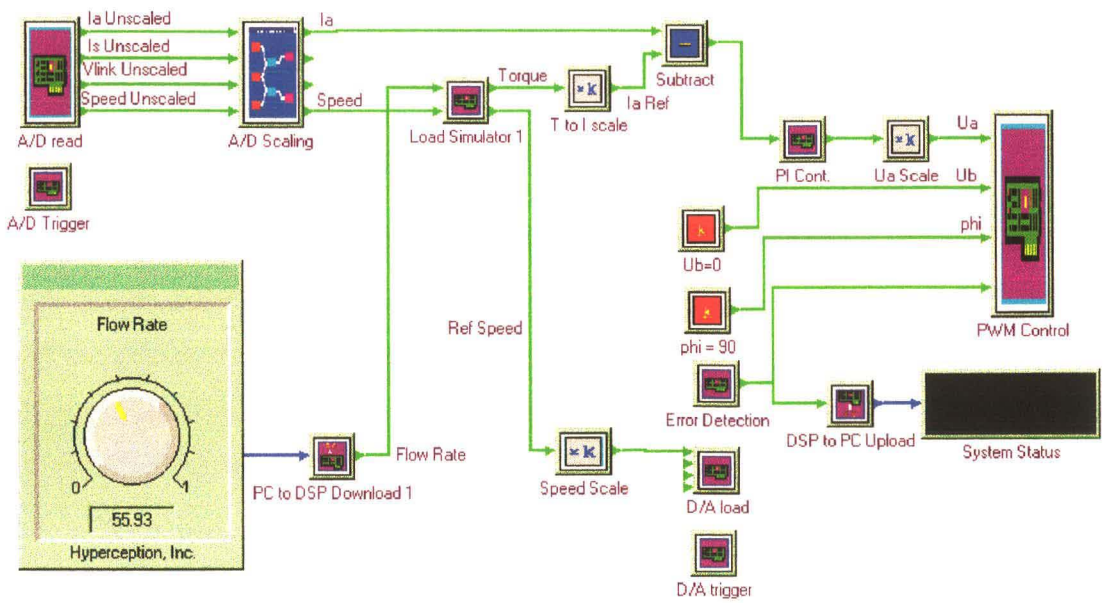


Fig. 5.6 Test bed controller DC motor control block diagram

The Load Simulator block calculates the set point torque required by the pump or fan that is to be simulated based on the rotational speed of the test bed system, the type of flow control and flow rate required. The Load Simulator block also calculates the speed set point for the AC VSD

according to the type of flow control and flow rate being simulated. The type of control to be simulated is selected from the Load Simulator's parameter box (parameter box of any block is attained by double clicking on the relevant box). The Load Simulator block calculates the load torque and speed required in the test bed system based on the input parameters and the equations determined in Chapter 3. The inputs to the Load Simulator block are the rotation speed of the test bed system in RPM and the flow rate in percent. The output of the Load Simulator is the set point speed for the AC VSD in rpm and the set point torque for the DC motor in Nm. A speed controller is not implemented in the test bed controller as the AC VSD in the test bed system contains a speed controller (PI controller) which is capable of maintaining the speed of the induction motor at the set point speed. The speed of the test bed system is measured and the load torque calculations are based on this speed when the flow rate is being controlled, this allows the test bed user to vary the speed manually while allowing the AC VSD to be loaded with a load that represents a pump or fan as would occur in the field where the AC VSD's speed is varied until the flow rate in the system is correct.

The load torque and speed set points are calculated for the various pump and fan systems as follows:

- **Flow control using discharge throttling for pump system A and B**

  As the pump described in Chapters 2 and 3 operates at 3000rpm, and since the test bed system has a maximum speed of 2000rpm the load torque curves calculated in Chapter 3 have to be normalized in order to simulate the pump system on the test bed system. The load torque curves are normalised such that the operating speed of the pump would vary between 0 and 1500rpm. The load torque placed on the AC VSD when delivering maximum flow rate is twice that for the system described in Chapter 3, which ensures that the AC VSD delivers the same power as if it were operating at 3000rpm. The effect of keeping the output power of the AC VSD equal to that of the system described in Chapter 3 is to ensure that the electrical losses in the system are the same. There is a slight difference in the mechanical power losses due to the lower windage and frictional losses in the system. The method used to normalizing the

pump load curves as described above are used to normalize all the pump load curves presented in Chapter 3 such that a direct comparison may be drawn between pump simulations performed on the test bed system.

As discharge throttling is being simulated and the flow rate is controlled using a throttling valve the speed of the AC VSD is held constant and set according to Eq. 5.1.

$$n_r = 1500 \tag{5.1}$$

The load torque placed on the AC VSD is calculated using Eq. 5.2 which is gained by normalising the straight line graph given in Fig. 3.22 (the full load torque required to simulate the pump is doubled).

$$T_l = \frac{(203 - 70)}{100} Q + 70 \tag{5.2}$$

where Q is the desired volume flow rate (%)

T_l is the load torque in Nm.

- **Flow control using speed regulation for pump system A**

The pump curves presented in Chapter 3 which describe pump system A in which speed regulation is used to control the flow rate are normalized as in the case of the pump system using discharge throttling.

Eq. 3.6 in Chapter 3 described the speed required to produce a specific flow rate for a pump which operates between 0 and 3000rpm. Normalising this curve such that it can operate on the test bed system results in Eq. 5.3.

$$n_r = \frac{(1500 - 550)}{100^{1.4}} Q^{1.4} + 550 \qquad (5.3)$$

where Q is the desired flow rate (%)

n_r is the required speed (rpm).

The above equation determines the speed set point of the AC VSD such that it runs at a speed which would produce the desired volume flow rate. The AC VSD has to be loaded with the pump characteristic for pump system A as calculated from Eq. 5.4 which is the normalized form of Eq. 3.7.

$$T_l = \frac{203.7}{1500^2} n_r^2 \qquad (5.4)$$

where T_l is the load torque to be simulated (Nm)

n_r is the rotational speed of the VSD in the test bed system (rpm).

It must be noted that at full speed (1500rpm) the power required by the simulated pump is 32kW, which is exactly the same power requirement for the pump when operating at 3000rpm (see Fig. 3.26).

- **Flow control using speed regulation for pump system B**

The equations used to describe the operation of pump system B when using speed regulation to control the flow rate were formulated in Chapter 3. These equations also have to be normalized as described earlier in order for the pump simulations to be performed on the test bed system. The speed of the pump required to produce a specific flow rate is given in Eq. 5.5 which is the normalized form of Eq. 3.8.

$$n_r = \frac{(1500 - 1240)}{100^2} Q^2 + 1240 \tag{5.5}$$

where n_r is the rotation speed required in the test bed system (rpm)

Q is the volume flow rate (%).

The torque that the pump places on the VSD for pump system B can be calculated from Eq. 5.6 which is the normalised form of Eq. 3.9.

$$T_l = -126.102 \times 10^{-9} n_r^3 + 108.685 \times 10^{-5} n_r^2 - 2.481 n_r + 1175.467 \tag{5.6}$$

where T_l is the torque required to simulate the pump (%)

n_r is the rotational speed of the VSD (rpm).

- **Flow control using dampers for fan system**

When dampers are used to control the flow rate of a fan the speed and the torque produced by the fan remains constant as described in Chapter 3. Unlike a pump the fan torque and speed do not have to be normalized as it functions within the operating range of the test bed system. The fan system in which dampers are used to control the flow rate is operated at constant speed, the speed reference to the AC VSD is set according to Eq. 5.7. As the torque required by the fan (at a specific speed) remains virtually constant irrespective of the flow rate the reference torque required is set according to Eq. 5.8

$$n_r = 1500 \tag{5.7}$$

$$T_l = 95 \tag{5.8}$$

- **Flow control using speed regulation for fan system**

The speed of the AC VSD is governed by the flow rate being simulated as defined in Eq. 3.10 and is normalized such that the desired flow rate appears in percent as shown by Eq. 5.9. The flow rate appears as a percent as this ensures that the flow rate control knob does not have to be altered when a pump or fan is simulated.

$$n_r = \frac{2000}{120} Q \tag{5.9}$$

where Q is the desired volume flow rate (%)

n_r is the speed require to produce the volume flow rate Q.

The flow rate and the torque required to move the fan are dependent on the fans rotation speed, which is in turn related to the flow rate required. As with the pumps characteristic equations described above the speed/flow rate relationship and the torque/speed relationship have been kept separate as this enables the user to vary the speed of the AC VSD while loaded with the fan characteristic. The torque required to rotate the fan at a specific speed is given by Eq. 5.10 and is given here for convenience from Eq. 3.11.

$$T_l = \frac{176}{2000^2} n_r^2 \tag{5.10}$$

The equations described above enable the speed and torque required to simulated a pump or fan to be calculated based on the flow rate being simulated by the user. The method in which data is calculated and passed to the load simulator, and the process by which the data calculated by the Load Simulator is used to control the test bed hardware is discussed below.

The incoming signals from the test bed system are read from the A/D converters on the PC32 card by the A/D read block. The output signals from the A/D read block are the output signals from the A/D converters, since the Load Simulator block requires the signals to be in engineering units (rpm, A, V) the signals are passed through the A/D scale block which converts the signals into engineering units (the internal structure of the A/D scale block is shown in Appendix G). The desired flow rate to be simulated is gained from the flow rate knob, as this is an input control the signal is passed to the PC32 card via a PC to DSP upload block. The simulated flow rate is passed to the Load Simulator block which calculates the speed set point for the AC VSD in the test bed system based on the type of system being simulated. The speed set point generated by the Load Simulator block is scaled prior to being passed to the D/A load block which outputs the speed set point to the AC VSD in the test bed system.

The rotational speed is read by the A/D read block and then scaled by the A/D scale block. This signal is fed to the Load Simulator block from which the required load torque is calculated based on the type of system being simulated. The load torque calculated by the load simulator block has to be converted to a current reference (Chapter 2 showed that the torque produced by the DC motor is proportional to the current flowing in the armature provided the field flux remains constant inside the machine), this is achieved by dividing the load torque by the DC motors armature constant (K_m) as described in equation Eq. 2.14.

The armature reference current is subtracted from the actual armature current (which is read by the A/D read block and scaled suitably by the A/D scale block). The difference between the armature current reference and the actual armature current is the error which is passed to the discrete PI controller as described in Chapter 3. The output of the PI controller is the desired output voltage from the H-Bridge inverter. The output of the PI controller is scaled before being passed to the U_a input of the PWM controller. The PWM controller has three further inputs which are the U_b voltage, the phase angle (ρ) and a disable input. The U_b and phase angle input are set to 0V and 90° respectively which is required to ensure that the PWM ASIC controls the H-Bridge inverter correctly as described in Chapter 4. The disable input of the PWM control block is used to enable or disable the PWM ASIC. If the disable input of the PWM control block is kept at zero

the PWM ASIC is enabled, as soon as the input goes above zero the PWM ASIC is disabled. The disable input of the PWM ASIC is fed from the error detect block. The error detect block monitors the signals on the digital input port of the PC32 card which is fed signals from the PC32 card digital interface. When no errors are present on the test bed system the output of the error detection block is zero, on detecting an error the error detection block outputs a number which corresponds to the error that occurred (see Appendix G for error numbers). This error number is passed to the PC via the DSP to PC upload where it is display so that the test bed user can determine which error resulted in the test bed being disabled. The A/D trigger and D/A trigger blocks are required to ensure that the A/D converters and D/A converters start their conversion and output the desired value in time with the interrupt generated by the PWM ASIC. A description of the PI controller, A/D read, A/D scale, A/D trigger, D/A load, D/A trigger and PWM control blocks are given in Appendix G and their associated code listings is given in Appendix I. The power usage of the AC VSD in the test bed system has to be measured while driving a pump or fan load to enable the power usage calculation to be performed.

5.3.2 Test Bed Controller Power Usage Measurement

The measurement of the AC VSD power usage is achieved by the controller configuration shown in Fig. 5.7.

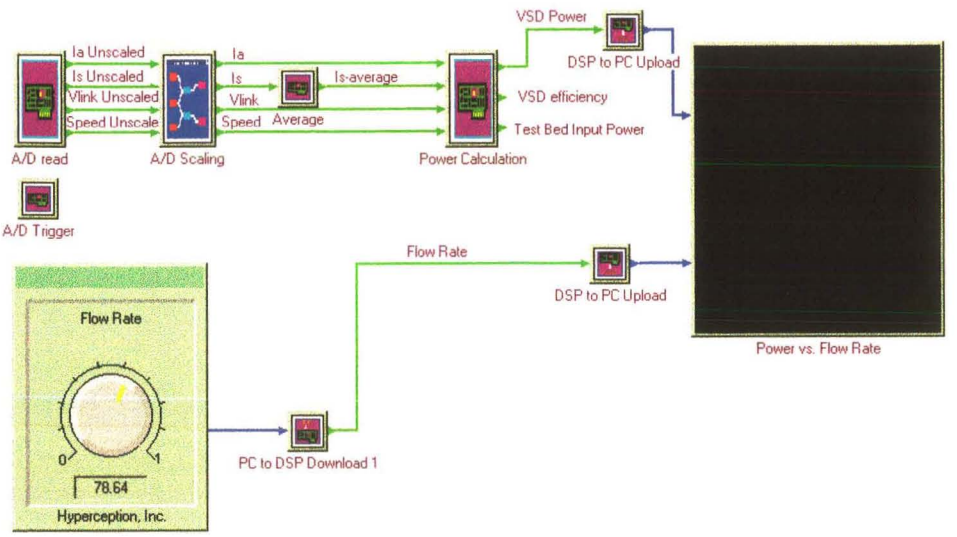


Fig. 5.7 Test bed power usage measurement block diagram

The A/D read block and A/D scale block are the same blocks as shown in Fig. 5.6, which read in the analogue values and scale them such that the output is in engineering units. The power calculation block performs all the calculations required to determine the power usage of the AC VSD in the test bed system. Besides calculating the AC VSD power usage the power calculation block also calculates the AC VSD efficiency (VSI and IM efficiencies combined) and the input power to the test bed system. The AC VSD efficiency and the test bed system input power are calculated for interest and are not used in determining the power usage of a pump or fan system. The AC VSD power usage is calculated from Eq. 5.11, which was derived in Chapter 2, Eq 2.24. The efficiency and test bed system input power is calculated from Eqs 5.12 and 5.13 respectively, these equations were derived in Chapter 2 Eqs 2.21 and 2.27.

$$P_{vsd} = I_r.V_{link} + E_a.I_a - \left(I_a^2.R_a + I_a.V_b\right) \tag{5.11}$$

$$P_r = I_r.V_{link} \tag{5.12}$$

$$Eff_{vsd} = \frac{E_a.I_a + B.\omega_r^2}{E_a.I_a - (I_a^2.R_a + I_aV_b) + V_{link}.I_r}100 \tag{5.13}$$

Where E_a is the armature back EMF

I_a is the DC motor armature Current

R_a is the total resistance in the armature circuit

V_b is the volt drop across the brushes in the DC motor

I_r is the average value of the supply current flowing from the diode rectifier of the VSD

V_{link} is the DC link voltage.

The inputs to the power calculation block come from the A/D scale block, except for the supply current (I_r). The measured value of the supply current is the instantaneous input current to the system which contains 300Hz pulses which relate to the current flowing through the diodes which charge the capacitors present in the DC link. Current will only flow through the diodes when the

supply voltage rises above the DC link voltage, therefore the diodes only conduct for a small period of time around the peak of the supply voltage. The equations used to calculate the power usage of the AC VSD, the AC VSD efficiency and the input power to the test bed system rely on the supply current being an average value not an instantaneous value. The supply current (I_r) is converted into an average value using the average block. A description of the average block is given in Appendix G and the associated code listing is given in Appendix H.

The power usage output of the power calculation block is passed with the flow rate to the PC via a DSP to PC upload block where it is displayed on an X-Y display. The X-Y display enables the test bed user to determined the power usage of the AC VSD for a specified flow rate. The power usage of the test bed system at a specified flow rate may then be used to calculate the energy usage of the pump or fan system over a period of time based on the duty cycle of the process. The execution of the test bed controller software on the PC32 card is governed by the interrupt generated by the PWM ASIC, this ensures that synchronous sampling is achieved and fixes the sampling and calculation rate of the PI controller and average calculation block. The accuracy of the measured system variables and the power usage measurements are compared against measurement equipment readings in Chapter 6 when the commissioning of the test bed system is described.

5.3.3 Test Bed Controller Interrupt Settings

The execution rate of test bed controller software is controlled by the interrupt of the PWM ASIC on the PWM/tachometer card. This is done such that the test bed controller software is executed at a fixed rate which is vital to ensure that the PI controller and average calculation block produce the correct results. Another advantage of synchronizing the execution rate of the test bed controller software to the PWM ASIC is that the A/D converters sample the system variables at a point where no switching of a device in the DC drive takes place. The advantage of using synchronous sampling is that it ensures little or no switching noise is sampled by the A/D converters.

The timing of the test bed controller is achieved by the use of two interrupts: interrupt 0 which is generated by the PWM ASIC; and interrupt 1 which is generated by an A/D converter conversion complete signal. The process by which the test bed controller is executed can be explained with the use of the flow charts shown in Fig. 5.8.

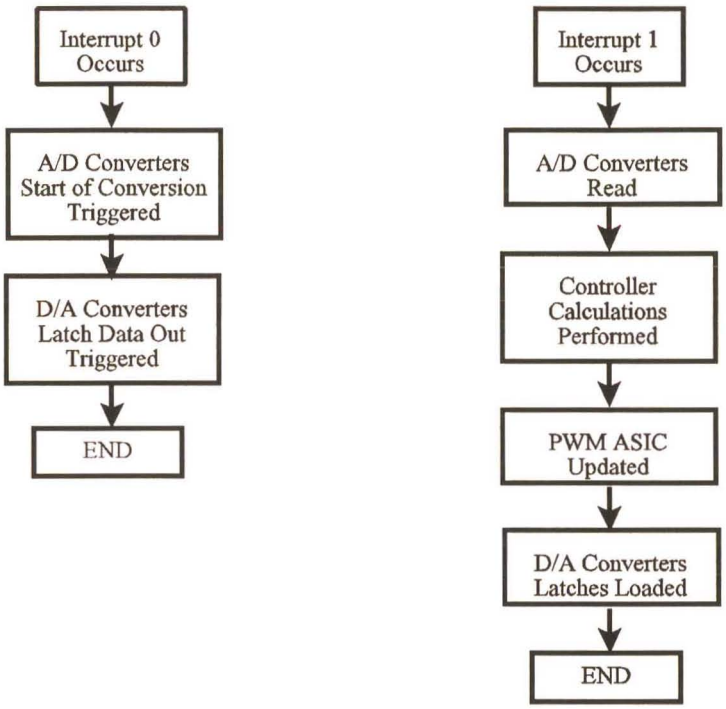


Fig. 5.8 Test bed controller flow charts

An interrupt is received from the PWM ASIC on interrupt pin 0, resulting in the *A/D Trigger* block being run which triggers the A/D converters to start a conversion by writing to their start of conversion memory address. The values loaded into the D/A converter latches on the last cycle, are output when the *D/A Trigger* Block is run. Since the test bed controller is event driven (execution controlled by interrupts 0 and 1), the period between events allows the controller to perform background operation, such as transferring data to the PC to update the digital displays. When the A/D converters have completed their conversion they signal an end of conversion on interrupt 1. When interrupt 1 occurs the A/D converters are read by *A/D Read* block, whereafter the remaining blocks as shown in Fig. 5.6 and 5.7 in the test be controller are run.

5.4 Conclusion

A brief overview of Hypersignal's RIDE software was given to demonstrate the ease with which a controller can be implemented. The main requirements of the test bed controller which enables the AC VSD to be loaded with a pump or fan load while its power usage is measured were highlighted. The method by which the test bed software calculates the load torque and controls the DC drive such that the desired torque is produced was explained with the aid of a RIDE block diagram. The equations used to calculate the load torque required to simulate the pump and fan systems described in Chapter 3 were presented as they were used directly in the test bed software. The means in which the power usage of the AC VSD is calculated by the software was demonstrated through the use of a RIDE block diagram showing the required blocks and data flow required in achieving the end results. The equations used to calculate the power usage, efficiency and input power to the test bed system as derived in Chapter 2 were presented as they are implemented in the test bed software.

For the successful operation of the test bed system to take place, the various gains in the test bed controller have to be set. Correct setup of the AC VSD is also important as this effects the performance of the AC VSD. Chapter 6 describes the commissioning of the various components that constitute the test bed system.

Chapter 6

Test Bed Commissioning

6.1 Introduction

The pump and fan loads determined in Chapter 3 are required so that the power usage of the AC VSD in the test bed system can be evaluated while driving a simulated pump or fan load. The hardware and the software which forms the test bed system was presented in Chapters 4 and 5. This hardware and software was required to load the AC VSD with a simulated pump or fan load in order to determine its power usage for different methods of flow control. Before any pump or fan simulations could be performed on the test bed system, the main components of the test bed system had to be commissioned individually to ensure that they operated correctly

The test bed system consists of three main components, the AC VSD, configurable load and the test bed controller each of which had to be commissioned. The AC VSD was commissioned first as its DC link supplies power to the DC drive and was required for the commissioning of the DC drive. The DC drive which forms the configurable load was then commissioned. The commissioning of the DC drive involved comparing the simulated results developed in Chapter 3 with the practical results measured on the DC drive. Lastly the test bed controller software had to be tested and commissioned. The commissioning of the test bed software involved checking the measured results from the controller against results from measurement equipment.

6.2 REFU Drive Commissioning

This section describes the procedure followed in commissioning the REFU drive. The procedures described in Chapter 2 were used to calculate the d-axis and q-axis current from the no load test performed on the induction motor and the induction motor's name plate parameters. The d-axis and q-axis current values are required for the commissioning of the REFU drive used in the test bed system.

A simplified block diagram of the REFU drive controller is shown in Fig. 6.1. Being an analogue FOC controller, all the set point parameters (d-axis current reference, q-axis current limit and the slip gain) are set using potentiometers (R5, R14 and R10) as shown in Fig. 6.1.

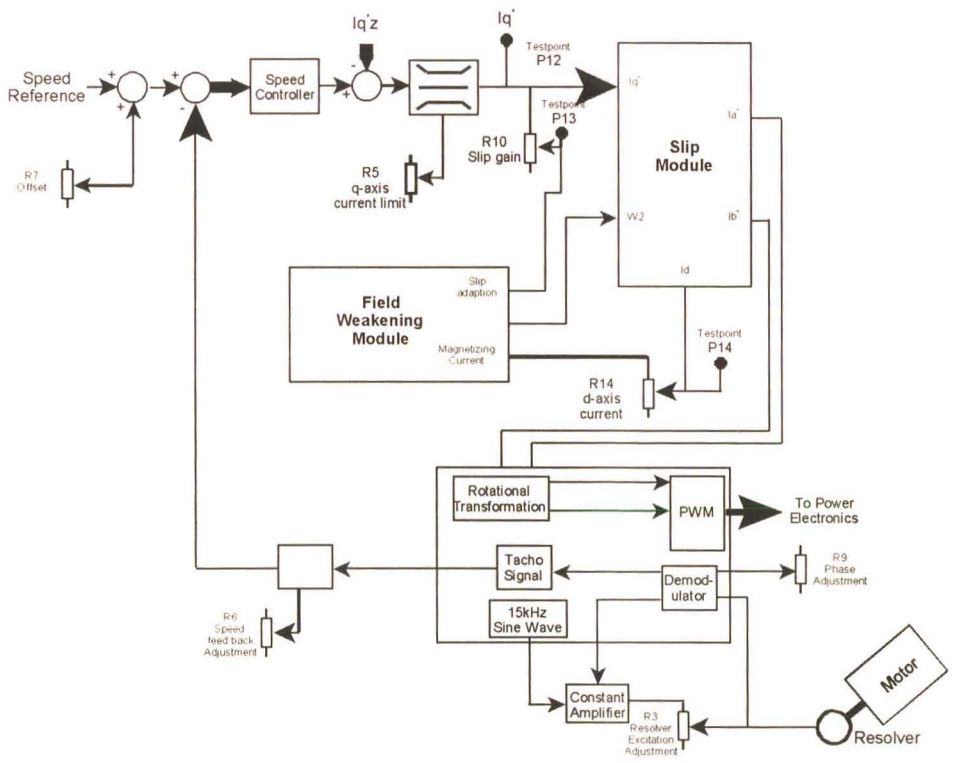


Fig. 6.1 Simplified block diagram of REFU drive controller

The method used to calculate the set point values of d-axis current, q-axis current and slip gain are described below; these values are required to determine the set point values of the various potentiometers (as shown in Fig. 6.1) on the REFU drive controller according to the procedures described in Appendix H.

6.2.1 Induction Motor Parameters

The parameters required for the calculation of the set point values for the REFU controller may be derived from a knowledge of the induction motor's name plate data and the no-load current shown below. Each parameter required for the calculations of the set point values for the REFU controller are described in detail

Power	: 30kW
Supply Voltage	: 380V
Connection type	: Delta
Full Load current (I_s)	:58A rms
$\cos\varphi_n$	:0.86
Pole Pairs (P)	:2
Nominal speed (n_{nm})	:1465rpm
Supply Frequency (f_{nm})	:50Hz
No Load current (I_{ds})	:25A rms

6.2.2 Magnetizing Current Calculations

When running under FOC control the magnetizing current for an induction motor is produced by the d-axis stator current. The performance of the motor is heavily reliant on this quantity: if the motor is under fluxed (too small a magnetizing current), it will not produce rated torque; on the other hand, if the machine is over fluxed, the magnetic circuit saturates leading to high losses and lower efficiency operation. The magnetizing current of the motor is found by running the motor at no load. When running at no load the synchronous speed and the rotor speed are almost

identical (slip is almost zero), resulting in the q-axis rotor flux being zero. The motor running at no load operates under a similar condition as occurs when operating under FOC (q-axis rotor flux is zero). The currents flowing into the machine are only setting up the flux (I_{qs}=0), thus $|I_{ds}| = |I_a| = |I_b| = |I_c|$ for a balanced three phase machine. The no load current of the induction motor was measured and found to be 25A, thus I_{ds} = 25A. The I_{ds} value is used in Appendix H to calculate the magnetizing current set point. The magnetizing current set point as calculated in Appendix H was not setup correctly on the drive. When the drive was run for the first time the induction motor produced a rough sound when rotated at low speeds. The set point values adjusted on the FOC controller card were checked and it was found that the magnetising current set point was set higher than the calculated value, resulting in the induction motor being over fluxed. The magnetising current was then setup correctly where after the induction motor ran smoothly even at very low speeds.

6.2.3 Q-axis Current Limit Calculations

The q-axis current limit (I_{qs}^*), limits the current that the REFU drive will allow to flow through the induction motor. The maximum I_{qs} value that the REFU drive should be set to is calculated from the name plate parameters on the machine, using Eq. 6.1. Setting the I_{qs} value higher than the value calculated using Eq. 6.1 can result in the induction motor failing on overload if the load placed on the induction motor remains high for an extended period of time.

$$I_{qs} = \sqrt{(I_s)^2 - (I_{ds})^2} \tag{6.1}$$

where I_{qs} is the maximum torque producing current that should be allowed to flow

I_s is the full load current of the machine.

With I_s = 58A and I_{ds} = 25A, I_{qs}= 52.3A. The value of I_{qs} is used to calculate the set point value required to be set on the REFU drive controller, the calculation of which is given in Appendix H.

6.2.4 Slip Frequency Calculation

As stated in Chapter 2 one of the conditions that has to be met in order for an induction motor to be controlled under FOC is given in Eq. 6.2 which is given in Appendix A, Eq. A.15. Eq. 6.2 states that the slip frequency ($s\omega$) is a function of the motor parameters and the q-axis stator current reference value. The motor parameters as given in Eq. 6.2 remain almost constant therefore if the slip frequency is known and the q-axis stator current is known at a particular slip frequency (full load as these parameters are given on the motor name plate) then the motor parameters can be calculated. The motor parameters are set on the REFU drive by adjusting the slip gain. The slip frequency which is required in setting the slip gain on the REFU drive is calculated using Eq. 6.3.

$$s\omega = \frac{L_m R_2 i_{qs}^*}{L_{22}\lambda_{dr}} \tag{6.2}$$

where s is the induction motor slip

ω is the induction motor rotation speed

L_m is the stator and rotor mutual inductance

R_2 is the rotor resistance

i^*_{qs} is the q-axis current reference

L_{22} is the rotor self inductance

λ_{dr} is the q-axis rotor flux linkage.

$$f_{2n} = f_{nm} - n\frac{n_{nm}}{60} \tag{6.3}$$

where f_{2n} is the slip frequency

f_{nm} is the supply frequency

n is the number of pole pairs in the IM

n_{nm} is the full load speed of the motor (as given on the name plate).

With n_{nm} = 1465rpm, f_{nm} = 50Hz and n=2, the slip frequency f_{2n} =1.17Hz. The value of f_{2n} is used in Appendix H when calculating the set point value for the slip gain to be set on REFU drive controller.

The set points required for the commissioning of the REFU drive were calculated from the values determined above as shown in Appendix H. The REFU drive controller card was then supplied with power, and the various set point parameters calculated in Appendix H were set on the card, whereafter the REFU drive was powered and enabled. The current flowing into the induction motor when running under no load at rated speed (when the correct magnetising current set point was adjusted) was equal to the no load current of the machine operating off the 50Hz mains supply. The next step was to commission the DC drive.

6.3 DC Drive Commissioning

The DC drive is an integral part of the test bed system as it forms the configurable load which simulates the pump and fan loads. The performance of the DC motor and PI current controller determines the accuracy and reliability of the configurable load. The DC drive was simulated in Chapter 3 where a PI controller was designed to give a first-order response (armature current did not exhibit any over shoot and had zero steady state error). The commissioning of the DC drive involved checking that the correct feed back signals were received, whereafter the simulated and practical results were compared to determine the DC drive's performance.

The discrete PI current controller, simulated in Chapter 3, was implemented in RIDE. The DC drive was supplied by a separate three phase supply for tests, so that the DC link voltage could be reduced during the initial commissioning phase. Only once the current feedback loop and the switching signals being fed to the SKIIP module were deemed correct was the DC drive connected to the DC link of the REFU drive. With the DC drive connected to the DC link results were taken in order to determine whether the DC drive's operation was suitable for use as a configurable load.

6.3.1 DC Motor Operation Under PWM Control

In Chapter 3 it was stated that the ripple present in the armature current due to the PWM switching waveforms was important as it affects the ripple torque produced by the DC motor. As the simulated ripple current flowing through the DC motor was found to be too high, the armature inductance was increased by the addition of external inductors to reduce the ripple current. With the additional inductance the simulated ripple current was approximately 5A peak. The measured DC motor armature current ripple is shown in Fig. 6.2. This measurement was taken while the DC motor was at stand still, with zero average voltage being applied to the DC motor armature terminals. From Fig. 6.2 it can be seen that the armature ripple current is approximately 5A (peak) when the DC link voltage was at 570V. The simulated and actual peak ripple current flowing in the armature are very close.

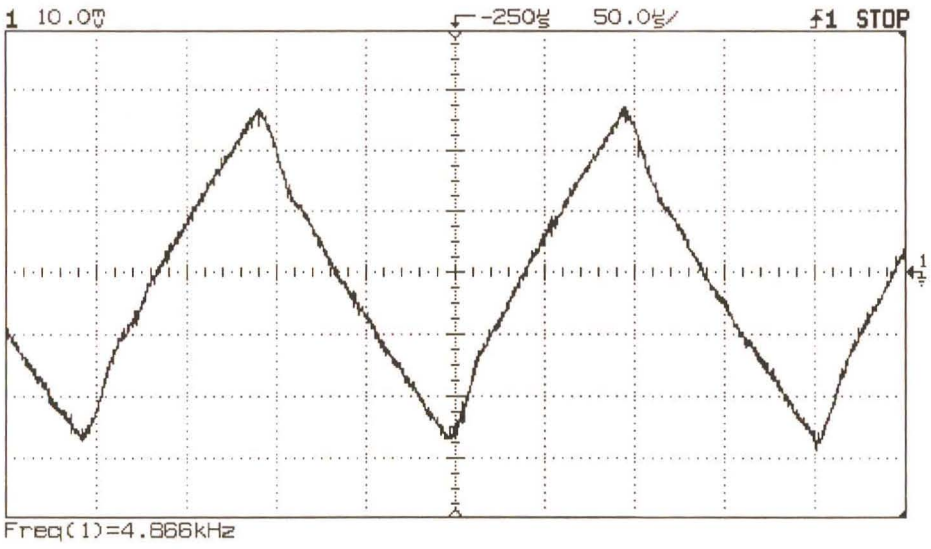


Fig. 6.2 DC motor armature current ripple at standstill (10mV = 2A)

The operation of the DC motor when supplied by the H-bridge inverter is therefore very similar to the simulated results presented in Chapter 3. The next step was to implement the PI controller simulated in Chapter 3 and test its suitability for use in the test bed system.

6.3.2 Armature Current Controller Performance

The PI controller gains determined from the simulations in Chapter 3 were used in the discrete PI controller in RIDE to determine if the operation of the DC drive was suitable for a configurable load. The response of the armature current to a step input of 100A is shown in Fig. 6.3. The ripple current due to the PWM switching can be clearly seen in Fig. 6.3. The PI armature current controller is run synchronously with the PWM ASIC interrupt as discussed in Chapter 4. Thus the PI armature current controller should not be affected by the ripple current due to the synchronous sampling used in the controller. Fig. 6.4 shows the sampled armature current as seen by the PI armature current controller and the 100A reference signal within the RIDE controller.

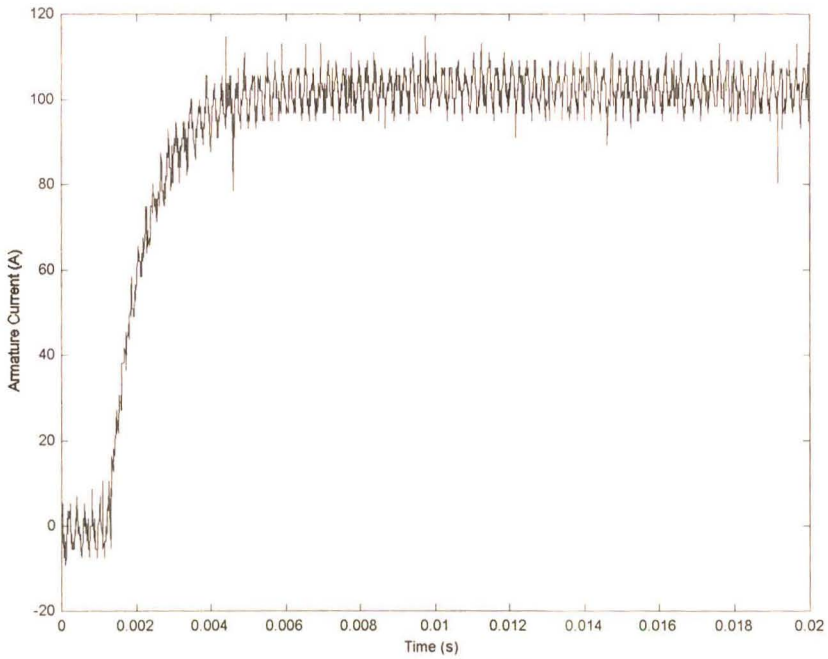


Fig. 6.3 Actual armature current response to 100A step input

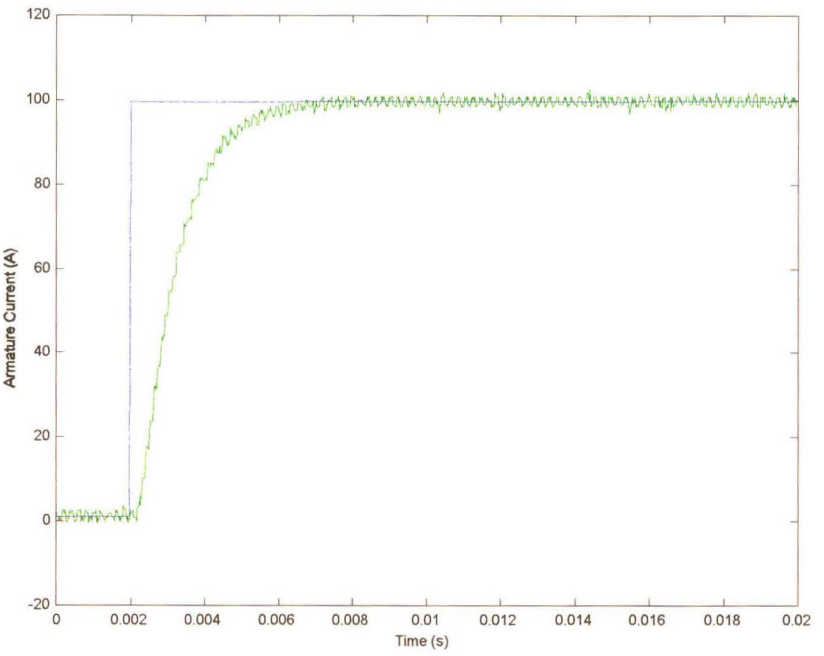


Fig. 6.4 Armature current and reference signals sampled by controller.

From Fig. 6.4 it can be seen how effective synchronous sampling is in rejecting the ripple current, resulting in a small ripple current being measured which is very close to the average DC armature current. The small ripple is due to the method in which the synchronous sampling is implemented, as the interrupt from the PWM ASIC triggers a hardware interrupt (interrupt 0) on the DSP processor which then executes a software interrupt routine which triggers the A/D converters to start their conversion. The problem with using the hardware interrupt on the DSP processor is the time it takes to execute the software code from when the interrupt is detected on the interrupt pin does not remain fixed. This time is known as interrupt latency and results in the A/D converter being triggered a short time after the PWM ASIC interrupt has occurred. The period of time that lapses from when the signal is received on the interrupt pin and when the A/D converter is triggered does not remain constant as it is dependent on the type of instruction being executed prior to the interrupt being detected. Due to the variation in the interrupt latency which result in the A/D converters being triggered at a varying interval around the PWM ASIC interrupt a small current ripple is measured.

To verify the performance of the PI armature current controller in the test bed system, the results shown in Fig. 6.4 are compared to the simulated results of Chapter 3, as shown in Fig. 6.5.

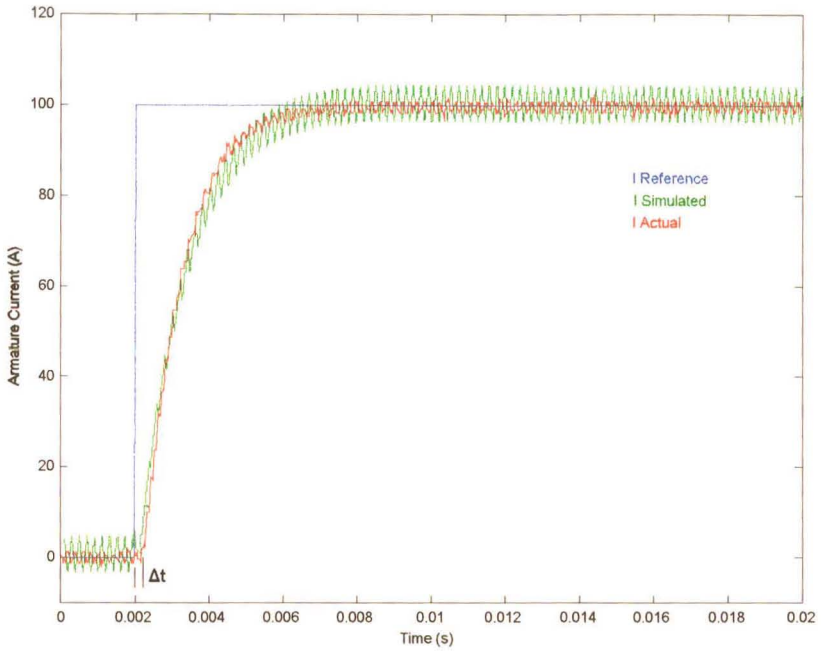


Fig. 6.5 Practical and simulated system response to 100A step input

From Fig. 6.5 it can be seen that the practical and simulated responses from the PI controller are similar, the practical system responds slightly faster than the simulated system but still exhibits no overshoot. The practical response shows no steady state error due to the integral component of the PI controller. The time delay (Δt) between the change in set point to the time when the controller begins to respond is two sample periods of the PI controller ($T_s = 102\mu S$, $\Delta t = 204\mu S$). The sample delay as shown by Fig. 6.5 is the same as the simulated sample delay as shown and explained in Chapter 3. From Fig. 6.5 it can be seen that the DC drive will operate as a configurable load as it is capable of responding quickly to a change in set point and exhibits no steady state error.

6.4 Test Bed Controller Software Commissioning

The DC motor armature current controller forms an integral part of the test bed controller software. After implementing and testing the DC motor armature current controller the remaining test bed controller software was implemented as described in Chapter 5. As the calculations performed in the test bed software rely on the data being in physical units (ie. volts, amperes, rpm) the values read by the A/D converters had to be scaled. The transducers used in the test bed system have a high accuracy (error < 1%) but the burden resistors used to terminate the transducers generally have a much lower accuracy (1%). The voltage signal from the burden resistors used to terminate the transducers are then passed through an amplifier as described in Chapter 4. The errors due to the burden resistors and the amplifiers are all linear errors and are greater than the error due to the transducers themselves. The errors due to the burden resistors and amplifiers can be compensated for in the test bed controller software by multiplying the A/D converter signal by a linear scaling factor thereby overcoming the linear errors. The scaling of the A/D converter input signals and the compensation of the linear errors are performed in the A/D scaling hierarchical block.

To ensure that the correct scaling factors were set in the test bed software the various system variables being measured were calibrated against a number of measurement instruments as follows: The DC link voltage being measured was compared against a reading gained from a digital multi-meter at the DC link operating voltage, this ensured that the highest accuracy of the measurement occurs at the nominal operating voltage. The DC motor armature current and the DC link current were calibrated against a reading from an oscilloscope connected to a current probe, using the oscilloscope to calibrate the current measurements is particularly important when calibrating the DC link current scaling factor as the current is discontinuous. The speed for the test bed system was calibrated against a hand held tachometer placed on the shaft of the induction motor. Once the scaling factors for the various inputs to the test bed controller had been setup the powers flowing in the test bed system calculated by the test bed controller were compared against power measurement devices to determine if the software was calculating the values correctly. A three phase watt meter capable of measuring non-sinusoidal waveforms was used to

measure the power drawn by the test bed system while various loads were applied to the VSD by the DC motor. The measured values of the power flowing into the test bed system from the three phase watt meter were compared against the values calculated by the test bed controller. It was found that the watt meter readings and the calculated value from the test bed controller were in agreement, which indicated that the test bed controller software was measuring the system variables and calculating the power flow in the test bed system accurately.

The scaling factors implemented in the A/D scaling hierarchical block are given in Appendix G. Once the test bed software scaling factors were setup correctly and the measurement results for the input power to the test bed system were verified the pump and fan systems described in Chapter 3 were simulated on the test bed system.

6.5 Conclusion

The commissioning of the test bed system involved three stages: the first stage was the commissioning of the REFU drive used in the test bed system. The second stage was the commissioning of the DC drive and the third stage involved checking the test bed system software and setting up the scaling parameters such that the correct measurement results were attained.

The REFU drive was commissioned based on the motor parameters and the commissioning procedure outlined in the drive's operations manual. The REFU drive commissioning was accomplished without any major faults. The DC drive commissioning was achieved in three stages: The first was determining the effects of the PWM switching waveforms applied to the DC motor where it was found that the simulated and practical results were in agreement. The second stage involved checking the signal integrity of the transducer used and the operation of the DC drive at a reduced DC link voltage. The third stage involved inter-connecting the DC drive's and REFU drive's DC link such that the DC drive operated at the correct voltage whereafter a step response was applied to the DC drive such that a comparison could be made between the practical results gained and the simulated results from Chapter 3. The practical and simulation results were found to be in agreement with each other. The commissioning of the test bed software involved

checking the variables measured by the software against the measured values gained by other test equipment. Once the scaling parameters were set in the test bed software, the calculated values from the test bed controller were compared against those measure by a watt meter where it was found the test bed software produced the same values as those of the watt meter. The watt meter readings being in agreement with the test bed controller software power calculations demonstrated that the test bed controller software was operating correctly.

With the successful commissioning of the hardware and software, the test bed system was used to perform the pump and fan simulations based on the load curves determined in Chapter 3. Chapter 7 discusses the simulation results obtained from the test bed system, whereafter the energy usage of the pump and fan systems are calculated based on a number of flow rate duty cycles.

Chapter 7

Test Bed Simulation Results

7.1 Introduction

Chapter 2 described the operation of centrifugal pumps and fans, whereafter Chapter 3 presented two pump system test cases and a single test case for a fan system, where the flow rate was controlled using discharge throttling or dampers, and speed regulation. To evaluate the power savings that are achievable by using speed regulation in place of discharge throttling or damper control, a test bed was constructed which enables pump and fan systems to be simulated. The design and construction of the test bed system was presented in Chapter 4, and the test bed controller software was presented in Chapter 5. The commissioning of the test bed system was presented in Chapter 6.

This Chapter presents the results (VSD power usage) of the pump and fan simulations conducted on the test bed system based on the pump and fan characteristics presented in Chapter 3. These results are then used to calculate the energy usage of the pump or fan system when operating under a specific duty cycle. The efficiencies of the pump and fan systems are also presented. The power consumption of the test bed system while simulating a pump load is presented as it demonstrates the advantages of linking the VSD and DC drives DC link to improve the efficiency of the test bed system.

7.2 Test Bed Pump Simulation Results

The test bed controller enables a user to simulate the operation of the pump systems described in Chapter 3. The type of pump system that is to be simulated is determined by the load simulator block based on the test bed users selection (see Chapter 5 and Appendix G). Two pump systems

were simulated, the first system had a large friction head and the second system had a large static head. The two pump systems were simulated when discharge throttling was used to control the flow rate whereafter the pump systems were simulated when speed regulation was used to control the flow rate.

The test bed system measures the power usage of the VSD while operating under a specific load. The results presented below were gained by selecting the appropriate pump system from the load simulator block. The flow rate knob was then altered (as shown in Chapter 5) such that the power flowing into the VSD could be measured at each simulated flow rate by the test bed system.

7.2.1 Pump System A Simulation Results

The simulated results for pump system A were achieved in two parts: the first was using discharge throttling to control the flow rate. The second simulation was using speed regulation to control the flow rate. Both results are presented on the same graph to enable comparisons to be drawn between the system using discharge throttling and speed regulation.

The simulated load torque placed on the VSD as its speed changes, is shown in Fig. 7.1. From Fig. 7.1 it can be seen that the pump load curves described in Chapter 3 had to be normalised as described in Chapter 5 due to the test bed system having a maximum speed of 2000rpm. The speed of the pump is fixed when discharge throttling is used to control the flow rate. When discharge throttling is simulated the change in speed is very small over the simulated range as the VSD in the test bed system is capable of maintaining the speed due to the speed controller present in the VSD. When speed regulation is used to control the flow rate, the pump's speed does not fall below 550rpm as shown in Fig. 7.1. When the pump speed is reduced below 550rpm the pump cannot pump any liquid as it is unable to produce the head required to overcome the static head of the system. No measurements were performed at speeds below 550rpm as the pump should never be operated below this speed.

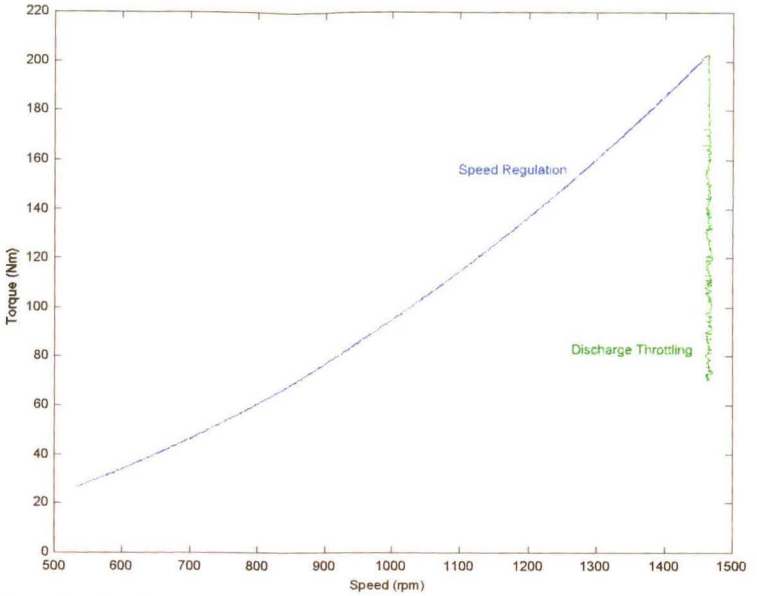


Fig. 7.1 Load torque placed on the VSD to simulate pump system B for discharge throttling and speed regulation.

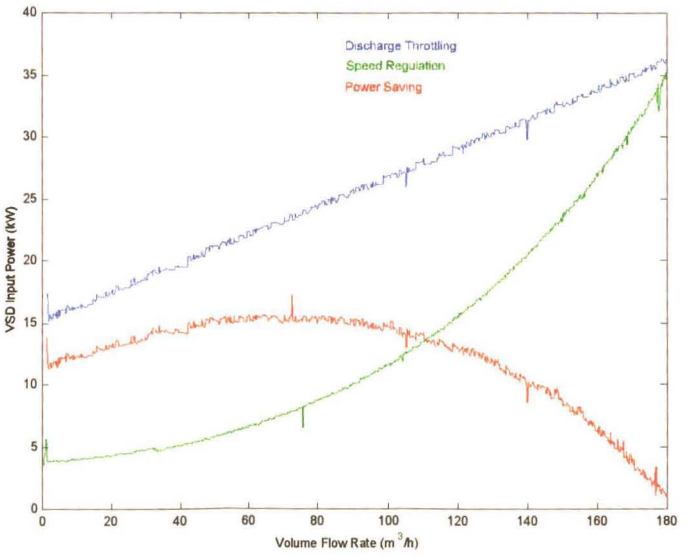


Fig. 7.2 Input power to the VSD and power difference as the flow rate is varied using discharge throttling and speed regulation for pump system A.

The power flowing into the VSD, for both discharge throttling and speed regulation is shown in Fig. 7.2. The difference between the power consumed by the VSD when discharge throttling is simulated and when speed regulation is simulated is shown in Fig. 7.2 (red plot). From Fig. 7.2 it can be seen that power can be saved for all flow rates when speed regulation is used to control the flow rate. The power saved when speed regulation is used to control the flow rate when the desired flow rate is below $140m^3/h$ is greater than 10kW.

The results presented in Fig. 7.2 are used to calculate the energy efficiency and energy usage for pump system A when discharge throttling and speed regulation are used to control the flow rate. The output power of a pump system can be defined as the useful energy gained by the liquid in being moved from one point to another per unit time. The liquid gains energy in two forms: kinetic energy which is the energy contained in the moving liquid; and potential energy which is the energy gained by the liquid in being lifted from the suction point to the discharge point. The efficiency of the pump system can be determined from the useful output power of the system as defined above (calculations are given in Appendix C) and the input power to the VSD in the pump system. The pump system efficiencies for pump system A using discharge throttling and speed regulation are shown in Fig. 7.3.

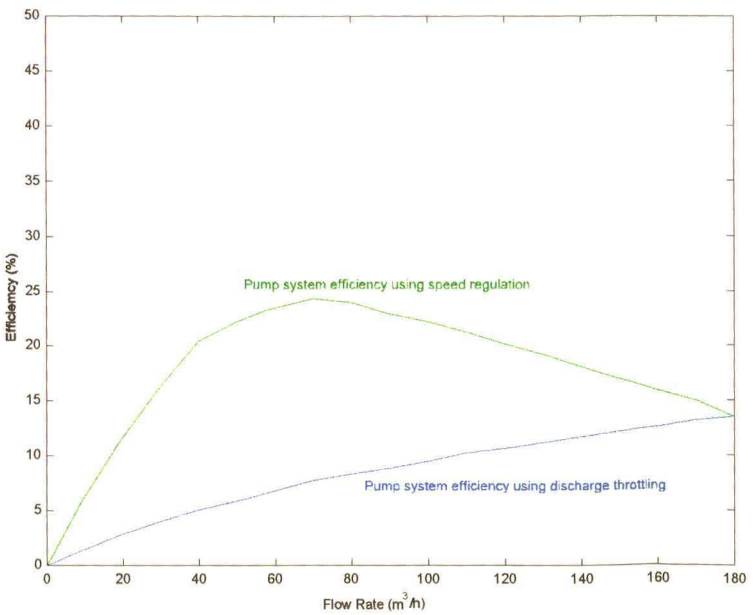


Fig. 7.3 System efficiency using discharge throttling and speed regulation for pump system A

The results displayed in Fig. 7.3 show that the pump system using speed regulation to control the flow rate is more efficient than the same system in which discharge throttling is used to control the flow rate. The result of the improvement in the pump system efficiency is the reduced power that the pump system requires to achieve the desired flow rate as shown in Fig. 7.2. The energy usage of pump system A is dependent on the duty cycle of the pumped liquid. The energy savings achievable by replacing discharge throttling with speed regulation is therefore determined by the duty cycle of the pumped liquid. To demonstrate the effect of the duty cycle on the energy savings two duty cycles are presented: the first duty cycle requires the flow rate to be low for a large portion of the time with high peak demands for short periods of time as shown in Fig. 7.4. This duty cycle has an average flow rate of $96m^3/h$. The second duty cycle is the reverse of the first with large flow rates being demanded for large periods of time with low flow rates for very short periods of time as shown in Fig. 7.5. This duty cycle has an average value of $166m^3/h$. The energy consumption of pump system A under discharge throttling and speed regulation are based upon the twenty four hour duty cycles mentioned above. The yearly energy consumption would be dependent on the number of twenty four hour periods that the pump system operates.

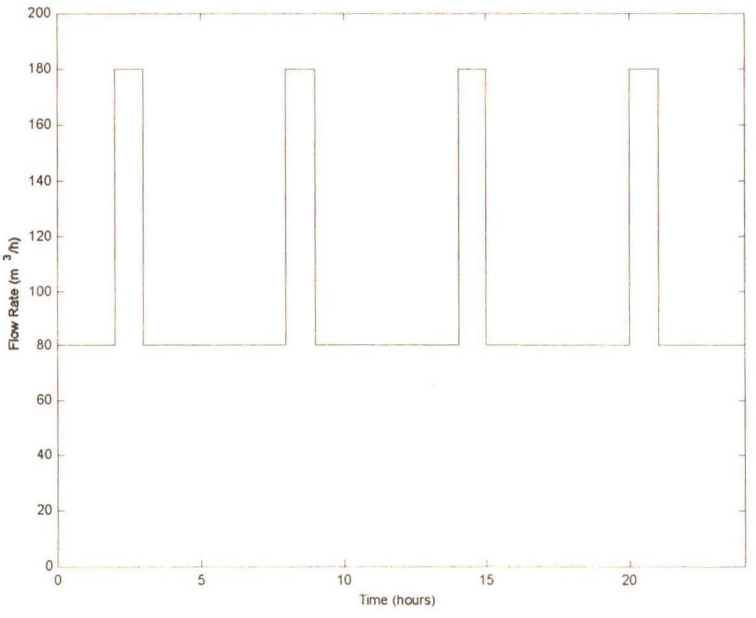


Fig. 7.4 Duty cycle with low average flow rate used to calculate energy consumption of pump systems A and B using discharge throttling and speed regulation.

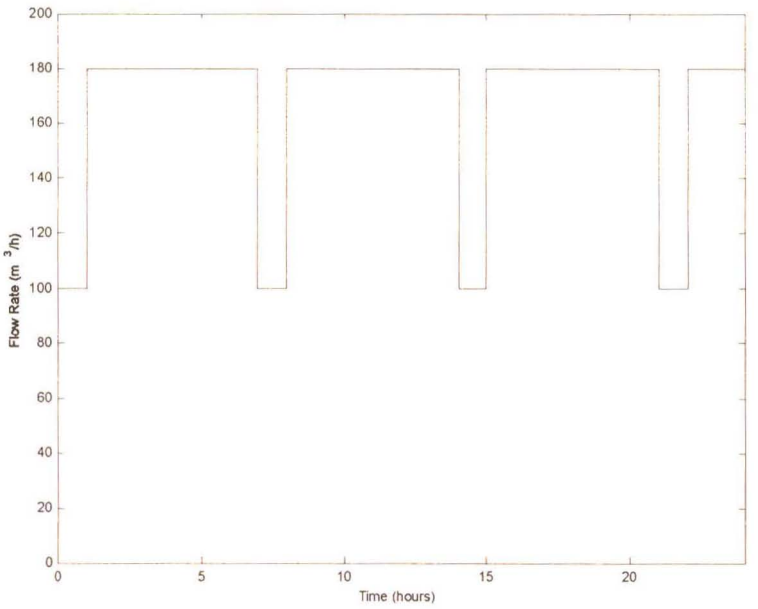


Fig. 7.5 Duty cycle with high average flow rate used to calculate energy of pump systems A and B using discharge throttling and speed regulation.

The energy usage of pump system A when using discharge throttling to achieve the flow rate duty cycle shown in Fig. 7.4 can be calculated using the results shown in Fig. 7.2 as 618.4 kWh (energy usage over a 24 hour period), and that when using speed regulation as 307.8 kWh. Thus the energy savings achieved by replacing discharge throttling with speed regulation in order to control the flow rate of pump system A is equal to 310.6 kWh or a 50.2% reduction in the energy consumption of the system.

In order to demonstrate the importance of the duty cycle when determining the energy usage of a pump system it will be shown how incorrect results may be obtained. The average flow rate for the duty cycle shown in Fig. 7.4 is 96m^3/h, the power required by the pump in order to achieve this flow rate when using discharge throttling is 26.32kW and when using speed regulation is 11.32kW. The energy consumed over a 24 hour period would thus be 631kWh and 271kWh

respectively. These calculations based on the average flow rate differ to those calculated using the duty cycle of the pumped liquid. This demonstrates the importance of considering the duty cycle of pumped liquid when calculating the energy usage as noted by Rice [34].

The energy usage of pump system A when discharge throttling and then speed regulation is used to achieve the flow rates shown in Fig. 7.5 is 809.0 kWh and 749.4 kWh respectively yielding a saving of 59.68kWh. The use of speed regulation therefore results in a 7.3% reduction in the energy consumption of the system. The comparison of the energy usage for pump system A operating under the duty cycles shown in Fig. 7.4 and 7.5 demonstrates the effect that the duty cycle has on the possible energy savings when discharge throttling is replaced by speed regulation.

The results shown above demonstrate that the energy savings which may be achieved in any pump system when discharge throttling is replaced by speed regulation is dependent on the duty cycle of the process. These results also show that the energy usage calculations have to be performed using the actual flow rate duty cycle and not the average flow rate. The power savings and therefore the energy savings that may be achieved by replacing discharge throttling with speed regulation in order to control the flow rate is also determined by the pump installation. To demonstrate this a second pump system (pump system B) shall now be considered.

7.2.2 Pump System B Simulation Results

Pump system B differs in that it has a large static head and a much smaller friction head than pump system A, therefore most of the energy supplied by the pump is used to lift the liquid from the suction point in the system to the discharge point. Due to the large static head the speed range over which the pump varies is greatly reduced, as shown in Fig. 7.6. The simulated load torque placed on the VSD as the simulated flow rate is varied is shown in Fig. 7.6. The load placed on the VSD as the flow rate is varied when discharge throttling is simulated is the same as that shown in Fig. 7.1 as discussed in section 3.3.2.

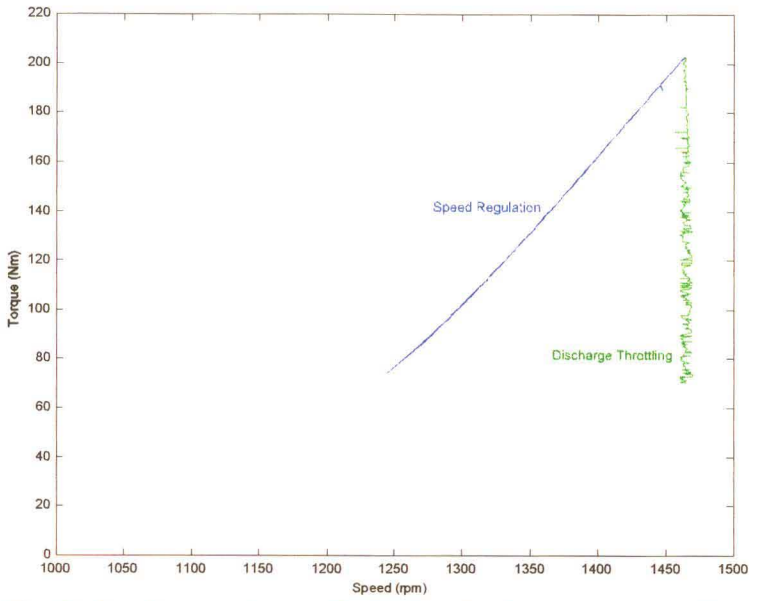


Fig. 7.6 Load torque place on the VSD to simulate pump system B controlling the flow rate for discharge throttling and speed regulation

The power consumed by the VSD for the load shown in Fig. 7.6 is shown in Fig. 7.7. The difference in the power consumed by the VSD when discharge throttling and speed regulation are used to control the flow rate is also displayed in Fig. 7.7 (red plot) and is seen to be less than that of pump system A when comparing Figs 7.2 and 7.7. Pump system B has a larger static head and a lower frictional head than pump system A, therefore the difference in the power used by pump system A and B when speed regulation is used to control the flow rate is due to the system characteristics. In pump system B most of the power supplied to the pump is used to lift the liquid from the suction point to the discharge point, this input power is required for both discharge throttling and speed regulation. The power savings achieved when using speed regulation for any pump system is gained by the reducing of the frictional losses in the system (throttling value is a frictional loss). As pump system B has a large static head the frictional losses required by the throttling valve in order to reduce the flow rate is smaller than that required in pump system A, therefore the reduction in the frictional losses in pump system B are smaller when speed regulation is used resulting in a lower power saving being achieved.

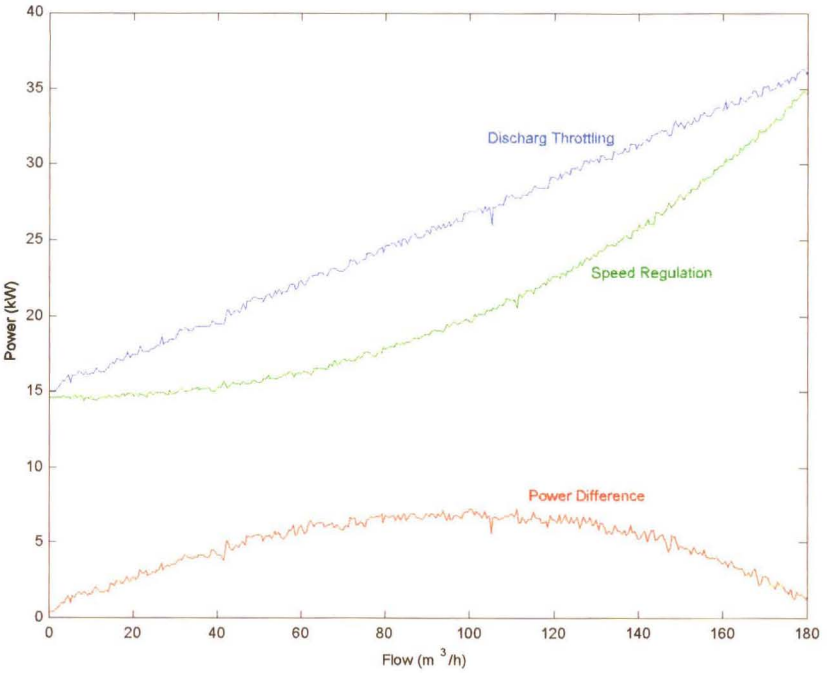


Fig. 7.7 Input power to VSD and power difference as flow rate is varied using discharge throttling and speed regulation for pump system B.

The curves shown in Fig. 7.7 are used to calculate the efficiency and energy usage of pump system B when discharge throttling and speed regulation are used to control the flow rate. The output power of pump system B (the energy gained by the liquid being pumped) as the flow rate is varied is calculated in Appendix C. The efficiency of pump system B when discharge throttling and speed regulation are used to control the flow rate is shown in Fig. 7.8. The efficiency of pump system B is higher when speed regulation is used to control the flow rate as shown in Fig. 7.8, resulting in reduced power usage as shown in Fig. 7.7. The efficiency of pump system B is higher than that of pump system A due to the shorter length of pipe used which results in a lower frictional loss in the system.

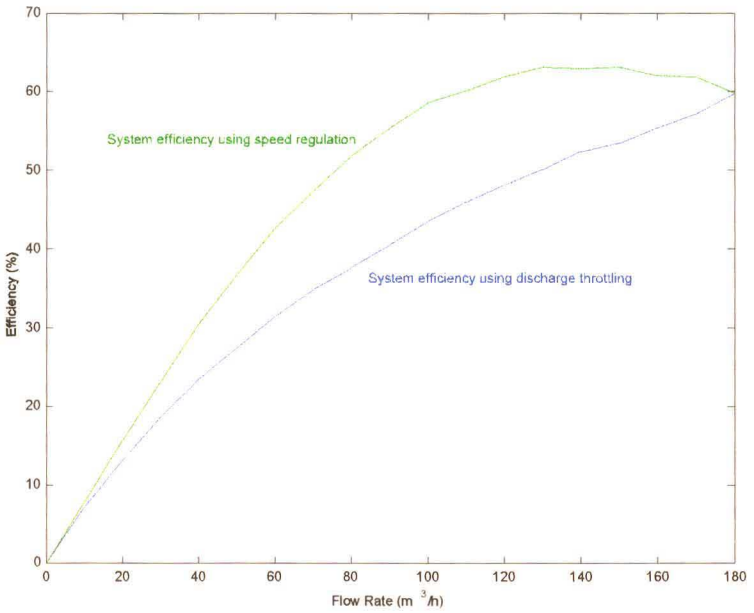


Fig. 7.8 System efficiency using discharge throttling and speed regulation for pump system B.

The energy usage of any pump system is dependent on the duty cycle of the flow rates required. The energy usage of pump system B using discharge throttling and speed regulation for the flow rates shown in Fig. 7.4 are 631.8kWh and 496.6kWh respectively. The energy that can be saved by replacing discharge throttling with speed regulation in pump system B when the flow rates shown in Fig. 7.4 are required is 135.2kWh. The use of speed regulation to control the flow rate of pump system B when the flow rates required follow that shown in Fig. 7.4 would result in a 21.4% reduction in the energy usage of the system.

The energy usage of pump system B using discharge throttling and speed regulation are used to control the flow rate as shown in Fig. 7.5 is 822.6kWh and 795.8kWh respectively, which gives an energy saving of 26.2kWh. The use of speed regulation to control the flow rate of pump system B when the flow rates required follows that shown in Fig. 7.5 would result in a 3.2% reduction in the energy consumption of the system.

The use of speed regulation in place of discharge throttling will always result in some energy saving provided that a reduced flow rate is required. From the simulated results shown in Fig. 7.2 and 7.7 it can be seen that pump system A requires less power than pump system B to pump the liquid when speed regulation is used. The difference in power usage is due to the system the pump is operated in. Pump system B has a larger static head than pump system A, this results in the speed range over which pump system B can operate being reduced, as the head the pump is able to produce is related to the pump speed. The larger the speed range over which the pump can operate the larger the power difference will be when discharge throttling is replaced by speed regulation as shown in Figs 7.2 and 7.7. A natural conclusion would be that pump system A would produce higher energy saving than pump system B if discharge throttling was replaced by speed regulation in order to control the flow rate. If the duty cycle of the pump system is now considered it can be shown that the above statement can often be in error. If pump system A has a duty cycle as shown in Fig. 7.5 and pump system B has a duty cycle as shown in Fig. 7.4 the result of replacing discharge throttling with speed regulation in order to save energy will result in pump system B producing a higher energy saving (21.4%) than pump system A (7.3%). The importance of considering both the system characteristics and the duty cycle of the pump system can now be seen as they both have an effect on the possible energy savings. The importance of using the actual flow rate duty cycle to calculate the energy usage of a pump system has also been demonstrated (for pump system A). Here it was shown that by using the average flow rate to calculate the energy usage an incorrect energy usage value is attained. These results are also supported by Rice [34] who expresses the importance of considering the whole pump system and the process when performing an energy evaluation.

The possible energy savings that can be achieved by replacing a fan system using dampers with a system in which speed regulation is used to control the flow rate is also investigated. The investigation into the fan system will demonstrate the importance of considering the fan system and the duty cycle of the flow rates when determining the possible energy savings that are achievable.

7.3 Test Bed Fan Simulation Results

The fan system described in Chapter 3 is simulated on the test bed system, where the flow rate of the fan is altered using dampers and speed regulation. The results will enable the efficiency and energy usage of the fan system to be calculated. The calculated energy saving will enable conclusions to be drawn on the effect that flow rate duty cycle has on the energy savings. Only one fan system is simulated as fans are chosen to operate in a specific system, and do not have an associated head as with pump systems. Two fan systems with the same system resistance will always produce the same flow rate. It is therefore not possible to setup two simple fan systems (fan system with ducting and dampers) as considered in this study that will produce identical flow rates at one fan speed and deviate as the fan speed is reduced. The load placed on the VSD versus the speed of the VSD, is shown in Fig. 7.8 for both methods of flow control.

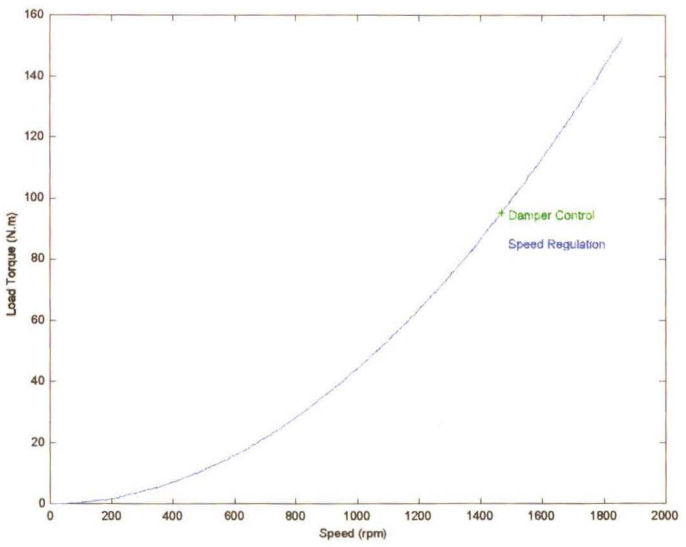


Fig. 7.8 Load torque placed on the VSD to simulate the fan load using damper control and speed regulation.

The load placed on the VSD when dampers are used to control the flow rate remains fixed as the load does not change when the damper settings are changed. The speed of the fan can be increased above 1500 rpm when speed regulation is used to control the flow rate, due to the ability of the VSD to run the induction motor above its nominal speed. The fan is also capable of

being run at speeds up to 2000rpm, as described in Chapter 3. The power consumed by the VSD as the flow rate is altered using dampers or speed regulation is shown in Fig. 7.9.

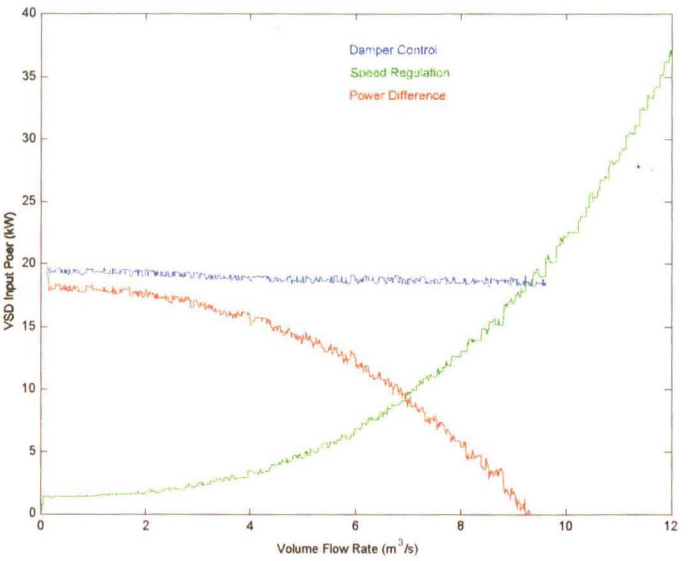


Fig. 7.9 Input power to the VSD and the power difference as the flow rate is varied using dampers and speed regulation.

Under speed regulation the power drawn by the VSD in achieving a desired flow rate is less than that for the fan running at fixed speed using dampers to control the flow rate. From Fig. 7.9 it can be seen that fans may produce higher flow rates than the system in which dampers are used, as the fan speed can be increased above 1500rpm due to the VSD being able to generate speed above the nominal speed of the induction motor.

The results from the test bed simulation for the fan system shown in Fig. 7.9 are used to calculate the efficiency and energy usage of the fan system when dampers and speed regulation are used to control the flow rate of the system. The output power of the fan system is the energy gained by the air as it is moved from the inlet of the fan system to the exit. No head exists in the fan system therefore only kinetic energy is gained by the air (the output power versus the flow rate is calculated in Appendix C). The efficiency of the fan as the flow rate is varied is shown in Fig. 7.10

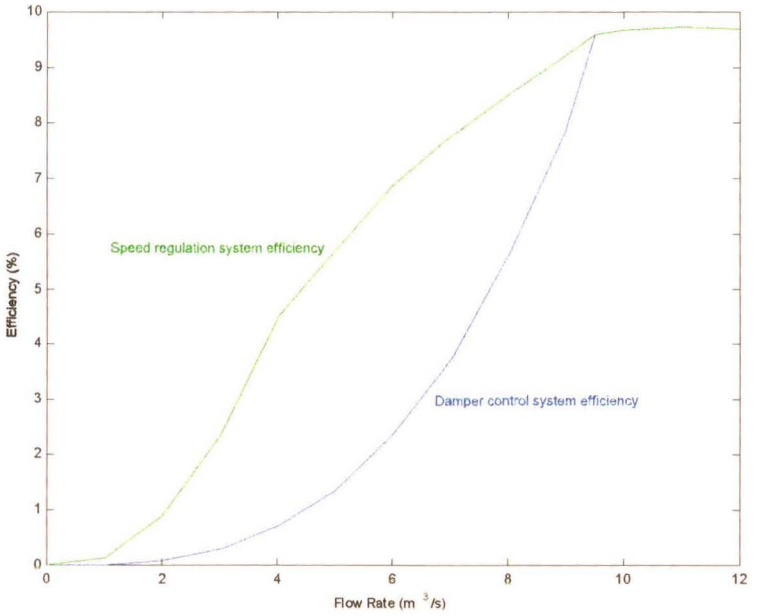


Fig. 7.10 Efficiency of fan system using dampers and speed regulation to control the flow rate.

The efficiency of the fan system using speed regulation to control the flow rate is higher than that of the same fan system using dampers as shown in Fig. 7.10. As with pump systems the energy usage of the fan system is dependent on the duty cycle of the flow rates required. Two duty cycles running over a twenty four hour period are presented in Figs 7.11 and 7.12. The two duty cycles are presented in order to evaluate the effect of the duty cycle on the possible energy savings. Fig. 7.11 was chosen as it has a low flow rate for extended periods of time with short periods where the flow rate is high and has an average flow rate of $4.5m^3/s$. Fig. 7.12 was chosen as it has long periods of time where the flow rate is high with short periods where the flow rate is low and has an average flow rate of $8.4m^3/s$.

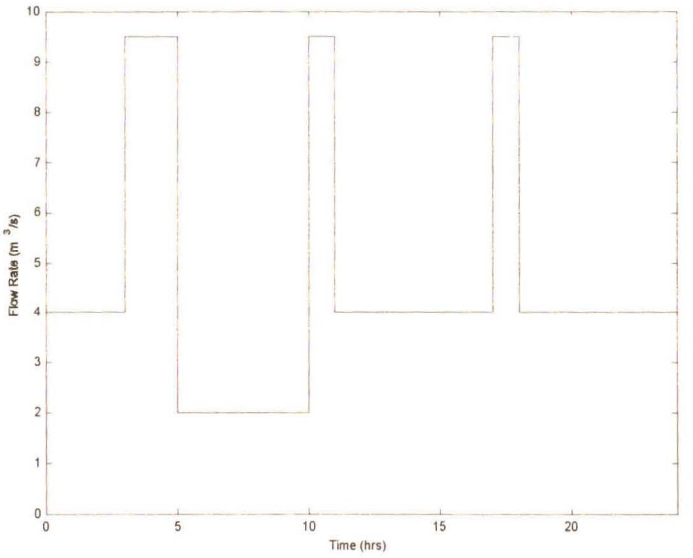


Fig. 7.11 Fan system duty cycle used to calculate the energy usage of the fan system.

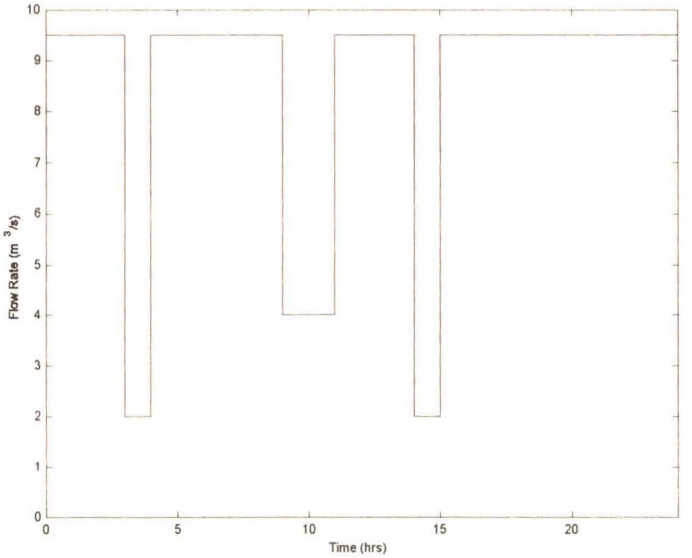


Fig. 7.12 Fan system duty cycle used to calculate energy usage of fan system.

The energy usage of the fan system over a twenty four hour period will be calculated based on the duty cycle presented in Figs 7.11 and 7.12. The energy usage of the fan system when dampers are used to control the flow rate in order to achieve the flow rates shown in Fig. 7.11 is 465.4kWh. The energy usage of the same fan system in which the flow rate is controlled using speed regulation in order to achieve the flow rates shown in Fig. 7.11 is 141.1kWh. The use of speed regulation results in a saving of 324.3kWh which is a 69.6% decrease in the energy usage of the fan system. In order to demonstrate the importance of considering the flow rate duty cycle it will be shown how the use of the average flow rate can yield incorrect results. The average flow rate for the duty cycle shown in Fig. 7.4 is $4.5m^3/s$, the power required by the fan in order to achieve this flow rate when using discharge throttling is 19.39kW (remains fixed for all flow rates) and when using speed regulation is 3.58kW. The energy consumed by the VSD in pumping this flow rate over a 24 hour period would thus be 465.4kWh and 85.9kWh respectively. The energy usage calculations based on the average flow rate for the fan system using speed regulation differs from those calculated using the duty cycle. This demonstrates the importance of the duty cycle when calculating the energy usage as demonstrated for the pump systems and noted by Rice [34]. The energy savings achievable are dependent on the duty cycle of the flow rates required, therefore a second duty cycle is presented to illustrate this.

The energy usage of the fan system using dampers to achieve the duty cycle in Fig. 7.12 is 465.4kWh and has the same energy usage as the duty cycle shown in Fig. 7.11 as the fan draws a constant power irrespective of the flow rate when dampers are used as described in Chapter 2. The corresponding energy consumption of the fan system using speed regulation is 398.7kWh. The energy savings that can be achieved by using speed regulation rather than dampers to achieve the duty cycle shown in Fig. 7.12 is 66kWh, and is a 14.3% decrease in the energy usage.

The calculations of the energy usage of the fan system based on the simulation results shown in Fig. 7.9 and the duty cycles shown in Figs 7.11 and 7.12 highlight the importance of understanding the full process of the fan system when evaluating the energy usage of fan systems. Failing to consider the fan system and the flow rate duty cycle will result in incorrect results being attained. The use of an unrealistic or incorrect energy evaluation will result in the return of

investment time calculated being incorrect which could have significant impacts on whether the VSD installation was viable or not.

7.4 Test Bed System Power Usage

One of the main advantages of the test bed system is its ability to simulate any load characteristic that VSDs can be loaded with, in industry. Another advantage of the test bed system is the reduced power usage: due to the power from the DC motor being returned back to the DC link, the power supplied by the three phase mains supply is less than the power being drawn by the VSD in maintaining a given speed under a specific load. The power usage of the test bed system depends on the load placed on the VSD and the speed of the VSD. Fig. 7.13 shows the power usage of the test bed system versus the simulated flow rate of a pump in which speed regulation is used to control the flow rate. The power drawn by the VSD is also plotted in Fig. 7.13 allowing a comparison to be drawn on the efficiency of the test bed system.

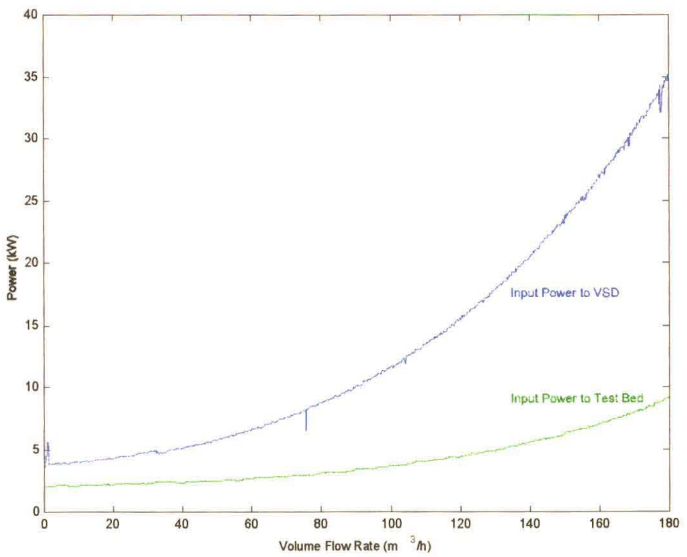


Fig. 7.13 Plot of power flow into test bed and power flow into VSD Vs simulated pump flow rate

The power flow into the test bed system is less than the power flow into the VSD, thus ensuring the test bed system is energy efficient. The reduced power usage of the test bed system reduces the cost of performing simulations on the test bed system. The efficiency of the test bed system, which relates the power flowing into the VSD to the total losses in the system (which is equal to the power flowing into the DC link from the three phase mains supply), can be calculated using Eq. 7.1, the results of which are shown in Fig. 7.14.

$$Efficiency = \left(1 - \frac{L_{tb}}{P_{vsi}}\right).100 \tag{7.1}$$

where L_{tb} is the total losses in the test bed system (input power to test bed $L_{tb} = P_s$).

P_{vsi} is the power flow to the VSI.

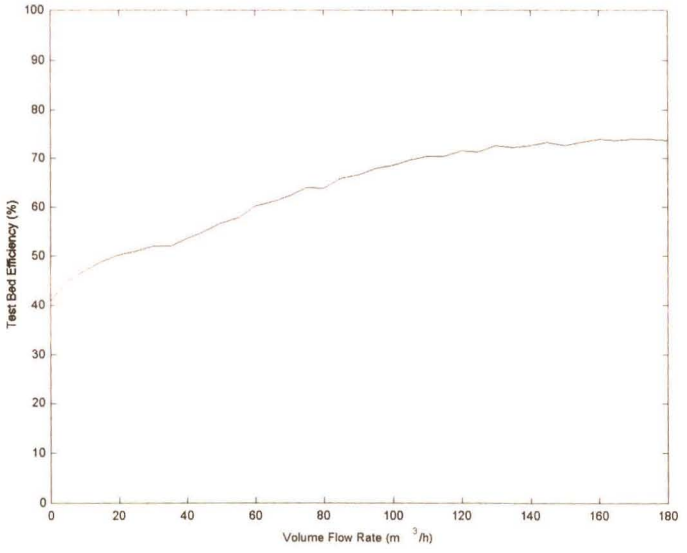


Fig. 7.14 Test bed efficiency for pump load simulation

The efficiency and power usage for test bed system are not shown for the remaining pump and fan simulations, as they follow similar trends.

7.5 Conclusion

A number of pump and fan simulations were performed on the test bed system based on load curve calculations performed in Chapter 3. The test bed simulation results showed that the power usage of any pump or fan system could be reduced by replacing discharge throttling or dampers with speed regulation in order to control the flow rate. The results presented in this chapter indicated that the achievable energy savings in any pump or fan system are dependent two factors: the pump or fan system (system head and characteristics); and the associated duty cycle. A failure to consider the pump or fan system characteristics will result in incorrect values being used in calculating the energy usage of the system. Using an incorrect duty cycle or using the average flow rate will result in an incorrect energy usage evaluation being performed.

The importance of the various factors involved in an energy savings evaluation can now be seen, as an incorrect evaluation of a pump or fan system will not result in realistic returns as described by Rice [34]. The advantage of the test bed system which enables the power usage of a VSD to be measured while driving a pump or fan is now evident as it enables a true reflection to be gained on the effect that the pump or fan system has on the power usage of the VSD.

The power usage of the test bed system was also presented where it was demonstrated that the test bed system has a high efficiency and requires less power to be drawn from the mains supply than that flowing into the VSD. This power usage is reduced due to the connection of the DC drive and VSD in the test bed system which enables the power generated by the DC drive to be used by the VSD in the test bed system.

Chapter 8

Conclusion

8.1 General

An increase in electricity consumption in South Africa (see Fig. 1.1) has resulted due to population and industrial growth. Resulting in a reduction in Eskom's spare generation capacity. Instead of building new power stations Eskom has set out an integrated electricity policy with the aim of reducing the electricity consumption through the more efficient use of electricity.

The use of VSDs are being considered as a means of improving the energy usage of systems. Due to the cost of VSDs it is important to consider their impact on the energy consumption of a system prior to being installed. A test bed system has been constructed which can measure the power usage of a VSD while driving a simulated industrial load. The results gained from the test bed system can be used to investigate the energy usage of the simulated system. Pumps and fans form a large part of the industrial load placed on the electrical grid. Therefore, by replacing fixed speed pump or fan systems with variable speed systems large energy savings could be achieved. The effect that the pump or fan system has on the power drawn by the VSD can be evaluated with the use of the test bed system, whereafter the energy usage of the pump or fan system can be determined from the associated flow rate duty cycle. To demonstrate the importance of investigating pump and fan systems thoroughly when determining the energy usage and savings that can be achieved by using variable speed drives a number of pump and fan simulations were performed on the test bed system. A conclusion will be drawn on the construction of the test bed system followed by the results gained from the test bed system. Suggestion for future work on both the test bed system and the results attained are also presented.

8.2 Test Bed System Implementation

The design and construction of the test bed system formed the majority of the work performed in this study. The work required in achieving the working test bed system can be split into two sections: the first was the design and construction of the test bed system hardware; the second was the development and implementation of the test bed control software.

Before the VSD in the test bed system could be setup and commissioned its operating principles had to be understood. With an understanding of the operating principles used in the FOC controller the VSD was setup and commissioned based on the procedures outlined in the VSD operation manual. The design and construction of the components required for the DC drive and the transducers used in the test bed system had to be understood and were carefully considered to ensure that the test bed system hardware would be reliable. To ensure the reliability of the hardware the following were considered: the earthing strategy used; the design and layout of the printed circuit boards; the component values and ratings; the layout of the individual components in the test bed system.

The test bed system software which is responsible for the simulation of the pump or fan load and the measurement of the power used by the VSD was implemented in RIDE software. The DC motor's armature current is controlled by the test bed software controller in order for it to produce the torque required to simulate a pump or fan load. The current controller used to control the DC motor' armature current was a discrete PI controller which was designed and simulated in Chapter 3. The simulated and actual system responses for the armature current controller matched each other indicating that the controller was performing correctly. The set point value used by the armature current controller is determined by the magnitude and type of flow control being simulated on the test bed system. The load curves for the various types of pump and fan systems from which the armature current controller set points are calculated were presented in Chapter 3. While the VSD is driving the simulated pump or fan load the test bed controller software calculates the VSD power usage from a number of measured variables. The VSD power usage is calculated based on the equations derived in Chapter 2. The variables measured by the

test bed controller matched that of the test equipment used in the setup and commissioning of the test bed system, this verified that the test bed hardware and software was operating correctly. Once the test bed system operation was verified the pump and fan simulations were performed.

8.3 Energy Usage of Pump and Fan Systems

The test bed system enables the power usage of a VSD to be measured while driving a simulated pump or fan load. Two pump systems and one fan system were simulated on the test bed system. Two pump systems were used to demonstrate the difference in power consumed by a VSD when the flow rate is controlled using discharge throttling and speed regulation, and the effect that the pump system has on the power usage of the VSD. The fan system was simulated to determine the power difference between a fan system using dampers and speed regulation to control the flow rate.

From the pump simulations it was found that the pump system has a large effect on the power usage of the VSD. In both pump system simulations the power required to pump a specific volume of liquid was less when speed regulation was used in place of discharge throttling. The energy usage of the pump systems were determined from the flow rate duty cycle. Two flow rate duty cycles were presented in order to demonstrate the effect of the duty cycle on the energy usage of the pump system. From the results presented it can be shown that the power drawn by the VSD while pumping at a specific flow rate was determined by the system characteristics. The energy usage of the pump system is determined from both the flow rate duty cycle and the power drawn by the VSD at the specific flow rates required in the flow rate duty cycle. The results have shown that energy can be saved in any pump system when replacing discharge throttling with speed regulation. Although energy can always be saved by using speed regulation it has also been shown that a system in which the VSD draws a low power for a specified flow rate (pump system A operating under the duty cycle shown in Fig. 7.5) will not necessarily result in larger energy savings than another pump system in which the VSD draws a larger power (pump system B operating under the duty cycle shown in Fig. 7.4) as the energy savings are directly related to the flow rate duty cycle. In general if the average flow rate required with respect to the maximum

flow rate is lower then the larger the possible energy savings will be when replacing discharge throttling with speed regulation. The importance of considering the flow rate duty cycle when calculating the energy usage of the pump system was also demonstrated, were it was shown that using the average flow rate gave incorrect results.

From the fan simulations it was found that the power usage of fixed speed fan systems remain constant for all settings of the dampers. The use of a VSD to control the flow rate of a fan system will always result in a reduction of the power usage of the fan system. The energy saving that will result from replacing a fixed speed fan system with a variable speed fan system depends on the flow rate duty cycle. As with pump systems the lower the average flow rate required with respect to the maximum flow rate the larger the energy savings will be when replacing dampers with speed regulation. The results presented on the use of VSDs in improving the energy usage of fan systems has shown the importance of considering both the power drawn by the VSD while moving a specific volume of air as well as the flow rate duty cycle of the air being moved. The results have also shown the importance of calculating the energy consumption of the fan system from the actual flow rate duty cycle, as using the average flow rate over a period of time will result in incorrect results. A failure to consider the power drawn by the VSD or the flow rate duty cycle will result in an incorrect energy evaluation being performed.

A large number of articles have been written on the use of VSDs to control the flow rate of pump or fan system and present energy saving results [5, 6, 34, 40, 41, 44, 51]. A number of the articles fail to examine both the physical system as well as the associated process [5, 44] just stating that energy can be saved through the use of VSDs in pump or fan systems. The results presented in this study demonstrate the importance of examining both the physical system and the associated duty cycle and are supported by Rice [34]. Rice [34] states that the person performing the energy evaluation must have an appreciation for both the process (flow rate duty cycle) and the mechanical aspects (pump or fan system) of the installation. The return on investment on the VSD purchased in order to affect the energy savings has not been investigated in this study, but will depend on capital cost of the VSD, the energy savings achievable when the VSD is installed and the cost of the electrical energy saved. The use of a VSD in a pump or fan system has a number

of advantages over fixed speed drives. If these advantages are taken into account the payback period of the VSD could possibly be reduced even further.

8.4 Advantages of Using VSDs in Pump or fan Systems

Besides the possible energy savings that can be achieved by using VSDs to control the speed of pumps and fans, there are other advantages that can also be considered :

- Soft start and stop, this reduces the mechanical stress on the pump or fan system during starting and stopping, resulting in reduced maintenance on the pump or fan system.
- The flow rate of the pump or fan is more controllable.
- Most AC VSDs draw unity power factor, due to the front end diode rectifier, thus no reactive power compensators are required to compensate for the reactive power drawn by induction motors.

8.5 Demonstration Use of Test Bed System

The test bed system can be used to evaluate the possible energy savings in any pump or fan system, this is done by altering the load simulator block such that the load placed on the VSD matches that of the physical pump or fan in the system. The pump or fan load simulation can then be performed in order to determine the energy usage of the pump or fan at various flow rates. The test bed system is a useful tool in making electricity users aware of the advantages of using VSDs to drive industrial loads.

8.6 Suggestions for Future Work

The test bed system has been constructed and demonstrates the power usage of pumps or fans using different methods of flow control. The following areas are suggested for future work:

- The test bed system should be evaluated against a pump or fan system in industry in order to demonstrate the ability of the test bed system to predict the power usage of the VSD for various flow rates.

- The energy savings that are achievable by using VSDs to control the speed of the driven equipment is not limited to pumps and fans. By performing simulations of other industrial loads, the energy usage of these systems can be attained and evaluated.

- An investigation into the minimum energy saving which should occur when discharge throttling is replaced by speed regulation which would result in a suitable ROI should be performed.

- It was noted that the VSD tripped when a mains dips occurred, the test bed system will allow further work to be performed on the operation of VSDs under mains fault conditions.

Appendix A

Field Oriented Control of the Induction Motor

This appendix gives a brief introduction to the method by which field oriented control is achieved in and induction motor.

A.1 Induction Motor Model

To understand the principles of FOC one first needs to understand the operating principles of the induction motor. When modeling an induction motor the following assumptions are made [7, 21].

- Magnetic saturation is neglected
- Air gap MMFs are represented by the fundamental components of their spatial distributions which are symmetrical about the magnetic axis of the windings of origin.
- Slotting effects are neglected.
- Magnetic Materials are free of eddy currents and hysteresis losses.
- The machine is perfectly symmetrical.

The theory on induction motors is well known [21] and is generally described in the dq reference frame, as this simplifies the computational requirements of the model when compared with three phase (ABC) models. In the dq reference frame the number of flux linkages are reduced as the induction motor is modeled as a two pole machine. The dq synchronous reference frame is often used to describe the operation of an induction motor under FOC as it further simplifies the describing equations [7]. An induction motor can be described in the dq synchronous reference frame by Eqs A.1 to A.4 [7].

$$v_{ds} = R_1 i_{ds} + p\lambda_{ds} - \omega\lambda_{qs} \quad (A.1)$$

$$v_{qs} = R_1 i_{qs} + p\lambda_{qs} + \omega\lambda_{ds} \quad (A.2)$$

$$v_{dr} = 0 = R_2 i_{dr} + p\lambda_{dr} - s\omega\lambda_{qr} \quad (A.3)$$

$$v_{qr} = 0 = R_2 i_{qr} + p\lambda_{qr} + s\omega\lambda_{dr} \quad (A.4)$$

where v_{ds} is the d-axis stator voltage

v_{qs} is the q-axis stator voltage

v_{dr} is the d-axis rotor voltage, which is zero in a squirrel cage IM

v_{qr} is the q-axis rotor voltage, which is zero in a squirrel cage IM

R_1 is the stator resistance

R_2 is the rotor resistance

i_{ds} is the d-axis stator current

i_{qs} is the q axis stator current

i_{dr} is the d-axis rotor current

i_{qr} is the q-axis rotor current

λ_{ds} is the d-axis stator flux linkage

λ_{qs} is the q-axis stator flux linkage

λ_{dr} is the d-axis stator flux linkage

λ_{qr} is the q-axis Stator flux linkage

s is the slip.

If the flux linkage terms (λ_{qs}, λ_{ds}, λ_{qr}, λ_{dr}) in Eqs A.1 to A.4 are expressed by the inductance components (L_{11}, L_{22}, L_m) of the induction motor as described in Eqs A.5 to A.8, the outcome is the set of Eqs A.9 to A.15.

$$\lambda_{ds} = L_{11} i_{ds} + L_m i_{dr} \quad (A.5)$$

$$\lambda_{qs} = L_{11} i_{qs} + L_m i_{qr} \quad (A.6)$$

$$\lambda_{dr} = L_{22} i_{dr} + L_m i_{ds} \quad (A.7)$$

$$\lambda_{qr} = L_{22} i_{qr} + L_m i_{qs} \quad (A.8)$$

where L_{11} is the stator self inductance

L_{22} is the rotor self inductance

L_m is the stator and rotor mutual inductances.

$$v_{ds} = (R_1 + L_{11}p)i_{ds} - \omega L_{11}i_{qs} + L_m p i_{dr} - \omega L_m i_{qr} \quad \text{(A.9)}$$

$$v_{qs} = (R_1 + L_{11}p)i_{qs} + \omega L_{11}i_{ds} + L_m p i_{qr} + \omega L_m i_{dr} \quad \text{(A.10)}$$

$$0 = (R_2 + L_{22}p)i_{dr} - s\omega L_{22}i_{qr} + L_m p i_{ds} - s\omega L_m i_{qs} \quad \text{(A.11)}$$

$$0 = (R_2 + L_{22}p)i_{dr} + s\omega L_{22}i_{dr} + L_m p i_{qs} + s\omega L_m i_{ds} \quad \text{(A.12)}$$

$$\lambda_{dr} = L_{22}i_{dr} + L_m i_{ds} \quad \text{(A.13)}$$

$$\lambda_{qr} = L_{22}i_{qr} + L_m i_{qs} \quad \text{(A.14)}$$

$$T_{em} = \frac{2}{3\omega_0}\left(\lambda_{dr}i_{dr} - \lambda_{qr}i_{qr}\right) \quad \text{(A.15)}$$

where ω_o is the synchronous speed of the machine.

The block diagram for an induction motor can be obtained from the formulae and is shown in Fig. A.1 (where $\sigma = 1 - \frac{L_m^2}{L_{11}L_{22}}$).

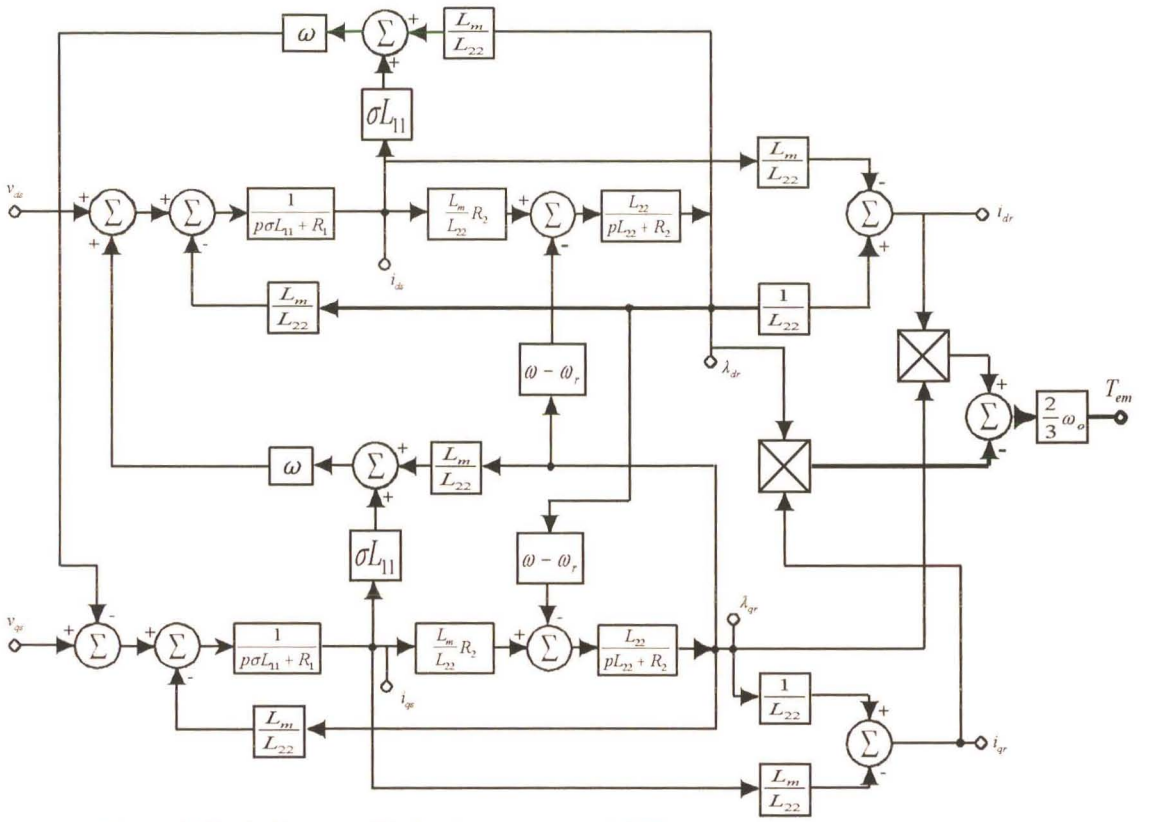


Fig. A.1 Block diagram of induction motor model [7].

A.2 Principles of Field Oriented Control

The objective of FOC is to establish and maintain an explicit angular relationship between the stator current and the rotor flux established in a induction motor [7, 25]. This angular relationship may be achieved by regulating the slip of the induction motor to a value which results in the rotor flux vector aligning with the d-axis component of the stator current vector, as shown in Fig. A.2. The magnetizing flux inside the induction motor is produced by the d-axis stator current vector, while the q-axis component of the stator current vector is used to regulate the torque produced by the induction motor. This results in a similar situation to that of the DC motor where the magnitude of the produced torque is a determined by the interaction of the magnetizing flux and the armature flux. Where the magnetizing flux (set by the field windings) is held constant the torque produced by the DC motor is controlled by the magnitude of the armature current.

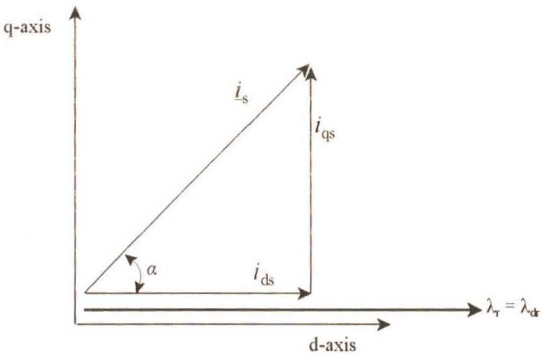


Fig. A.2 Conditions for FOC

For the conditions of Fig. A.2 to be met the following conditions have to hold

$$\lambda_{dr} = L_m i_{mr}^* = \text{constant} \tag{A.16}$$

$$\lambda_{qr} = 0 \tag{A.17}$$

For the above conditions to hold the following conditions also have to be met.

$$i_{qr} = 0 \tag{A.18}$$

also

$$s\omega = L_m R_2 i_{qs}^* / L_{22} \lambda_{dr} \tag{A.19}$$

where * denotes a reference value set by the controller to achieve the required performance from the motor.

The block diagram shown in Fig. A.1 can be simplified by assuming that a controller which will orientate $\underline{i}_s$ such that λ_{qr} equals zero can be designed. If λ_{qr} is to be zero then the output at point A in Fig. A.3 has to be zero, for this to hold point B has to be zero. For the output at point B to be zero then the two inputs C and D to the summer have to be equal [7].

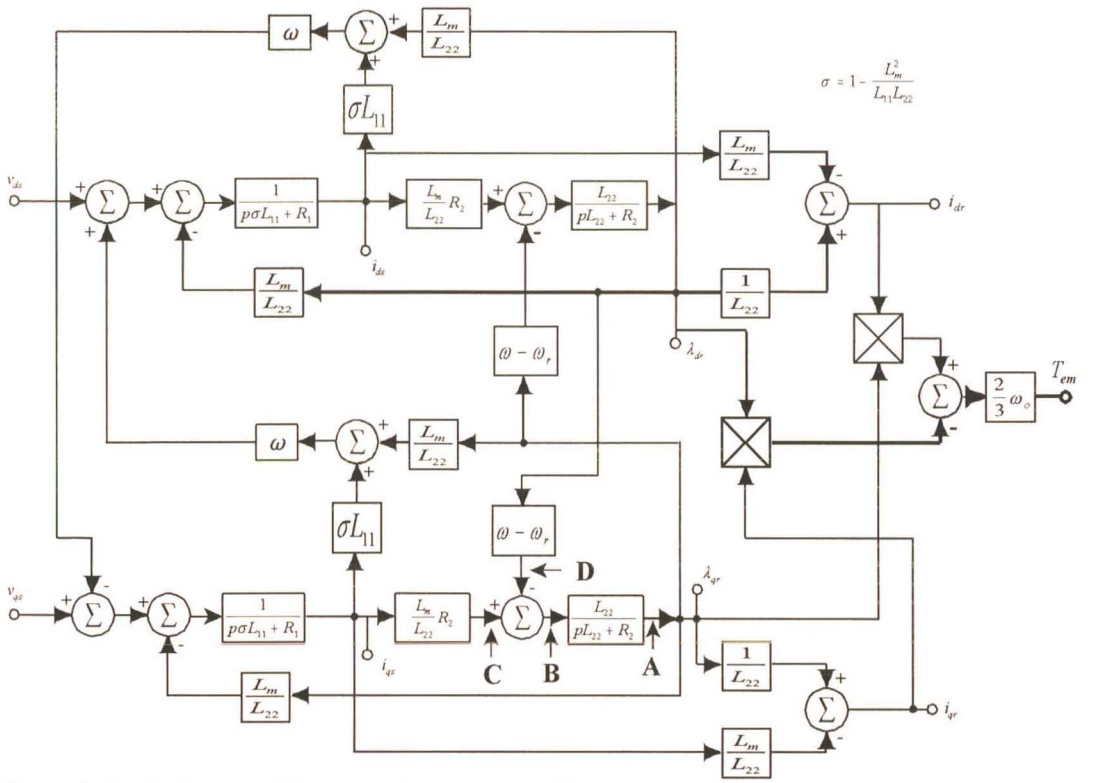


Fig. A.3 Block diagram of the induction motor model.

If a controller is designed (FOC controller) which ensures that the condition of FOC are assured, then any lines and blocks connected to points A, B, C and D in Fig. A.3 may be removed as they no longer contribute to the dynamics [7]. The resultant block diagram is shown in Fig. A.4.

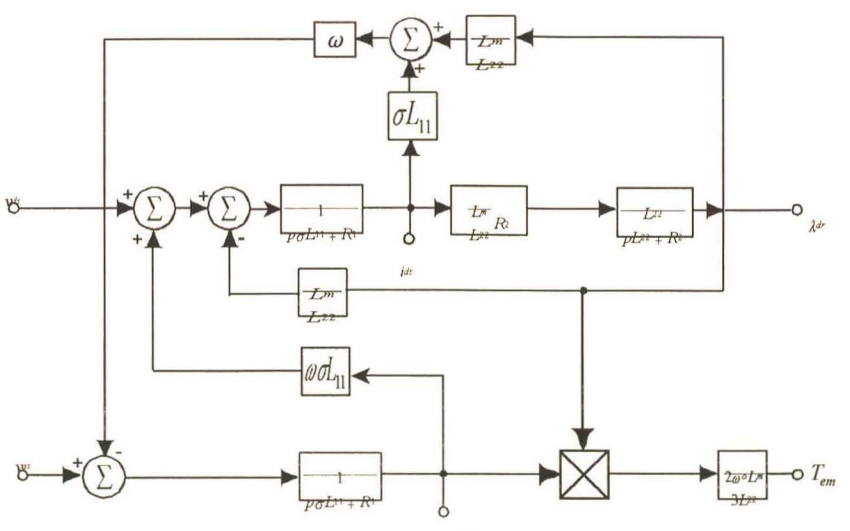


Fig. A.4 Block diagram of the induction motor when q-axis rotor flux is zero.

If the stator currents being fed to an induction motor are controlled to ensure that the conditions required for FOC are maintained, the stator currents may be treated as the inputs to the induction motor and the stator dynamics may then be omitted [7]. With the knowledge that the FOC controller will feed the motor with the required dq currents and the induction motor will in turn set up its own dq voltages to balance these currents, the block diagram of Fig. A.4 can be further simplified as shown in Fig A.5.

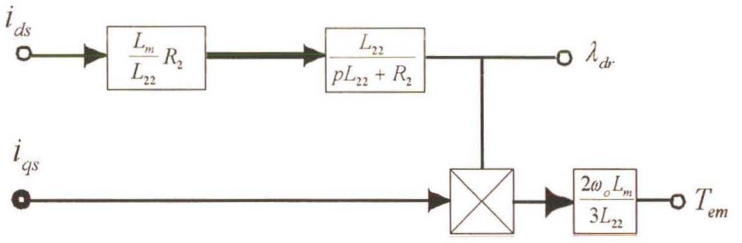


Fig. A.5 Induction motor under FOC with stator current control

From Fig. A.5 it can be shown that the induction motor performs in the same manner as a separately excited DC motor. This operation is evident when comparing the torque equation for the DC motor given in Eq. 2.14 and that of an induction motor running under FOC as given in Eq. A.20.

$$T_{em} = \lambda_{dr} i_{qs} K \tag{A.20}$$

From the above equation and block diagram, it can be seen that if the d-axis current is held constant thereby holding the magnetizing flux in the machine constant, the q-axis current can be used to control the torque of the SCIM by controlling the magnitude of the stator flux [7].

Appendix B

DC Motor Parameter Calculations

This Appendix presents the tests performed on the DC motor. The test results and calculations used to determine the DC motor parameter are presented.

B.1 Open Circuit Generator Test

The objective of the open circuit generator test is to identify the motor constant (K_m) which relates the induced armature voltage to the armatures rotational speed for a specific flux level inside the machine. This test was performed with the field windings being fed by the field supply that was implemented in the test bed system, such that the same flux level was present inside the machine for the test as would be when the DC motor was in operation.

The DC motor was driven by the induction motor at different speeds and the armature voltage was measured at each speed. The speed was measured in rpm using an optical tachometer. This measured speed was then converted into rad/sec as this is the DC motor model's units of speed, the result of which are shown in Table B.1.

The motor constant (K_m) was then calculated using Eq. B.1

$$K_m = \frac{V_a}{\omega} \tag{B.1}$$

The motor constant was found to be equal to 2.07 V/rad/sec which is the average of all the calculated motor constants shown in Table A.1.

Table B.1 Open circuit generator test results

Speed (rpm)	Speed - ω (Rad/sec)	Armature Voltage - V_a (V)	Motor Constant - K_m (V/rad/sec)
0	0	0	----------
118.6	12.4	26.0	2.09
233.3	24.4	51.0	2.09
346.9	36.3	75.8	2.09
462.7	48.5	100.7	2.08
571.9	59.9	125.0	2.09
693.5	72.6	150.8	2.08
803.9	84.2	174.9	2.08
921.9	96.5	200.0	2.07
1034.0	108.3	225.0	2.08
1153.0	120.7	250.0	2.07
1265.0	132.5	275.0	2.08
1387.0	145.3	301.0	2.07
1496.0	156.7	325.0	2.07
1607.0	168.28	349.0	2.07
1715.0	179.6	373.0	2.08
1847.0	193.4	401.0	2.07

B.2 No Load Motor Test

The objective of the no load motor test is to identify the coefficient of viscous friction. This is done by applying a fixed voltage on the armature and measuring the armature current and speed, the results of which are shown in Table B.2. The coefficient of viscous friction can be calculated using Eq. B.2 and B.3

Table B.2 DC motor no load test results

Va (V)	Speed (rpm)	Speed (rad/sec)	Ia (A)	Total Losses (W)	Elec Losses (W)	Mech. Losses (W)	B (N.m/rad/sec)
0	0	0	0	0	0	0	----------
23.9	100.0	10.5	1.7	40.63	0.32	40.31	0.368
46.4	201.0	21.1	1.79	83.06	0.35	82.70	0.187
68.7	302.0	31.6	1.87	128.47	0.38	128.08	0.128
92.1	408.0	42.7	1.97	181.44	0.43	181.01	0.099
113.7	500.0	52.4	2.09	237.63	0.48	237.15	0.087
135.0	605.0	63.36	2.28	307.80	0.57	307.23	0.077
156.2	700.0	73.3	2.36	368.63	0.61	368.02	0.068
178.3	801.0	83.9	2.44	435.05	0.65	434.40	0.062
199.6	900.0	94.3	2.56	510.98	0.72	510.26	0.057
222.0	1004.0	105.1	2.63	583.86	0.76	583.10	0.053
245.0	1109.0	116.1	2.73	668.85	0.82	668.03	0.050
266.0	1203.0	125.9	2.83	752.78	0.88	751.90	0.047
289.0	1308.0	136.9	2.98	861.22	0.98	860.24	0.046
309.0	1403.0	146.9	3.13	967.17	1.08	966.09	0.045
332.0	1505.0	157.6	3.30	1095.60	1.20	1094.40	0.044
354.0	1605.0	168.1	3.45	1221.30	1.31	1219.99	0.043
376.0	1702.0	178.2	3.65	1372.40	1.47	1370.93	0.043
397.0	1808.0	189.3	3.75	1488.75	1.55	1487.20	0.043

The energy entering the DC motor under steady state no load conditions is used to keep the motor rotating at a constant speed, this energy comprises of two components, the electrical losses and the mechanical losses. The mechanical losses can be calculated using Eq. B.2

$$M_{loss} = I_a V_a - (I_a^2 R_a + I_a V_b) \tag{B.2}$$

The product of I_a and V_a is the power entering the DC motor, the electrical losses at steady state comprise of the power dissipated in the armature resistance R_a and the power lost due to the volt drop across the brushes (V_b) in the machine.

The mechanical losses is the energy required to overcome the viscous friction of the rotational components in the system and can calculated using Eq. B.3.

$$B = \frac{M_{loss}}{\omega_r^{\,2}} \tag{B.3}$$

where ω_r is the rotational speed of the armature.

The coefficient of viscous friction spans a large range, due to static forces of friction which remain constant for any rotational speed [4]. These static forces of friction dominate the mechanical losses at low speed, due to the low mechanical losses at low speed the coefficient of friction has been taken as 0.044 N.m/rad/sec since the speed range of interest in the test bed system is generally of speeds higher than 500rpm (see pump and fan simulations in Chapter 3).

Appendix C

Pump and Fan System Calculations

This appendix gives the data used to calculate the pump and fan system characteristics as well as the pump and fan system useful output power calculations.

C.1 Tables Used For System Head Calculations

In a system where a pump is used, there is a need for piping which transports the fluid from one point to another. Whenever the pipes change direction a bend or elbow is used, this bend or elbow impedes the flow of liquid as the liquid has to change direction. When calculating system heads for pumps these bends or elbows are represented by an added length of pipe to the system which would provide the same impedance as that of the bend or elbow, these equivalent lengths are found experimentally and provided in tables as shown in Table C.1 [22]. The equivalent length of pipe that is used in the system head calculations in Chapter 3 is shaded.

The friction that a liquid experiences as it moves through a pipe is effected by the pipes internal diameter and the velocity of the liquid flowing through the pipe, this information is presented in tables and is determined experimentally. Table C.2 shows the friction head per 10m of pipe generated due to water flowing through a (5in) 127mm pipe at various velocities the table was adapted to SI units from [22]. Table C.2 was used to calculate the friction head of the pump systems described in Chapter 3, the shaded row indicates the data used to calculate the friction head for the pipe used in the pump systems described in Chapter 3.

Table C.1 Equivalent lengths of straight pipe for standard elbows

Inside Diameter of Elbow		Equivalent Length of Pipe	
(inches)	(millimeters)	(feet)	(meters)
0.5	12.7	1	0.305
0.75	19.05	1.5	0.457
1	25.4	2	0.610
1.25	31.75	3	0.915
1.5	38.1	4	1.220
2	50.8	5	1.525
2.5	63.5	6	1.830
3	76.2	8	2.440
4	101.6	11	3.355
5	127.0	15	4.575
6	152.4	18	5.490
7	177.8	20	6.10
8	203.2	24	7.320
10	254.0	32	9.760
12	304.8	40	12.200
14	355.6	48	14.640
16	406.4	58	17.690

Table C.2 Friction head for specific flow rates in a 127mm (5in) Pipe

Flow Rate		Liquid Velocity		Friction Head	
gpm	m^3/h	ft/s	m/s	ft/100ft	m/10m
70	15.90	1.12	0.34	0.21	0.021
75	17.03	1.14	0.34	0.24	0.024
100	22.71	1.63	0.49	0.41	0.041
125	28.39	2.04	0.62	0.64	0.064
150	34.07	2.45	0.74	0.88	0.088
175	39.75	2.86	0.87	1.18	0.118
200	45.43	3.27	0.99	1.48	0.148
225	51.10	3.67	1.11	1.86	0.186
250	56.78	4.08	1.24	2.24	0.224
275	62.46	4.50	1.37	2.72	0.272
300	68.14	4.90	1.49	3.14	0.314
350	79.49	5.72	1.74	4.19	0.419
400	90.85	6.54	1.99	5.40	0.54
450	102.21	7.35	2.24	6.70	0.67
475	107.89	7.76	2.36	7.42	0.742
500	113.56	8.17	2.49	8.12	0.812
550	124.92	8.99	2.74	9.60	0.96
600	136.28	9.80	2.98	11.30	1.13
650	147.63	10.62	3.23	13.20	1.32
700	158.99	11.44	3.48	15.10	1.51
750	170.34	12.26	3.73	17.20	1.72
800	181.70	13.07	3.98	19.40	1.94
850	193.06	13.89	4.23	21.70	2.17
900	204.41	14.71	4.48	24.00	2.4

C.2 System Head Calculations For Pump System A

The total system head has to be calculated for each flow rate in order for the system curve to be plotted on the pump characteristic curve in Chapter 3, Fig. 3.20. The method used to calculate the system head was discussed in Chapter 2, the head for pump system A system for a flow rate of 181m^3/h can be calculated as shown below.

Pump system A comprises of two static heads namely the suction head and the discharge head.

Static suction lift = 3m

Static discharge head = 6m

Total Static head = 9m

The equivalent length of piping is calculated as follows, using Table C.1, a 127mm elbow is equivalent to 4.575m of straight pipe, the total length of piping is now the sum of all the lengths of pipe, including the equivalent length of straight pipe representing the elbows.

Length of straight pipe = 3 +4.575+5+100+4.575+6+4.575+100 = 228.73m

The friction head due to the pipe can then be calculated from Table C.2, this table gives the friction head in meters per 10m of piping for a specific flow rate, the friction head can be calculated from Eq. 2.5. The friction head loss in meters per 10m is 1.94 m /10m for a flow rate of 181m^3/h, therefore the friction head is equal to 44.37m

The velocity head is calculated from Eq 2.6, and is equal to 0.8m. The total system head is thus the sum of all the heads or head losses and is calculated using Eq. 2.4

Total System head = 9 + 44.37 + 0.8 = 54.17m

The total system head the pump has to work against when the flow rate is 181m^3/h, is 54.17m.

The system heads for the other flow rates for pump system A are calculated in the same manner as shown above and are given in Table C.3. The calculated system head are used in Fig. 3.20 when plotting the pump and system characteristic curves in order to determine the torque required to drive the pump at a specific flow rate.

Table C.3 Total system head for varying flow rates for pump system A as shown in Fig. 3.18

Flow Rate (m^3/h)	Velocity (m/s)	Static Head (m)	Velocity Head (m)	Friction Heat (m)	Total Head (m)
0	0	9.00	0	0	9.000
15.9	0.348	9.00	0.006	0.480	9.486
17.0	0.342	9.00	0.006	0.549	9.555
22.7	0.497	9.00	0.013	0.938	9.950
28.4	0.622	9.00	0.020	1.464	10.484
34.1	0.747	9.00	0.028	2.013	11.041
39.7	0.872	9.00	0.039	2.699	11.738
45.4	0.997	9.00	0.051	3.385	12.436
51.1	1.119	9.00	0.064	4.254	13.318
56.7	1.244	9.00	0.079	5.124	14.202
62.4	1.373	9.00	0.096	6.221	15.317
68.1	1.495	9.00	0.114	7.182	16.296
79.5	1.745	9.00	0.155	9.584	18.739
90.8	1.995	9.00	0.203	12.351	21.554
102.2	2.242	9.00	0.256	15.325	24.581
107.8	2.367	9.00	0.286	16.972	16.257
113.6	2.492	9.00	0.316	18.573	27.889
124.9	2.742	9.00	0.383	21.958	31.341
136.3	2.989	9.00	0.455	25.846	35.302
147.6	3.239	9.00	0.535	30.192	39.727

158.9	3.489	9.00	0.621	34.538	44.159
170.3	3.739	9.00	0.713	39.342	49.054
181.7	3.986	9.00	0.810	44.374	54.184
193.1	4.236	9.00	0.915	49.634	59.549
204.4	4.487	9.00	1.026	54.895	64.921

C.3 System Head Calculations For Pump System B

The system head calculations for pump system B are performed in the same manner as that of pump system A. The system heads for various flow rates are shown in Table C.4. The calculated system head values are used in Fig. 3.20 when plotting the pump and system characteristics curves in order to determine the torque required to drive the pump at a specific flow rate.

Table C.4 System head for various flow rates for pump system B as shown in Fig. 3.19

Flow Rate (m^3/h)	Velocity (m/s)	Static Head (m)	Velocity Head (m)	Friction Head (m)	Total Head (m)
0	0.000	42	0.000	0.00	42.00
15.9	0.348	42	0.006	0.12	42.13
17	0.342	42	0.007	0.14	42.15
22.7	0.497	42	0.013	0.24	42.26
28.4	0.622	42	0.020	0.38	42.40
34.1	0.747	42	0.028	0.52	42.55
39.7	0.872	42	0.039	0.70	42.74
45.4	0.997	42	0.051	0.88	42.93
51.1	1.119	42	0.064	1.10	43.17
56.7	1.244	42	0.079	1.33	43.41
62.4	1.373	42	0.096	1.61	43.71

68.1	1.495	42	0.114	1.86	43.97
79.5	1.745	42	0.155	2.48	44.64
90.8	1.995	42	0.203	3.20	45.40
102.2	2.242	42	0.256	3.97	46.22
107.8	2.367	42	0.286	4.39	46.68
113.6	2.492	42	0.317	4.81	47.13
124.9	2.742	42	0.383	5.69	48.07
136.3	2.989	42	0.455	6.69	49.15
147.6	3.239	42	0.535	7.82	50.35
158.9	3.489	42	0.620	8.94	51.56
170.3	3.739	42	0.713	10.19	52.90
181.7	3.986	42	0.810	11.49	54.30
193.1	4.236	42	0.915	12.85	55.77
204.4	4.487	42	1.026	14.21	57.24

C.4 Output Power Calculations For Pump System A

The output power of a pump system can be defined as the energy gained by the liquid per unit time at the point where it is discharged. At the discharge point the liquid contains energy in two forms: kinetic energy which is the energy stored in the liquid due to it moving; and potential energy which is the energy gained by the water as it was moved from the inlet of the pump system to the discharge point. The output power of the pump system can be calculated from the flow rate and the static head in the pump system. A sample calculation of the method used to determining the output power of pump system A is shown. The remaining data is presented in tables. The output power of the system will be calculated for the system shown in Fig. 3.18, for a rate of 78 m^3/h (it is assumed that the pumped liquid is clean fresh water, where $1m^3$ of water weights 1000kg).

Firstly the energy required to lift the liquid from the inlet to the discharge point is calculated. From Fig. 3.18 it can be seen that the liquid is lifted through a vertical distance of 9m. The mass of liquid lifted in one second can be calculated using Eq. C.1.

$$W_l = \frac{Q}{(60 \cdot 60)} \cdot 1000 \tag{C.1}$$

where W_l is the mass of liquid lifted in one second in kg

Q is the flow rate in m^3/h.

Thus for a flow rate of $78m^3/h$ the mass of liquid lifted every second is 21.66kg. The potential energy gained by the liquid can be calculated using Eq. C.2.

$$E_p = W_l g h \tag{C.2}$$

where E_p is the gain of potential energy of the liquid lifted in Joules

g is the constant of gravity ($9.81 m/s^2$)

h is the vertical distance that the liquid is lifted in m.

Thus the energy gained by 21.66kg of liquid as it is lifted from the inlet point to the discharge point is 1912 J. Since this amount of energy is required every second (21.66kg is lifted every second) to maintain the flow rate, the power required to lift the liquid at a rate of $78m^3/h$ is equal to 1.912kW.

The energy stored in the liquid as kinetic energy has to be calculated, the velocity of the liquid has to be known prior to this and is calculated using Eq. C.3.

$$V_l = \frac{Q}{(60 \cdot 60) \cdot A_p} \tag{C.3}$$

where V_l is the velocity of the liquid in m/s

Q is the flow rate in m^3/h

A_p is the cross sectional area of the pipe in m^2.

The pipe used in Fig. 3.18 has an internal diameter of 127mm, thus the velocity of the liquid is 1.71 m/s for a flow rate of $78m^3/h$. The kinetic energy can now be calculated using Eq. C.4.

$$E_k = \frac{1}{2} W_l V_l^2 \tag{C.4}$$

where E_k is the kinetic energy of the liquid in joules

V_l is the velocity of the liquid in the pipe.

The kinetic energy in 21.66kg of liquid being pumped at $78m^3/h$ is equal to 31.67 joules. Since 21.66 kg of liquid is pumped every second the power required to keep the liquid flowing at $78m^3/h$ will be 0.03167 kW.

The total output power from the pipe system is the sum of the power required to lift the liquid and the power contained in the liquid itself, thus the total power equals 1.944kW.

The output power (energy gained by the liquid per unit time) of pump system A for discharge throttling and speed regulation is given in Table C.5. The output power for pump system A is the same for the system using discharge throttling and speed regulation due to the output power being calculated from the flow rate of the liquid.

Table C.5 Output power for pump system A

Flow (m^3/h)	Output Power (kW)
0	0
10	0.245
20	0.491
30	0.738
30	0.985
50	1.235
60	1.486
70	1.740
80	1.996
90	2.256
100	2.519
110	2.787
120	3.059
130	3.335
140	3.617
150	3.904
160	4.198
170	4.498
180	4.804

C.5 Output Power Calculations For Pump System B

The same method used to calculate the output power of pump system A is used to calculate the output power for pump system B, the largest difference is the head that the liquid is lifted (42m). The results of the output power calculations for pump system B are given in Table C.6. The

output power for pump system B operating under discharge throttling and speed regulation is the same as the output power is based on the flow rate of the system.

Table C.6 Output power for pump system B

Flow (m^3/h)	Output Power (kW)
0	0
10	1.14
20	2.29
30	3.43
40	4.58
50	5.73
60	6.88
70	8.03
80	9.19
90	10.35
100	11.51
110	12.67
120	13.85
130	15.03
140	16.21
150	17.39
160	18.58
170	19.78
180	20.99

C.6 Output Power Calculations For the Fan System

The output power of a fan system can be defined as the energy gained by the air being moved per unit time. The air gains kinetic energy as it moves through the fan, it is assumed that the air is not lifted but travels in a horizontal plane from the inlet of the fan to the outlet of the ducting. The output of the fan system is thus represented by the kinetic energy gained by the air. A sample calculation of the output power of the fan system described in Chapter 3 for a flow rate of 9 m^3/s is described whereafter the results of the remaining calculations are shown in Table C.7 (it is assumed that the air has a density of 1.2kg/m^3, and the cross sectional area of the ducting is 0.532m^2). The output power of the fan system for a specific flow rate is constant irrespective of the type of control used as the output power calculations are based on the flow rate.

The velocity of the air in the ducting can be calculated using Eq. C.5.

$$V_{air} = \frac{Q}{A_d} \tag{C.5}$$

where V_{air} is the velocity of the air in m/s

Q is the flow rate in m^3/s

A_d is the cross sectional area of the ducting in m^2.

The velocity of the air in the ducting is 16.917m/s when the flow rate is 9m^3/s. In one second 9m^3 of air is moved, thus in one second 10.8kg of air is moved. The kinetic energy of the air moved in one second can be calculated using Eq. C.4. and results in 1545 joules of energy being used to move the air. To maintain the above flow rate 1.545kW of power is absorbed by the air and is the useful energy gained from the system.

Table C.7 Output power for fan system described in Chapter 3

Flow (m^3/s)	Output Power (kW)
0	0
1	0.002
2	0.017
3	0.057
4	0.136
5	0.265
6	0.458
7	0.727
8	1.085
9	1.545
10	2.120

Appendix D

PWM/Tachometer Card Additional Data

This appendix gives additional data required when using the PWM and tachometer controller ASICs and the circuit diagram of the PWM/tachometer card.

D.1 PBM 1/87 Register Structure

For the PWM ASIC to perform the user required tasks, there are several registers that control the operation and output of the PWM ASIC. There are two main types of registers namely the control or status register and the data registers. The PWM ASIC uses the address line A0 and A1 to determine which type of register is being written to or read from. If the device is used in 16 bit bus mode only A1 is used to determine which register is used, A0 must be connected to ground. Table D.1 shows the value on the address line A1 and the register that is accessed in this mode.

Table D.1 Register addressing 16 bit mode

Address line A1	Register
0	Status Register / Control Register
1	Data Register

The main register is the control register and is only accessed during a write cycle to the device when A1 is set low. The Control Register controls the operation of the device, the structure of the control register is shown in Table D.2

Table D.2 structure of the control register

Bit	Name	Function
0	EIN	Global Enable : (1) enable, (0) diabled
1	IINT	Determines weather current polarity is internally /externally controlled
2	IINT1	Current polarity of inverter branch 1
3	IINT2	Current polarity of inverter branch 2
4	IINT3	Current polarity of inverter branch 3
5	-	Not Used
6	TESTFL	Test Flag, set to zero during normal operation
7	BUS16	Used to select 8 bit (0) or 16 bit (1) bus mode
8	WAD0	write address 0
9	WAD1	write address 1
10	WAD2	write address 2
11	WAD3	write address 3
12	RAD0	read address 0
13	RAD1	read address 1
14	RDSTART	Start read Cylcle, a high initiates the read function
15	-	Not Used

When reading from the Control register address the Status register is read, this register indicates the status of the PWM ASIC, the structure of the Status register is shown in Table D.3

Table D.3 structure of Status Register

Bit	Name	Function
0	WRFLAG	Indicated weather write register is clear (0)
1	RDFLAG	Indicates weather a read cycle is complete (0)
2	CALCFLAG	Indicates weather an internal processing cycle is in progress (1)
3..15		Used for testing purpose only (not specified)

When data is written to the data register the address of the data register has to be set by the address pointer in the control register (WAD0 to WAD3). After each write to the data register

the PWM ASIC automatically increments the address pointer. The address and functions of the data registers are shown in Table D.4. Data may only be written to the data registers when the WRFLAG in the Status register is low otherwise the data that is written during this period may be lost.

Table D.4 Address of data words for PWM ASIC

WAD0..3	Name	Function
0000 (0)	UA	Voltage component U_a
0001 (1)	UB	Voltage component U_b
0010 (2)	PHI1	phase angle, upper half (high 16 bits)
0011 (3)	DPHI1	Frequency, upper half (high 16 bits)
0100 (4)	PHI0	phase angle, lower half (low 16 bits)
0101 (5)	DPHI0	Frequency, lower half (low 16 bits)
0110 (6)	PHIADD	Difference phase angle upper half (high 16 bits)
0111 (7)	-	Not Used
1000 (8)	TAUS	Turn off Time
1001 (9)	TTOT	Dead Band
1010 (10)	TMIN	Turn on TIme
1011 (11)	VORTL	Switching Frequency
1100 (12)	TSTART	Start of Processing Cycle
1101 (13)	-	Not Used
1110 (14)	-	Not Used
1111 (15)	-	Not Used

Values can be read from the PWM ASIC using the RDSTART bits and the read address bits RAD, a read cycle can only take place when a write cycle is not in process. The RDSTART bit is set high to initialize a read cycle , the host processor may read the required values when this bit is set low, Table D.5 gives the addresses of the values when reading back data.

Table D.5 Address of data words which can be read from the PWM ASIC

RAD1..0	Name	Function
00 (0)	PHI0	phase angle, Lower half (low 16 bits)
01 (1)	PHI1	phase angle, Upper half (high 16 bits)
10 (2)	Uo	Voltage Value
11 (3)	-	Not Used

The data that is written to or read from the PWM ASIC has to be normalized such that the correct output is achieved, the input data values are normalized as follows :

- **Switching Frequency**

  The inverter switching frequency may be set by dividing the CLK input by means of a programmable pre-scaler VORTL, the switching frequency can be calculated using Eq. D.1 and D.2

$$n_{vor} = VORTL + 1 \tag{D.1}$$

$$f_{switch} = \frac{f_{clk}}{n_{vor} \bullet 1024} \tag{D.2}$$

  where f_{clk} is the clock frequency of the PWM ASIC.

- **Voltage components**

  The Voltage components UA and UB must be written in two's complement format using bit 15 as the sign bit.

- **Phase angle**

  The phase angle component is made up of two values giving a total of 24 bits of accuracy where PHI1 is the upper 16 bits and PHI0 is the lower 12 bits. The value of Phi (φ) can be calculated using Eq. D.3

$$phi = PHI1_{14...0} \bullet \frac{360°}{6 \bullet 2^{12}} + PHI0_{15...4} \bullet \frac{360°}{6 \bullet 2^{24}} \quad \text{(D.3)}$$

This gives us an angle resolution of 0.00000357° on the lower half (PHI0) and a resolution of 0.01465° on the upper half (PHI1)

- **Output Frequency**

The output frequency is achieved by adding a difference angle ($d\varphi$) to the total angle phi every calculation cycle, DPH0 is the lower half of the difference angle and DPHI1 is the upper half of the difference angle these values are represented in two's complement. The difference angle value required to output a specific frequency is given in Eq. D.4.

$$DPHI1_{11..0} \bullet 2^{12} + DPHI0_{15..4} = 3 \bullet 2^{34} \bullet n_{vor} \bullet \frac{f}{f_{clk}} \quad \text{(D.4)}$$

where f is the desired output frequency.

For the PWM ASIC running off a 20MHz clock the lower difference angle (DPHI0) gives us a resolution of 0.0003881 Hz and the higher difference angle (DPHI1) gives us a resolution of 1.589 Hz.

- **Turn on, Turn off and Dead band**

These values are related to the resolution of the switching signals, the minimum off time for a switch T_{aus} is given Eq. D.5, the dead-band between switches T_{tot} is given by Eq. D.6 and the minimum on time for a switch T_{min} is given by Eq. D.7.

$$T_{aus} = TAUS_{5..0} \bullet \frac{n_{vor}}{f_{clk}} \quad \text{(D.5)}$$

$$T_{tot} = TTOT_{5..0} \frac{n_{vr}}{f_{clk}} \tag{D.6}$$

$$T_{\min} = 2 \bullet TMIN_{5..0} \bullet \frac{n_{vor}}{f_{clk}} \tag{D.7}$$

- **Starting Time of the Calculation Cycle**

  The calculation cycle in each half switching period lasts 314 clock cycles, TSTART is dependent upon the scalling factor VORTL and must be set according to Eq. D.8 and is an integer value, this is done as the point where TSTART occurs in a half switching period varies, thus the calculation cycle has to be started earlier enough so that the switching data is available for the following switching period.

$$TSTART = \text{int}\left(512 - \frac{322}{VORTL + 1}\right) \tag{D.8}$$

D.2 TC3005 Tachometer ASIC

The tachometer controller ASIC, used on the PWM/tachometer cards is the TC3005H manufactured by Hanning Elektro-Werke [17]. Incremental rotary encoders working on an optical or magnetic basis produce three signals. Two of these signals are sine waves or square waves which are displaced by 90°, and are referred to as channel A and channel B signals , the third signal indicates the end of a revolution. By counting the number of sine or square wave pulses the position or speed of the encoder can be obtained, and the direction of rotation can be determined by the phase difference between channel A and channel B signals. The resolution of an encoder is limited by the number of output pulses per revolution, this resolution cannot be increased. If the encoder used, outputs sine and cosine waves, the tacho controller ASIC allows for the channel

A and channel B signals to be scanned by two flash A/D converters. This enables extra resolution to be gained within each half cycle of the input waveform, which are known as the quadrants of the encoder. The block diagram of the tacho controller ASIC, is shown in Fig. D.1

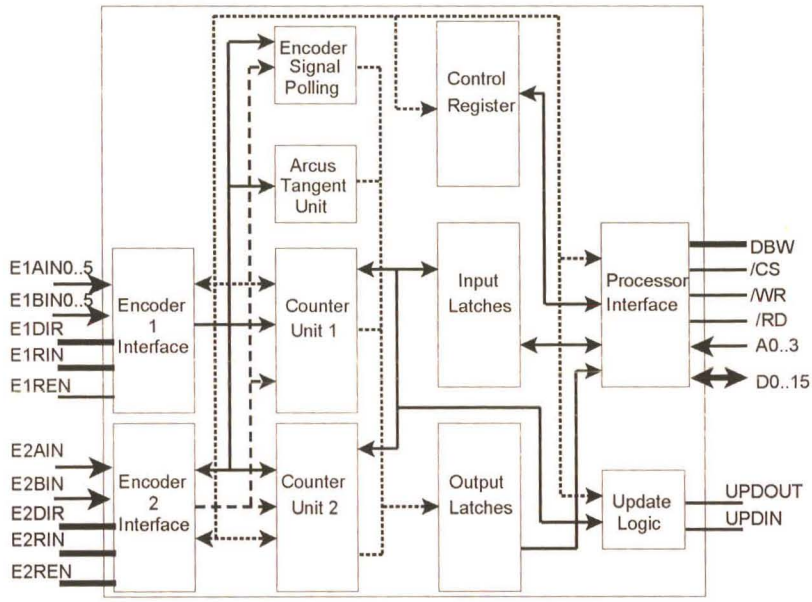


Fig. D.1 Block diagram of TC3005H

The main components of the tacho ASIC [17], as shown in Fig. D.1, are :

- **Encoder Interface**
- **Counter Units**
- **Output Latches**
- **Processor Interface**

Each of the components of the tacho ASIC will be discussed in detail

D.2.1 Encoder Interface

The TC3005H has two independent encoder interfaces. The first encoder interface (encoder 1) makes provision to connect two 6 bit flash A/D converters to determine a higher resolution angle, which can be obtained from an encoder that produces sine and cosine waves. It can also be used without the A/D converters, in which case the channel A and channel B signals (TTL signals) are fed to E1AIN5 and E1BIN5. This interface also has an input for the reference signal (E1RIN) which occurs once every cycle. The second encoder interface (encoder 2) accepts square wave pulses only from the channel A and Channel B signals (E2AIN and E2BIN) and a reference pulse(R2RIN).

There are two ways of connecting encoders to the TC3005H, the first is shown in Fig D.2. In this situation the speed or position information is gained from two encoders, if only one encoder is needed then it could be coupled to either of the encoder interfaces.

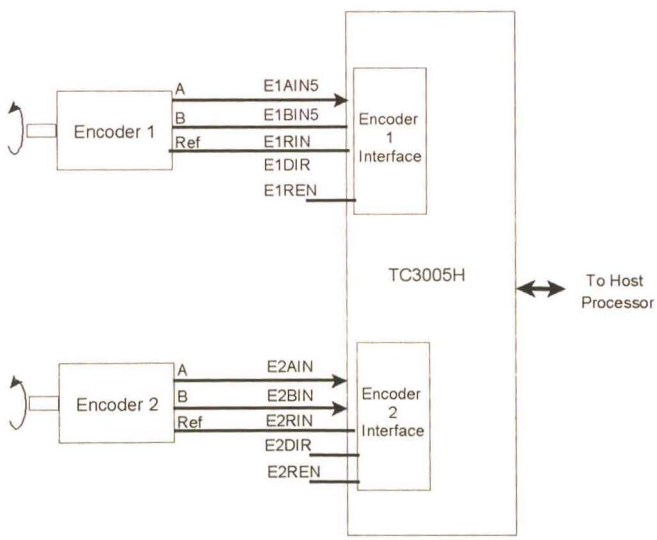


Fig. D.2 Connection of two encoders

If it is required that the high resolution angle be measured by the two flash A/D converters, then the connection shown in Fig. D.3 is used. In this situation encoder 2 interface and counter two,

count the quadrants from the encoder, while encoder 1 interface is used in conjunction with the ARCTAN register to measure the high resolution angle.

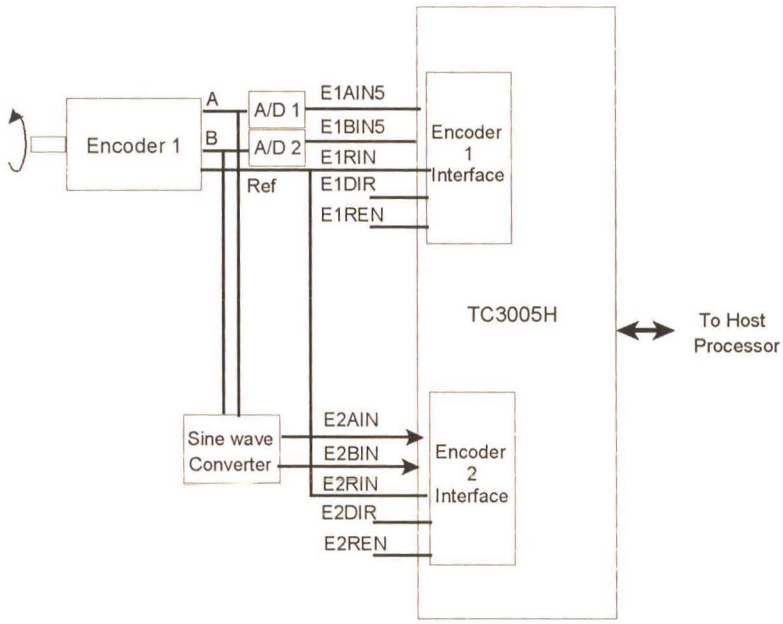


Fig D.3 Using encoder to get high resolution angle

D.2.2 Counter Units

The Counter units are used to count the incoming pulses to the encoder interface. The encoder interface determines whether the counter should be incremented or decremented.

- Counter 1 has 32 bits of resolution, which is used to give the position information on the encoder, together with the 8 bits (ARCTAN) of higher resolution that can be obtained when using the A/D converters.

- Counter 2 has 16 bits of resolution and is used to gain the position information on the encoder, connected to encoder 2 interface.

D.2.3 Output Latches

The output latches hold the values from the two counters and the ARCTAN unit. These latches can be updated by using the update logic, which is achieved by writing to the UPIN pin, or by writing to the SYSCON register. An internal update cycle, set by the UPDCYC register can also update the counter output latches.

D.2.4 Processor Interface

The processor interface allows the host processor to read from and write to the various control registers, status registers, and input and output latches. The addresses of all the registers are shown in Table D.6, with the bit width, address and access type for a 16 bit bus interface (set by the external DBW pin DBW = 0 for 8 bit mode, DBW = 1 for 16 bit mode). The addresses of the various register which the host processor has access to are shown in Table D.6.

Table D.6 Addresses of control and data registers

Name	Function	bit width	Address	Access type
SYSCON	System Control Register	16	0	R/W
CNTCON	Counter Control Register	16	2	R/W
ENSIG	Encoder Signal register	8	4	R
ARCTAN	High resolution angle	8	5	R
CT1DAT	Counter 1 data	32	6,8	R/W
CT2DAT	Counter 2 data	16	10	R/W
UPDCYC	Update cycle register	16	12	R/W
LINCNT	Line count register	16	14	R/W

The primary control register is the SYSCON register, which is used to set up the operation of the Tachometer ASIC. The CNTCON register is the counter control register and is used to setup the operation of the counter units. The ENSIG register holds the values of the encoder signal, while the output latches are updated, and the ARCTAN register holds the high resolution angle which is read from the A/D converters. CT1DAT and CD2DAT are the counter one and counter two registers, which hold the value of the counters after an update signal is given. When written to

these registers hold the counter values that are loaded into counter one or counter two after a counter reset has occurred. The UPDCYC register determines the period between up date cycles of the CT1DAT and CT2DAT registers when the tacho ASIC is setup to update these register automatically. The LINCNT register holds the maximum line count for counter two, once counter two reaches this value it is reset back to zero. For more detail on the tacho ASIC registers refer to [17].

D.3 PWM/Tachometer Card Encoder Interface

Due to the large number of configurations allowed by the Tachometer ASIC, there is a need for an encoder interface, which is configured with the use of jumpers (see Fig. D.4). Table D.7 describes the function of the jumpers.

Table D.7 Encoder interface jumper functions

Jumper	Function
JMP2	Encoder 2 external reference enable
JMP3	Encoder 2 external direction control
JMP4	Encoder 1 external reference enable
JMP5	Encoder 1 external direction control
JMP6	Encoder 2 Reference pulse selector
JMP7	Encoder 2 channel A selector
JMP8	Encoder 2 channel B selector
JMP9	Encoder 1 channel B input Selector
JMP10	Encoder 1 channel A input Selector

A description of the jumper functions are described below

- Jumper JMP2 and JMP4 set the E2REN pins high when the jumper is in place. This enables the counters to be reset when the reference pulse from the encoder(s) occur (when counters

are in counter modes 2 to 7).

- Jumpers JMP3 and JMP5 allow the encoder input channels (A and B) to be exchanged. These inputs are exclusive OR connected with the EnxDIR bits in the Status register, thus for the channels to be interchanged only one of the two inputs must be high.

- Jumpers JMP6, JMP7 and JMP8 select the source location of the encoder pulses for encoder 2 interface. These jumpers allow pulses coming into encoder 1 interface to be counted on counter two. The encoder 1 interface converts the sine wave signals to a square wave signals via a comparator circuit. This mode is used when the high resolution angle is being read by the A/D converters.

- Jumpers JMP9 and JMP10 are used to select the input for channel A and B signals for the encoder 1 interface, these signals can either be from the encoder 1 interface circuit which outputs square waves, or from the A/D converter if the high resolution angle is being measured.

D.4 PWM/Tachometer Card Address Decoding

The address decoding on the PWM/Tachometer Card relies on partial address decoding performed by the PC32 card in the form of decoded lines DECODES0 and DECODES1 [18]. These lines are selectable using jumper JMP1 on the PWM/tachometer card (see Fig. D.8) and determines the base address for the PWM/tachometer card according to Table D.8

Table D.8 Address of PC32 Card address decoded lines

Decodes Line	Address
DECODES0	819800
DECODES1	81A000

There may be a need to connect other interface cards or PWM/tachometer cards to the PC32 Card, therefore further address decoding is performed on address lines A8 to A10 to allow more cards to be connected to a single decodes line (DECODES0 or DECODES1). This gives the PWM/tachometer card an offset address which is selected by the use of a DIP switch. Table D.9 shows the DIP switch settings and the offset address.

Table D.9 Switch setting and offset address

SW1	SW2	SW3	SW4	Offset
off	off	off	off	Disabled
on	off	off	off	0x000
on	on	off	off	0x100
on	off	on	off	0x200
on	on	on	off	0x300
on	off	off	on	0x400
on	on	off	on	0x500
on	off	on	on	0x600
on	on	on	on	0x700

The address decoding performed on address lines A8 to A10 allows up to 8 cards to be connected to a single decodes line. Address line A11 is not decoded, as this address line has two different states in the DECODES0 and DECODES1 address range. Address Lines A7 to A4 are used to address or control the peripheral devices on the PWM/tachometer card. Table D.10 gives the offset address for the respective peripherals on the card.

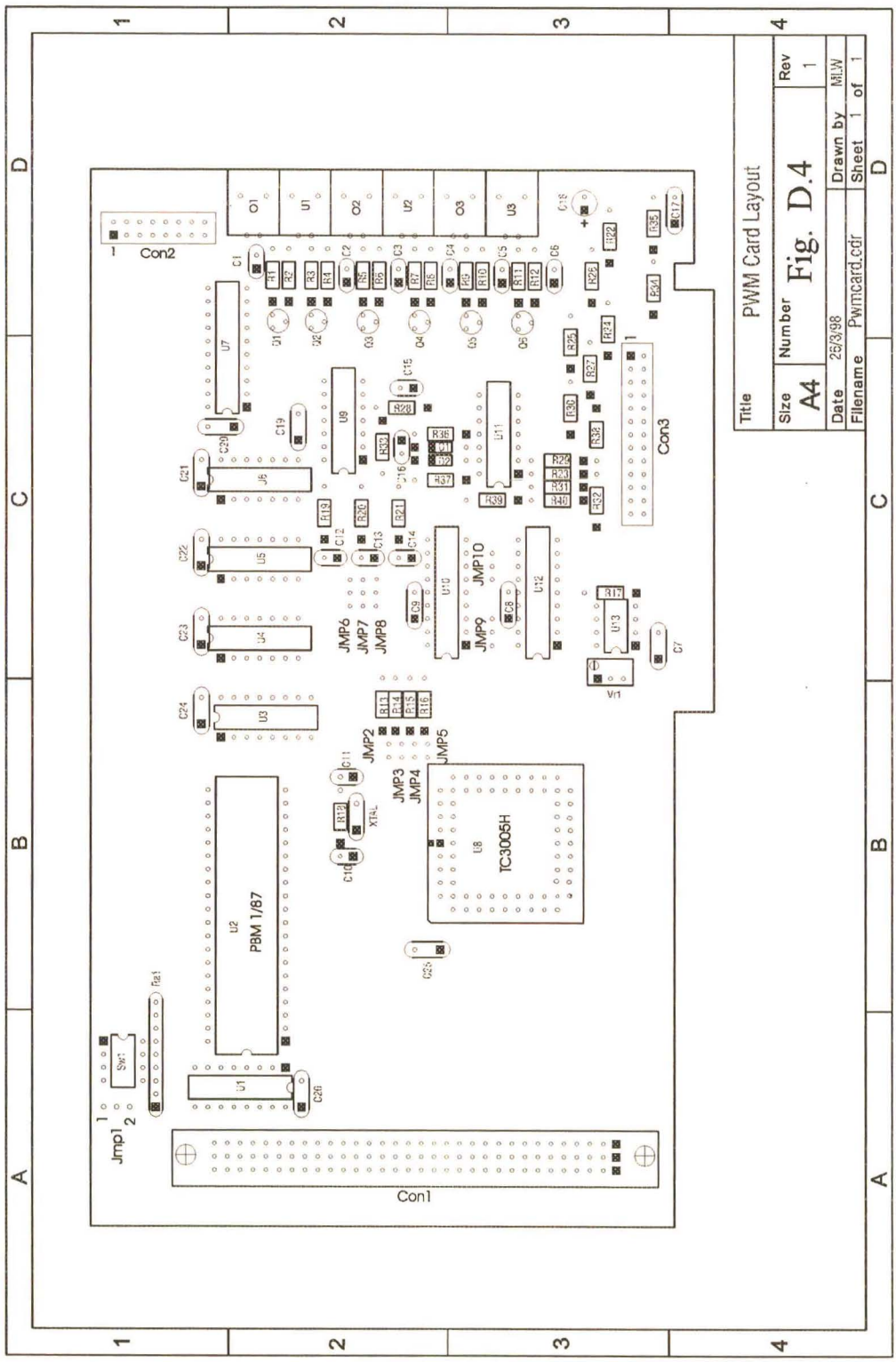
Table D.10 offset addresses of peripherals on PWM/Tachometer Card

Device/function	Offset Address
Chip Select PBM 1/87	0x00
Reset PBM 1/87	0x10
Chip Select TC3005H	0x20
UPDIN select TC3005H	0x30
Reset TC3005H	0x40

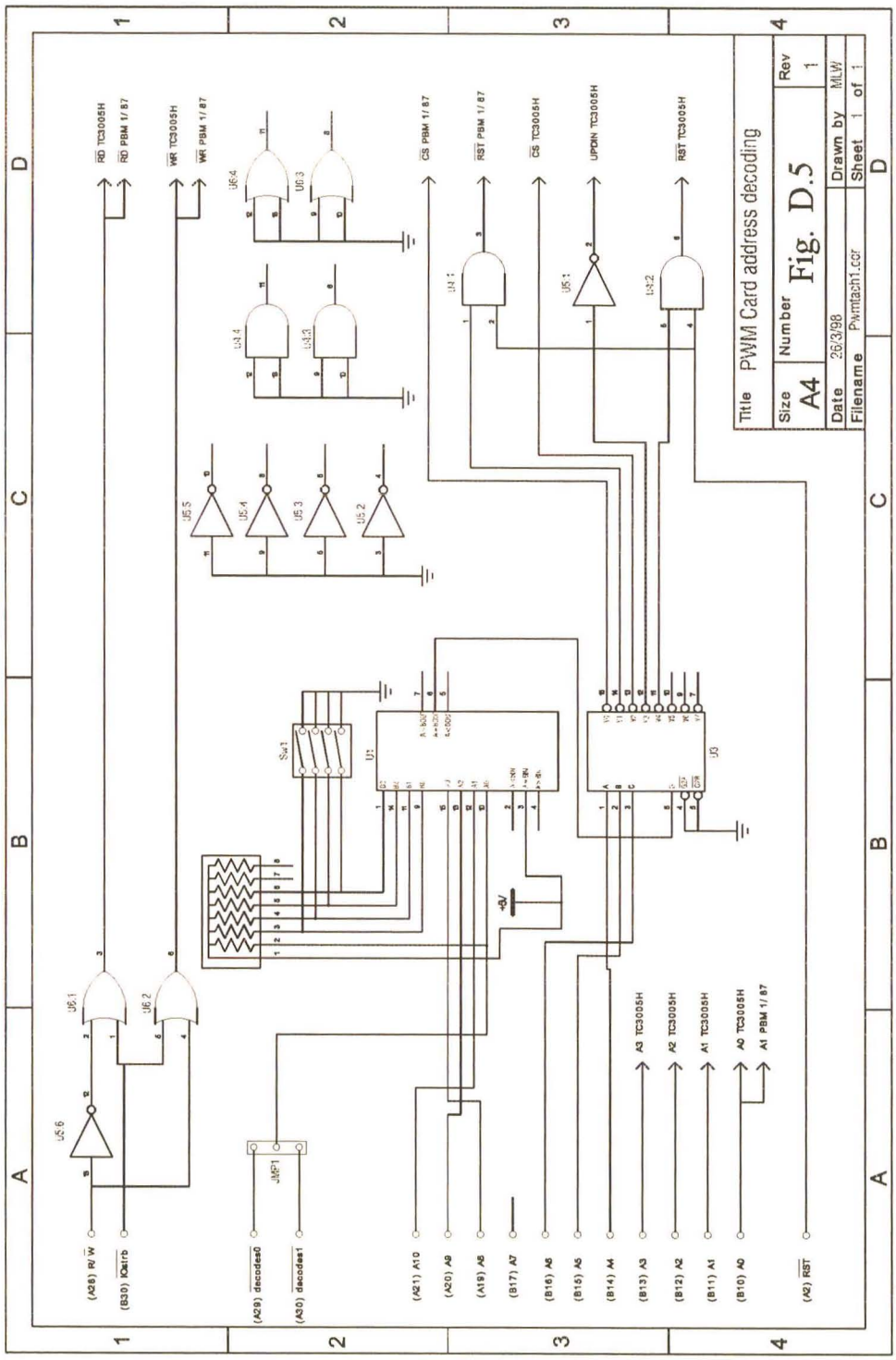
The lower four address lines A3 to A0 are used for internal addressing on TC3005H ASICs, with address line A0 connected to the A1 address line of the PWM ASIC (address line A0 on PWM ASIC connected to ground). Both ASICs are operated in 16 bit bus mode to get maximum performance from both the TMS320C32 processor and the ASICs.

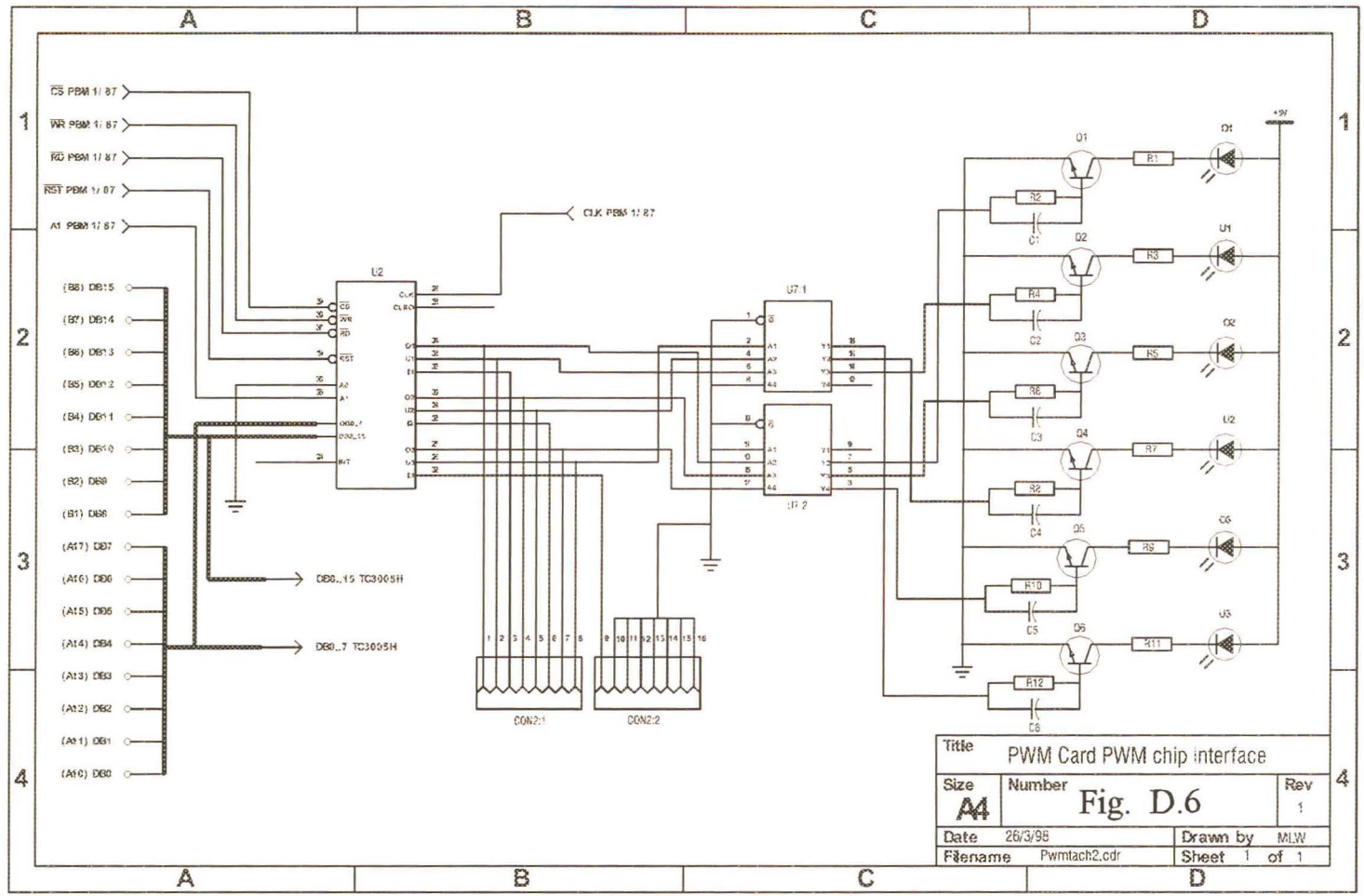
To determine the address to access a device, the base address (set by JMP1), the card offset address (set by dip switches), the peripheral offset (fixed) and the internal address of the relevant device have to be summed, giving the control address or device register address. Thus for the card operation in the test bed system, DECODES0 is used and the card is set as board 0. Thus to read the STATUS register from the PWM ASIC, the address would be calculated as follows, DECODES0 address = 0x819800 (Table D.8), board offset address = 0x000 (Table D.9), the device offset = 0x00 (Table D.10) and the STATUS register address 0x0 (Table D.6) are summed together to give an address in hex of 0x8198000.

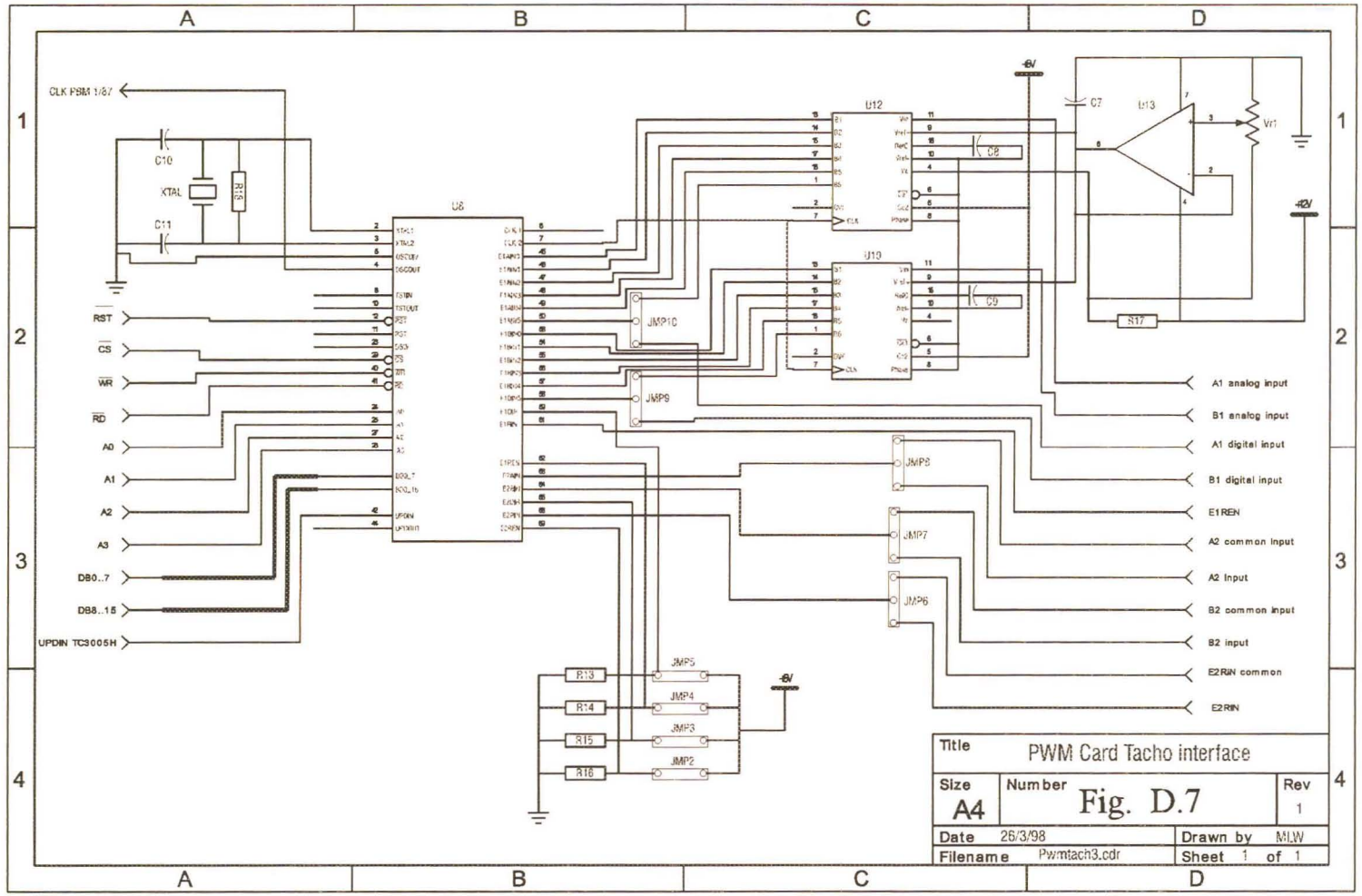
D.5 PWM/Tachometer Card Layout

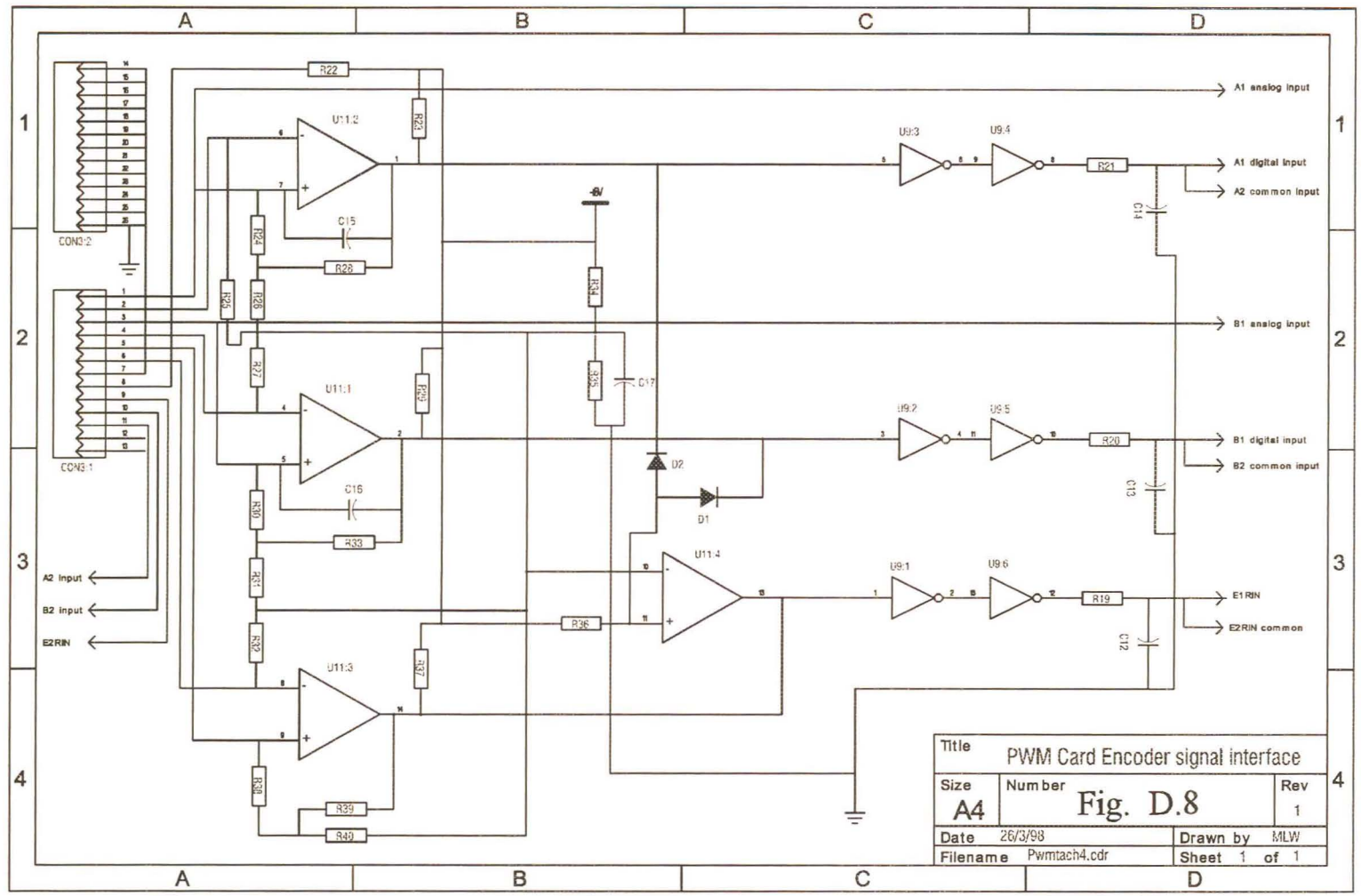


D.6 PWM/Tachometer Card Schematics




CS PBM 1/ 87
WR PBM 1/ 87
RD PBM 1/ 87
RST PBM 1/ 87
A1 PBM 1/ 87
CLK PBM 1/ 87
(B8) DB15
(B7) DB14
(B6) DB13
(B5) DB12
(B4) DB11
(B3) DB10
(B2) DB9
(B1) DB8
(A17) DB7
(A16) DB6
(A15) DB5
(A14) DB4
(A13) DB3
(A12) DB2
(A11) DB1
(A10) DB0
U2
U7:1
U7:2
DB8..15 TC3005H
DB0..7 TC3005H
CON2:1
CON2:2
Q1
Q2
Q3
Q4
Q5
Q6
D1
D2
D3
D4
D5
D6
R1
R2
R3
R4
R5
R6
R7
R8
R9
R10
R11
R12
C1
C2
C3
C4
C5
C6
Title PWM Card PWM chip interface
Size A4
Number Fig. D.6
Rev 1
Date 26/3/98
Drawn by MLW
Filename Pwmtach2.cdr
Sheet 1 of 1


A
B
C
D
1
2
3
4
CLK PSM 1/87
C10
XTAL
R16
C11
U8
U12
U13
U10
C7
C8
C9
Vr1
R17
+8V
+12V
RST
CS
WR
RD
A0
A1
A2
A3
DB0..7
DB8..15
UPDIN TC3005H
JMP10
JMP9
JMP8
JMP7
JMP6
JMP5
JMP4
JMP3
JMP2
R13
R14
R15
R16
A1 analog input
B1 analog input
A1 digital input
B1 digital input
E1REN
A2 common input
A2 input
B2 common input
B2 input
E2RIN common
E2RIN
Title
PWM Card Tacho interface
Size
A4
Number
Fig. D.7
Rev
1
Date
26/3/98
Drawn by
MLW
Filename
Pwmtach3.cdr
Sheet
1
of
1


A
B
C
D
1
2
3
4
CON3:2
CON3:1
R22
R23
R24
R25
R26
R27
R28
R29
R30
R31
R32
R33
R34
R35
R36
R37
R38
R39
R40
C15
C16
C17
U11:1
U11:2
U11:3
U11:4
-6V
D1
D2
U9:1
U9:2
U9:3
U9:4
U9:5
U9:6
R19
R20
R21
C12
C13
C14
A1 analog input
A1 digital input
A2 common input
B1 analog input
B1 digital input
B2 common input
E1RIN
E2RIN common
A2 input
B2 input
E2RIN
Title
PWM Card Encoder signal interface
Size
A4
Number
Fig. D.8
Rev
1
Date
26/3/98
Drawn by
MLW
Filename
Pwmtach4.cdr
Sheet 1 of 1

Appendix E

Circuit Diagrams, PCB Layouts and Component Calculations

This appendix describes the method used to calculating the burden resistor values used to terminate the transducers used in the system. The circuit diagrams and the PCB layouts are presented for reference.

E.1 Transducer Burden Resistor Calculations

The current and voltage measurement transducers used in the test bed system produce a current output proportional to the measured current and voltage respectively. The tachometer used in the test bed system produces a voltage higher than the allowable input voltage to the PC32 card analog interface, thus a resistor divider is required to reduce this voltage to a desirable level. The burden resistors and resistor divider required to ensure correct operation of the voltage and current transducers and the tachometer are discussed below.

E.1.1 Current Transducers

The current transducers used in the test bed system are LEM LA205-S [23], these sensors are Hall effect devices capable of measuring 200A RMS. The LA205-S produces a current proportional to the measured current. The LA205-S has an effective turns ratio of 1:2000 with a maximum output current of 100mA. It is important to ensure that the LA205-S is operated in its linear region, it is therefore important that the burden resistor used in conjunction with the LA205-S is less than 68Ω when the sensor is operated from a 12V supply.

The DC motor current measurement transducer is required to measure 100A DC (full load motor current), while the current transducer measuring the DC link current in the REFU drive is required to measure a peak input current of 90.5A. Both current sensors are setup to measure 125A peak (positive and negative) by using a burden resistor of 62Ω (Resistor R1 and R8, Fig. E.5), this given an output voltage of 3.88V. The gain of the amplifier in the PC32 analog interface has to be set to 2.57 (10/3.88) so that the input voltage to the PC32 card is 10V when the maximum current is being measured (125A).

E.1.2 Voltage Transducer

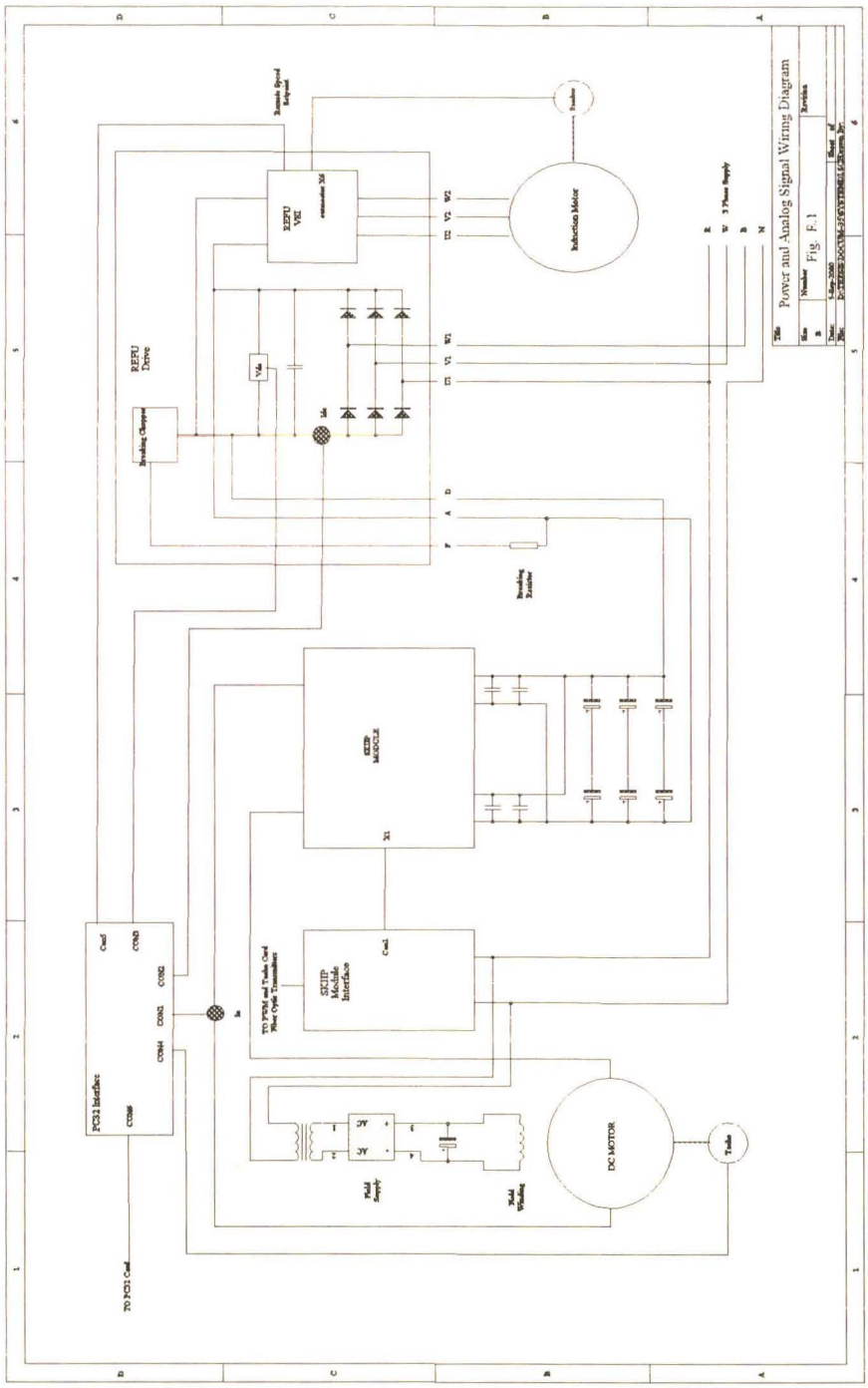
The transducer used to measure the DC link voltage is the LEM LV25-P [24]. The LV25-P is a Hall effect device which outputs a current proportional to the current flowing in the primary. The DC link voltage is measure by placing a resistor in line with the LV25-P, the current flowing through the primary of the device is then proportional to the DC link voltage. The LV25-P has a nominal primary current of 10mA and an effective turns ratio of 2500:1000 which results in a nominal primary current of 25mA. To ensure that the LV25-P operates in its linear region the burden resistor used on the secondary should not exceed 190Ω.

The voltage transducer is expected to measure a maximum DC link voltage of 680V which occurs prior to the REFU drive tripping out on a DC link over voltage fault. The LV25-P is setup in such a manner that it can measure 700V DC. A 69kΩ resistor is placed in series with the voltage transducer to limit the current flowing through the device to 10,29mA when the DC link voltage is 700V. When the DC link voltage is 700V the current flowing from the output of the LV25-P will be 25.74mA. A burden resistor of 180Ω is used on the output of the LV25-P (Resistor R15, Fig. E.5) which will produce a voltage of 4.63V. The gain of the amplifier used in the PC32 analog interface has to be set to 2.16 (10/4.63) such that input to the PC32 card is 10V when the maximum DC link voltage is being measured.

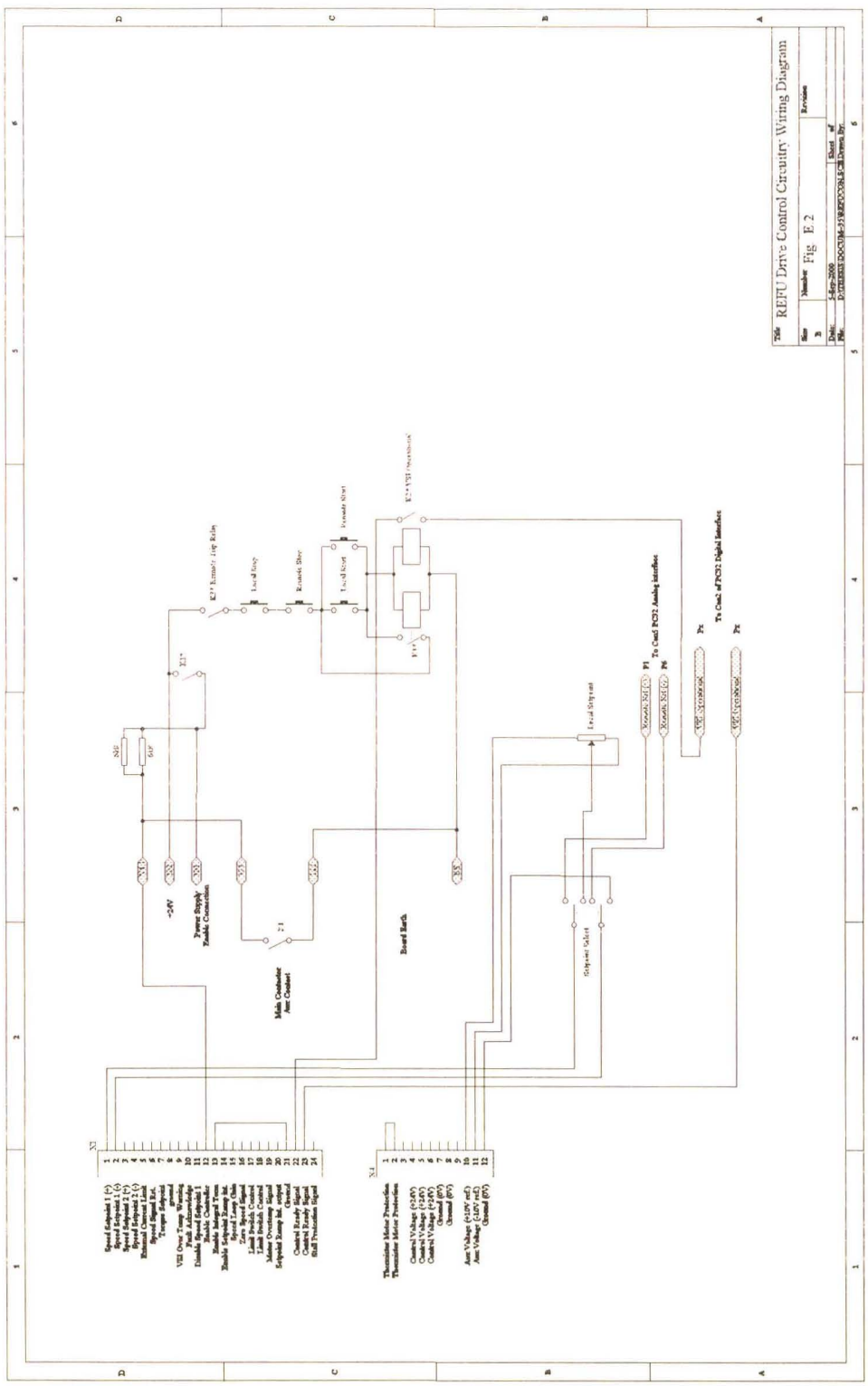
E.1.3 Speed Transducer

The analog tachometer used to measure the rotational speed of the IM and DC motor in the test bed system is a RADIO-ENERGIE Reo 444-RC1/CA. The Reo 444-RC1/CA produces 0.06V/rpm. The Reo 444-RC1/CA will therefore produce 120V at 2000rpm (2000rpm is the upper speed limit of the test bed system). A resistor divider is placed before the speed signal is fed to the analog interface, the resistor divider reduces the speed signal to 10V when the DC motor is running at 2000rpm. The gain of the amplifier in the PC32 card analog interface is set to unity to ensure that the entire input range of the PC32 card's A/D converter are used.

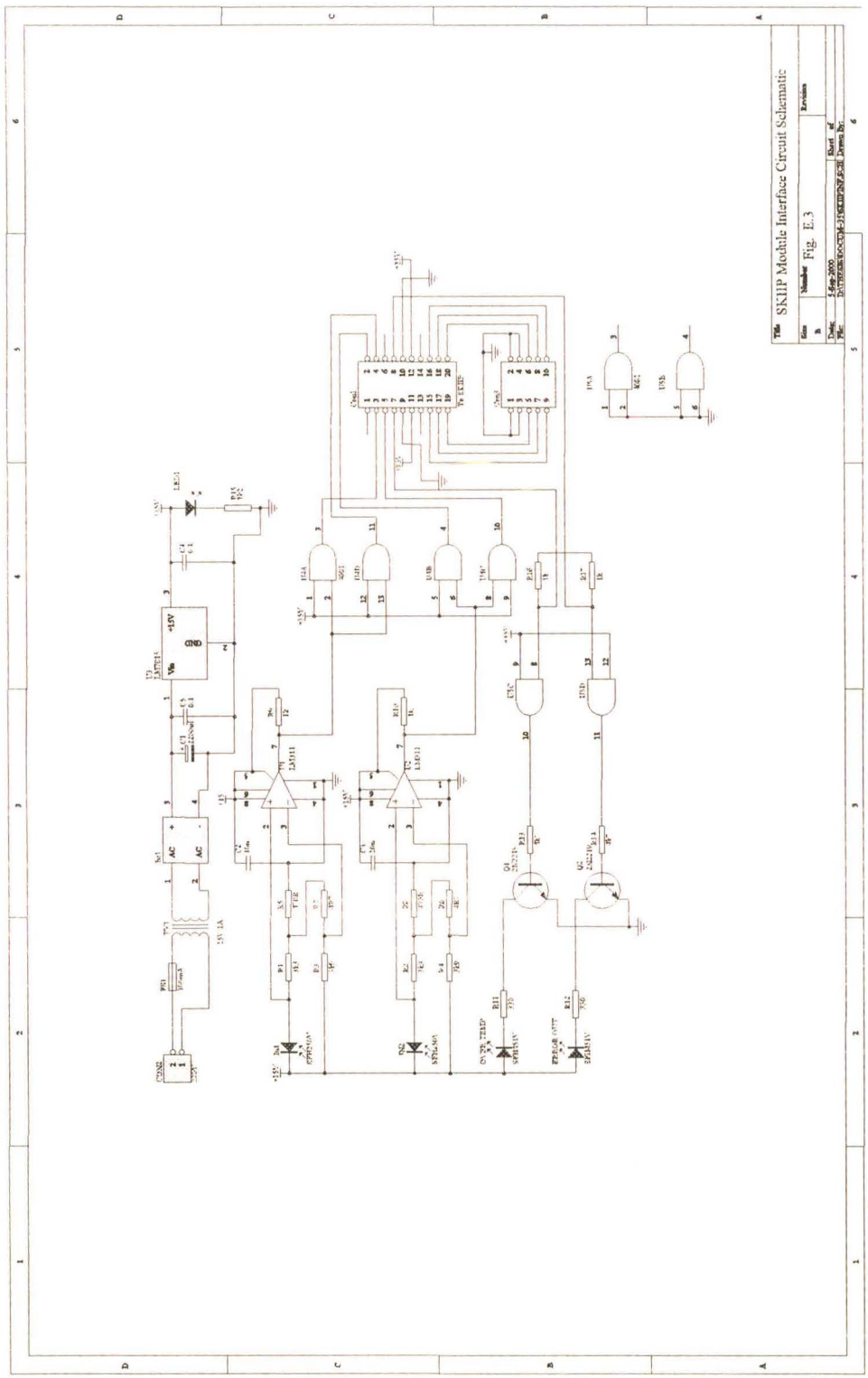
E.2 Test Bed System Wiring Diagrams

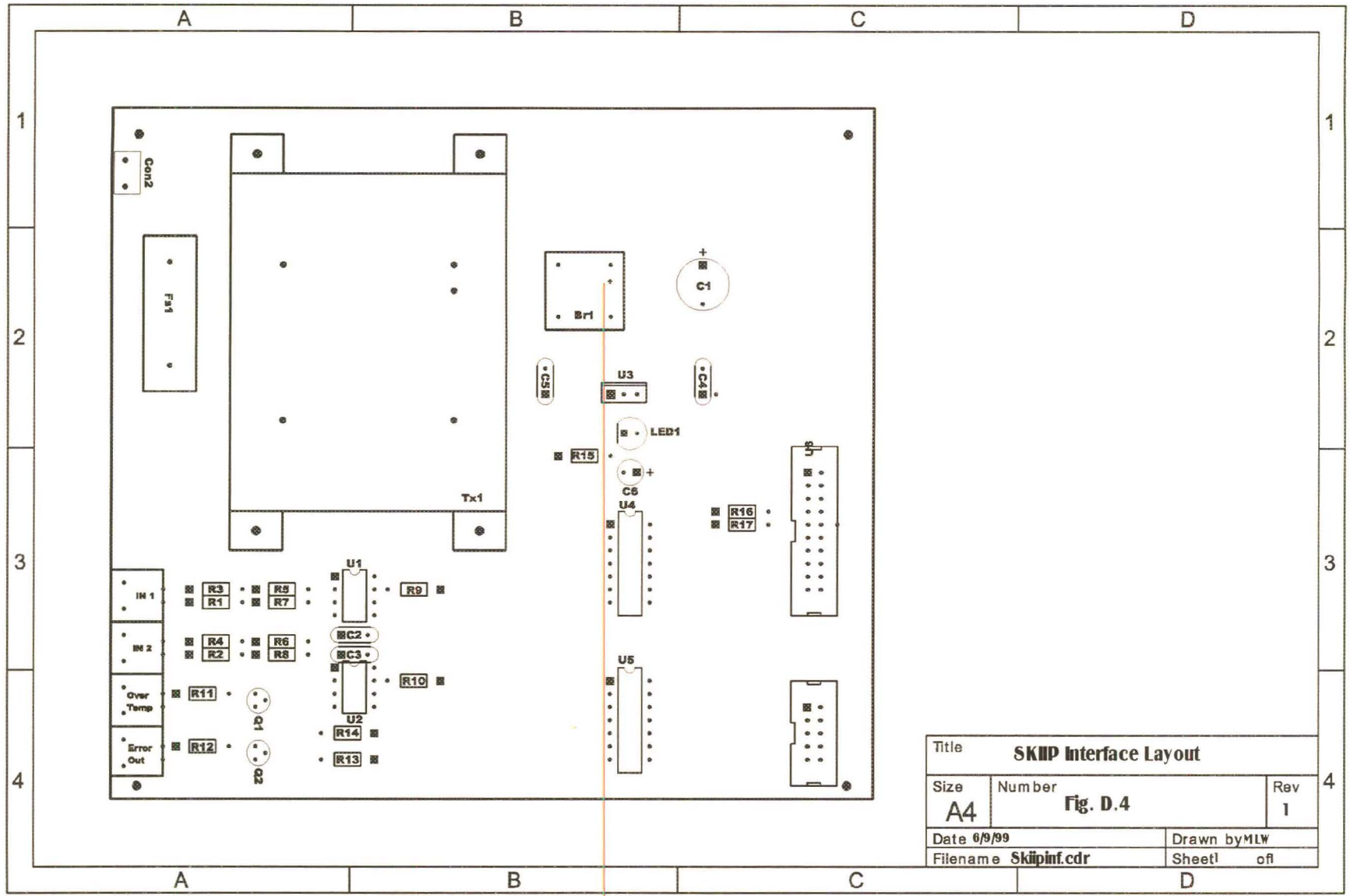


E.3 REFU Drive Control Signal Wiring Diagram

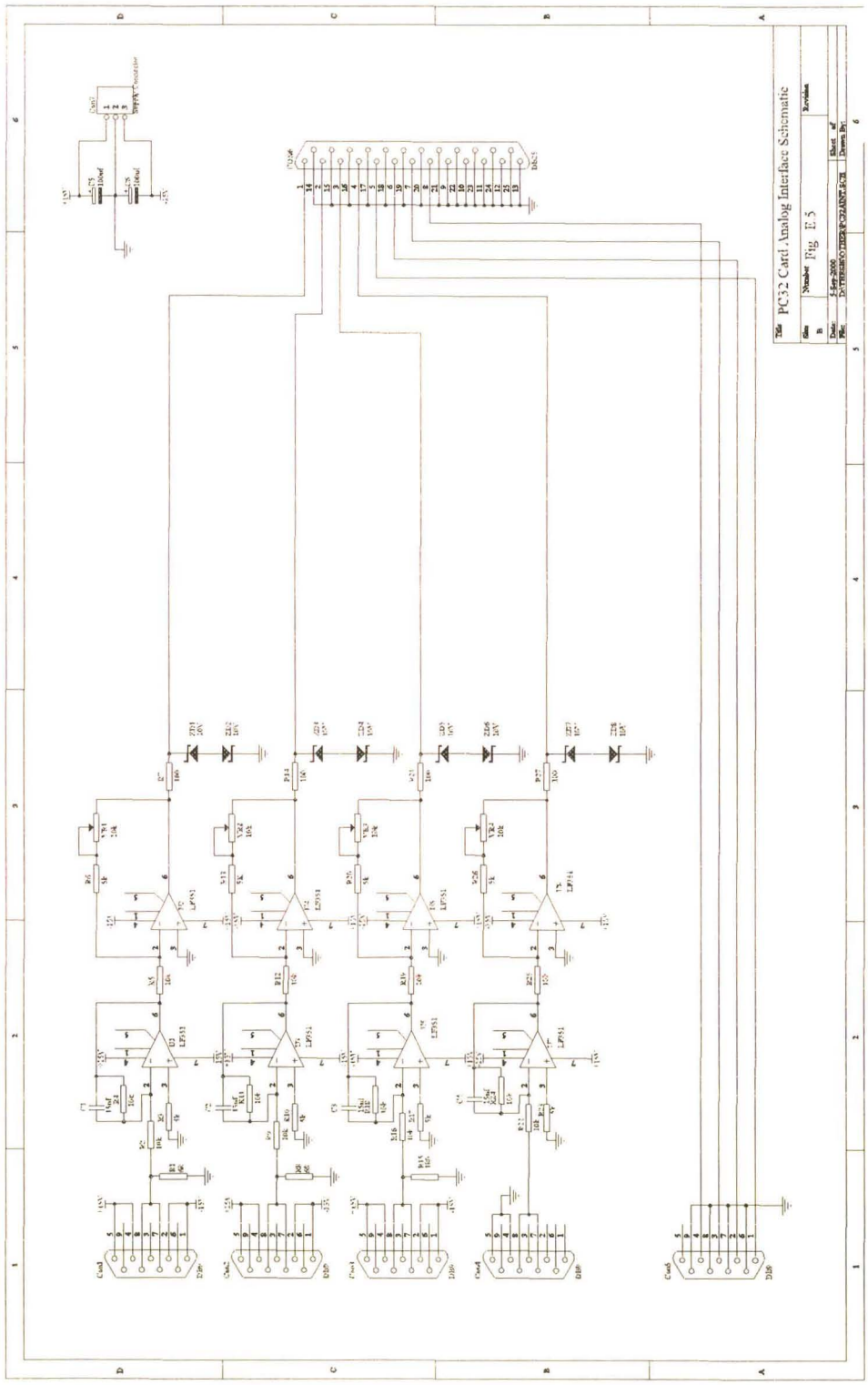


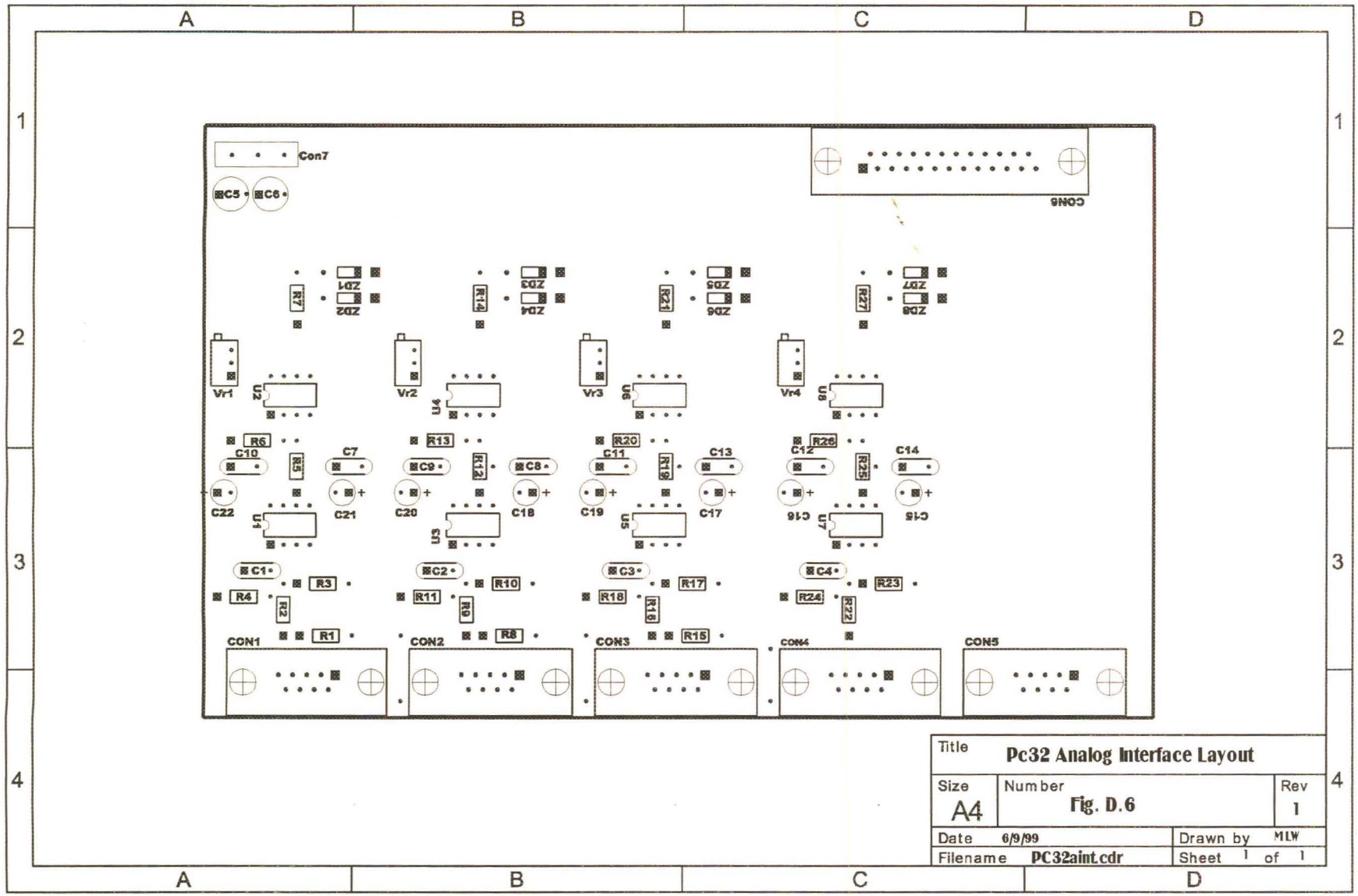
E.4 SKIIP Module Interface Circuit Schematic and PCB Layout



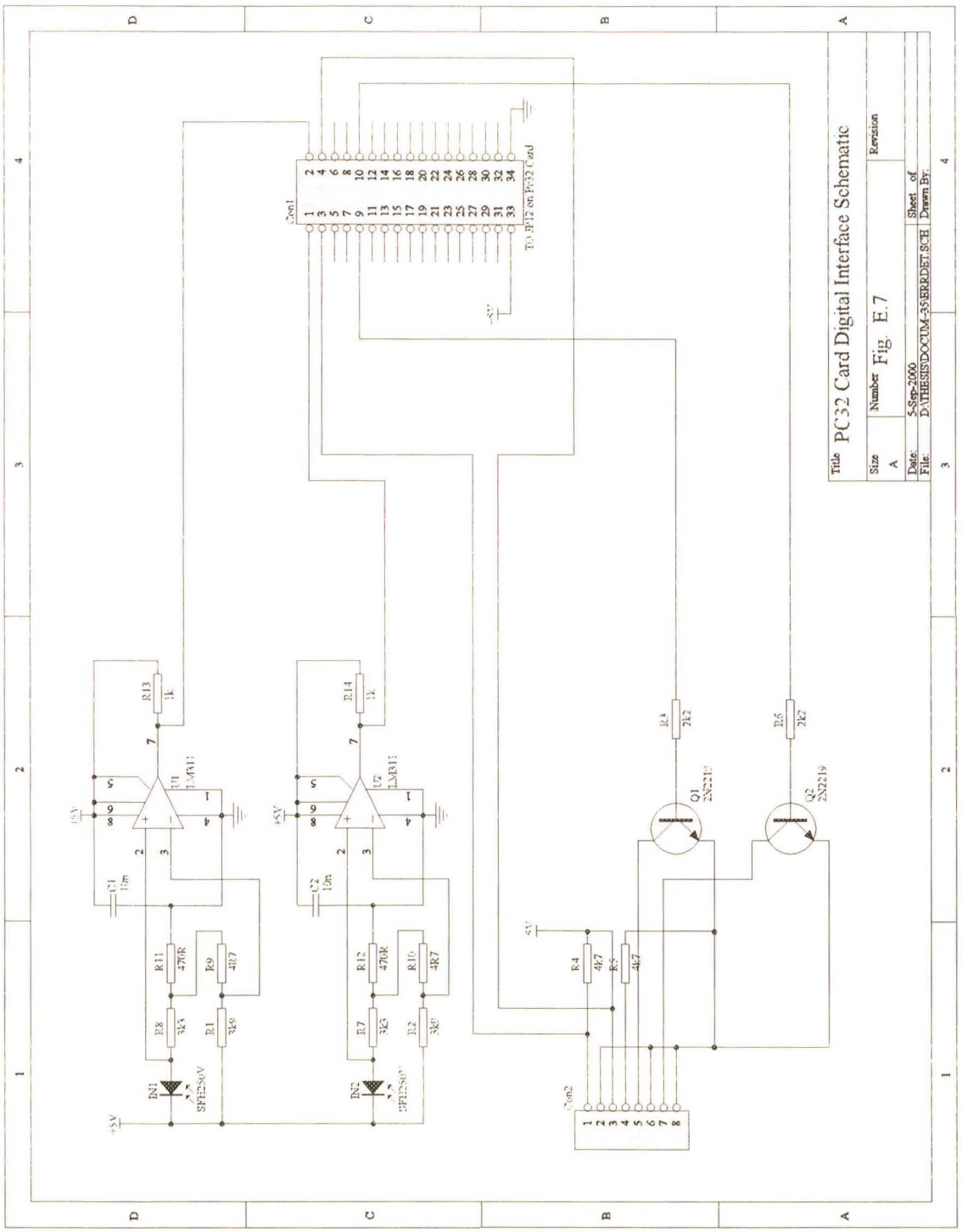

A
B
C
D
1
2
3
4
Con2
Fs1
Tx1
Br1
C1
C5
C4
U3
LED1
R15
C6
U4
R16
R17
Sh
U1
R3
R1
R5
R7
R9
IN 1
IN 2
Over Temp
Error Out
R4
R2
R6
R8
C2
C3
R11
R10
Q1
Q2
U2
R14
R13
R12
U5
Title
SKIIP Interface Layout
Size
A4
Number
Fig. D.4
Rev
1
Date 6/9/99
Drawn by MLW
Filename Skiipinf.cdr
Sheet1 of1

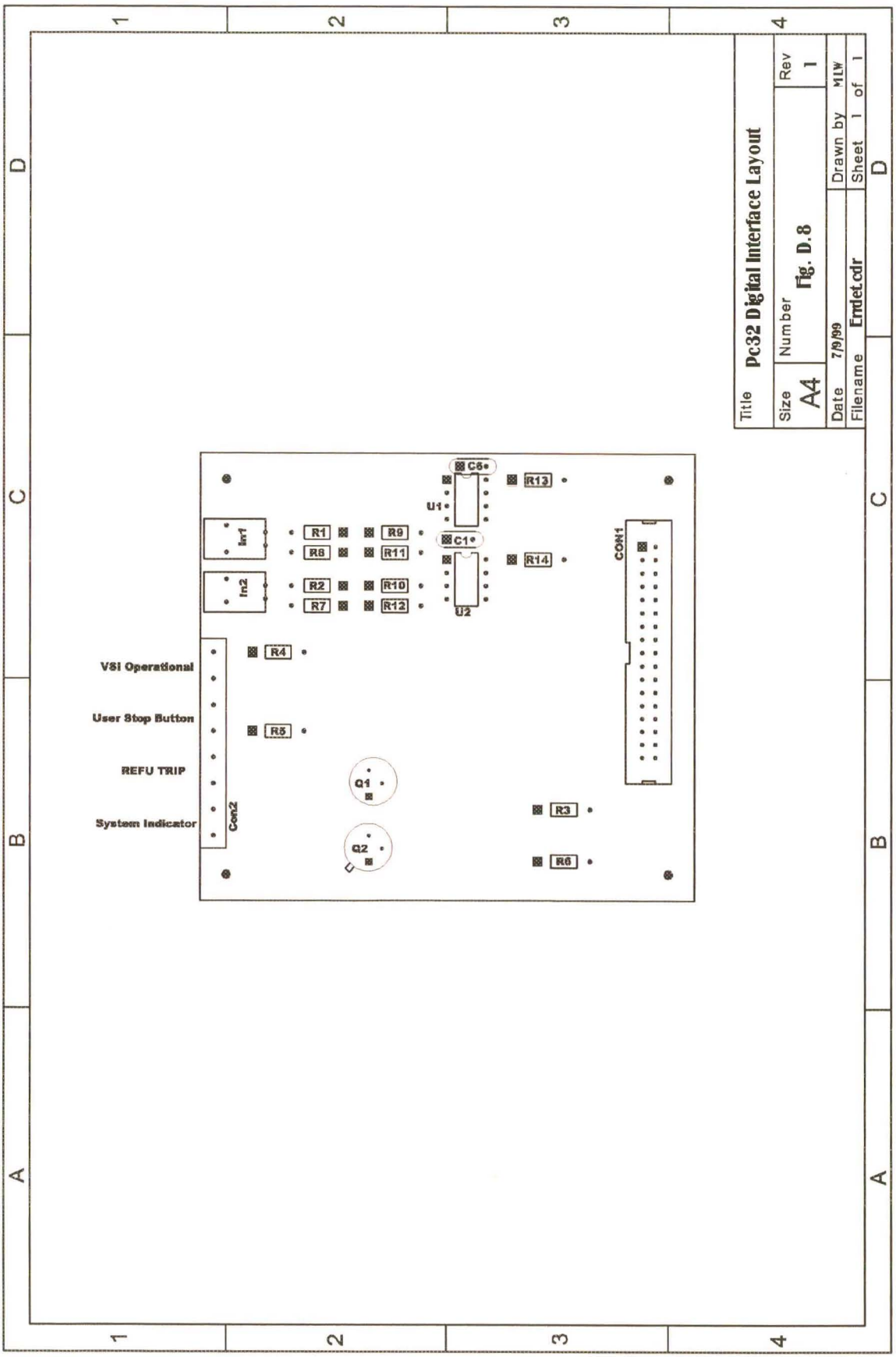
E.5 PC32 Card Analog Interface Circuit Schematic and PCB Layout




A
B
C
D
1
2
3
4
Con7
C5
C6
CON6
ZD1
ZD2
ZD3
ZD4
ZD5
ZD6
ZD7
ZD8
R7
R14
R21
R27
Vr1
Vr2
Vr3
Vr4
U2
U4
U6
U8
R6
R13
R20
R26
C10
C9
C11
C12
C7
C8
C13
C14
R5
R12
R19
R25
C22
C21
C20
C18
C19
C17
C16
C15
U1
U3
U5
U7
C1
C2
C3
C4
R4
R3
R11
R10
R18
R17
R24
R23
R2
R9
R16
R22
R1
R8
R15
CON1
CON2
CON3
CON4
CON5
Title
Pc32 Analog Interface Layout
Size
A4
Number
Fig. D.6
Rev
1
Date
6/9/99
Drawn by
MLW
Filename
PC32aint.cdr
Sheet
1
of
1

E.6 PC32 Card Digital Interface Circuit Schematic and PCB Layout




Title Pc32 Digital Interface Layout
Size A4
Number Fig. D.8
Rev 1
Date 7/9/99
Drawn by MLW
Filename Emdet.cdr
Sheet 1 of 1
C6
U1
C1
U2
R13
R14
R1
R8
R9
R11
R2
R7
R10
R12
In1
In2
CON1
R4
R5
Q1
Q2
R3
R6
VSI Operational
User Stop Button
REFU TRIP
System Indicator
Con2

Appendix F

PI Controller Discrete Transform

For a PI controller to be implemented on a micro-controller or a digital system a discrete transform of the PI controller is required. A PI controller in the time domain can be expressed using Eq. F.1 [29].

$$U(t) = K_p\left[e(t) + \frac{1}{T_i}\int e(t)dt\right] \tag{F.1}$$

where e(t) is the error

K_p is the Proportional Gain

T_i is the integral time constant.

Eq. F.1 may be transformed into the s-domain using the Laplace transform which results in Eq. F.2 [29].

$$U(s) = K_p\left[e(s) + \frac{e(s)}{T_i S}\right] \tag{F.2}$$

The transfer function G(s) for the PI controller in the S-domain can be found by dividing the output by the input and is given in Eq. F.3.

$$G(s) = \frac{U(s)}{e(s)} = K_p\left[1 + \frac{1}{T_i S}\right] \tag{F.3}$$

Eq. F.3 has to be transformed into the discrete domain using the discrete transform which relates the S-domain to the discrete or Z-domain, the transform is shown in Eq. F.4 [30].

$$S = \frac{2}{T_s}\frac{(z-1)}{(z+1)} \tag{F.4}$$

where T_s is the sampling time of the discrete PI controller.

Eq. F.2 can be transformed into the z-domain using the transformation shown in Eq. F.4 and results in Eq. F.5.

$$U(z) = K_p\left[e(z) + \frac{e(z)}{\frac{2}{T_s}\frac{(z-1)}{(z+1)}T_i}\right] \tag{F.5}$$

Simplifying the above equation results in Eq. F.6.

$$U(z)(z-1) = K_p\left[e(z)(z-1) + \frac{T_s}{2T_i}(z+1)e(z)\right] \tag{F.6}$$

By dividing Eq. F.6 by z results in Eq. F.7.

$$U(z)(1-z^{-1}) = K_p\left[e(z)(1-z^{-1}) + \frac{T_s}{2T_i}(1+z^{-1})\right] \tag{F.7}$$

Eq. F.7 can be expanded further as shown below.

$$U(z) = U(z)z^{-1} + K_p\left[(e(z) + e(z)z^{-1}) + \frac{T_s}{2T_i}(e(z) + e(z)z^{-1})\right] \quad \text{(F.8)}$$

From the definition of z in the discrete transform, z^{-1} is a unit sample delay, thus Eq. F.8 can be written in the discrete domain as shown in Eq. F.9.

$$U(k) = U(k-1) + K_p(e(k) - e(k-1)) + \frac{K_p T_s}{2T_i}(e(k) + e(k-1)) \quad \text{(F.9)}$$

where U(k) is the present output of the PI controller

U(k-1) was the previous output of the PI controller

e(k) is the present error between the actual process variable and the reference value

e(k-1) was the previous error between the actual process variable and the reference value.

The final form of the discrete PI controller as used in Chapter 3 and Chapter 6 is obtained by substituting Eq. F.10 into Eq. F.9 which results in Eq. F.11.

$$K_i = \frac{1}{T_i} \quad \text{(F.10)}$$

where K_i is known as the integral gain.

$$U(k) = U(k-1) + K_p(e(k) - e(k-1)) + \frac{K_p K_i T_s}{2}(e(k) + e(k-1)) \quad \text{(F.11)}$$

Integral windup occurs when the input error signal does not change quickly, the integral component of the PI controller increases above the maximum output value, if this is allowed to happen there is a large delay from when the error signal reaches zero until the PI controller begins to react again due to the integrator having a large value, this results in unstable operation of the PI controller when large disturbances in a system are present. Integral windup is prevented by disabling the integral term of the PI controller as soon as the output of the PI controller reaches it's limiting value, as soon as the error signal begins to drop the PI controller operation continues without any delay.

The above objective is achieved by limiting U(k) to the maximum output this results in U(k-1) being held at a limit, the following rules are applied to U(k) to ensure that integral windup does not occur.

$$\text{If } U(k) > U_{limit} \qquad (F.12)$$
$$U(k) = U_{limit}$$

As the PI controller output can vary in both direction the above rule has to be applied to the negative values of U(k) as shown in Eq. F.13.

$$\text{If } U(k) < -U_{limit} \qquad (F.13)$$
$$U(k) = -U_{limit}$$

Appendix G

User Define Function Block Description

This appendix describes the operation and functions of the user define blocks created, such that the test bed controller described in Chapter 5 could be implemented. The DSP source code for the user define blocks, is given in Appendix I.

G.1 A/D Trigger and A/D Read Blocks

Two separate blocks are required to gain data from the A/D converters. The first block is the *A/D Trigger* block, this block writes to the A/D converters memory location, which starts a conversion on the relevant A/D converter [18]. All the A/D converters are signaled to start a conversion when this block is run.

The *A/D Read* block reads the converted analog values from the A/D converters. This block has to be connected to an interrupt, which is connected to the end of conversion signal from the A/D converter. All the A/D converters are read when this block is executed. The C code listing for A/D Trigger and the A/D Read blocks are given in Appendix I.

G.2 D/A Trigger and D/A Load Blocks

Data has to be written to the D/A converters in two stages, the first stage is to latch the data onto the D/A converters holding latches. The second stage is to trigger the D/A to output the analog value that corresponds to the digital value in the holding latches [18].

The D/A converter functions have been split into two blocks, which enables the system to load and output data in synchronism with external events. The *D/A Load* block performs the first

function by writing the digital value, supplied to its inputs, to the corresponding D/A converter holding latches. The second block is the *D/A Trigger* block, this block writes to the D/A converter, indicating that the data in the holding latches must be output on the analog output port. The C code listing for D/A Trigger and the D/A Load blocks is given in Appendix I.

G.3 Average Conversion Block

The signal being fed from the A/D converter connected to the LEM sensor measuring the DC link current is discontinuous, due to the conduction of the diodes in the front end rectifier. The measured value has to be converted to its average value, this is achieved by using Eq. G.1 to find the average value of the waveform [9].

$$I_{avg} = \frac{1}{T}\int_{0}^{T} I(t)dt \qquad \text{(G.1)}$$

The real time integration is achieved by using the trapezoidal rule to perform the integration [14]. The period over which the signal is integrated is also of importance, if the period is too short, the output signal will not represent the true average value. The integration is performed over a period of 1000 samples, with a sample rate of 9.76kHz, the waveform is integrated over a period of 102.46mS, which ensures that a significant number of the current pulses are integrated to provide a reliable result. The C code listing for the Average block is given in Appendix I.

G.4 Power Calculation Block

The power usage and efficiency of the VSD is calculated in the *Power Calc* block. This block performs the calculations described in Chapter 2. The operation of the power calculation block was described in Chapter 6 based on the equations derived in Chapter 2. This section reiterates the information presented in Chapter 6 and defines the inputs and outputs of the block.

The block has four inputs : the DC motor current (I_a); the DC link supply current (I_s); the DC link Voltage (V_{link}) and the test bed system speed (ω_r)

The block has three outputs : the VSD power usage (P_{vsd}) ; the VSD efficiency and the power drawn by test bed system from three phase mains (P_s)

The VSD power usage (P_{vsd}), VSD efficiency and the three phase supply power (P_s) are calculated using Eqs G.2 to G.4 respectively.

$$P_{vsd} = I_r.V_{link} + E_a.I_a - (I_a^2.R_a + I_aV_b) \tag{G.2}$$

$$Efficiency(\%) = \frac{E_a.I_a + B\omega_r^{\,2}}{E_a.I_a - (I_a^2R_a + I_aV_b) + V_{link}.I_r}100 \tag{G.3}$$

$$P_s = I_r.V_{link} \tag{G.4}$$

where E_a is the armature back EMF and is calculated using Eq. 2.14.

I_a is the DC motor armature current.

R_a is the total resistance in the armature circuit

V_b is the volt drop across the brushes in the armature circuit

I_r is the average value of the supply current flowing from the diode rectifier of the VSD

V_{link} is the DC link voltage

ω_r is the test bed system rotational speed in (rad/sec).

The efficiency block parameters shown in Fig. G.1 are the DC motor constant (K_m), the DC motor armature resistance (R_a), which includes the resistance of the inductors in series with the armature, the coefficient of viscous friction (B) and the voltage dropped across the brushes (V_b).

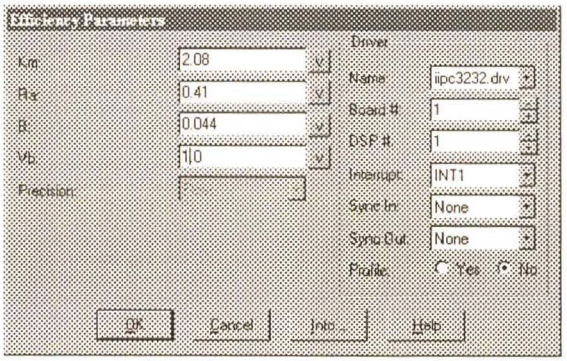


Fig. G.1 Efficiency Block Parameters

The C code listing for Efficiency block is given in Appendix I.

G.5 Load Simulator

The load simulation block enables the test bed system to simulate the load that the pumps and fans, described in Chapter 3, would place on the VSD while operating under different conditions. The block has two inputs, the first is the desired flow rate (0-100%) that has to be simulated, and the second input is the rotational speed of the motors in the test bed system, which is used to calculate the load torque. The block has two outputs, the first is the speed set point, which is output to the VSD such that the VSD runs at a speed which would produce the specified flow rate when speed regulation is used to control the flow rate. The second output is the load torque that is required to simulate the pump or fan operating under either throttling or speed regulation in order to control the flow rate. The parameters for the load simulation block are shown in Fig. G.2, where the load type selection box allows the test bed user to select which system is to be simulated. The equations used to describe the operation of a pump or fan which are used in the implementation of the load simulator block are described in Chapter 6.

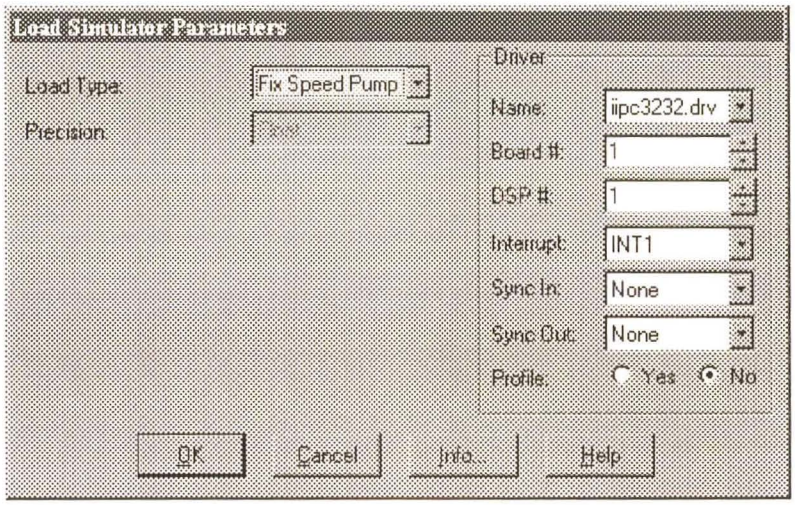


Fig. G.2 Parameters for Load Simulator Block

The C code listing for the load simulation block is given in Appendix I.

G.6 Error Detection Block

The error detection block checks the digital I/O port for any error signals generated by the components in the test bed system. The output of the block gives a number that corresponds to a specific error. The error number and the fault that is associated with it, is given in Table G.1.

Table G.1 Error Detection Block Output Values

Error Number	Fault
0	No error
1	SKIIP Module Group Fault
2	SKIIP Module Over Temperature
4	REFU Group Fault or not running
8	Stop Button pressed

The error detection block also sets two bits on the I/O port in accordance with the error that occurs. When an error from the SKIIP module is detected a bit is set which trips the REFU drive and turns on a red LED on the user interface. When the test bed system is running and no faults are present, a second bit is set which turns on a green LED on the user interface. The C code listing for Error detection block is given in Appendix I.

G.7 PWM Controller Block

The PWM control block communicates with the PWM ASIC on the PWM/tachometer card. The function block has four inputs, U_a Voltage, U_b Voltage, phase/frequency input and an external disable input. The PWM controller Block parameters are shown in Fig. G.3. The Global enable radio button allows the block to be enabled or disabled, if "No" is selected then the PWM ASIC is disabled and no switching pulses are output. The Internal Enable button allows the user to select whether the fourth input to the PWM controller block can disable the PWM ASIC. If the Internal Enable button is checked "No", whenever the fourth input to the block is not zero the PWM ASIC is disabled. The Input 3 type allows the third input into the PWM controller block to be configured as a phase or frequency input. The decodes combo box and the board number combo box are used such that the PWM controller block can be configured to point to the correct memory location when communicating with the PWM/tachometer card.

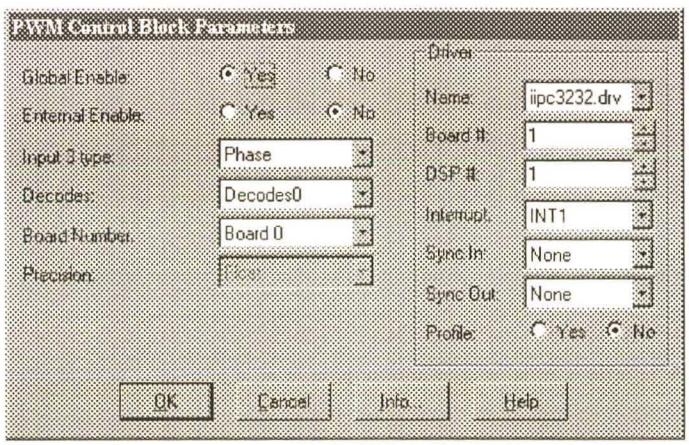


Fig. G.3 PWM Controller Block Parameters

The C code listing for PWM Controller block is given in Appendix I.

G.8 PI Controller Block

The PI controller block has one input and one output, the input is the error signal which is the difference between the desired or reference value and the actual value. The output of the block is the demand value that the PI controller requires to reduce the error to zero. The output of the PI controller block is calculated using Eq. G.5. The derivation of this formula can be found in Appendix F.

$$O_n = O_{n-1} + K_p(e_n - e_{n-1}) + \frac{K_p K_i}{2T_s}(e_n + e_{n-1}) \quad \text{(G.5)}$$

where O_n is the current output

O_{n-1} was the previous output

e_n is the current error (input value to the block)

e_{n-1} is the previous error

T_s is the sampling rate (number of times the block is run every second)

K_p is the proportional gain

K_i is the integral gain.

To ensure that integral windup does not occur, the following set of rules which limit the output as shown in Eqs G.6 and G.7 are implemented.

On>Omax (G.6)

On=Omax

On<-Omax (G.7)

On=-Omax

Where Omax is the maximum output value of the PI controller that results in the saturation of the

power controlling devices output (PWM ASIC's maximum input value).

The PI controller parameter block is shown in Fig. G.4

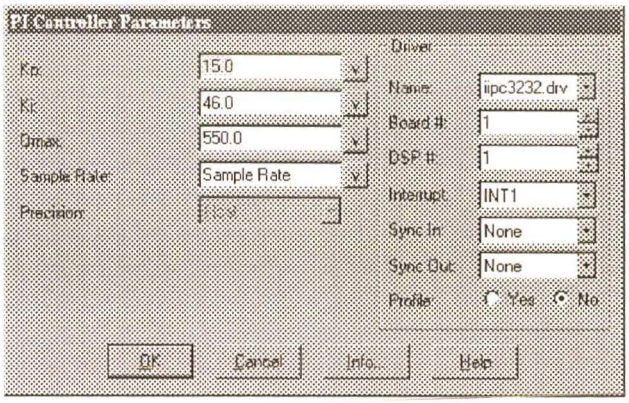


Fig. G.4 PI Controller Parameter Block

The PI controller block has the following parameters; Kp is the proportional gain and Ki is the integral gain which are set equal to the proportional and integral gain values described in Chapter 3. Omax is the maximum output limit and is used to limit the integral action such that integral windup does nor occur, and is set equal to the DC link voltage. The sampling rate is set equal to the sample rate at which the block is run (9760 samples per second).

G.7 Analogue Input Scaling Hierarchical Block

The analogue signals being measure by the A/D converters are derived from various transducers, measuring various system parameters. The A/D scale block is required to convert the digital value from the A/D converters to a value of the correct level for internal use. The input signals being fed to the Analogue Input Scaling block (from the A/D converters) are signed integer values (-32767 to +32767), these values need to be scaled to give a representative value of the parameter being measured. The value input into the block is firstly added to a constant, to null offset errors, whereafter the value is multiplied by a gain constant. The output values are in amps for the current measurements, volts for the voltage measurement and rpm for the speed measurement. The structure of the *A/D scale* block is shown in Fig. G.5.

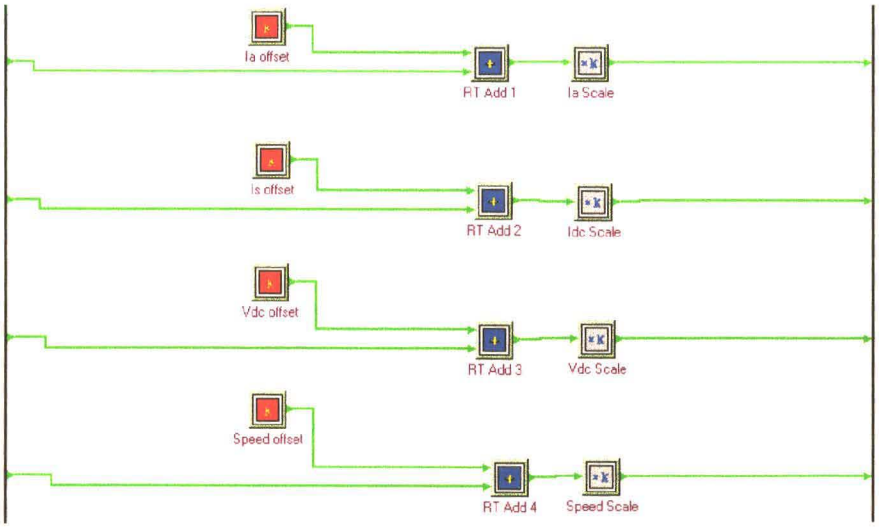


Fig. G.5 A/D Scale Hierarchical Block Diagram

The setting of the scaling parameters was done by measuring the current fed back signals with a current probe and oscilloscope, the voltage signals with a voltmeter and the speed signals with a digital tachometer. The values measured by the test equipment were compared to the values output by the Analogue input scaling block to determine if the scaling values were correct. The following scaling values are used to convert the digital input value from the A/D converter to a value which represents the sign and magnitude of the measured value

DC motor Armature Current Scaling factor : -0.004180
DC link Input Current Scaling Factor : -0.004086
DC link Voltage Scaling Factor : -0.02076
Test bed system Speed Scaling Factor : -0.06097

Due to analogue inputs to the differential inputs of the PC32 card being swapped the sign of the scaling factors are negative.

Appendix H

REFU Drive Controller Set Point Calculations

This appendix describes the methods used to calculate the set point values required to setup the REFU drive for the induction motor used in the test bed system.

H.1 REFU Drive Controller

The REFU drive is a 40kVA, FOC drive, which produces sinusoidal current waveforms and achieves speed feed back by means of a resolver [33]. The REFU drive controller is an analog controller therefore all the set point parameters are adjusted with the use of preset potentiometers. A simplified block diagram of the REFU drive controller is shown in Fig. H.1.

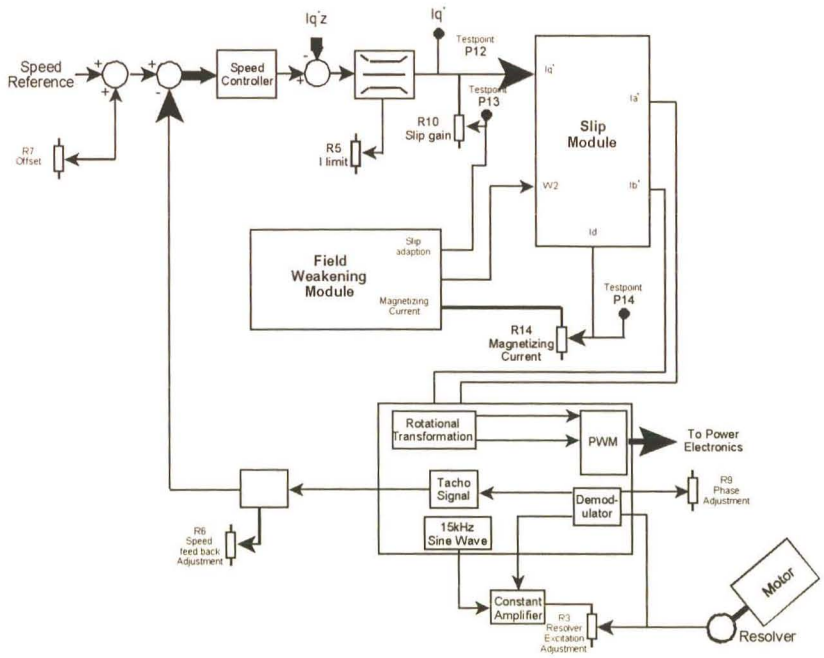


Fig. H.1 Simplified block diagram of REFU drive controller

The parameters that have to be set on the REFU drive controller card (as shown in Fig. H.1) to enable the card to control the IM are the: magnetizing current; torque current limit; and the slip frequency. The internal signals used inside the REFU drive are all scaled due to the transducers used in the drive. The scaling factor has to be known prior to the commissioning of the drive in order to calculate the set point values correctly. The method used to calculate the set point values based on the motor parameters described in Chapter 6 are described below.

H.2 REFU Drive Controller Current Scaling Factor

The REFU drive controller card accepts ±10 volt analog signals, thus all measured signals are scaled to conform to these signal levels. The motor currents are measured using LEM sensors, which provide an output current proportional to the current being measured. The output current of the LEM sensor is terminated in a resistor, which converts the output current to a voltage signal level, required by the REFU drive controller.

The REFU drive is rated to deliver a peak current (I_{max}) of 128A, a scaling factor (S_f) for the current can be calculated using Eq. H.1.

$$10V = S_f \cdot I_{\max} \tag{H.1}$$

The Scaling factor used is set by the drive manufacturer and can be calculated by substituting the maximum current in Eq. H.1, to give the value of the scaling factor, as shown in Eq. H.2

$$S_f = \frac{10V}{128A} = 0.078 \tag{H.2}$$

This current scaling factor is used when calculating the remaining parameter set points for the FOC drive.

H.3 REFU Drive Magnetizing Current Setup

The magnetizing current for an induction motor, running under FOC control, is produced by the d-axis stator current. The performance of the motor is heavily reliant on this parameter, if the motor is under fluxed (too small a magnetizing current), it will not produce rated torque, on the other hand, if the machine is over fluxed, the magnetic circuit saturates leading to high losses and lower efficiency operation. The magnetizing current of the motor is found by running the motor at no load. When running at no load the slip is almost zero, resulting in the rotor flux and the rotor d-axis flux lining up, thus the motor operates under the same conditions as under FOC. The currents flowing into the machine are only setting up the flux (i_{qs}=0), thus $|i_{ds}| = |i_a| = |i_b| = |i_c|$ for a balanced three phase machine. The no load current of the induction motor was measured and found to be 25A, thus i_{ds} = 25A.

The set point (U_{pf}) required to produce the flux is determined using Eq. H.3

$$U_{pf} = i_{ds} \cdot S_f \tag{H.3}$$

The value of U_{pf} = 1.95V for the induction motor used in the test bed system.

This value (U_{pf}) is set using potentiometer R14 and the set point value is measured on test point P14, as shown in Fig. H.1.

H.4 REFU Drive Torque Limit Setup

The torque limit, limits the maximum i_{qs} current (thus the torque) that the REFU drive will allow to flow through the induction motor. The maximum name i_{qs} current that should be allowed to flow through the induction motor is 52.3A as calculated in Chapter 6.

The set point value (U_{pb}) is then calculated using Eq. H.4

$$U_{pb} = i_{qs} \cdot S_f \tag{H.4}$$

The value of U_{pb} = 4,08V for the induction motor used in the test bed system.

This Value (U_{pb}) is set using potentiometer R5 and the set point value is measured at test point P12, as shown in Fig H.1.

H.5 REFU Drive Slip Frequency Setup

When an induction motor is being controlled by an FOC controller the controller attempts to maintain specific conditions inside the motor. If the conditions of FOC are not met then the induction motor will not operate correctly with a resultant reduction in performance. One of the conditions for FOC is given in Eq. H.5, which was given in Appendix A, Eq A.15.

$$s\omega = L_m R_2 i^*_{qs} / L_{22} \lambda_{dr} \tag{H.5}$$

where:

s is the induction motor slip

ω is the IM rotation speed

L_m is the stator and rotor mutual incuctance

R_2 is the rotor resistance

i^*_{qs} is the q-axis current reference value

L_{22} is the rotor self inductance

λ_{dr} is the q-axis Stator flux linkage.

The term $s\omega$ is known as the slip frequency, if the q-axis current reference (i_{qs}) is known and the slip frequency is known then the induction motor parameters can be calculated from H.5. The REFU drive controller requires the values of the IM parameters such that it can calculate the desired d and q axis currents in order to control the IM. The slip frequency ($f_{2n} = s\omega$) of the motor can be calculated from Eq. H.6

$$f_{2n} = f_{nm} - n\frac{n_{nm}}{60} \tag{H.6}$$

Where f_{nm} is the mains frequency that the motor is designed for, n is the number of pole pairs and n_{nm} is the full load speed of the motor. With n_{nm} = 1465rpm, f_{nm} = 50Hz and n = 2, the slip frequency f_{2n} = 1.17Hz.

The set point value (U_{ps}) can now be calculated using Eq. H.7, where a set point value of 10V relates to a slip frequency of 10.8Hz.

$$U_{ps} = \frac{10V}{10.8Hz} f_{2n} \tag{H.7}$$

The value of U_{ps} = 1.07V for the induction motor used in the test bed system.

This set point (U_{ps}) is measured at test point P13 and set by potentiometer R10, as shown in Fig. H.1.

Appendix I

User Defined Blocks DSP Source Code

This Appendix presents the DSP C source code used to generate the object files (extension .obj) for the user defined blocks which are used for the implementation of the test bed controller. For each block there are two file which are used to generate the object file, the first is the header file (extension .h) which contains the input and output data structure and any persistent variables used to implement the DSP block and the function definitions for the DSP block. The second file is the DSP source code file (extension .c) this contains the functional code for the DSP block

I.1 A/D Read and A/D Trigger Block Source Code

A/D Read

```
/*=============================================================================//
//      FILE NAME :    _Adread.h                                                //
//      BLOCK NAME: A/D read                                                    //
//      GROUP NAME: Motion Control                                              //
//      PURPOSE :              Provides the real-time block's DSP header file.  //
//      Hypersignal Block Wizard Version 4.00.14 Auto-Generated Block           //
//      Number of Inputs :     0                                                //
//      Number of Outputs:     4                                                //
//      Creation Date:   Wed - 13 January 1999                                  //
//      Creation Time:   07:55 AM                                               //
//=============================================================================*/

/*****************************************************************************/
/*                                DEFINITIONS                                */
/*****************************************************************************/
/* boolean macros */
typedef  unsigned int  BOOL;
```

```
#define FALSE  0
#define TRUE   1

/*****************************************************************************/
/*                          Data Pointers / Parameter Structure              */
/* NOTE:      This structure was auto-generated by Hypersignal               */
/*            Block Wizard application.  DO NOT MODIFY THIS STRUCTURE!       */
/*****************************************************************************/

typedef struct
{
/*****************************************************************************/
/*                          Synchronization Addresses                        */
/*****************************************************************************/
  unsigned int *SyncIn;              /* ptr to input sync value  */
  unsigned int *SyncOut;             /* ptr to output sync value */

/*****************************************************************************/
/*                          Output Data Pointers / Framesize                 */
/*****************************************************************************/
  float    *OutPtr0;                       /* output 0 data pointer */
  float    *OutPtr1;                       /* output 1 data pointer */
  float    *OutPtr2;                       /* output 2 data pointer */
  float    *OutPtr3;                       /* output 3 data pointer */
  unsigned int FramesizeOut0;        /* output 0 framesize */
  unsigned int FramesizeOut1;        /* output 1 framesize */
  unsigned int FramesizeOut2;        /* output 2 framesize */
  unsigned int FramesizeOut3;        /* output 3 framesize */

/*****************************************************************************/
/*                          Internal Persistent Variables                    */
/*****************************************************************************/
/* NOTE:      If your block requires persistent variables, then they must    */
/*            be put here.  DO NOT USE global variables or statics in the    */
/*            C source file unless you really want them to be shared by      */
/*            all instances of this block.  If you do add variables,         */
```

```
/*              update the "#define INTVAR_SIZE" in the PC's header (.h)                */
/*              file to reflect the proper internal variable storage.                    */
/*****************************************************************************************/

} PARAMS;

/*****************************************************************************************/
/*                                  FUNCTION PROTOTYPES                                  */
/*****************************************************************************************/

void ADread_INT  (PARAMS *pParam);          /* optional block interrupt routine */

void ADread_INIT (PARAMS *pParam);          /* optional block initialization routine */

void ADread_STOP (PARAMS *pParam);          /* optional block stop routine */

void ADread_RESTART (PARAMS *pParam);       /* optional block restart routine */

void ADread(PARAMS *pParam);                /* main block routine */


/*=======================================================================================//
//      FILE NAME :     _Adread.c                                                        //
//      BLOCK NAME: A/D read                                                             //
//      GROUP NAME: Motion Control                                                       //
//      PURPOSE:            Provides the real-time block's DSP C source code.            //
//      Hypersignal Block Wizard Version 4.00.14 Auto-Generated Block                    //
//      Number of Inputs :      0                                                        //
//      Number of Outputs:      4                                                        //
//      Creation Date:  Wed - 13 January 1999                                            //
//      Creation Time:  07:55 AM                                                         //
//=======================================================================================*/

#include "_adread.h"

/*---------------------------------------------------------------------------------------*/
```

```
/*                                  Real-time block routine                                  */
/*-------------------------------------------------------------------------------------------*/
/* This is the main block routine.  It is called during each loop of the main                */
/* application.  If an interrupt is selected, and the interrupt routine above                */
/* is not activated, this routine will be called in response to the selected                 */
/* interrupt instead of during the main application loop.                                    */
/*-------------------------------------------------------------------------------------------*/

void ADread(PARAMS *pParam)
{

unsigned int index;                     /* index for frame processing loop */
int AD0,AD1,AD2,AD3;                    /* AD holding Variables */

#define ADC0 (volatile int*) 0x810000   /* address for ADC0 channel */
#define ADC1 (volatile int*) 0x810800   /* address for ADC1 channel */
#define ADC2 (volatile int*) 0x811000   /* address for ADC2 channel */
#define ADC3 (volatile int*) 0x811800   /* address for ADC3 channel */

AD0 = (0xFFFF&(*(ADC0)));               /* read ADC0 */
AD1 = (0xFFFF&(*(ADC1)));               /* read ADC1 */
AD2 = (0xFFFF&(*(ADC2)));               /* read ADC2 */
AD3 = (0xFFFF&(*(ADC3)));               /* read ADC3 */

if (AD0<32767)                          /* sort out two's Compliment with ADC0 Result*/
 {  AD0 = -AD0;  }
else
 {  AD0 = 65536 - AD0;  }

if (AD1<32767)                          /* sort out two's Compliment with ADC1 Result*/
 {  AD1 = -AD1;  }
else
 {  AD1 = 65536 - AD1;  }

if (AD2<32767)                          /* sort out two's Compliment with ADC2 Result*/
 {  AD2 = -AD2;  }
```

```
else
 {  AD2 = 65536 - AD2;  }

if (AD3<32767)                          /* sort out two's Compliment with ADC3 Result*/
 {  AD3 = -AD3;  }
else
 {  AD3 = 65536 - AD3;  }

  for (index = 0; index<pParam->FramesizeOut0; index++)
   {
    *(pParam->OutPtr0+index) = ((float)AD0);        /* save data */
   }

  for (index = 0; index<pParam->FramesizeOut1; index++)
   {
    *(pParam->OutPtr1+index) = ((float)AD1);        /* save data */
   }

  for (index = 0; index<pParam->FramesizeOut2; index++)
   {
    *(pParam->OutPtr2+index) = ((float)AD2);        /* save data */
   }

  for (index = 0; index<pParam->FramesizeOut3; index++)
   {
    *(pParam->OutPtr3+index) = ((float)AD3);        /* save data */
   }

}
```

A/D Trigger

```
/*=============================================================================*//
//      FILE NAME :    _Adtrig.h                                                //
//      BLOCK NAME: A/D Trigger                                                 //
//      GROUP NAME: Motion Control                                              //
//      PURPOSE:            Provides the real-time block's DSP header file.     //
//      Hypersignal Block Wizard Version 4.00.14 Auto-Generated Block           //
//      Number of Inputs :      0                                               //
//      Number of Outputs:      0                                               //
//      Creation Date:    Wed - 13 January 1999                                 //
//      Creation Time:    07:49 AM                                              //
//=============================================================================*/

/*****************************************************************************/
/*                                DEFINITIONS                                */
/*****************************************************************************/
/* boolean macros */
typedef  unsigned int  BOOL;
#define  FALSE   0
#define  TRUE    1

/*****************************************************************************/
/*                          Data Pointers / Parameter Structure              */
/* NOTE:       This structure was auto-generated by Hypersignal              */
/*             Block Wizard application.  DO NOT MODIFY THIS STRUCTURE!      */
/*****************************************************************************/

typedef struct
{
/*****************************************************************************/
/*                          Synchronization Addresses                        */
/*****************************************************************************/
  unsigned int *SyncIn;             /* ptr to input sync value  */
  unsigned int *SyncOut;            /* ptr to output sync value */
```

```
/*****************************************************************************/
/*                          Internal Persistent Variables                    */
/*****************************************************************************/
/* NOTE:    If your block requires persistent variables. then they must      */
/*          be put here.  DO NOT USE global variables or statics in the      */
/*          C source file unless you really want them to be shared by        */
/*          all instances of this block.  If you do add variables.           */
/*          update the "#define INTVAR_SIZE" in the PC's header (.h)         */
/*          file to reflect the proper internal variable storage.            */
/*****************************************************************************/

} PARAMS;

/*****************************************************************************/
/*                          FUNCTION PROTOTYPES                              */
/*****************************************************************************/

void ADtrig_INT  (PARAMS *pParam);          /* optional block interrupt routine */

void ADtrig_INIT (PARAMS *pParam);          /* optional block initialization routine */

void ADtrig_STOP (PARAMS *pParam);          /* optional block stop routine */

void ADtrig_RESTART (PARAMS *pParam);       /* optional block restart routine */

void ADtrig(PARAMS *pParam);            /* main block routine */

/*===========================================================================*/
//      FILE NAME :    _Adtrig.c                                             //
//      BLOCK NAME: A/D Trigger                                              //
//      GROUP NAME: Motion Control                                           //
//      PURPOSE:            Provides the real-time block's DSP C source code.//
//      Hypersignal Block Wizard Version 4.00.14 Auto-Generated Block        //
//      Number of Inputs :      0                                            //
//      Number of Outputs:      0                                            //
//      Creation Date:    Wed - 13 January 1999                              //
```

```
//        Creation Time:   07:49 AM                                                    //
//=============================================================================*/

#include "_ADtrig.h"

/*-----------------------------------------------------------------------------*/
/*                                Real-time block routine                      */
/*-----------------------------------------------------------------------------*/
/* This is the main block routine.  It is called during each loop of the main  */
/* application.  If an interrupt is selected, and the interrupt routine above  */
/* is not activated, this routine will be called in response to the selected    */
/* interrupt instead of during the main application loop.                       */
/*-----------------------------------------------------------------------------*/
void ADtrig(PARAMS *pParam)
{
#define ADC0 (volatile int*) 0x810000        /* define ADC0 memory address */
#define ADC1 (volatile int*) 0x810800        /* define ADC1 memory address */
#define ADC2 (volatile int*) 0x811000        /* define ADC2 memory address */
#define ADC3 (volatile int*) 0x811800        /* define ADC3 memory address */

*(ADC0)=0;                                   /* Start ADC0 conversion */
*(ADC1)=0;                                   /* Start ADC1 conversion */
*(ADC2)=0;                                   /* Start ADC2 conversion */
*(ADC3)=0;                                   /* Start ADC3 conversion */
}
```

I.2 D/A Trigger and D/A Load Source Code

D/A Trigger

```
/*=============================================================================//
//        FILE NAME:          _Datrig.h                                        //
//        BLOCK NAME: D/A triggers                                             //
//        GROUP NAME: Motion Control                                           //
//        PURPOSE:            Provides the real-time block's DSP header file.  //
```

```
//      Hypersignal Block Wizard Version 4.00.14 Auto-Generated Block                    //
//      Number of Inputs :      0                                                        //
//      Number of Outputs:      0                                                        //
//      Creation Date:   Wed - 13 January 1999                                           //
//      Creation Time:   07:51 AM                                                        //
//===================================================================================*/

/*************************************************************************************/
/*                                    DEFINITIONS                                    */
/*************************************************************************************/
/* boolean macros */
typedef unsigned int  BOOL;
#define  FALSE   0
#define  TRUE    1

/*************************************************************************************/
/*                          Data Pointers / Parameter Structure                      */
/* NOTE:       This structure was auto-generated by Hypersignal                      */
/*             Block Wizard application.  DO NOT MODIFY THIS STRUCTURE!              */
/*************************************************************************************/

typedef struct
{
/*************************************************************************************/
/*                          Synchronization Addresses                                */
/*************************************************************************************/
 unsigned int *SyncIn;              /* ptr to input sync value  */
 unsigned int *SyncOut;             /* ptr to output sync value */

/*************************************************************************************/
/*                          Internal Persistent Variables                            */
/*************************************************************************************/
/* NOTE:       If your block requires persistent variables, then they must           */
/*             be put here.  DO NOT USE global variables or statics in the           */
/*             C source file unless you really want them to be shared by             */
/*             all instances of this block.  If you do add variables,                */
```

```
/*              update the "#define INTVAR_SIZE" in the PC's header (.h)                */
/*              file to reflect the proper internal variable storage.                    */
/*****************************************************************************************/

} PARAMS;

/*****************************************************************************************/
/*                                  FUNCTION PROTOTYPES                                  */
/*****************************************************************************************/

void DAtrig_INT  (PARAMS *pParam);          /* optional block interrupt routine */

void DAtrig_INIT (PARAMS *pParam);          /* optional block initialization routine */

void DAtrig_STOP (PARAMS *pParam);          /* optional block stop routine */

void DAtrig_RESTART (PARAMS *pParam);       /* optional block restart routine */

void DAtrig(PARAMS *pParam);                /* main block routine */


/*=======================================================================================//
//      FILE NAME:              _Datrig.c                                                //
//      BLOCK NAME: D/A triggers                                                         //
//      GROUP NAME: Motion Control                                                       //
//      PURPOSE:                Provides the real-time block's DSP C source code.        //
//      Hypersignal Block Wizard Version 4.00.14 Auto-Generated Block                    //
//      Number of Inputs :      0                                                        //
//      Number of Outputs:      0                                                        //
//      Creation Date:  Wed - 13 January 1999                                            //
//      Creation Time:  07:51 AM                                                         //
//=======================================================================================*/

#include "_DAtrig.h"

/*---------------------------------------------------------------------------------------*/
```

```
/*                                  Real-time block routine                                  */
/*-------------------------------------------------------------------------------------------*/
/* This is the main block routine.  It is called during each loop of the main                */
/* application.  If an interrupt is selected, and the interrupt routine above                */
/* is not activated, this routine will be called in response to the selected                 */
/* interrupt instead of during the main application loop.                                    */
/*-------------------------------------------------------------------------------------------*/

void DAtrig(PARAMS *pParam)
{

#define DAC0 (volatile int*) 0x816000          /* address for DAC0 update */
#define DAC1 (volatile int*) 0x816800          /* address for DAC1 update */
#define DAC2 (volatile int*) 0x817000          /* address for DAC2 update */
#define DAC3 (volatile int*) 0x817800          /* address for DAC3 update */

*(DAC0)=0;                                     /* update DAC0 output */
*(DAC1)=0;                                     /* update DAC1 output */
*(DAC2)=0;                                     /* update DAC2 output */
*(DAC3)=0;                                     /* update DAC3 output */
}
```

D/A Load

```
/*=============================================================================*//
//      FILE NAME:          _Daload.h                                            //
//      BLOCK NAME: D/A load                                                     //
//      GROUP NAME: Motion Control                                               //
//      PURPOSE:            Provides the real-time block's DSP header file.      //
//      Hypersignal Block Wizard Version 4.00.14 Auto-Generated Block            //
//      Number of Inputs :  4                                                    //
//      Number of Outputs:  0                                                    //
//      Creation Date:   Wed - 13 January 1999                                   //
//      Creation Time:   07:57 AM                                                //
//=============================================================================*/
```

```
/*****************************************************************************/
/*                              DEFINITIONS                                  */
/*****************************************************************************/
/* boolean macros */
typedef unsigned int  BOOL;
#define FALSE   0
#define TRUE    1

/*****************************************************************************/
/*                          Data Pointers / Parameter Structure              */
/* NOTE:        This structure was auto-generated by Hypersignal            */
/*              Block Wizard application.  DO NOT MODIFY THIS STRUCTURE!    */
/*****************************************************************************/

typedef struct
{
/*****************************************************************************/
/*                          Synchronization Addresses                        */
/*****************************************************************************/
  unsigned int *SyncIn;                 /* ptr to input sync value */
  unsigned int *SyncOut;                /* ptr to output sync value */

/*****************************************************************************/
/*                          Input Data Pointers / Framesizes                 */
/*****************************************************************************/
  float     *InPtr0;                    /* input 0 data pointer */
  float     *InPtr1;                    /* input 1 data pointer */
  float     *InPtr2;                    /* input 2 data pointer */
  float     *InPtr3;                    /* input 3 data pointer */
  unsigned int FramesizeIn0;            /* input 0 framesize */
  unsigned int FramesizeIn1;            /* input 1 framesize */
  unsigned int FramesizeIn2;            /* input 2 framesize */
  unsigned int FramesizeIn3;            /* input 3 framesize */

/*****************************************************************************/
/*                          Internal Persistent Variables                    */
```

```
/*****************************************************************************/
/* NOTE:        If your block requires persistent variables, then they must  */
/*              be put here.  DO NOT USE global variables or statics in the  */
/*              C source file unless you really want them to be shared by    */
/*              all instances of this block.  If you do add variables,       */
/*              update the "#define INTVAR_SIZE" in the PC's header (.h)      */
/*              file to reflect the proper internal variable storage.        */
/*****************************************************************************/

} PARAMS;

/*****************************************************************************/
/*                          FUNCTION PROTOTYPES                              */
/*****************************************************************************/

void DAload_INT  (PARAMS *pParam);      /* optional block interrupt routine */

void DAload_INIT (PARAMS *pParam);      /* optional block initialization routine */

void DAload_STOP (PARAMS *pParam);      /* optional block stop routine */

void DAload_RESTART (PARAMS *pParam);   /* optional block restart routine */

void DAload(PARAMS *pParam);            /* main block routine */



/*===========================================================================*/
//      FILE NAME:          _Daload.c                                        //
//      BLOCK NAME: D/A load                                                 //
//      GROUP NAME: Motion Control                                           //
//      PURPOSE:            Provides the real-time block's DSP C source code.//
//      Hypersignal Block Wizard Version 4.00.14 Auto-Generated Block        //
//      Number of Inputs :  4                                                //
//      Number of Outputs:  0                                                //
//      Creation Date:  Wed - 13 January 1999                                //
```

```
//          Creation Time:   07:57 AM                                                    //
//=============================================================================*/

#include "_DAload.h"

/*-----------------------------------------------------------------------------*/
/*                                Real-time block routine                      */
/*-----------------------------------------------------------------------------*/
/* This is the main block routine.  It is called during each loop of the main  */
/* application.  If an interrupt is selected, and the interrupt routine above  */
/* is not activated, this routine will be called in response to the selected    */
/* interrupt instead of during the main application loop.                       */
/*-----------------------------------------------------------------------------*/
void DAload(PARAMS *pParam)
{

#define DAC0 (volatile int*) 0x814000          /* address for DAC0 channel */
#define DAC1 (volatile int*) 0x814800          /* address for DAC1 channel */
#define DAC2 (volatile int*) 0x815000          /* address for DAC2 channel */
#define DAC3 (volatile int*) 0x815800          /* address for DAC3 channel */

float DA0,DA1,DA2,DA3;                         /* Values to be output to latches */

  DA0 = *(pParam->InPtr0);                     /* accessing input data 0 */
  DA1 = *(pParam->InPtr1);                     /* accessing input data 1 */
  DA2 = *(pParam->InPtr2);                     /* accessing input data 2 */
  DA3 = *(pParam->InPtr3);                     /* accessing input data 3 */

  *(DAC0) = ((int)*(pParam->InPtr0));          /* latch DAC0 Value */
  *(DAC1) = ((int)*(pParam->InPtr1));          /* latch DAC1 Value */
  *(DAC2) = ((int)*(pParam->InPtr2));          /* latch DAC2 Value */
  *(DAC3) = ((int)*(pParam->InPtr3));          /* latch DAC3 Value */
}
```

I.3 Average Conversion Block Source Code

```
/*=============================================================================*//
//      FILE NAME:              _ave1.h                                        //
//      BLOCK NAME: ave1                                                       //
//      GROUP NAME: Motion Control                                             //
//      PURPOSE:                Provides the real-time block's DSP header file.//
//      Hypersignal Block Wizard Version 4.00.14 Auto-Generated Block          //
//      Number of Inputs:       1                                              //
//      Number of Outputs:      1                                              //
//      Creation Date:  Fri - 16 July 1999                                     //
//      Creation Time:  07:03 AM                                               //
//=============================================================================*/

/*****************************************************************************/
/*                                DEFINITIONS                                */
/*****************************************************************************/
/* boolean macros */
typedef unsigned int  BOOL;
#define FALSE   0
#define TRUE    1

/*****************************************************************************/
/*                        Data Pointers / Parameter Structure                */
/* NOTE:        This structure was auto-generated by Hypersignal              */
/*              Block Wizard application.  DO NOT MODIFY THIS STRUCTURE!      */
/*****************************************************************************/

typedef struct
{
/*****************************************************************************/
/*                          Synchronization Addresses                        */
/*****************************************************************************/
  unsigned int *SyncIn;                 /* ptr to input sync value  */
  unsigned int *SyncOut;                /* ptr to output sync value */
```

```
/*****************************************************************************/
/*                          Input Data Pointers / Framesizes                 */
/*****************************************************************************/
 float      *InPtr0;                          /* input 0 data pointer */
 unsigned int FramesizeIn0;                   /* input 0 framesize */

/*****************************************************************************/
/*                          Output Data Pointers / Framesize                 */
/*****************************************************************************/
 float      *OutPtr0;                         /* output 0 data pointer */
 unsigned int FramesizeOut0;                  /* output 0 framesize */

/*****************************************************************************/
/*                                Block Parameters                           */
/*****************************************************************************/
 int        _SampleRate;

/*****************************************************************************/
/*                          Internal Persistent Variables                    */
/*****************************************************************************/
/* NOTE:        If your block requires persistent variables, then they must  */
/*              be put here.  DO NOT USE global variables or statics in the  */
/*              C source file unless you really want them to be shared by    */
/*              all instances of this block.  If you do add variables,       */
/*              update the "#define INTVAR_SIZE" in the PC's header (.h)      */
/*              file to reflect the proper internal variable storage.        */
/*****************************************************************************/
 float    Samples[10000];          /* holding array for integration */
 int      point;                   /* pointer to current position in Samples array */
 float    sumarea;                 /* holding value of all summed areas */
 float    Ilast;                   /* last current Value read */

} PARAMS;

/*****************************************************************************/
/*                          FUNCTION PROTOTYPES                              */
```

```
/*****************************************************************************/

void ave1_INT  (PARAMS *pParam);          /* optional block interrupt routine */

void ave1_INIT (PARAMS *pParam);          /* optional block initialization routine */

void ave1_STOP (PARAMS *pParam);          /* optional block stop routine */

void ave1_RESTART (PARAMS *pParam);       /* optional block restart routine */

void ave1(PARAMS *pParam);                /* main block routine */


/*===========================================================================//
//      FILE NAME:              _ave1.c                                      //
//      BLOCK NAME: ave1                                                     //
//      GROUP NAME: Motion Control                                           //
//      PURPOSE:                Provides the real-time block's DSP C source code.  //
//      Hypersignal Block Wizard Version 4.00.14 Auto-Generated Block        //
//      Number of Inputs:       1                                            //
//      Number of Outputs:      1                                            //
//      Creation Date:  Fri - 16 July 1999                                   //
//      Creation Time:  07:03 AM                                             //
//===========================================================================*/

#include "_ave1.h"
#include "math.h"

/*---------------------------------------------------------------------------*/
/*                    Optional real-time block initialization routine        */
/*---------------------------------------------------------------------------*/
/* If this routine is activated, it will be called one time before the main  */
/* application begins.  This allows for any required software or hardware    */
/* initialization to be performed before the block executes.                 */
/*---------------------------------------------------------------------------*/
```

```
void ave1_INIT(PARAMS *pParam)
{
 int point2;                          /*temporary array pointer */

 (pParam->point) = 0;                 /* set array pointer to zero */
 for (point2 = 0; point2<9999; point2++)
  {
   (pParam->Samples[point2]) = 0;     /* set all values to zero */
  }
 (pParam->sumarea) = 0;               /* value of all summed areas = 0 */
 (pParam->Ilast) = 0;                 /* last current Value =0 */
}

/*-------------------------------------------------------------------------------------------------------------------*/
/*                                  Optional real-time block restart routine                                         */
/*-------------------------------------------------------------------------------------------------------------------*/
/* If this routine is activated, it will be called whenever a block diagram                                          */
/* worksheet is executed after being stopped.  Blocks that deal with hardware                                        */
/* may need this routine to restart the hardware's execution.                                                        */
/*-------------------------------------------------------------------------------------------------------------------*/

void ave1_RESTART(PARAMS *pParam)
{
 int point2;                          /* temporary array pointer */

 (pParam->point) = 0;                 /* set array pointer to zero */
 for (point2 = 0; point2<9999; point2++)
  {
   (pParam->Samples[point2]) = 0;     /* set all values to zero */
  }
 (pParam->sumarea) = 0;               /* value of all summed areas = 0 */
 (pParam->Ilast) = 0;                 /* last current Value =0 */

}
```

```
/*-------------------------------------------------------------------------------------------------------------------*/
/*                                        Real-time block routine                                                    */
/*-------------------------------------------------------------------------------------------------------------------*/
/* This is the main block routine.  It is called during each loop of the main                                        */
/* application.  If an interrupt is selected, and the interrupt routine above                                        */
/* is not activated, this routine will be called in response to the selected                                         */
/* interrupt instead of during the main application loop.                                                            */
/*-------------------------------------------------------------------------------------------------------------------*/
void ave1(PARAMS *pParam)
{
unsigned int index;                          /* index for frame processing loop */
float old,new;                               /* values of areas calculated */
float Iave;                                  /* rms value of input waveform */
float Iin;                                   /* current Value squared */
int point2;                                  /* pointer to area in array */

point2 = (pParam->point);                    /* Retrieve pointer value */
point2++;                                    /* increment pointer to point to old area */
if (point2>9999)                             /* check if at end of array */
  {  point2 = 0;  }
(pParam->point) = point2;                    /* save pointer value */
old = (pParam->Samples[point2]);             /* recall old area */
Iin = *(pParam->InPtr0);
new = ((Iin+(pParam->Ilast))*(1.0/(2*pParam->_SampleRate)));      /* trapizoidal area rule */
(pParam->Ilast) = Iin;                       /* save current I squared value for next calculation */
(pParam->Samples[point2]) = new;             /* save new area */
(pParam->sumarea)=(pParam->sumarea)+new-old; /* get new area under waveform */
Iave = (pParam->sumarea)*(float)((pParam->_SampleRate)/10000.0);

   for (index = 0; index<pParam->FramesizeOut0; index++) {
     *(pParam->OutPtr0+index) = Iave;        /* output average value of input */
   }
}
```

I.4 Efficiency Calculation Block Source Code

```
/*=============================================================================*//
//      FILE NAME:              _effic.h                                       //
//      BLOCK NAME: Efficiency                                                 //
//      GROUP NAME: Motion Control                                             //
//      PURPOSE:                Provides the real-time block's DSP header file.//
//      Hypersignal Block Wizard Version 4.00.14 Auto-Generated Block          //
//      Number of Inputs :      4                                              //
//      Number of Outputs:      3                                              //
//      Creation Date:  Thu - 15 July 1999                                     //
//      Creation Time:  02:39 PM                                               //
//=============================================================================*/

/*****************************************************************************/
/*                                DEFINITIONS                                */
/*****************************************************************************/
/* boolean macros */
typedef unsigned int  BOOL;
#define FALSE   0
#define TRUE    1

/*****************************************************************************/
/*                          Data Pointers / Parameter Structure              */
/* NOTE:        This structure was auto-generated by Hypersignal             */
/*              Block Wizard application.  DO NOT MODIFY THIS STRUCTURE!     */
/*****************************************************************************/

typedef struct
{
/*****************************************************************************/
/*                          Synchronization Addresses                        */
/*****************************************************************************/
  unsigned int *SyncIn;                 /* ptr to input sync value  */
  unsigned int *SyncOut;                /* ptr to output sync value */
```

```
/*****************************************************************************/
/*                        Input Data Pointers / Framesizes                   */
/*****************************************************************************/
 float      *InPtr0;                        /* input 0 data pointer */
 float      *InPtr1;                        /* input 1 data pointer */
 float      *InPtr2;                        /* input 2 data pointer */
 float      *InPtr3;                        /* input 3 data pointer */
 unsigned int FramesizeIn0;                 /* input 0 framesize */
 unsigned int FramesizeIn1;                 /* input 1 framesize */
 unsigned int FramesizeIn2;                 /* input 2 framesize */
 unsigned int FramesizeIn3;                 /* input 3 framesize */

/*****************************************************************************/
/*                        Output Data Pointers / Framesize                   */
/*****************************************************************************/
 float      *OutPtr0;                       /* output 0 data pointer */
 float      *OutPtr1;                       /* output 1 data pointer */
 float      *OutPtr2;                       /* output 2 data pointer */
 unsigned int FramesizeOut0;                /* output 0 framesize */
 unsigned int FramesizeOut1;                /* output 1 framesize */
 unsigned int FramesizeOut2;                /* output 2 framesize */

/*****************************************************************************/
/*                        Block Parameters                                   */
/*****************************************************************************/
 float      _Km;                            /* motor constant parameter */
 float      _Ra;                            /* armature resistance Parameter */
 float      _B;                             /* DC motor Viscous fric. Parameter */
 float      _Vb;                            /* DC motor Brush Volt Drop Parameter*/

/*****************************************************************************/
/*                        Internal Persistent Variables                      */
/*****************************************************************************/
/* NOTE:      If your block requires persistent variables, then they must    */
/*            be put here.  DO NOT USE global variables or statics in the    */
/*            C source file unless you really want them to be shared by       */
```

```
/*              all instances of this block.  If you do add variables.                  */
/*              update the "#define INTVAR_SIZE" in the PC's header (.h)               */
/*              file to reflect the proper internal variable storage.                   */
/*******************************************************************************/

} PARAMS;

/*******************************************************************************/
/*                                  FUNCTION PROTOTYPES                        */
/*******************************************************************************/

void effic_INT  (PARAMS *pParam);          /* optional block interrupt routine */

void effic_INIT (PARAMS *pParam);          /* optional block initialization routine */

void effic_STOP (PARAMS *pParam);          /* optional block stop routine */

void effic_RESTART (PARAMS *pParam);       /* optional block restart routine */

void effic(PARAMS *pParam);                /* main block routine */


/*=============================================================================*//
//      FILE NAME:          _effic.c                                           //
//      BLOCK NAME: Efficiency                                                 //
//      GROUP NAME: Motion Control                                             //
//      PURPOSE:            Provides the real-time block's DSP C source code.  //
//      Hypersignal Block Wizard Version 4.00.14 Auto-Generated Block          //
//      Number of Inputs:   4                                                  //
//      Number of Outputs:  1                                                  //
//      Creation Date:  Thu - 15 July 1999                                     //
//      Creation Time:  01:50 PM                                               //
//=============================================================================*/

#include "_effic.h"
```

```
/*-------------------------------------------------------------------------------------------------------------------*/
/*                                        Real-time block routine                                                    */
/*-------------------------------------------------------------------------------------------------------------------*/
/* This is the main block routine.  It is called during each loop of the main                                        */
/* application.  If an interrupt is selected, and the interrupt routine above                                        */
/* is not activated, this routine will be called in response to the selected                                         */
/* interrupt instead of during the main application loop.                                                            */
/*-------------------------------------------------------------------------------------------------------------------*/
void effic(PARAMS *pParam)
{
unsigned int index;                                  /* index for frame processing loop */
float Vsd_out,Vsd_in,efficiency;                     /* define local parameters */
float Ea,Ia,Is,Vl,Speed,sup_power,dc_power;          /* parameter to improve readability */

index = 0;

Speed = *(pParam->InPtr0+index);                     /* read inputs into variables */
Ia    = *(pParam->InPtr1+index);
Is    = *(pParam->InPtr2+index);
Vl    = *(pParam->InPtr3+index);

Speed = Speed*0.1047197;                             /* convert speed to rad/sec */

Ea = (Speed*(pParam->_Km));                          /* calculate armature voltage */

Vsd_out = (Ea*Ia) + ((pParam->_B)*Speed*Speed);      /* calculate output power from VSD*/

sup_power = Is*Vl;                    /* calculate power flowing to Dc link from 3 phase supply */

dc_power  = (Ea*Ia)-(Ia*Ia*(pParam->_Ra)+(pParam->_Vb)*Ia);   /* calculate power flowing into dc link
                                                                from DC motor */

Vsd_in = sup_power+dc_power;                         /* calculate power flowing into VSD */

efficiency = ((Vsd_out*100)/Vsd_in);                 /* calculate Vsd efficiency in % */
```

```
    for (index = 0; index<pParam->FramesizeOut0; index++)
    {
      *(pParam->OutPtr0+index) = efficiency;          /* output efficiency */
      *(pParam->OutPtr1+index) = Vsd_in;              /* output, input power to VSD */
      *(pParam->OutPtr2+index) = sup_power;           /* output power supplied to test bed */
    }

}
```

I.5 Load Simulator Source Code

```
/*=============================================================================//
//      FILE NAME:              _trqcalc.h                                      //
//      BLOCK NAME: Load Simulator                                              //
//      GROUP NAME: Motion Control                                              //
//      PURPOSE:                Provides the real-time block's DSP header file. //
//      Hypersignal Block Wizard Version 4.00.14 Auto-Generated Block           //
//      Number of Inputs :      2                                               //
//      Number of Outputs:      2                                               //
//      Creation Date:  Tue - 31 August 1999                                    //
//      Creation Time:  07:32 AM                                                //
//=============================================================================*/

/*****************************************************************************/
/*                                  DEFINITIONS                              */
/*****************************************************************************/
/* boolean macros */
typedef  unsigned int  BOOL;
#define  FALSE   0
#define  TRUE    1

/*****************************************************************************/
/*                          Data Pointers / Parameter Structure              */
/* NOTE:        This structure was auto-generated by Hypersignal            */
/*              Block Wizard application.  DO NOT MODIFY THIS STRUCTURE!    */
```

```
/*****************************************************************************/

typedef struct
{
/*****************************************************************************/
/*                          Synchronization Addresses                        */
/*****************************************************************************/
 unsigned int *SyncIn;                          /* ptr to input sync value  */
 unsigned int *SyncOut;                         /* ptr to output sync value */

/*****************************************************************************/
/*                          Input Data Pointers / Framesizes*                /
/*****************************************************************************/
 float    *InPtr0;                              /* input 0 data pointer */
 float    *InPtr1;                              /* input 1 data pointer */
 unsigned int FramesizeIn0;                     /* input 0 framesize */
 unsigned int FramesizeIn1;                     /* input 1 framesize */

/*****************************************************************************/
/*                          Output Data Pointers / Framesize                 */
/*****************************************************************************/
 float    *OutPtr0;                             /* output 0 data pointer */
 float    *OutPtr1;                             /* output 1 data pointer */
 unsigned int FramesizeOut0;                    /* output 0 framesize */
 unsigned int FramesizeOut1;                    /* output 1 framesize */

/*****************************************************************************/
/*                          Block Parameters                                 */
/*****************************************************************************/
 int      _LoadType;                            /* load type parameter passed from PC */

/*****************************************************************************/
/*                          Internal Persistent Variables                    */
/*****************************************************************************/
/* NOTE:    If your block requires persistent variables, then they must      */
/*          be put here.  DO NOT USE global variables or statics in the      */
```

```
/*          C source file unless you really want them to be shared by                */
/*          all instances of this block.  If you do add variables,                   */
/*          update the "#define INTVAR_SIZE" in the PC's header (.h)                 */
/*          file to reflect the proper internal variable storage.                    */
/*****************************************************************************/

} PARAMS;

/*****************************************************************************/
/*                              FUNCTION PROTOTYPES                              */
/*****************************************************************************/

void trqcalc_INT  (PARAMS *pParam);        /* optional block interrupt routine */

void trqcalc_INIT (PARAMS *pParam);        /* optional block initialization routine */

void trqcalc_STOP (PARAMS *pParam);        /* optional block stop routine */

void trqcalc_RESTART (PARAMS *pParam);     /* optional block restart routine */

void trqcalc(PARAMS *pParam);              /* main block routine */


/*===========================================================================*/
//      FILE NAME:          _trqcalc.c                                        //
//      BLOCK NAME: Load Simulator                                            //
//      GROUP NAME: Motion Control                                            //
//      PURPOSE:            Provides the real-time block's DSP C source code. //
//      Hypersignal Block Wizard Version 4.00.14 Auto-Generated Block         //
//      Number of Inputs :  2                                                 //
//      Number of Outputs:  2                                                 //
//      Creation Date:  Tue - 31 August 1999                                  //
//      Creation Time:  07:32 AM                                              //
//===========================================================================*/

#include "_trqcalc.h"
```

```
#include "MATH.h"

/*-------------------------------------------------------------------------------------------------------------------*/
/*                                        Real-time block routine                                                    */
/*-------------------------------------------------------------------------------------------------------------------*/
/* This is the main block routine.  It is called during each loop of the main                                        */
/* application.  If an interrupt is selected, and the interrupt routine above                                        */
/* is not activated, this routine will be called in response to the selected                                         */
/* interrupt instead of during the main application loop.                                                            */
/*-------------------------------------------------------------------------------------------------------------------*/
void trqcalc(PARAMS *pParam)
{
unsigned int index;                          /* index for frame processing loop */
float flow;                                  /* flow demand from 0 to 100% */
float Torque;                                /* output load torque in N.m */
float speedout;                              /* ouput speed to VSD in rpm */
float speedin;                               /* input speed from test bed in rpm */

index = 0;

flow = *(pParam->InPtr0);                    /* get required flow rate */
speedin = *(pParam->InPtr1);                 /* get system speed */


if((pParam->_LoadType) == 0)                 /* fixed speed pump simulation control */
 {
 Torque = ((32.0-11.0)*(flow/100.0)+11.0)*6.37;   /* calculate load torque based on flow demand */
 speedout = 1500;                            /* output speed of test bed set to 1500rpm */
 }

else if((pParam->_LoadType) == 1)            /* variable speed pump simulation control */
 {
 speedout = 1.51*(exp(1.4*log(flow)))+550.0;      /* calculate output speed for VSD */
 Torque = (203.0/(1500.0*1500.0))*speedin*speedin;  /* calculate load torque */
 }
```

```
else if((pParam->_LoadType) == 2)                    /* fixed speed fan simulation control */
 {
  Torque = (15000.0/157.08);                         /* load torque remains constant */
  speedout = 1500;                                   /* output speed of test bed set to 1500rpm */
 }
else if((pParam->_LoadType) == 3)                    /* variable speed fan simulation Control */
 {
  speedout = (flow/100)*1500;                        /* speed and flow is a linear relationship */
  Torque = speedin*speedin*(99.31/(1500.0*1500.0));  /* calculate torque*/
 }

else if((pParam->_LoadType) == 4)                    /* pump system B fixed speed simulation*/
 {                                                   /* Calcs same as pump system A fixed speed*/
 Torques = ((32.0-11.0)*(flow/100.0)+11.0)*6.37      /* calculate torque reference */
 speedout = 1500;                                    /* set speed at 1500rpm - fixed speed*/
 }

else if((pParam->_LoadType) == 5)                    /* pump system B variable speed simulation */
 {
  speedout = 16.049E-3*flow^2+2480;                          /* speed and flow is a linear relationship
                                                             */
  Torque = -630.511E-10*flow^3 + 543.429E-6*flow^2 - 142.07E-2 * flow + 1175.467; /*calculate torque
                                                                                    reference */
 }

   for (index = 0; index<pParam->FramesizeOut0; index++)
    {
     *(pParam->OutPtr0+index) = speedout;                    /* output Data */
     *(pParam->OutPtr1+index) = Torque;                      /* output Data */
    }
}
```

I.6 Error Detection Source Code

```
/*==========================================================================================//
//      FILE NAME:              _errdet.h                                                    //
//      BLOCK NAME: Error Detection                                                          //
//      GROUP NAME: Motion Control                                                           //
//      PURPOSE:                Provides the real-time block's DSP header file.              //
//      Hypersignal Block Wizard Version 4.00.14 Auto-Generated Block                        //
//      Number of Inputs :      0                                                            //
//      Number of Outputs:      1                                                            //
//      Creation Date:  Wed - 19 May 1999                                                    //
//      Creation Time:  12:35 PM                                                             //
//==========================================================================================*/

/*******************************************************************************************/
/*                                      DEFINITIONS                                        */
/*******************************************************************************************/
/* boolean macros */
typedef  unsigned int  BOOL;
#define  FALSE   0
#define  TRUE    1
#define Dio (volatile int*) 0x818000            /* address for digital i/o port */

/*******************************************************************************************/
/*                              Data Pointers / Parameter Structure                        */
/* NOTE:        This structure was auto-generated by Hypersignal                           */
/*              Block Wizard application.  DO NOT MODIFY THIS STRUCTURE!                   */
/*******************************************************************************************/

typedef struct
{
/*******************************************************************************************/
/*                              Synchronization Addresses                                  */
/*******************************************************************************************/
  unsigned int *SyncIn;                         /* ptr to input sync value  */
```

```
 unsigned int *SyncOut;                    /* ptr to output sync value */

/*****************************************************************************/
/*                          Output Data Pointers / Framesize                 */
/*****************************************************************************/
 float      *OutPtr0;                      /* output 0 data pointer */
 unsigned int FramesizeOut0;               /* output 0 framesize */

/*****************************************************************************/
/*                          Internal Persistent Variables                    */
/*****************************************************************************/
/* NOTE:       If your block requires persistent variables, then they must   */
/*             be put here.  DO NOT USE global variables or statics in the   */
/*             C source file unless you really want them to be shared by     */
/*             all instances of this block.  If you do add variables.        */
/*             update the "#define INTVAR_SIZE" in the PC's header (.h)      */
/*             file to reflect the proper internal variable storage.         */
/*****************************************************************************/

} PARAMS;


/*****************************************************************************/
/*                          FUNCTION PROTOTYPES                              */
/*****************************************************************************/

void errdet_INT  (PARAMS *pParam);        /* optional block interrupt routine */

void errdet_INIT (PARAMS *pParam);/       /* optional block initialization routine */

void errdet_STOP (PARAMS *pParam);        /* optional block stop routine */

void errdet_RESTART (PARAMS *pParam);     /* optional block restart routine */

void errdet(PARAMS *pParam);              /* main block routine */
```

```
/*=============================================================================*//
//      FILE NAME:              _errdet.c                                      //
//      BLOCK NAME: Error Detection                                            //
//      GROUP NAME: Motion Control                                             //
//      PURPOSE:                Provides the real-time block's DSP C source code.  //
//      Hypersignal Block Wizard Version 4.00.14 Auto-Generated Block          //
//      Number of Inputs :      0                                              //
//      Number of Outputs:      1                                              //
//      Creation Date:  Wed - 19 May 1999                                      //
//      Creation Time:  12:35 PM                                               //
//=============================================================================*/

#include "_errdet.h"


/*-----------------------------------------------------------------------------*/
/*                      Optional real-time block initialization routine        */
/*-----------------------------------------------------------------------------*/
/* If this routine is activated, it will be called one time before the main    */
/* application begins.  This allows for any required software or hardware      */
/* initialization to be performed before the block executes.                   */
/*-----------------------------------------------------------------------------*/

void errdet_INIT(PARAMS *pParam)
{
 unsigned int index;                            /* index for frame processing loop */

  for (index = 0; index<pParam->FramesizeOut0; index++) {

    *(pParam->OutPtr0+index) = 0x0;             /* output zero no error */
  }
 *(Dio) = 512;                                  /* set green led on -system active */
}

/*-----------------------------------------------------------------------------*/
```

```
/*                          Optional real-time block stop routine                          */
/*-------------------------------------------------------------------------------------------*/
/* If this routine is activated, it will be called whenever the block diagram              */
/* worksheet's execution is stopped.  Blocks that deal with hardware may need              */
/* this routine to stop the hardware's execution.                                          */
/*-------------------------------------------------------------------------------------------*/

void errdet_STOP(PARAMS *pParam)
{
  *(Dio) = 0;                                   /* turn green Led off */

}

/*-------------------------------------------------------------------------------------------*/
/*                          Optional real-time block restart routine                       */
/*-------------------------------------------------------------------------------------------*/
/* If this routine is activated, it will be called whenever a block diagram                */
/* worksheet is executed after being stopped.  Blocks that deal with hardware              */
/* may need this routine to restart the hardware's execution.                              */
/*-------------------------------------------------------------------------------------------*/

void errdet_RESTART(PARAMS *pParam)
{
  unsigned int index;                           /* index for frame processing loop */

    for (index = 0; index<pParam->FramesizeOut0; index++) {

      *(pParam->OutPtr0+index) = 0x0;           /* output zero no error */
    }
   *(Dio) = 512;                                /* turn green Led on */
}

/*-------------------------------------------------------------------------------------------*/
/*                          Real-time block routine                                        */
/*-------------------------------------------------------------------------------------------*/
/* This is the main block routine.  It is called during each loop of the main              */
```

```
/* application.  If an interrupt is selected, and the interrupt routine above                    */
/* is not activated, this routine will be called in response to the selected                     */
/* interrupt instead of during the main application loop.                                        */
/*-------------------------------------------------------------------------------------------------*/
void errdet(PARAMS *pParam)
{

unsigned int index;                                   /* index for frame processing loop */
unsigned int temp;                                    /* temp variable used determining dig o/p */
 temp = (*(Dio)&0x000F);                              /* read digital inputs - remove unwanted bits */
 for (index = 0; index<pParam->FramesizeOut0; index++) {
  *(pParam->OutPtr0+index) = temp;                    /* full output array */
  }

 if (temp == 0)
  {
  *(Dio) = 512;                                       /* system okay, running */
  }
 else if ((temp & 0x01) == 1)
  {
  *(Dio) = 256;                                       /* skiip fault trip refu */
  }
 else if ((temp & 0x02) == 2)
  {
  *(Dio) = 256;                                       /* skiip fault trip refu */
  }
 else if ((temp & 0x04) == 4)
  {
  *(Dio) = 0;                                         /* stop refu drive already off */
  }
 else if ((temp & 0x08) == 8)
  {
  *(Dio) = 0;                                         /* configuarable load disable */
  }
}
```

I.7 PWM Controller Source Code

```
/*==========================================================================================//
//      FILE NAME:              _PWMcon.h                                                   //
//      BLOCK NAME: PWM Control Block                                                       //
//      GROUP NAME: Motion Control                                                          //
//      PURPOSE:                Provides the real-time block's DSP header file.             //
//      Hypersignal Block Wizard Version 4.00.14 Auto-Generated Block                       //
//      Number of Inputs :      4                                                           //
//      Number of Outputs:      0                                                           //
//      Creation Date:  Thu - 04 March 1999                                                 //
//      Creation Time:  08:17 AM                                                            //
//==========================================================================================*/

/*******************************************************************************************/
/*                                      DEFINITIONS                                        */
/*******************************************************************************************/
/* boolean macros */
typedef unsigned int  BOOL;
#define FALSE   0
#define TRUE    1

#define Decodes0 0
#define Decodes1 1

#define dec0 (int) 0x819800                 /* Decodes 0 address */
#define dec1 (int) 0x81a000                 /* Decodes 1 address */


/*******************************************************************************************/
/*                                  Data Pointers / Parameter Structure                    */
/* NOTE:        This structure was auto-generated by Hypersignal                           */
/*              Block Wizard application.  DO NOT MODIFY THIS STRUCTURE!                   */
/*******************************************************************************************/
```

```
typedef struct
{
/*****************************************************************************/
/*                          Synchronization Addresses                        */
/*****************************************************************************/
 unsigned int *SyncIn;                  /* ptr to input sync value  */
 unsigned int *SyncOut;                 /* ptr to output sync value */

/*****************************************************************************/
/*                          Input Data Pointers / Framesizes                 */
/*****************************************************************************/
 float    *InPtr0;                      /* input 0 data pointer */
 float    *InPtr1;                      /* input 1 data pointer */
 float    *InPtr2;                      /* input 2 data pointer */
 float    *InPtr3;                      /* input 3 data pointer */
 unsigned int FramesizeIn0;             /* input 0 framesize */
 unsigned int FramesizeIn1;             /* input 1 framesize */
 unsigned int FramesizeIn2;             /* input 2 framesize */
 unsigned int FramesizeIn3;             /* input 3 framesize */

/*****************************************************************************/
/*                          Block Parameters                                 */
/*****************************************************************************/
 int      _GlobalEnable;                /* global enable parameter */
 int      _EnternalEnable;              /* external/internal enable parameter */
 int      _Input3type;                  /* frequency/phase selection */
 int      _Decodes;                     /* decodes input selection */
 int      _BoardNumber;                 /* board number selection */

 int     *Data_word;                    /* board data word address */
 int     *Status_word;                  /* doard status word address */

/*****************************************************************************/
/*                          Internal Persistent Variables                    */
/*****************************************************************************/
/* NOTE:       If your block requires persistent variables, then they must   */
```

```
/*              be put here.  DO NOT USE global variables or statics in the                */
/*              C source file unless you really want them to be shared by                  */
/*              all instances of this block.  If you do add variables,                     */
/*              update the "#define INTVAR_SIZE" in the PC's header (.h)                  */
/*              file to reflect the proper internal variable storage.                      */
/*******************************************************************************************/

} PARAMS;

/*******************************************************************************************/
/*                                      FUNCTION PROTOTYPES                                */
/*******************************************************************************************/

void PWMcon_INT  (PARAMS *pParam);          /* optional block interrupt routine */

void PWMcon_INIT (PARAMS *pParam);          /* optional block initialization routine */

void PWMcon_STOP (PARAMS *pParam);          /* optional block stop routine */

void PWMcon_RESTART (PARAMS *pParam);       /* optional block restart routine */

void PWMcon(PARAMS *pParam);                /* main block routine */

void pollpwm(int *S_word);                  /* routine to poll PWM asic */


/*=============================================================================*/
//      FILE NAME :     _PWMcon.c                                              //
//      BLOCK NAME: PWM Control Block                                          //
//      GROUP NAME: Motion Control                                             //
//      PURPOSE:            Provides the real-time block's DSP C source code.  //
//      Hypersignal Block Wizard Version 4.00.14 Auto-Generated Block          //
//      Number of Inputs :      4                                              //
//      Number of Outputs:      0                                              //
//      Creation Date:  Thu - 04 March 1999                                    //
//      Creation Time:  08:17 AM                                               //
```

```
//=============================================================================*/

#include "_pwmcon.h"

#define TAUS   ((int)0)                          /* min off time */
#define TTOT   ((int)0)                          /* dead band */
#define TMIN   ((int)0)                          /* min on time */
#define VORTL  ((int)3)                          /*set switching frequency to 4.88kHz*/
#define TSTART ((int)(512-(322/(VORTL+1))))      /* calc Tstart time */

/*-----------------------------------------------------------------------------*/
/*                    Optional real-time block initialization routine          */
/*-----------------------------------------------------------------------------*/
/* If this routine is activated, it will be called one time before the main    */
/* application begins.  This allows for any required software or hardware      */
/* initialization to be performed before the block executes.                   */
/*-----------------------------------------------------------------------------*/

void PWMcon_INIT(PARAMS *pParam)
{ int *temp;                                     /* temporary pointer value */

  if(pParam->_Decodes == Decodes0)               /* check if board on DEC 0 */
   {                                             /* calculate board addresses */
     (pParam->Data_word)   = (int*)(dec0 + ((pParam->_BoardNumber)*256));
     (pParam->Status_word) = (int*)(dec0 + 1 + ((pParam->_BoardNumber)*256));
   }
  else                                           /* board on DEC 1 */
   {   (pParam->Status_word) = (int*)(dec1 + 1+((pParam->_BoardNumber)*256));
       (pParam->Data_word)   = (int*)(dec1 + ((pParam->_BoardNumber)*256));
   }

  *(pParam->Status_word) = 128;                  /* set up 16 bit addressing mode */
  *(pParam->Status_word) = 128;                  /* set address to zero */

  pollpwm((pParam->Status_word));                /* wait for PWM ASIC */
  *(pParam->Data_word) = ((int)0);               /* write Ua */
```

```
  pollpwm((pParam->Status_word));                /* wait for PWM ASIC */
  *(pParam->Data_word) = ((int)0);               /* write Ub */
  pollpwm((pParam->Status_word));                /* wait for PWM ASIC */
  *(pParam->Data_word) = ((int)0);               /* write phi1 */
  pollpwm((pParam->Status_word));                /* wait for PWM ASIC */
  *(pParam->Data_word) = ((int)0);               /* write dphi1 */
  pollpwm((pParam->Status_word));                /* wait for PWM ASIC */
  *(pParam->Data_word) = ((int)0);               /* write phi0 */
  pollpwm((pParam->Status_word));                /* wait for PWM ASIC */
  *(pParam->Data_word) = ((int)0);               /* write dphi0 */
  pollpwm((pParam->Status_word));                /* wait for PWM ASIC */
  *(pParam->Data_word) = ((int)0);               /* write phiadd */
  pollpwm((pParam->Status_word));                /* wait for PWM ASIC */
  *(pParam->Data_word) = ((int)0);               /* unused */
  pollpwm((pParam->Status_word));                /* wait for PWM ASIC */
  *(pParam->Data_word) = TAUS;                   /* write turn off time */
  pollpwm((pParam->Status_word));                /* wait for PWM ASIC */
  *(pParam->Data_word) = TTOT;                   /* write dead band */
  pollpwm((pParam->Status_word));                /* wait for PWM ASIC */
  *(pParam->Data_word) = TMIN;                   /* write turn on time */
  pollpwm((pParam->Status_word));                /* wait for PWM ASIC */
  *(pParam->Data_word) = VORTL;                  /* write switching frequency scale value */
  pollpwm((pParam->Status_word));                /* wait for PWM ASIC */
  *(pParam->Data_word) = TSTART;                 /* write start of processing cycle */
  *(pParam->Status_word) = 129;                  /* ENABLE CHIP */

}

/*-------------------------------------------------------------------------------------------------------------------*/
/*                                    Optional real-time block stop routine                                          */
/*-------------------------------------------------------------------------------------------------------------------*/
/* If this routine is activated, it will be called whenever the block diagram                                        */
/* worksheet's execution is stopped.  Blocks that deal with hardware may need                                        */
/* this routine to stop the hardware's execution.                                                                    */
/*-------------------------------------------------------------------------------------------------------------------*/
```

```
void PWMcon_STOP(PARAMS *pParam)
{
 *(pParam->Status_word) = 0;                    /* disable chip */
 *(pParam->Status_word) = 0;                    /* disable chip */
}

/*-------------------------------------------------------------------------------------------------------------------*/
/*                                  Optional real-time block restart routine                                         */
/*-------------------------------------------------------------------------------------------------------------------*/
/* If this routine is activated, it will be called whenever a block diagram                                          */
/* worksheet is executed after being stopped.  Blocks that deal with hardware                                        */
/* may need this routine to restart the hardware's execution.                                                        */
/*-------------------------------------------------------------------------------------------------------------------*/

void PWMcon_RESTART(PARAMS *pParam)
{ int *temp;                                    /* temporary pointer variable */

  if(pParam->_Decodes == Decodes0)              /* check if dec 0 is used */
   {                                            /* calculate board address */
     (pParam->Data_word)   = (int*)(dec0 + ((pParam->_BoardNumber)*256));
     (pParam->Status_word) = (int*)(dec0 + 1 + ((pParam->_BoardNumber)*256));
   }
  else                                          /* else used DEC1 */
   {   (pParam->Status_word) = (int*)(dec1 + 1+((pParam->_BoardNumber)*256));
       (pParam->Data_word)   = (int*)(dec1 + ((pParam->_BoardNumber)*256));
   }

  *(pParam->Status_word) = 128;                 /* set up 16 bit addressing mode */
  *(pParam->Status_word) = 128;                 /* set address to zero */

  pollpwm((pParam->Status_word));               /* wait for PWM ASIC */
  *(pParam->Data_word) = ((int)0);              /* write Ua */
  pollpwm((pParam->Status_word));               /* wait for PWM ASIC */
  *(pParam->Data_word) = ((int)0);              /* write Ub */
  pollpwm((pParam->Status_word));               /* wait for PWM ASIC */
  *(pParam->Data_word) = ((int)0);              /* write phi1 */
```

```
  pollpwm((pParam->Status_word));                    /* wait for PWM ASIC */
  *(pParam->Data_word) = ((int)0);                   /* write dphi1 */
  pollpwm((pParam->Status_word));                    /* wait for PWM ASIC */
  *(pParam->Data_word) = ((int)0);                   /* write phi0 */
  pollpwm((pParam->Status_word));                    /* wait for PWM ASIC */
  *(pParam->Data_word) = ((int)0);                   /* write dphi0 */
  pollpwm((pParam->Status_word));                    /* wait for PWM ASIC */
  *(pParam->Data_word) = ((int)0);                   /* write phiadd */
  pollpwm((pParam->Status_word));                    /* wait for PWM ASIC */
  *(pParam->Data_word) = ((int)0);                   /* unused */
  pollpwm((pParam->Status_word));                    /* wait for PWM ASIC */
  *(pParam->Data_word) = TAUS;                       /* write turn off time */
  pollpwm((pParam->Status_word));                    /* wait for PWM ASIC */
  *(pParam->Data_word) = TTOT;                       /* write dead band */
  pollpwm((pParam->Status_word));                    /* wait for PWM ASIC */
  *(pParam->Data_word) = TMIN;                       /* turn on time */
  pollpwm((pParam->Status_word));                    /* wait for PWM ASIC */
  *(pParam->Data_word) = VORTL;                      /* write switching frequency scale value */
  pollpwm((pParam->Status_word));                    /* wait for PWM ASIC */
  *(pParam->Data_word) = TSTART;                     /* write start of processing cycle */
  *(pParam->Status_word) = 129;                      /* ENABLE CHIP */
}

/*-------------------------------------------------------------------------------------------------------------------*/
/*                                          Real-time block routine                                                  */
/*-------------------------------------------------------------------------------------------------------------------*/
/* This is the main block routine.  It is called during each loop of the main                                        */
/* application.  If an interrupt is selected, and the interrupt routine above                                        */
/* is not activated, this routine will be called in response to the selected                                         */
/* interrupt instead of during the main application loop.                                                            */
/*-------------------------------------------------------------------------------------------------------------------*/
void PWMcon(PARAMS *pParam)
{
  unsigned int index;                                /* index for frame processing loop */
  float  control,Ua,Ub,freq ;                        /* variable used */
```

```
Ua = *(pParam->InPtr0);                    /* accessing input data 0 */
Ub = *(pParam->InPtr1);                    /* accessing input data 1 */
freq = *(pParam->InPtr2);                  /* accessing input data 2 */
control = *(pParam->InPtr3);               /* accessing input data 3 */

if(pParam->_GlobalEnable)                  /* only run if global enable true */
 {
  if((pParam->_EnternalEnable)|(control<1))  /* check if external enable is used */
   {
    *(pParam->Status_word) = 129;          /* set address to zero */
    pollpwm((pParam->Status_word));        /* wait for PWM ASIC */
    *(pParam->Data_word) = (int)Ua;        /* write out Ua voltage */
    pollpwm((pParam->Status_word));        /* wait for PWM ASIC */
    *(pParam->Data_word) = (int)Ub;        /* write out Ub voltage */

    if (pParam->_Input3type)               /* determine weather parameter is freq of phase */
     {
      pollpwm((pParam->Status_word));      /* wait for PWM ASIC */
      *(pParam->Data_word) = (int)freq;    /* write out phase */
     }
    else
     {
      *(pParam->Status_word) = 897;        /* change address in PWM ASIC
      pollpwm((pParam->Status_word));      /* wait for PWM ASIC */
      *(pParam->Data_word) = (int)freq;    /* write out frequency */
     }
   }
  else
   {
    *(pParam->Status_word) = 0;            /* disable chip */
   }
 }
else
 {
  *(pParam->Status_word) = 0;              /* disable chip */
 }
```

```
}

void pollpwm(int *S_word)                          /* routine to check if PWM ASIC ready */
{
 int check;

 check = *(S_word);
  while ((check & 0x1) != 0)                       /* if zero then data can be written */
  {
     check = *(S_word);
  }
}
```

G.8 PI Controller Source Code

```
/*=============================================================================//
//      FILE NAME:              _picontr.h                                      //
//      BLOCK NAME: PI Controller                                               //
//      GROUP NAME: Motion Control                                              //
//      PURPOSE:                Provides the real-time block's DSP header file. //
//      Hypersignal Block Wizard Version 4.00.14 Auto-Generated Block           //
//      Number of Inputs :      1                                               //
//      Number of Outputs:      1                                               //
//      Creation Date:  Thu - 19 March 1998                                     //
//      Creation Time:  02:33 PM                                                //
//=============================================================================*/

/*****************************************************************************/
/*                                DEFINITIONS                                */
/*****************************************************************************/
/* boolean macros */
typedef unsigned int  BOOL;
#define FALSE   0
#define TRUE    1
```

```
/*****************************************************************************/
/*                        Data Pointers / Parameter Structure                */
/* NOTE:    This structure was auto-generated by Hypersignal                 */
/*          Block Wizard application.  DO NOT MODIFY THIS STRUCTURE!         */
/*****************************************************************************/

typedef struct
{
/*****************************************************************************/
/*                          Synchronization Addresses                        */
/*****************************************************************************/
 unsigned int *SyncIn;          /* ptr to input sync value */
 unsigned int *SyncOut;         /* ptr to output sync value */

/*****************************************************************************/
/*                        Input Data Pointers / Framesizes                   */
/*****************************************************************************/
 float    *InPtr0;                      /* input 0 data pointer */
 unsigned int FramesizeIn0;             /* input 0 framesize */

/*****************************************************************************/
/*                        Output Data Pointers / Framesize                   */
/*****************************************************************************/
 float    *OutPtr0;                     /* output 0 data pointer */
 unsigned int FramesizeOut0;            /* output 0 framesize */

/*****************************************************************************/
/*                             Block Parameters                              */
/*****************************************************************************/
 float    _Kp;                          /* proportional gain value */
 float    _Ki;                          /* integral gain value */
 float    _Omax;                        /* saturation limit to prevent windup */
 float    _SampleRate;                  /* sample rate for PI controller */
 float    Oprev;                        /* previous output */
 float    Eprev;                        /* previous error */
```

```
/*****************************************************************************/
/*                          Internal Persistent Variables                    */
/*****************************************************************************/
/* NOTE:    If your block requires persistent variables, then they must      */
/*          be put here.  DO NOT USE global variables or statics in the      */
/*          C source file unless you really want them to be shared by        */
/*          all instances of this block.  If you do add variables,           */
/*          update the "#define INTVAR_SIZE" in the PC's header (.h)         */
/*          file to reflect the proper internal variable storage.            */
/*****************************************************************************/

} PARAMS;

/*****************************************************************************/
/*                          FUNCTION PROTOTYPES                              */
/*****************************************************************************/

void picontr_INT  (PARAMS *pParam);        /* optional block interrupt routine */

void picontr_INIT (PARAMS *pParam);        /* optional block initialization routine */

void picontr_STOP (PARAMS *pParam);        /* optional block stop routine */

void picontr_RESTART (PARAMS *pParam);     /* optional block restart routine */

void picontr(PARAMS *pParam);              /* main block routine */


/*===========================================================================*/
//      FILE NAME:          _picontr.c                                       //
//      BLOCK NAME: PI Controller                                            //
//      GROUP NAME: Motion Control                                           //
//      PURPOSE:    Provides the real-time block's DSP C source code.        //
//      Hypersignal Block Wizard Version 4.00.14 Auto-Generated Block        //
//      Number of Inputs :  1                                                //
//      Number of Outputs:  1                                                //
```

```
//      Creation Date:   Thu - 19 March 1998                                    //
//      Creation Time:   02:33 PM                                               //
//=============================================================================*/

#include "_picontr.h"

/*-----------------------------------------------------------------------------*/
/*                    Optional real-time block initialization routine          */
/*-----------------------------------------------------------------------------*/
/* If this routine is activated, it will be called one time before the main    */
/* application begins.  This allows for any required software or hardware      */
/* initialization to be performed before the block executes.                   */
/*-----------------------------------------------------------------------------*/

void picontr_INIT(PARAMS *pParam)
{
(pParam->Oprev) = 0;                    /* reset previous output to zero */
(pParam->Eprev) = 0;                    /* reset previous error to zero */
}

/*-----------------------------------------------------------------------------*/
/*                    Optional real-time block restart routine                 */
/*-----------------------------------------------------------------------------*/
/* If this routine is activated, it will be called whenever a block diagram    */
/* worksheet is executed after being stopped.  Blocks that deal with hardware  */
/* may need this routine to restart the hardware's execution.                  */
/*-----------------------------------------------------------------------------*/

void picontr_RESTART(PARAMS *pParam)
{
(pParam->Oprev) = 0;                    /* reset previous output to zero */
(pParam->Eprev) = 0;                    /* reset previous error to zero  */
}

/*-----------------------------------------------------------------------------*/
/*                          Real-time block routine                            */
```

```
/*-------------------------------------------------------------------------------------------------------------------*/
/* This is the main block routine.  It is called during each loop of the main                                        */
/* application.  If an interrupt is selected, and the interrupt routine above                                        */
/* is not activated, this routine will be called in response to the selected                                         */
/* interrupt instead of during the main application loop.                                                            */
/*-------------------------------------------------------------------------------------------------------------------*/
void picontr(PARAMS *pParam)
{
unsigned int index;                                   /* index for frame processing loop */
float error,kp,ki,Omax;                               /* variables to make code readable */
float output;                                         /* output value of PI controller */

 kp = (pParam->_Kp);                                  /* get proportional gain value */
 ki = (pParam->_Ki);                                  /* get iontegral gain value */
 error = *(pParam->InPtr0+0);                         /* get last error */
 Omax = (pParam->_Omax);                              /* get saturation limit - anti windup */

 output = kp*(error-(pParam->Eprev)) +
        ((kp*ki)/(2*(pParam->_SampleRate)))*(error+(pParam->Eprev)) +
        (pParam->Oprev);                              /* calculate PI controller output value */

 if (output > Omax)                                   /* check for integral windup */
  {
   output = Omax;
  }
 else {}
 if (output < -Omax)
  {
   output = -Omax;
   }
 else {}

 (pParam->Oprev)=output;                              /* save output for next calculation */
 (pParam->Eprev)=error;                               /* save error for next calculation */

   for (index = 0; index<pParam->FramesizeOut0; index++) {
```

```
        *(pParam->OutPtr0+index) = output;                    /* full output array */
    }
}
```

References

[1] African Energy, "Where will Eskom's Future SA Generation Capacity Come From?" , African Energy, February - March 1999, Vol. 1, No.1, Pg 37 - 38.

[2] African Energy, "Why SA is Pursuing Demand Side Management", African Energy, February - March 1999, Vol. 1, No. 1, Pg 32 - 33.

[3] Associated Pumping Services CC, P.O Box 286, New Germany, 3620 South Africa.

[4] I Boldea, SA Nasar, "Vector Control of AC Drives" CRC Press, 1992, ISBN 0-8493-4408-5, Pg 3 - 10.

[5] Stafford S. Cuffe, Peter W. Hammond, "A Variable Frequency AC Blower Drive Installation for Efficient and Accurate Control of Glass Tempering", IEEE Transactions on Industry Applications, Vol. IA-21, No. 4, July/Aug 1985, pg. 1047-1052.

[6] F.A. DeWinter and B.J. Kedrosky, "The Application of a 3500-hp Variable Frequency Drive for Pipeline Pump Control", IEEE Transactions on Industry Applications, Vol. 25, No. 6, Nov/Dec 1989, pg. 1019-1024.

[7] Diana G, Harley RG, "An Aid for Teaching Field Oriented Control Applied to Induction Machines", IEEE Trans. On Power Systems, Vol. 4 N0. 3, Aug 1989, Pg. 1258 - 1262.

[8] Howden Donkin (PTY) LTD, P.O Box 15196 Westmead, 3620 South Africa.

[9] JA Edminister, "Electric Circuits", 2nd edition, McGraw-Hill, 1983, ISBN 0-07-018984-6, Pg 283 - 284

[10] "Semiconductor growth continuing", Information from Electronica '98 Trade Fair, Munich, Elektron, Vol. 15, No. 6, June 1998, pg. 10.

[11] "Eskom Annual Report 1998", Eskom, 1998, Pg 17

[12] "Eskom Annual Report 1999", Eskom, 1999

[13] J.H.Eto and A de Almeida, "Saving Electricity in Commercial Buildings with Adjustable-Speed Drives", IEEE Transaction on Industry Applications, Vol. 24, No. 3, May/June 1988, pg. 439 - 443.

[14] CF Gerald, PO Wheatly, "Applied Numerical Analysis", 5th Edition, Addison and Weasly, 1994, ISBN 0-0201-59290-8, Pg 213 - 215

[15] CF Gerald, PO Wheatly, "Applied Numerical Analysis", 5th Edition, Addison and Weasly, 1994, ISBN 0-0201-59290-8, Pg 330 - 370

[16] Hanning Electro-Werke, "Pulse Width Modulator PBM 1/87. PBM 1/89 Data Sheet", Revision 2.0, 1993

[17] Hanning Electro-Werke, "Tacho Controller TC3005H Data Sheet", Revision 1.1, 1994

[18] Innovative Integration. "PC32 Hardware User's Manual" Innovative Integration, 1995

[19] IJ Karassik, WC Krutzsch, WH Fraser, JP Messina, "Pump Handbook", 2nd Edition, McGraw-Hill, 1986, ISBN 0-07-033702-05

[20] Kleinhans CE, "Simulation and Practical Implementation of Field Oriented Control on the Current Source Inverter-Fed Induction Machine", Msc Thesis, University of Natal, 1995, Durban, South Africa.

[21] PC Krause, "Analysis of Electric Machinery", McGraw-Hill, 1986, ISBN 0-07-035436-7

[22] FA Kristal, FA Annett, "Pumps", McGraw-Hill, 1940, pg 272 - 276

[23] LEM, "LA205-S data sheet", LEM SA.

[24] LEM, "LV25-P data sheet", LEM SA

[25] W. K. Loh, "Performance of vector drives", Elektron, Vol. 14, No. 2, February 1997, pg. 10-12.

[26] J Mately, "Fluid Movers pumps, compressors, fans and blowers", McGraw- Hill, 1979, ISBN 0-07-010769-6, pg163-165.

[27] DB Miron, "Design of Feedback Control Systems", Harcourt Brace Jovanovich Inc., 1989, ISBN 0-15-517368-5, Pg. 43 - 47.

[28] T.A. Nondahl, "Basic Features of a Microprocessor-Controlled DC Drive", IEEE Industry Applications Society Annual Meeting, Houston, Texas, Oct 1992, pg 6.1 - 6.20.

[29] K Ogata, "Modern Control Engineering", 2nd Edition, Prentice-Hall International, 1990, ISBN 0-13-598731-8

[30] K Ogata, "Discrete-Time Control Systems", Prentice Hall International, 1987, ISBN 0-13-216102-8

[31] WC OSBORNE, "Fans", Pergamon Press, 1966, Library of Congress Catalog No. 66-18408.

[32] "Pumping Manual", Trade & Technical Press Ltd, 1964, pg 469 - 611

[33] REFU, "Operating Instructions REFU 401-K",1990

[34] D.E. Rice, "A Suggested Energy-Savings Evaluation Method for AC Adjustable Speed Drive Applications", IEEE Transaction on Industry Applications, Vol. 24, No. 6, Nov/Dec 1988, pg 1107 - 1117.

[35] Radio Energie, "DYNAMOS TACHYMETRIQUES Reo 444 TACHODYNAMOS", Radio Energie.

[36] Hypersignal, "Block Diagram/RIDE User's Manual" Hypersignal, 1996

[37] J.D. Rozner, P.W. Meyers, C.J. Robb, "The Application of Adjustable Frequency Controllers to Force Draft Fans for Improved Reliability and Energy Savings", IEEE Transaction on Industry Applications, Vol. IA-21, No. 6, Nov/Dec 1985, pg 1482 - 1490.

[38] MG Say, EO Taylor, "Direct Current Machines", 2nd Edition, Pitman, 1986, ISBN 0-273-02457-4, Pg 256- 263.

[39] Semikron, "Semikron Data Book", Semikron, 1998.

[40] SHILSTON PD, "Variable Speed Pump Drives - The Way Ahead", Pumps - The heart of the Matter, English Technical conference of the British Pump Manufacturers' Association, 1983, Cambridge, Pg 23 - 36.

[41] R.E. SIEK, C.L. BECNEL, "Pipeline Characteristics and Economic Consideration of Electric Pump Drives", IEEE Transaction of Industry Applications, Vol. IA-16, No. 5, pg. 633-640, Sept/Oct. 1980.

[42] email Correspondence with Siemens Germany. Karl.Reuss@nbg7.siemens.de

[43] The Math Works Inc., "Simulink, Using Simulink Version 2", Math Works Inc, 1999.

[44] J.J. Smit, P. Bento, "Energy Savings Through the Use of Variable Speed Drives", Elektron, Vol. 8, No. 10, Oct 1991, pg 23 - 29.

[45] Texas Instruments, "TMS320C3x User's Guide", Revision L, Texas Instruments, 1997, Literature number SPRU031E

[46] Van Blerk BD, "Development of a Scaled - Down Paper Machine to Demonstrate the Principles of Tension Control", Msc Thesis, University of Natal, 1997, Durban, South Africa.

[47] WALDRON WJ, "A System For Achieving Variable Flow From Fixed Speed Pumps" Pumps - The heart of the Matter, English Technical conference of the British Pump Manufacturers' Association, 1983, Cambridge, Pg 67 - 77.

[48] Walker ML, Diana G, "Test Bed System to Evaluate the Efficiency of Variable Speed Drives Under Varying Load Conditions", Seventh South African Universities Power Engineering Conference, Pg 223 - 226., Stelenbosh, South Africa, January 1998.

[49] Walker ML, Diana G, "Test Bed System to Evaluate the Efficiency of Variable Speed Drives Under Varying Load Conditions", IEEE International Symposium on Industrial Electronics, Pg 345-348, Pretoria, South Africa, July 1998.

[50] Walker ML, Diana G, "Motion Control Card for the Implementation of Variable Speed Drives and Associated Controllers", Eighth South African Universities Power Engineering Conference, Pg 46 - 51, Potchefstroom, South Africa, January 1999.

[51] J.R. Pottebaum, "Optimal Characteristics of a Variable-Frequency Centrifugal Pump Motor Drive", IEEE Transaction on Industry Applications, Vol. IA-20, No. 1, January/February 1984, Pg. 23 - 31.