AN ABSTRACT OF THE THESIS OF

Shibashis Bhowmik for the degree of Doctor of Philosophy in Electrical and Computer Engineering presented on August 26, 1997.

Title: **Performance Optimization for Doubly-Fed Generation Systems**

Redacted for Privacy

Abstract approved: _____
René Spée

A variable speed generation (VSG) system converts energy from a variable resource such as wind or water flow into variable rotational mechanical energy of a turbine or a similar device that converts translational kinetic energy into rotational mechanical energy. The mechanical energy is then converted into electrical energy by an electrical generator. Presently available and proposed generators include systems based mainly on dc machines, synchronous and induction machine technology as well as reluctance machines. While extracting more energy from the resource, most proposed VSG systems suffer a cost disadvantage due to the required rating of the power electronic interface. This cost penalty may eventually render the additional energy capture meaningless. Thus, reducing the cost of the power electronic hardware is essential for VSG systems to achieve viable and competitive $/kWh ratios when compared to fossil fuel-based generating systems.

A variable speed constant frequency (VSCF) system and controller are proposed that utilize a doubly-fed machine (DFM) as the energy conversion device. The system includes a power converter that provides the current excitation for the control winding of the DFM. Both the magnitude and frequency of the excitation is determined by an adaptive model-based controller which maximizes the power flow from the mechanical turbine to the electrical grid and reduces the generator losses by maintaining the maximum efficiency point throughout the mechanical input power range.

The proposed strategy has been experimentally verified in controlled laboratory conditions for a proof-of-concept brushless doubly-fed machine (BDFM) system of 1500 Watts power rating. Issues relating to power converter development and its incorporation in the system have been investigated. The controller and circuit design of a four quadrant, AC/AC power converter is presented and a novel sensorless current controller for the active rectifier stage is presented in detail.

While presented for a wind turbine application, the philosophy of the control algorithms are equally applicable for any variable speed application (motoring, generating, with and without gearbox, etc.) Also, the control can be augmented to not only maximize power and efficiency, but also provide for harmonic and reactive power compensation via the energy conversion system.

Performance Optimization

for Doubly-Fed Generation Systems

by

Shibashis Bhowmik

A Thesis Submitted

to

Oregon State University

In Partial Fulfillment of
the requirements for the
degree of

Doctor of Philosophy

Presented August 26, 1997
Commencement June 1998

Doctor of Philosophy thesis of **Shibashis Bhowmik** presented on **August 26, 1997**

Approved:

Redacted for Privacy

Major professor representing Electrical and Computer Engineering

Redacted for Privacy

Chair of the Department of Electrical and Computer Engineering

Redacted for Privacy

Dean of Graduate School

I understand that my thesis will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my thesis to any reader upon request.

Redacted for Privacy

Shibashis Bhowmik, Author

Acknowledgements

It has been quite a few years at Oregon State University since my initiation into the M.S. program in the Department of Electrical and Computer Engineering in the fall of 1990. During this period, I was fortunate to have come across a spectrum of some of the most highly accomplished individuals as my gurus and some eagerly ambitious colleagues who, I am sure, would one day be well recognized in their vocation as well.

My special and sincere note of acknowledgement goes to Dr. René Spée, my mentor for almost all these years at Oregon State University. He has been a constant source of guidance, help and tutelage for me and has been, as in all other projects, a significant contributor towards the completion of my dissertation. Kudos is also due to Profs. G.C. Alexander and A.K. Wallace of the Energy Systems group. Despite their busy schedules, they were always willing to help whenever I needed it. My, brief but rewarding, association with Hian (Prof. H.K. Lauw), while working at Electronic Power Conditioning Inc., Corvallis, OR, had been, mostly, a didactic experience. He had me develop invaluable engineering skills and technical knowledge. I also thank Prof. W. Kolodziej for serving on my doctoral committee and for those elucidating control courses that he taught us.

Amongst my colleagues, the names that readily come to mind are those of Patrick Rochelle, Ashok (Dr. Ramchandran), Viren Javadekar, D.K. Ravi, Dongsheng (Dr. Zhou), Brian Wiley, Chris Brune, Bhanu (Dr. Gorti), Mike (Dr. Boger), Ernesto Weidenbrüg, Trey Kemp, Manfred Dittrich, Annabelle van Zyl, Alex Faveluke, Brian Koch and Bob Bellagh. I have benefited from each of them, at some time or the other, during these years. I would especially cherish the most recent memories of working with the last five, mentioned above, and the team spirit that prevailed all throughout the duration of that project. Here, I would also like to thank my very special friend Rita Wells who had been the Graduate Secretary throughout my stay at OSU. Rita has come

to my rescue innumerable number of times. She always saved the day for me with a smile never letting me know that a favor was being done.

But I can never recount how much, my wife, Neeta helped me to see to the completion of this ordeal. I am indebted to her patience, her faith in me and her motivation allthroughtout. Likewise, a special note of thanks also go to my sister-in-law, Cheenu, for all the cheering and encouragement that she had provided me. Still, this work would never have been possible without the constant emotional support and understanding of my parents. I admire and respect them for their guidance and advice all throughout.

Acknowledgements

The work described in this thesis has been supported by a consortium consisting of Electric Power Research Institute, Bonneville Power Administration and Puget Sound Power and Light. The author is appreciative of the funding which made this work possible.

Table of Contents

Table of Contents (Continued)

Table of Contents (Continued)

Table of Contents (Continued)

List of Figures

List of Figures (Continued)

List of Figures (Continued)

List of Figures (Continued)

List of Figures (Continued)

List of Tables

List of Symbols

E_{ke}	Kinetic energy of moving air
m	mass of air
v	speed of moving air
v_p	tip speed of turbine blade
ρ	air density
P_w	Power in moving wind
P_{wt}	mechanical power in the shaft of the wind turbine
C_p	power coefficient of a wind turbine; ratio of P_{wt} to P_w
T_{wt}	mechanical torque in the shaft of the wind turbine
λ	tip-speed ratio; ratio of v_p to v
f_p	frequency of DFM power winding excitation
f_c	frequency of DFM control winding excitation
p_p	number of pole pairs of the DFM power winding
p_c	number of pole pairs of the DFM control winding
P_T	total generated power
η_{max}	estimated maximum DFM efficiency
I_c^{opt}	optimum DFM control winding rms current
v_{dc}	instantaneous dc-bus voltage
V_{dc}^*	dc-bus voltage reference
V_{ac}, V_{LL}	rms grid voltage (line-to-line)
i_a, i_b, i_c	instantaneous phase currents
L	input line inductance
C	capacitance of the dc-bus
v_{an}, v_{bn}, v_{cn}	instantaneous input phase voltages

Performance Optimization for Doubly-Fed Generation Systems

1. Introduction [1]

Wind power is gradually gaining prominence as a suitable source of renewable energy. Implementations range from the propeller-type, horizontal axis wind turbine (HAWT) to vertical axis or VAWT systems. Power ratings vary from several kW for remote area, off-grid applications to several MW for systems connected to the ac power grid. As wind speeds, in practice, vary over a wide range, so does the amount of power generated by the turbine. Hence, if wind turbines are only operated at a constant rotor speed, they are not capable of optimizing power extraction over a wide range of wind speeds. This can be improved, mechanically, by providing for adjustments in the pitch of the turbine blades or by utilizing the electrical system for simple, step-wise speed control employing pole-changing techniques. However, implementing mechanical controls renders the system structure more complicated and less reliable. Also, due to the narrow control bandwidth of the mechanical controls, improvement in energy capture is only marginal. With the advent of modern power electronic converters, it is possible to refer the entire control function to the electrical side and consequently improve energy capture and system reliability substantially. Many modern, commercial wind turbines are of the variable speed design, requiring a power electronic interface between the variable frequency wind turbine and the fixed frequency electric utility grid.

Horizontal axis turbines can have blade spans of up to approximately 90 m, with the majority of commercial systems being in the range of 15 - 45 m, with a power rating of 100 to 600 kW. Rotational speeds are in the range of 30 - 100 r/min, necessitating a gearbox to increase the speed to a value suitable for low pole number electric machines, typically around 1800 r/min. Fixed speed systems utilize either conventional synchronous machines or squirrel cage induction machines. Converter-fed variable speed systems can be based on conventional generators or can be implemented utilizing doubly-

fed machines to reduce rating and cost of the power electronic converter [1,2]. In HAWT systems, the gearbox and the generator are housed in the nacelle at the top of the tower structure, while in VAWT installations, all generating equipment can be positioned on the ground at the base of the turbine. In order to increase efficiency and decrease complexity, low speed generators can be utilized to eliminate the need for a step-up gearbox and couple the electric generator to the turbine rotor in a direct-drive configuration.

Wind power is a renewable resource and freely available worldwide. However, site preparation, system installation and provision for maintenance require significant investment. Economic considerations require that the cost of wind energy has to be competitive with conventional fossil resources in order to be a viable alternative. This becomes important in variable speed systems, which, while extracting more energy from the wind, suffer an initial cost disadvantage due to either the increased mechanical complexity or the power electronic converter requirement [3]. This cost penalty may eventually negate the gains associated with additional energy capture. Thus, reducing the cost of the power generating hardware is essential for variable-speed generating systems to achieve viable and competitive $/kWh ratios.

The power electronic utility interface necessary for converter based variable speed systems provides many additional benefits beyond enhanced energy capture. While acoustic noise produced during low power operation can be minimized, it is also possible to improve the poor displacement power factor associated with induction generators. At the same time, harmonics from other generators or non-linear loads can be compensated using appropriate converter control algorithms and utilizing passive or active harmonic filters.

While offering significant benefits in operational performance, variable-speed generators have the drawbacks of higher initial cost as well as increased complexity and potentially lower reliability. Hence, the need for a low-cost, robust variable speed generation (VSG) system is essential for enhanced energy capture with a reduced capital

investment. The brushless doubly-fed machine retains the ruggedness of an induction machine while allowing VSG operation at a significantly lower initial investment due to the reduction in rating of its power electronic energy conversion interface.

1.1. Basic wind turbine characteristics

The conversion of wind energy into electricity on a large scale raises the problem of energy storage. Work is in progress on a variety of energy storage systems, including batteries, pumped storage, compressed air and flywheels. Still, the electrical supply grid remains a convenient absorber of electrical energy provided the capacity of the wind generation equipment is still small in relation to the total generating capacity coupled to the supply grid. Thus, other than in relatively low power, remote area applications, most wind turbines are coupled to the ac utility grid.

1.1.1. Wind energy conversion

The energy in the wind as a result of the kinetic energy of the moving air mass is described by :

$$E_{ke} = \frac{1}{2} m v^2$$
$$\textit{with : } m \textit{ - air mass;}$$
$$v \textit{ - speed of moving} \qquad \textbf{(1.1)}$$

Air density is a function of altitude and temperature, given by:

$$\rho = 3443 \frac{P_a}{T}$$
$$\textit{with : } \rho \textit{ - air density;}$$
$$P_a \textit{ - atmospheric pressure;}$$
$$T \textit{ - absolute temperature.} \qquad \textbf{(1.2)}$$

The mass of air moving through a wind turbine of sweep area A per unit time is

$$m = \rho A v \tag{1.3}$$

and the total available power from the air movement through area A is expressed as

$$P_w = \frac{1}{2}\rho A v^3 \tag{1.4}$$

Wind power is thus proportional to the cube of the wind speed, v, and the power extracted by the wind turbine in mechanical form can be described as

$$P_{wt} = C_p P_w \tag{1.5}$$

with: C_p - *wind turbine power coefficient.*

It is not possible to extract all energy from the moving air, since in this case the air movement would seize and the air would pile up behind the turbine. It was shown by Betz [4] that for maximum power extraction, the air velocity at the wind turbine is $\frac{2}{3}$ of the upstream wind speed, further decreasing to $\frac{1}{3}$ well downstream of the turbine. Thus, the theoretical maximum power coefficient is $C_p = \frac{16}{27}$ or 59.3%. Practical wind turbines do not reach this Betz limit and the wind velocity at the turbine is somewhat greater than $\frac{2}{3} v$. The value of C_p is a function of wind and rotational speeds as well as form and pitch of the wind turbine and has a maximum at a fixed operating point, i.e. at a constant ratio of rotor speed to wind speed. Thus, the power coefficient is normally expressed in terms of the tip-speed-ratio λ, which is defined as:

$$\lambda = \frac{v_p}{v} = \frac{\Omega_T R}{v} \tag{1.6}$$

with: v_p - *tip - speed of turbine blade;*
R - *turbine rotor radius;*
Ω_T - *rotational turbine angular velocity;*
v - *wind speed.*

Figure 1.1 illustrates the variation of the power coefficient for a conceptual 100 kW horizontal axis wind turbine [1,2].

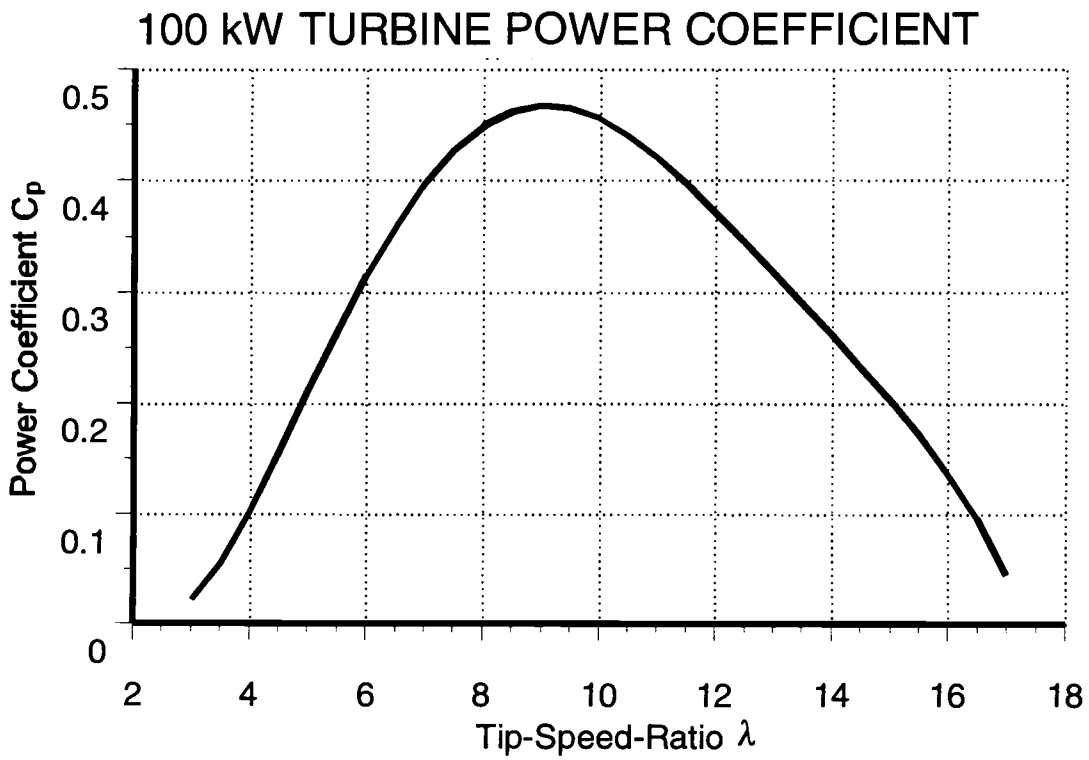


Figure 1.1 Power coefficient of a 100 kW wind turbine.

The torque coefficient, C_T, of the wind turbine is related to C_p as

$$C_T = \frac{Torque}{\frac{1}{2}\rho v^2 A R} = \frac{1}{\lambda} C_p \tag{1.7}$$

1.1.2. Fixed and variable speed generation

It is evident that for optimum energy extraction, the maximum power coefficient and thus the optimum tip-speed-ratio should be maintained at all wind speeds. With a constant speed constant frequency (CSCF) wind turbine system, it may be necessary to

use pitch control of the blades to limit power input to the system [5], thus introducing additional mechanical control systems. Variable-speed generation can track the changes in wind speed by adapting shaft speed and thus maintaining optimal energy generation.

In Fig. 1.2 the rotor power for the wind turbine characteristic of Fig. 1.1 is plotted using variable speed generation (VSG) and fixed speed generation (FSG) strategies. The fixed speed system is optimized at only a single speed, generally representing maximum energy capture potential for a given site. Thus, power output at near cut-in and maximum power are significantly lower than in the case of the variable speed system. The fixed speed fixed pitch system is inherently power limited, as increasing wind speeds lower the turbine power coefficient, a phenomenon referred to as stall regulation. On the other hand, the variable speed system will attempt to track maximum power at any wind speed within the design limitations of the system. Once rated generator power or maximum rotational speed is reached, it becomes necessary to limit the turbine power by abandoning the optimum tip-speed-ratio. During variable speed operation, wind and rotor speeds are related linearly, whereas in the constant power regime, shaft speed drops off sharply and is kept essentially constant thereafter. The speed range for VSG operation is selected such that it only commences when the cut-in wind speed is reached and torque is sufficient to operate. Due to the cubic power-speed relationship, the loss of energy is minimal. The upper cut-out or furling speed is determined by the limits of the turbine and its structural strength.

The improved energy capture for VSG systems is obtained at the expense of additional system components, most often in the form of a power electronic converter. The cost of this additional hardware needs to be low enough to achieve a reasonable rate of amortization given the additional energy generated. In addition to maximizing energy extraction from the wind, the generation system efficiency needs to be as high as economically feasible in order to maximize the energy delivered from generator to the electric power grid.

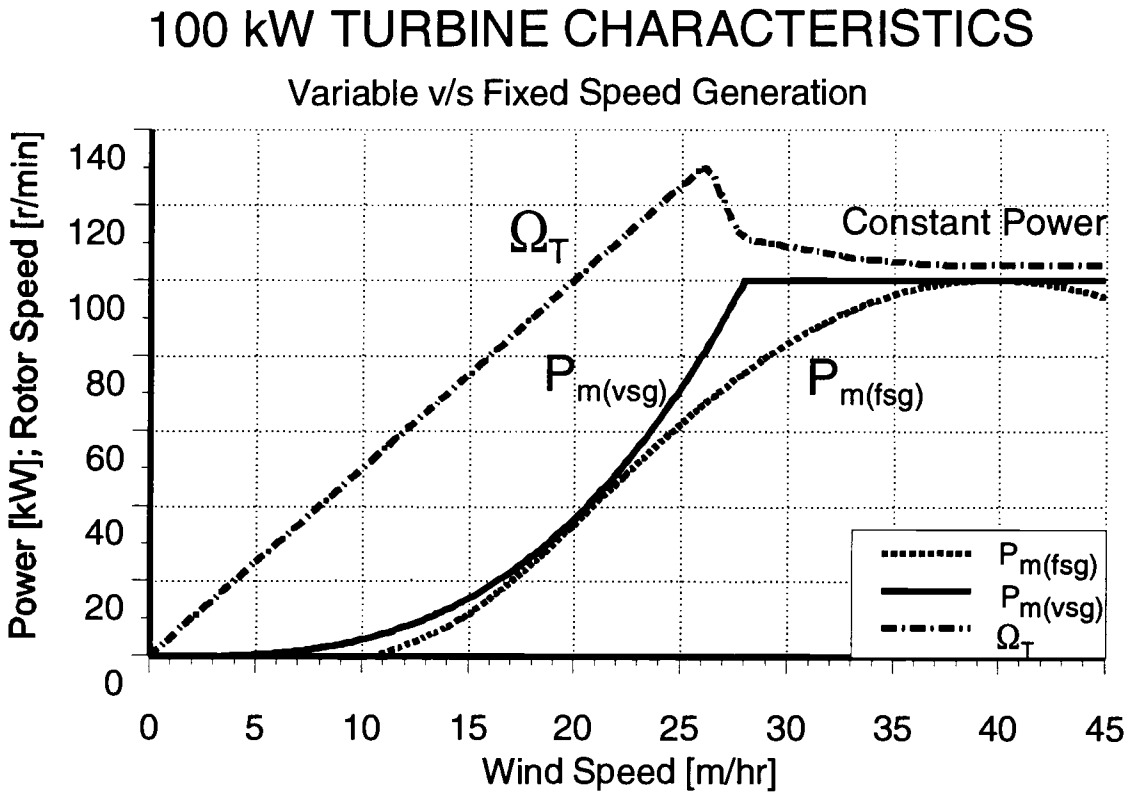


Figure 1.2 Variable and fixed speed power and speed characteristics of a 100 kW turbine.

1.1.3. Wind generation system design considerations

As illustrated in Fig. 1.2, wind turbines operate at a relatively low speed, in general below 100 r/min. Utilizing off-the-shelf electrical machinery with four (1800 r/min) to eight poles (900 r/min) requires a step-up gearbox with a gear ratio of at least 9 to 18, but often higher. This sometimes requires a multi-stage gearbox, with the resultant mechanical complexity and additional losses, which can be in the order of 4-8% of mechanical turbine power [6]. Eliminating the gearbox leads to significant system benefits, but requires generators of very high pole number, e.g. 160 for a turbine speed of 45 r/min. Given a minimum pole pitch, machine diameter for a high number of poles becomes very large, sometimes unacceptably so, given manufacturability and cost constraints and space restrictions in the nacelle at the top of the supporting tower in

HAWT systems. System design needs to optimize the trade-off between using an off-the-shelf, relatively high speed generator in conjunction with a gearbox and utilizing a special purpose, directly coupled generator.

HAWTs produce a non-constant torque due to tower shadow [7]. As each blade passes the supporting tower, the output torque decreases. Thus, the torque produced by a two-blade wind turbine contains a harmonic at twice the rotational speed, the so-called 2P harmonic. Also contributing to torque pulsations is a wind shear effect due to the wind speed gradient along the height of the area swept by the wind. Typically, a fixed-speed system is unable to mitigate this effect in order to improve the quality of the output power. In large turbines, the power pulsations passed to the network can become unacceptable, especially when the penetration level is high. In VSG systems, though, appropriate control of the power electronic converter can minimize torque ripple and thus output power pulsations [8,9]. Here, the inertias of turbine and generator are used to store energy and mitigate torque pulsations. This not only improves the utility interface characteristics, but also damps mechanical stresses on the system, thus allowing for relaxed safety factors, lighter construction and hence, higher reliability and longer useful life.

Some large wind systems are not self-starting or self-stopping. Mechanical tower resonances at low frequencies require that the wind turbine be motored up to operating speed. Fixed-speed systems require the starting of a large induction machine with the resulting expense of soft start mechanisms; stopping a fixed-speed system usually requires a large mechanical braking system. A mechanical brake may also be required in converter fed systems, since the stopping capability needs to be available even in the absence of electrical excitation, such as during fault conditions.

While sharing many operational characteristics with HAWT systems, VAWTs in general are not self starting and require the generator to run as a motor during start up. Certain VAWT systems, such as those of the Darrieus type [8], are not susceptible to

tower shadow, but wind shear is still present and torque pulsations are produced by the continuously changing angle of attack between wind and turbine blades.

Wind generation system design needs to account for the extreme environmental conditions encountered. Depending on the site, temperatures between -30 °C to +45 °C as well as internal condensation must be allowed for. Additionally, the differential temperature of system parts will vary with time and load. The nacelle cover can only provide for some protection against environmental effects, such as rain, snow, ice, particulates and chemical pollution. Appropriate design of not only the mechanical system, but especially the electronics for control and power processing is essential.
The design criteria for wind generation system design are summarized as follows:

- Reliable and low maintenance design optimized for the environmental constraints of a given site;
- Minimum number of components and cost;
- Maximum energy capture and efficiency;
- Integration of mechanical and electrical systems (start-up, shut-down, mechanical stresses);
- Utility supply interaction and contamination.

1.2. Conversion systems for wind power generation

1.2.1. Constant speed generators

CSCF systems use a grid-connected synchronous or induction generator, so that variations in wind speed have to be accommodated by pitch control of the wind turbine itself to limit input power at or below rating. This refers the primary power flow control function to the mechanical side of the system. Synchronous machines which can be utilized include conventional, wound-field machines as well as synchronous reluctance and permanent magnet machines. Conventional synchronous machines have the advantage of providing for reactive power control as well as voltage regulation via the

rotor field winding. Many different power ratings are available off-the-shelf from established manufacturers and machine development costs are minimized.

Permanent magnet (pm) synchronous machines have the advantage of a higher efficiency due to the absence of excitation losses. However, unlike in conventional synchronous generators, output control via the excitation is lost. Consequently, magnet flux and the number of stator turns have to be chosen such that satisfactory operation over the entire load and voltage range is possible while minimizing transient currents during synchronization. Many different permanent magnet designs are feasible [6], including conventional radial field machines, axial flux machines, transverse flux geometries and hybrid claw pole designs. Present day implementations are mainly based on the radial flux concept, utilizing either a buried magnet structure or surface magnets. Interior pm machines can be implemented using high energy, rare earth (SmCo or NeFeBo) magnets or less expensive ferrites. Surface magnet machines in general require rare earth magnets to achieve the desired airgap flux density.

Synchronous reluctance machines also do not provide for excitation control, but avoid the use of magnets and the associated costs. However, rotor designs to achieve the desired reluctance ratios are difficult to manufacture, which partially negates the cost advantage. In the absence of rotor excitation, synchronous reluctance machines require excitation from the connected power grid and have relatively poor displacement power factors. Thus, power factor correction capacitors are often employed.

All synchronous machines provide for a stiff coupling between turbine and the grid. Neither pitch nor stall control can respond fast enough to the torque fluctuations due to tower shadow and wind gust effects. This transmits the torque pulsations directly to the generator and requires an increased torque capability of approximately 50% above nominal. This stiff coupling also subjects the mechanical system to considerable stresses. Subsequently, increase in compliance and damping is required in the power train to

alleviate structural resonances. This discussion also applies to low-slip induction generators.

Many wind turbines utilize low cost, mass produced cage rotor induction machines. In the absence of rotor connections, these systems are also very robust. However, as with synchronous reluctance machines, excitation is required via the stator, which leads to poor power factors and requires power factor correction capacitors at the point of common coupling to the grid. The slip characteristic of induction machines is beneficial for wind turbine control, as it reduces stiffness and provides for additional compliance and damping in the electrical system as compared to synchronous generators.

1.2.2. Variable speed generators

In variable speed constant frequency (VSCF) systems the wind turbine operates at variable speed. If this speed range is made large enough, it is possible to operate a fixed pitch wind turbine and refer the entire power control function to the electrical side. Figure 1.3 reviews variable speed systems based on ac machines and voltage source inverter technology [1]. These represent the desired implementation for medium power (approximately 100 kW - 500 kW) wind systems; higher power turbines may still utilize current-fed, supply commutated topologies. In most cases, a gearbox is used to interface the low speed wind turbine with the high speed generator (not shown in Fig. 1.3).

The system shown in Fig. 1.3(a) can be implemented using conventional, wound-field synchronous machines or the permanent magnet or synchronous reluctance options discussed previously. The system shown in Fig. 1.3(c) utilizes conventional, off-the-shelf, cage rotor induction machines. In both cases, all generated power is processed by a power electronic converter. In the case of reluctance and induction machines, the required reactive power is also provided by the converter, leading to large and expensive converters, typically with a continuous rating in excess of 125% of machine rating. The size of the converter also negatively influences the supply interaction, i.e. the harmonics

injected into the power grid due to converter switching. Despite these disadvantages, many commercial variable speed systems still utilize the cage rotor induction machine due to its simplicity, ruggedness and low cost.

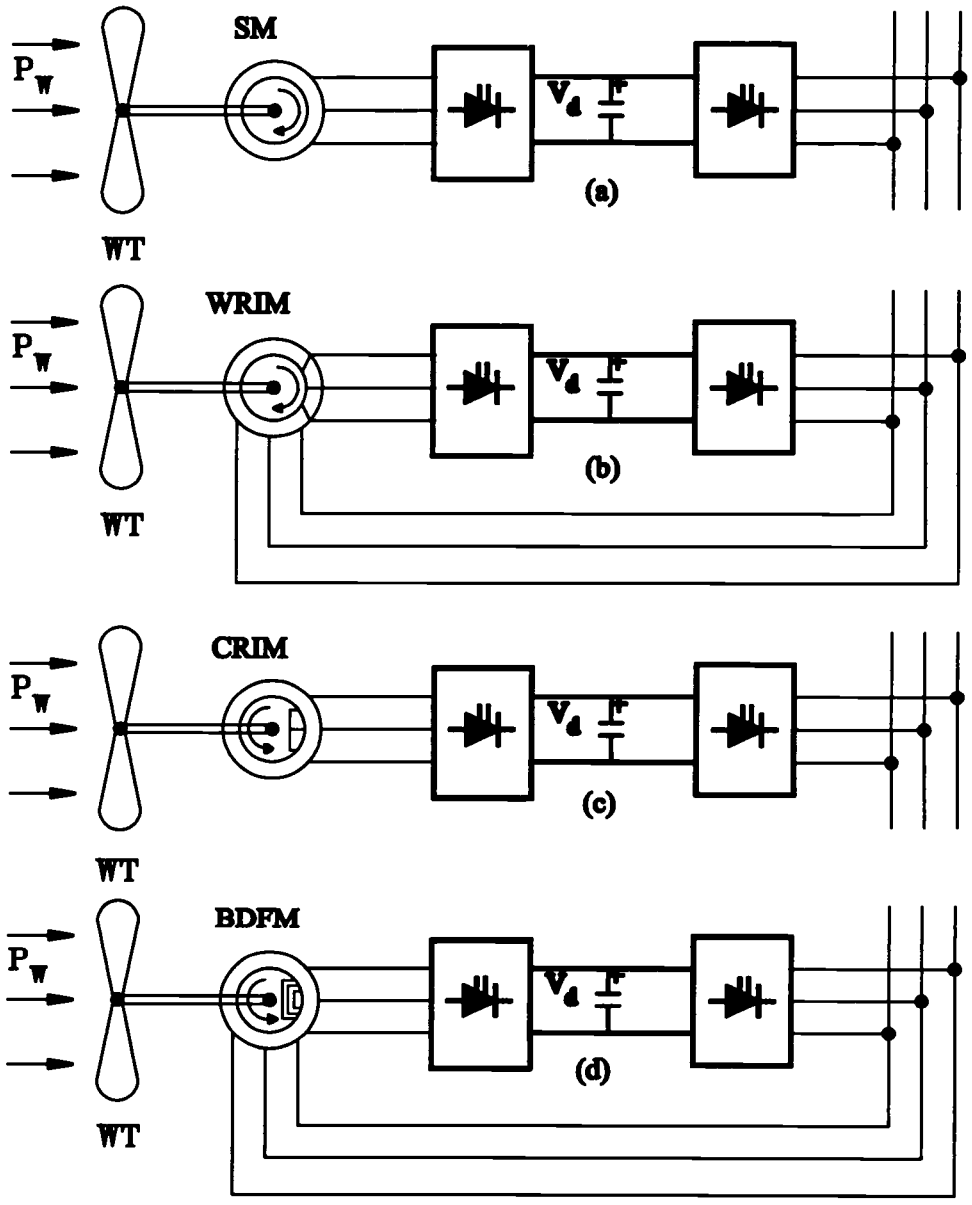


Figure 1.3 Energy conversion systems for variable speed generation: (a) synchronous, (b) wound rotor induction, (c) cage rotor induction and (d) brushless doubly-fed.

Only limited variable speed operation is possible with a conventional induction generator, since otherwise the efficiency becomes proportional to speed. This has led to the investigation of oversynchronous operation of slip ring or doubly-fed induction generators with rotor and stator power fed back to the supply [3,8]. In this case, variable speed operation is possible while only controlling the electrical side. The system shown in Fig. 1.3(b) utilizes a wound-rotor induction machine, the stator of which is directly connected to the utility grid. Only slip power is processed by the rotor power electronic converter, reducing its required rating, size and cost. If this so-called electronic Scherbius system is rated such that maximum power operation corresponds to rated torque at twice synchronous speed, the required rating of the power electronic converter is only 50% of that necessary for the synchronous machine system [3]. This leads to an appreciable saving in capital investment and also reduces the negative impact of converter harmonics on the utility system. From a maintenance and reliability point of view, the use of slip rings is a disadvantage, especially considering that the wind turbine should be capable of working unattended, in adverse conditions, for extended periods.

Recently, doubly-fed machines without slip rings or brushes have been proposed as viable variable speed generators for wind turbines [2,10]. Figure 1.3(d) shows the Brushless Doubly-Fed Machine (BDFM) which is based on the induction principle and combines the advantages of the cage rotor systems (low machine cost, robust brushless machine construction) with the benefits of the electronic Scherbius configuration (reduced power converter rating and cost). The BDFM has two stator windings of different pole number to avoid direct transformer coupling [11]. The power and control stator windings interact through the rotor, which has a specialized cage structure with a number of identical sections corresponding to the sum of the pole pairs of the stator windings. The machine exhibits synchronous behavior where stator frequencies and shaft speed are related by [11] :

$$f_c = f_r(p_p + p_c) - f_p$$

with: f_p – *utility grid frequency (60 Hz);*
f_r – *shaft speed dictated by the variable-speed generation algorithm;*
f_c – *required converter output frequency;*
p_p – *pole pairs of grid connected winding; and*
p_c – *pole pairs of converter controlled winding.*

(1.8)

In this thesis, a VSG controller is proposed which exploits the double stator winding feature to operate the machine in a maximum efficiency mode while tracking the input maximum power. In other words, the excitation of the machine is distributed in its two windings such that it operates "optimally" – in this embodiment, – the maximum efficiency mode. The optimality condition could be determined based on suitable user defined criteria such as maximum power factor mode at the grid, or a weighed product of the efficiency and the power factor at the grid. The philosophy of the controller is also applicable to any VSG or adjustable speed drive (ASD) system that utilizes a DFM as the energy conversion device from mechanical to electrical and vice versa.

In a DFM, only a fraction of the generator power is processed electronically, resulting in reduced size and cost as well as improved power quality. The die-castable rotor cage ensures robust and inexpensive machine construction. An alternative design with very similar characteristics is the doubly-fed reluctance machine [10], which uses an equivalent stator configuration, but replaces the rotor with a reluctance geometry similar to the one found in the synchronous reluctance machine. The dual power flow paths found in doubly-fed machines enables efficiency-optimized controls not possible with singly-fed machines.

The generator systems discussed so far are all based on three-phase technology with distributed stator windings. Another system which has been suggested for wind power applications is the variable reluctance machine, which differs considerably from the synchronous reluctance generator discussed previously [12]. The variable reluctance machine has a doubly salient structure, with distinct stator and rotor poles, much like a

stepper motor. Unlike in conventional ac machines, the stator windings are not distributed, but rather are concentrated on the stator poles. Combined with the simple rotor structure, this leads to ease of manufacturing and potential economic advantages. The converter topology for variable reluctance machines differs significantly from its counterpart for conventional machines. Development work is also under way to introduce magnets into variable reluctance machines in an effort to enhance generator performance [13].

Rather than follow the optimum VSG power curve (see Fig. 1.2) in a continuous fashion using the VSG systems described above, often a two-speed system is implemented to improve upon fixed speed generator performance. This can be realized with pole changing induction machines and the resulting system avoids the use of a power converter completely.

1.2.3. Power converter considerations

Technological advances in gate turn-off, high power semiconductor devices have revolutionized the power electronics applications industry, especially in the medium power range. Extremely fast turn-on and turn-off characteristics are obtained for Insulated Gate Bipolar Transistor (IGBT) modules rated in excess of 2000 A. These devices allow for high switching frequency and control bandwidth with relatively simple drive requirements. Hence, for medium power (500 kW – 1 MW) wind generation applications, hard switched, pulse-width-modulation (PWM) based power interfaces are typically employed. To ameliorate the performance of the switches by reducing the switching stresses and losses and to reduce the unwanted $\frac{dv}{dt}$ and $\frac{di}{dt}$ effects of hard switching, topologies utilizing soft switching techniques can be incorporated in wind generation systems [14].

For wind generation systems rated at above 500 kW – 1 MW, present day technology still favors Silicon Controlled Rectifiers (SCR) and Gate Turn-Off Thyristors (GTO),

which are available in ratings of over 4000 A. However, only slow switching frequencies, typically less than 500 Hz, can be attained employing SCRs and GTOs at the high power levels. Generally, SCR-based current link converters utilizing either load or forced commutation are used for high power synchronous or induction generator based systems. Cycloconverters provide the electronic grid interface for high power, direct drive synchronous generators and limited speed range, high power doubly-fed induction machines.

1.2.4. Direct drive wind turbines

The gearbox used to interface slow speed wind turbines with conventional, relatively high speed electric machines is a source of additional cost and losses. Direct-drive wind turbines require generators with very high pole numbers and, given a minimum pole pitch requirement, this leads to very large generators, with approximately double the outer diameter and weight of high speed machines. Low speed generator efficiency is also lower by a few percentage points and machine cost is higher, due to both the increase in size and the higher cost for these special purpose machines, which do not enjoy the economies of scale of mass manufacturing. Nevertheless, it has been shown [5,14] that the overall direct-drive system can have significant advantages in terms of initial cost, efficiency, reliability and return on investment, when compared to the high speed, geared generators.

Due to the benefits associated with direct drive systems, numerous development programs are under way investigating suitable machine geometries. Proposed machine geometries include variable-reluctance generators [11], surface-mount [14] and buried magnet permanent magnet machines. It is expected that significant progress will be made in this area over the coming decade and that market penetration of direct-drive wind turbines will increase significantly.

1.3. Wind turbine control systems

1.3.1. Fixed speed generator control

The simplest wind-turbine configuration is without pitch control and a synchronous link to the grid, where the turbine speed is constant due to the generator being directly connected to the fixed-frequency grid. While it is possible to limit the power input by stall regulation with this arrangement, it is not possible to control the amount of power delivered by the turbine to the electrical generator within its operating range. It is solely dependent on the wind variations. However, since the available aerodynamic power is a function of both the wind speed, v, and the angular speed of the turbine, Ω_T, the power delivered by the turbine could possibly be only a fraction of the maximum available power. As shown in Fig. 1.1, a fixed speed system is typically optimized at only a single speed of operation. This inherently limits the deliverable power, as increasing wind speeds lower the power coefficient. Although fixed speed wind turbine systems do not employ sophisticated speed control systems, they may still need to be equipped with electrical or mechanical control for power limiting, torque-ripple mitigation and resonance avoidance.

The tower passing effect of the turbine blades causes dynamic speed variations even for fixed speed generation systems, which could affect their performance. The use of tip vanes for dynamic rotational speed control of HAWT has been reported [16]. The tip vane is located at the tip of a turbine blade and functions as a device for power augmentation by varying the power coefficient, C_p. It can also be operated as a rotor speed braking device by varying the sweep angle between the vane orientation and the axis of rotation of the turbine blades. At zero sweep angle, the turbine power coefficient, C_p, is at its maximum. Any change in the sweep angle provides a means for turbine deceleration or acceleration. The tip vane sweep angle controller utilizes turbine speed error and its derivative to determine the absolute value of the sweep angle. In situations

where large wind fluctuations occur, tip vane control in conjunction with blade pitch control can provide for effective fixed speed operation.

The wind turbine and the energy conversion system can be represented by a rigid rotor of inertia J_R with an applied aerodynamic torque of T_A. This torque is transmitted as T_L to a shaft of rigidity K_L and damping C_L as shown in Fig. 1.4. Upon simple analysis, the transfer function *H(s)* between the transmitted torque, T_L and the aerodynamic torque, T_A can be formulated as

$$H(s) = \frac{T_L}{T_A} = \frac{C_L s + K_L}{J_R s^2 + C_L s + K_L} \tag{1.9}$$

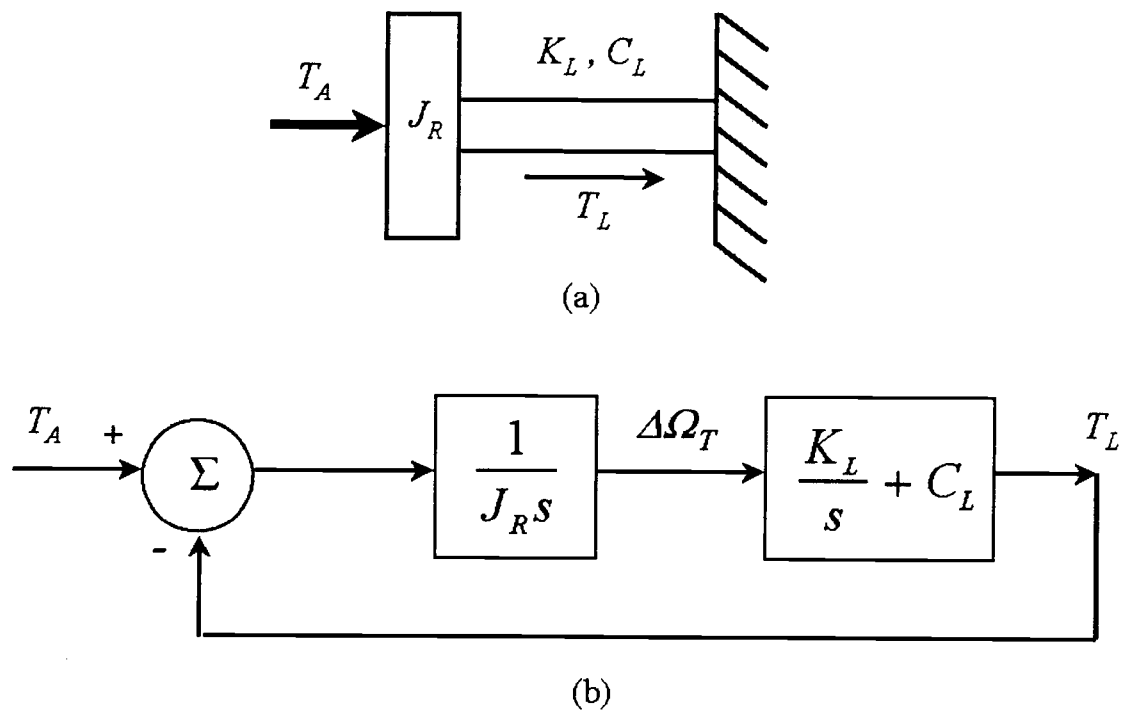


Figure 1.4 Simplified wind turbine (a) mechanical model and (b) block diagram, typically utilized for determination of mechanical system resonances and responses

The natural frequency, ω_N of the denominator of $H(s)$ (i.e. the characteristic equation) can be determined to be

$$\omega_N = \sqrt{\frac{K_L}{J_R}} \tag{1.10}$$

which corresponds to the first torsional mode or the system mode [7]. Developing a more elaborate model of the wind turbine system by representing the wind turbine by two discs of equal inertia, it is observed that there exists a second resonant peak of the transfer function when plotted in the frequency domain [17]. Due to the existence of these two resonant peaks in the transfer function of the wind turbine system, usually both the 2P and the 4P (for a vertical axis turbine) harmonics are minimally attenuated, enabling them to propagate through the drive train and appear as electrical torque ripple. The ripple, if not compensated, affects the quality of electrical power generated and causes torsional fatigue of the components of the drive train.

In an effort to increase the difference between the 2P harmonic of a turbine and the natural frequency, ω_N, various mechanical methods have been devised to reduce the rigidity of the low speed shaft. From eq. (1.10) above, it is readily seen that ω_N decreases as K_L, the rigidity of the low speed shaft, decreases. This has led to the use of a low rigidity shaft, also known as the quill shaft, on the low speed, turbine side of the gear-box. Further separation in the 2P harmonic and the natural frequency ω_N is possible by electrical means as discussed in section 1.2.2.

Torque pulsations and the associated power angle oscillations in slow speed direct coupled generators (small pole pitch permanent magnet synchronous generators) cannot be damped using damper windings on the rotor, as they would incur unacceptable losses as well as increase the physical size of the generator. Once the torque oscillations are passed on to the stator, they can be damped using a passive viscous damper, consisting of a spring and a mechanical damper, which connects the stator of the generator to the wind turbine housing. An adaptive mechanical damping method for wind turbine torque

stabilization [17] uses a bearing-mounted stator, which oscillates about the machine axis against a torsional spring. By selecting an appropriate stiffness, K_S, and inertia, J_S, of the stator damper, the "blank" frequency,

$$f_{blank} = \frac{1}{2\pi}\sqrt{\frac{K_S}{J_S}} \tag{1.11}$$

can be set equal to the 2P frequency. The blank frequency is the frequency at which stator and rotor oscillate as a single mass, thus rendering the system non-responsive to any input. Hence, when the blank frequency equals the 2P frequency, all torque pulsations can be eliminated. However, this results in excessively large stator oscillations and requires slight stator damping, thus sacrificing transient response to improved tower passing response. An adaptive damping system can be based upon the fact that an optimum system allows the stator to oscillate freely at the tower passing frequency, but damps oscillations at any other frequency [17].

1.3.2. Variable speed generator control

In addition to dynamically maximizing the wind turbine efficiency, variable speed can also serve to reduce system stresses such as torque pulsations. The following advantages are beneficial with respect to system reliability and generated power quality:

- Variable speed control can lead to a substantial reduction of the torque ripple in the drive train and hence improved power quality. This is achieved by an increase in compliance, whereby a variation in the aerodynamic power results in system acceleration or deceleration rather than in output power fluctuations.
- The attenuation of torsional mode resonances in variable speed operation allows for a relaxation of design requirements.
- Dynamic loads are reduced and safety margins can be relaxed.

Some electrical systems available for variable speed generation (VSG) operation are shown in Fig. 1.3. As illustrated, both singly and doubly-fed electrical generators have been employed in variable speed wind generation configurations [1].

1.3.2.1. Singly-fed generator control

Speed control of a wind turbine system can be accomplished by controlling either the turbine blade angle (pitch control) or the electrical generator speed if an electronic power converter is available. The compliance of the electrical coupling between the grid and the generator varies based on the type of generator. Typically, the mechanical compliance in the drive train is higher than the electrical compliance. This leads to a lightly damped, low frequency torsional mode [18]. Speed control allows for efficient system operation and also provides for an effective way of damping the low frequency torsional mode.

From Figs. 1.1 and 1.2 it is clear that in order to obtain maximum power from the wind turbine, it is necessary to keep the tip speed ratio, λ, constant over a wide range of wind speeds. This is achieved by a maximum power point tracking (MPPT) system, an essential component of any VSG controller, which tracks the optimum tip speed ratio for large variations of wind speeds. Here, it should be noted that the MPPT speed range needs to avoid areas of mechanical resonance. Wind speed is a difficult quantity to measure and should not be incorporated as a control input for MPPT design. A basic search algorithm involves sweeping the generator speed command over a certain range until a maximum output power is measured [19]. Other algorithms involve the determination of the wind generator model [20,21], employ techniques of system identification [1] or utilize a model independent control method [22].

Unlike in a dc machine, in a cage rotor induction machine the magnetic flux and the electromechanical torque are coupled, i.e. the torque controller affects the flux and vice versa when scalar control methods are employed. Field oriented or vector control of induction machines utilizes real-time mathematical transformations to effectively

decouple the control of flux and torque production, thus duplicating the features inherently available in a dc machine. The availability of sophisticated low-cost microcontrollers and digital signal processors has led to wide spread implementation of field oriented control techniques for variable speed induction generators. Employing field oriented control techniques for effective torque control and fast speed response allows for both maximum power point tracking (MPPT) and the compensation of torque pulsations [9]. Reactive power control is also possible by appropriately controlling the power electronic grid interface.

Traditionally, the MPPT controller is based upon a search algorithm [3, 19] where the rotational speed of the electrical generator shaft is varied over a range determined by the design of the installation site. While the strategy is extremely simple, it is slow in responding to frequent changes in the wind speed. Hence, the VSG may not be capable of operating at the MPP all throughout the possible wind speed variations. Hence, the benefits of the VSG system may not be exploited to the maximum extent despite the increased investment incurred due to the power converter.

In this thesis, an wind speed estimation based MPPT algorithm is proposed that employs principles of the generator characterization. The optimum tip-speed ratio is maintained by determining the wind speed iteratively from the total generated power and the $C_p - \lambda$ profile of the turbine in use. The proposed MPPT algorithm, though more complicated than the simple search based method, can still be implemented in a relatively low-end floating-point microprocessor. The ease of implementation guarantees fast control updates and ensures maximum power point operation for all wind speeds within the design limitations. While the controller has been developed based on a BDFM, the principles can be extended to other DFMs and singly-fed electrical machines.

1.3.2.2. Doubly-fed generator control

Utilizing a PWM converter for slip power recovery, a doubly-fed wound rotor induction machine can be operated as a variable speed generator [21]. The power converter rating and, hence, the initial cost is reduced as the converter handles only the slip power. Control of torque and thus active power can be achieved by controlling the rotor current component orthogonal to the stator flux; stator reactive power is controlled by the in-phase rotor current. The decoupling of stator active and reactive power flows, allows for an additional control function, which can be used to maintain optimal stator flux and minimize copper losses [21].

The modeling of wind turbine systems is an extremely difficult task due to the uncertainties in the modeling of the aerodynamics, such as the effects of tower shadow, wind shear, blade surface smoothness, wind turbulence and others. This suggests the use of approaches which do not rely on the physical model, as outlined in [1] for an efficiency maximization algorithm for doubly-fed generators. The doubly-fed machine provides an additional degree of control freedom, not present in singly-fed system, whereby stable operation is possible for a variety of control winding current levels at a particular operating speed [1,22]. The overall optimization problem for the mechanical (turbine) and electrical (generator) systems involves finding the maximum of power output as a function of both speed and control current, a three-dimensional optimization problem [1].

In order to keep the controller simple, mechanically robust and inexpensive, no mechanical inputs such as shaft torque should be required for the control algorithm. In addition to the power and efficiency maximizing function, the controller can also perform reactive power as well as harmonic compensation. Based on output power measurement, the controller sets converter frequency and current magnitude. Hence, no mechanical feedback signals are required. However, the compensation of torque pulsations requires an inner speed loop, which is based on field oriented control and requires rotor position feedback [24].

The proposed system optimization controller utilizes the characterization of the DFM in a certain "optimum" mode of operation, based on a user defined criterion of optimality. To create a knowledge base for the system with respect to the "optimum" operation, a set of open-loop experiments are conducted on the system to determine the output power while maintaining that "optimum" condition, such as maximum efficiency or maximum power factor operation at the grid. This information can also be constructed from initial acceptance testing of the system which is typically conducted before actual system installation. As determined in this embodiment, the control variable is the control winding current of the BDFM. Applying duality principles of electrical networks, the control winding voltage could also be employed as the control variable that ensures "optimal" operation of the system.

The optimality characterization thus obtained is then utilized for the overall system operation. If the system can be operated "optimally", the DFM characteristics obtained are employed by the wind speed estimation based MPPT algorithm discussed in section 1.3.2.1.

1.3.3. Compensation of torque pulsations

Torque pulsations are a nuisance problem even in VSG systems. They corrupt power quality as explained in the section on fixed speed generator control. The use of a power converter in variable speed generation enhances the control options for mitigation of torque oscillations. The increase in the available power due to a tower passing spike can be stored either

- in the inertia of the system by speeding up the generator utilizing field oriented control (7) ; or
- in the energy storage link, capacitors in voltage source inverters (VSI) or inductors in current source inverters (CSI).

The second option is impractical due to the significant increase in the required capacitance or inductance of the converter. Hence, the first option is the only practical solution and is commonly implemented in wind generation systems. The torque pulsation compensation is usually a fast inner control loop, with the MPPT algorithm implemented in a much slower and overall system control loop.

An older electrical approach for torque pulsation mitigation is to reduce the stiffness of the electric coupling with the help of a current source power electronic converter between the synchronous generator and the grid [7]. Employing a proportional-integral dc-link current regulator and operating the synchronous generator with constant air gap flux yields the following relationship between electromagnetic torque and generator speed:

$$\frac{\Delta T_E}{\Delta \omega_E} = K_P + \frac{K_I}{s} \tag{1.12}$$

Equation (1.12) above corresponds to the model of an electric generator with electrical stiffness, K_I, and electric damping, K_P. Hence, the PI dc-link current regulator can be adjusted for the parameters of the electromagnetic coupling to the electric grid. Also, the integral constant K_I can be tuned to fix the first torsional mode at any desired low value of frequency.

1.4. Wind turbine operation

The amount of power generated by a wind turbine depends on the design characteristics of turbine and generator as well as on the properties of the wind resource. As an example, Fig. 1.5 shows the annual wind speed distribution for a wind site on the Oregon coast [1]. This variation of wind speed, v, can be represented by a Weibull or Rayleigh probability distribution [25] :

$$f(v) = \frac{k}{c}\left(\frac{v}{c}\right)^{k-1} \exp\left[-\left(\frac{v}{c}\right)^{k}\right] \tag{1.13}$$

with: c – *scale factor*;

k – *shape factor*;

The turbine power output characteristic as shown in Fig. 1.2 can also be expressed as a function of wind speed, v:

$$P_m = P_{\max}\, g(v) \tag{1.14}$$

where $g(v)$ will be different for fixed and variable speed generators. The average power output of the turbine can now be determined by

$$P_{avg} = P_{\max} \int f(v)\, g(v)\, dv \tag{1.15}$$

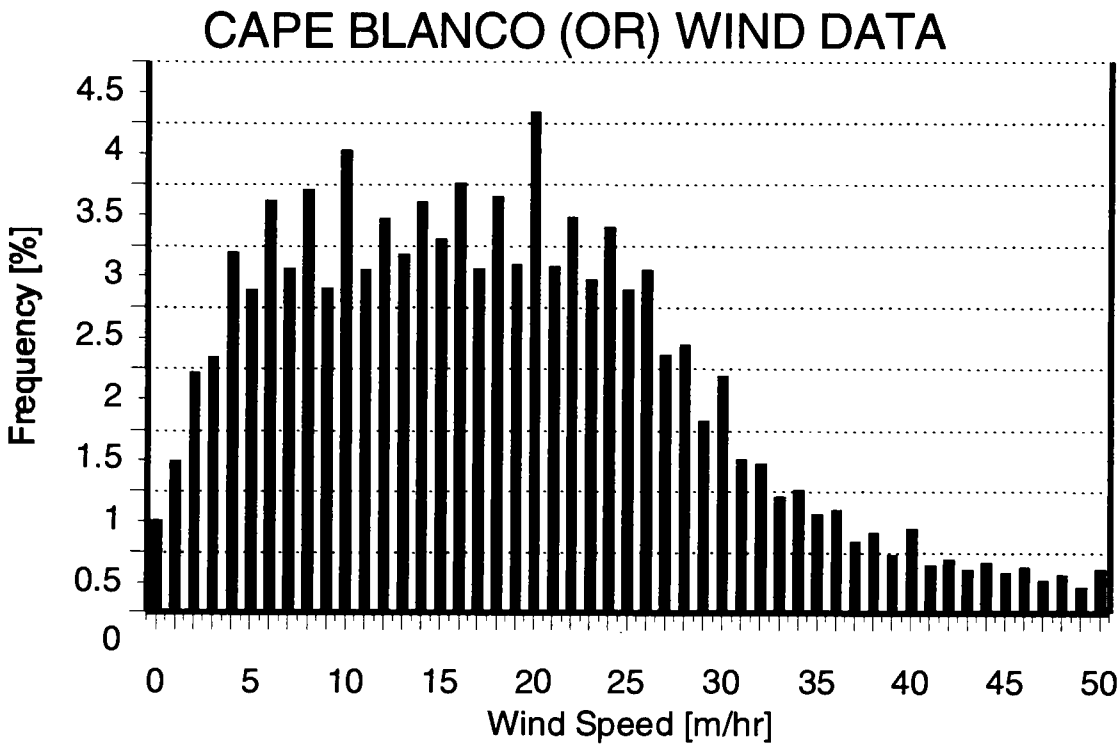


Figure 1.5 Sample annual wind speed distribution, illustrating the Rayleigh distribution of wind speeds.

The quantity inside the integral is defined as the wind turbine capacity factor and represents the ratio of average to maximum turbine outputs. Typical values are

significantly below unity, in the range of 0.3 to 0.4, reflecting the variable nature of the wind resource.

Figure 1.6 [2] shows the result of the integration in eq. (15) and illustrates the annual wind power availability and power extraction possible with CSCF and VSCF systems. The area shown in white represents the additional energy captured using VSCF generation systems and has to be weighed against the added complexity and cost of variable speed generators.

Wind power is a very competitive industry and plants are required to be competitive with fossil resources and yield a satisfactory return on investment. Issues of initial cost, generator sizing and efficiency as well as maintenance and reliability have to be carefully weighed in order to achieve satisfactory $/kWh results from a wind power plant.

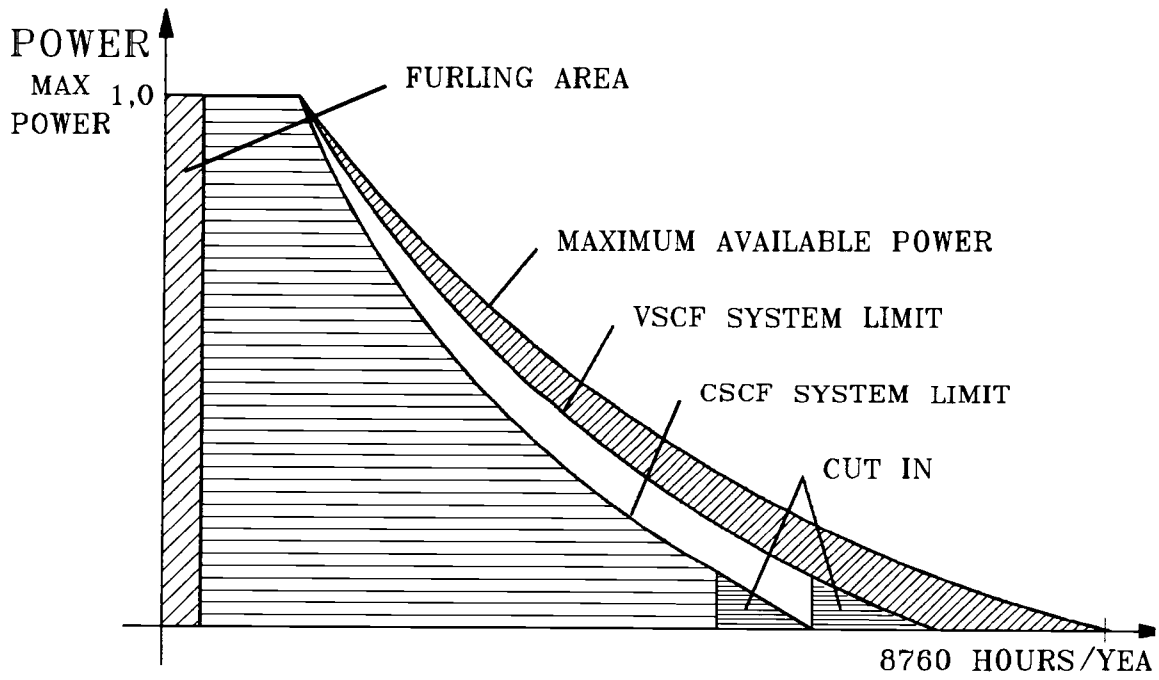


Figure 1.6 Power extraction from wind using VSCF and CSCF systems

1.5. Thesis Outline

Given the advantages of variable speed wind generation as explained in section 1.2.2, it could be made economically viable by reducing the rating of the power converter. The BDFM combines the benefits of the ruggedness of a squirrel cage induction machine with the reduced rating requirements of the power converter of a wound-rotor induction machine [26]. Furthermore, the availability of the extra control variable (the control winding current) can be utilized for additional performance enhancement of the generation system.

As described in Chapter 2, an efficiency maximization controller was developed for a BDFM based VSG application. This controller can be readily modified to accommodate the optimization of a user defined performance criteria, e.g. power factor or a weighted product of power factor and efficiency. The system controller, which includes the MPPT control loop, utilizes the efficiency maximization to track the optimum tip speed ratio under varying wind speed conditions. Controller development and algorithm are presented in Chapter 2, and implementation issues are detailed in Chapter 4.

The BDFM requires a four quadrant power converter for interfacing the control winding to the electrical grid. The output stage of the power converter is required to provide electrical excitation of a certain magnitude and frequency to the control winding of the BDFM, as determined by the user in the open-loop mode or, in the case of the closed loop implementation, the optimization controller. Hence, in the process of the implementation of the system controller for proof-of-concept laboratory verification, power converter control algorithms were developed. These include a simple current controller for the inverter and two control algorithms, sensor-based and sensorless versions, for the rectifier stage. The converter controller development and associated issues are detailed in Chapter 3. There, the switching algorithms for the inverter and the rectifier stages are discussed in detail. Theoretical issues of the sensorless controller development based upon the sensor-based version are presented Implementational details of the power converter utilized for the laboratory VSG system are presented in Chapter 4.

The different controllers were implemented employing Intel 80C196KC microcontroller based systems and detailed software code can be found in Appendix A. Representative waveforms of the converter during experimental evaluation are presented.

A 115 V, 1.5 kW prototype BDFM was utilized for the development of the system controller consisting of the MPPT and the efficiency maximization controller. The details of the experimental BDFM and dc-machine based wind turbine emulator development are presented in Chapter 4. The implementation of the system controller in an Intel 80486 based computer supplemented by a TMS320C3X DSP for power measurement analysis is also detailed in Chapter 4. Further software details regarding the controller and its development can be found in Appendix B. Software code utilized for the emulator development is presented in Appendix C. The data acquisition software developed for measurement and analysis of the different electrical quantities is listed in Appendix D.

Steady state performance evaluation of the closed loop system is described in Chapter 5. Although only efficiency maximization has been evaluated, performance optimization based on any other user defined criteria can be readily accommodated within the same control philosophy. Chapter 6 refers to issues that need further attention in the future and possible enhancement of the system controller presented in this thesis.

2. System and Controller Design

In the area of wind power generation and other power generation systems where the input resource power varies considerably, the benefits of adjustable speed generation over fixed speed systems have been known for some time [1-5,9]. In these systems, a maximum power point tracker adaptively adjusts a system quantity (such as speed in the case of wind turbines) to maximize turbine power output. Using singly fed systems, such as induction, synchronous or reluctance drives, the maximum power point tracking sets the operating point of the electrical machine and thus the losses of the electrical generator associated with that operating point. As the power flow path is fixed (turbine to motor to converter to grid) at a given speed and magnetic flux level of the generator, losses and thus conversion efficiency are fixed. It should be noted here, that in some singly-fed implementations efficiency of the system can be improved by adjusting the magnetizing component of the machine. In doubly fed systems, however, at a fixed operating point (power and speed), power flow can be regulated between both winding systems on the machine. This feature can be utilized to essentially minimize losses associated with the given operating point or achieve other desired performance enhancements. This capability is directly related to the fact that a doubly fed system requires one degree of freedom (frequency) to establish the maximum turbine power point, leaving one degree of freedom (current magnitude) for other control laws, such as efficiency maximization. The strategy is applicable to all doubly fed configurations, including conventional wound rotor induction machines, Scherbius cascades, brushless doubly-fed machines, doubly-fed reluctance machines.

The proposed strategy has been experimentally verified in controlled laboratory conditions for a proof-of-concept brushless doubly-fed machine system of 1.5 kW power rating. Verifying the controller concepts for other forms of doubly-fed machines is relatively simple and involves, mainly, machine characterization and determination of controller parameters. The controller described herein discusses the use of an adaptive Maximum Power Point Tracking (MPPT) strategy to implement an efficiency

maximization loop in parallel with the regular maximum tip-speed-ratio tracker, without the measurement of mechanical quantities. While the presented approach uses a simple model-based approach, this should only be taken as one embodiment; i.e. other adaptive control strategies are feasible as well. The system described here increases the overall power output of the generation system with a minimal increase in controller size and cost.

2.1. Control of DFM wind generator

From Figs. 1.1 and 1.2 it is clear that in order to obtain maximum power from the wind turbine it is necessary to keep the tip-speed-ratio, λ, constant over a wide range of wind speeds. This is achieved by a maximum power point tracking (MPPT) system which tracks the optimum tip speed ratio for large variations of wind speeds [1,3,27]. As wind speed varies, this involves appropriate control [19] of generator shaft speed to ensure that the maximum turbine power is followed, as illustrated in Fig. 1.2.

In the doubly-fed machine, the mechanical MPPT as discussed above, varies the converter output frequency and thus the rotational speed according to eq. (1.8). At the established shaft speed, stable operation is possible for a variety of control winding current levels, thus allowing for an additional degree of control freedom not present in singly-fed systems. For a given mechanical input power, this can be used to regulate the power flow in the stator windings and power converter and thus maximize efficiency and output power. This has been verified experimentally in section 2.2.1.2 for a 7.5 kW prototype BDFM and in Chapter 4 for the 1.5 kW BDFM utilized for the closed-loop verification. Thus, the overall optimization problem for the mechanical (turbine) and electrical (generator) systems involves finding the maximum of power output as a function of both speed and control current, as illustrated conceptually in Fig. 2.1.

In order to keep the controller simple, mechanically robust and inexpensive, no mechanical inputs such as shaft torque or speed should be required for the control algorithm. This has lead to the development of a system controller as illustrated in Fig. 2.2 which relies only on electrical feedback parameters of real power (for MPPT). The

concept can be extended to include reactive power feedback for optional power factor control. Chapter 6 describes the fundamentals of a reactive power controller based on the optimization controller proposed in the thesis for future enhancements to the original algorithm.

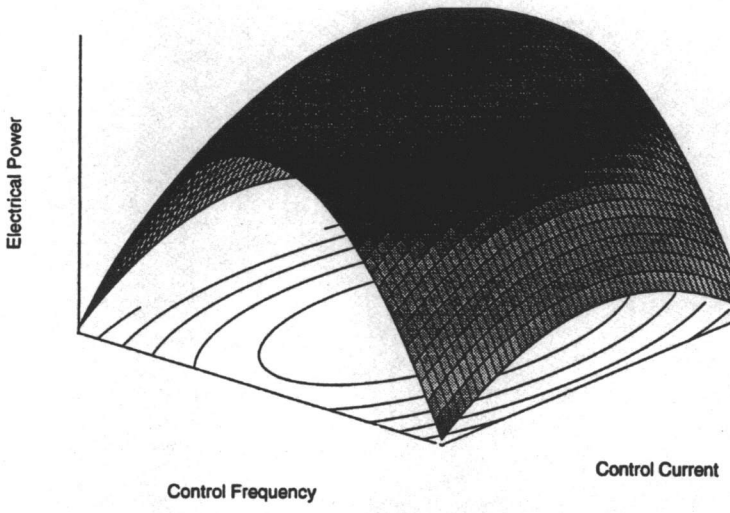


Figure 2.1 Conceptual representation of the VSG optimization controller as a three-dimensional problem.

In addition to the power and efficiency maximizing function, the controller can potentially also perform reactive power as well as harmonic compensation. It is also not necessary for the control to be implemented exclusively in a generation system (power flow from left to right in Fig. 2.2), but it is equally applicable for motoring or drive systems (power flow from right to left in Fig. 2.2).

Figure 2.2 illustrates the control algorithm in block diagram form. Based on output power measurement, the controller sets converter frequency and current magnitude. While currents for optimum efficiency and thus maximum output power could be calculated based on machine parameters, this should be avoided due to the variations of internal parameters such as rotor time constant. Since mechanical power is also not available, an adaptive approach is proposed to aid in the determination of the maximum power point as illustrated in Fig. 2.1. The adaptive controller discussed here is based on a system model as described in section 2.2. This is, however, only one simple implementational strategy. Other possible implementations could involve, any one or

combination of the following: 1) neural networks, 2) the use of other forms of adaptive or intelligent control, 3) the use of mechanical feedbacks, or 4) the determination and use of system parameters.

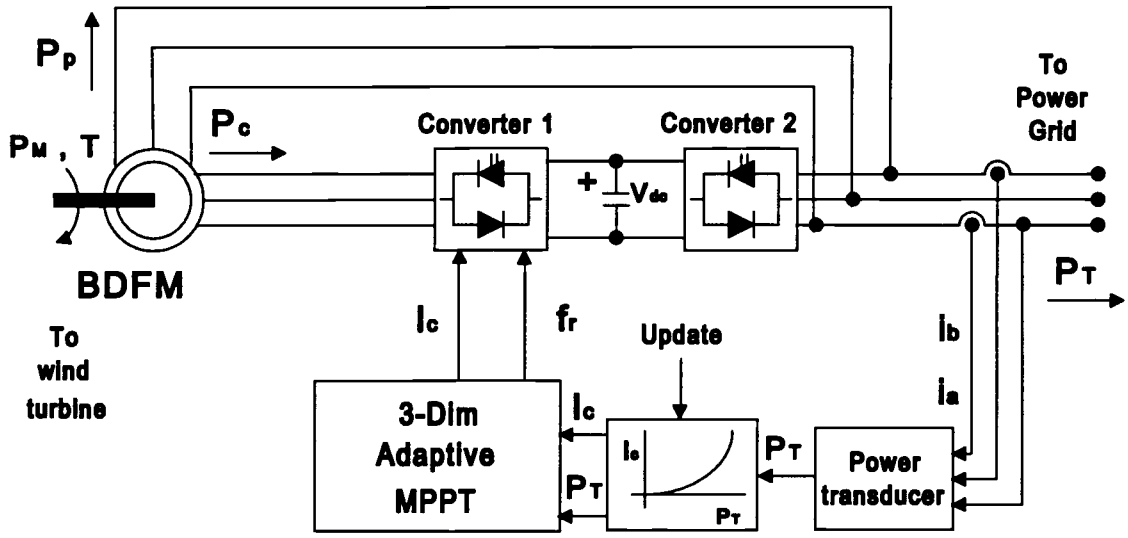


Figure 2.2 Block diagram representation of the controller in a wind generation application.

2.2. Optimization control algorithm

Figure 2.3 shows a general representation of the output power maximization algorithm for a doubly-fed variable speed wind generation system. It should be emphasized that there could be a multitude of implementation methods for the individual blocks of the flowchart in Fig. 2.3. While implementation could possibly vary considerably, the basic optimization algorithm would remain the same.

The algorithm begins by initializing, in 60, the variables, P_T, which represents the total measured power of the generator system, and f_r, the mechanical or the shaft speed of the DFM-turbine system. Typically, upon system startup, these variables would be initialized non-optimally, and solely based upon the wind speed and the generated power thereof at the time of initialization. Measurement of the total output power P_T is utilized

to determine the optimal control winding current setting, in 62.

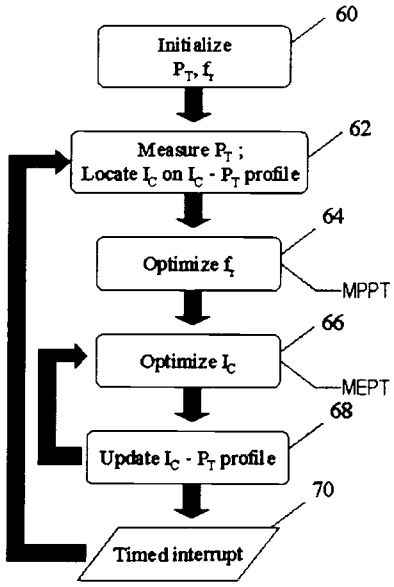


Figure 2.3 Generalized flowchart representation of the proposed optimization algorithm

The I_c-P_T profile is a simple model representation of the DFM system with respect to the performance optimization criteria chosen for the implementation. In this thesis, as implemented in the laboratory verification of the system controller, generation of maximum power at the output, i.e. at the grid, is the optimization criterion. Hence, to satisfy the control requirements, besides tracking the maximum power point (MPP) of the resource input, the BDFM generator was operated in the maximum efficiency mode. Other possible criteria could be the maximization of the power factor at the point of common coupling to the grid or a combinatorial performance index consisting of the the BDFM efficiency and the system power factor.

Once the performance criterion is chosen for an implementation, characterization of the DFM is done off-line or in the open-loop mode. The control variable, in the proposed strategy the control winding current, I_c, is determined as a function of the feedback quantity, which in this case is the measured generated power. The characterization method is elaborated in Chapter 4.

To ensure operation at the maximum power point of the wind turbine, Fig. 1.1, the mechanical speed, f_r, is iteratively determined in 64 by the MPPT controller. While attaining the maximum power point, the operation of the system is further optimized, by adjusting the control winding current, I_c, by the Maximum Efficiency Point Tracker (MEPT) controller in 66. The new optimized operation point, if different from the stored information, as determined by I_c and P_T, is used to update the I_c-P_T profile in 68. The algorithm repeats with a new measured output power P_T after a fixed delay, in 70, timed by an interrupt in the controller.

2.3. Flowchart implementation

As mentioned above, the individual control blocks of Fig. 2.3, namely MPPT and MEPT could have various implementation and control methods. Two possible implementations of the MPPT block are presented here in detail; one, Fig 2.4, utilizes a simple but slow search-based algorithm while the other, Fig. 2.7, applies model-based control techniques to estimate the wind speed. Knowledge of the wind speed helps in determination of the desired rotational speed for the turbine in operation. While the search-based algorithm is presented here merely as a reference, the wind speed estimated approach will be referred to in detail in the following sections.

2.3.1. Search-based controller

Figure 2.4 details a simplified realization of the basic algorithm as shown in Fig. 2.3. As mentioned before, the mechanical speed, f_r, and the total output power P_T are initialized, in 101 and 102 respectively, dependent upon the startup conditions. Based upon the measured output power, P_T, the control winding of the doubly-fed machine (DFM) is excited with a current magnitude as determined from the existing I_c-P_T profile as shown in 103. As the DFM control winding excitation is varied, the output power P_T could possibly vary too. If the new measured value of output power $P_{T(1)}$ is greater than that of the initial value, then $P_{T(1)}$ is retained as is shown in 104 and 105. Determination of the initial optimum control winding current initiates the f_r optimization algorithm

detailed in 106-113. In 106, the initialized mechanical speed, f_r, is incremented by a predetermined quantity, Δf_r. This change in the mechanical speed possibly varies the output power, P_T, and thus the output power, $P_{T(2)}$, is measured in 107. The new output power, if determined, in 109, to be greater than the previous P_T, is saved, in 108, as the latest operating point for maximum output power. The process of incrementing the mechanical speed f_r is repeated until the latest measured output power $P_{T(2)}$ is determined to be lower than that of the previous iteration. That leads to the part of the algorithm where the mechanical speed is decremented by a predetermined quantity, Δf_r, as shown in 110-113. The basic operation of the algorithm in the speed decremental mode is similar to that in the speed incremental mode as detailed above.

Once the maximum power point is established by varying the mechanical speed, the system is further optimized by fine tuning the optimal control winding current setting, in 114-122. The process of varying the control winding current magnitude, I_c, is very similar to that of f_r as explained above. In 114-118, the control winding current is incremented till the maximum power point is attained, while in 119-122 the current is decremented to obtain the maximum power operating point. The new operating point, in the I_c - P_T domain, obtained as detailed above, is updated in the profile, in 123. The update is required due to the continuously varying system parameters caused by changes in temperature, humidity, magnetic properties of the material of the machine, wear and tear, etc. As in Fig. 2.3, the algorithm repeats with a new measured output power P_T after a fixed delay, in 124, timed by an interrupt in the controller.

As discussed above, the maximum power and efficiency tracking algorithm presented uses a perturbation algorithm, which incrementally changes excitation current frequency and magnitude to obtain maximum overall output power (thus maximizing energy extraction from the wind and minimizing electrical losses in the generator/converter system). The following discusses an alternate method for the wind turbine maximum power point tracking.

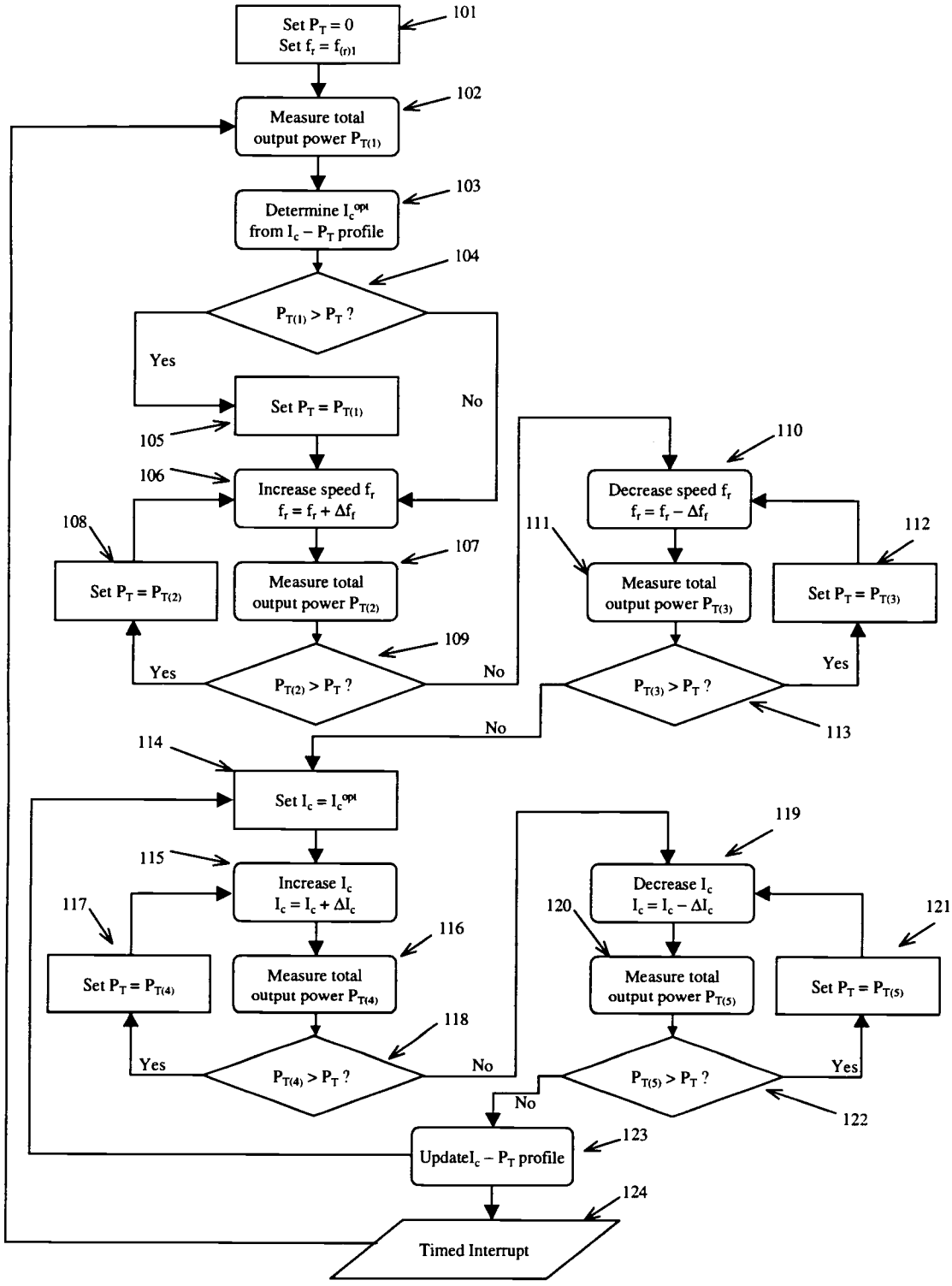


Figure 2.4 Perturbation based search algorithm for maximum power point tracking.

2.3.2. Wind speed estimation based controller

This section describes an alternative for the maximum power point tracking (MPPT) algorithm. The algorithm discussed is a possible realization of block 64 in Fig. 2.3. In Fig. 2.4, the algorithm would replace blocks 106-113. The new algorithm enables a faster determination of the maximum power point (MPP) as compared to the search based varying shaft speed approach outlined above in section 2.2.1.1.

The main control blocks of this novel algorithm are as shown in Fig. 2.2. The total output power, P_T, is the only sensed quantity (electrical or mechanical) for the MPPT control loop. A typical I_c vs P_T profile for the block of Fig. 2.2 is illustrated in Fig. 2.5. It should be noted here that Figs. 2.5 and 2.6 were obtained while characterizing a 230 V, 7.5 kW prototype BDFM. Similar characterizations, presented in Chapter 4, were conducted on the 115 V, 1.5 kW BDFM utilized in the closed-loop VSG system development.

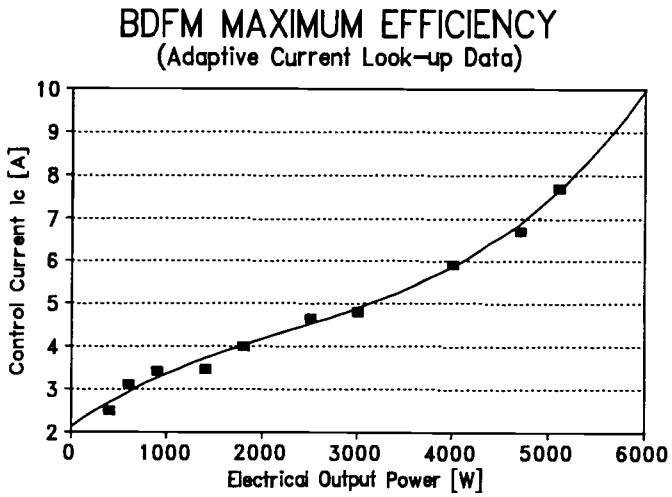


Figure 2.5 Optimal control winding current requirements to ensure maximum efficiency for a 7.5kW BDFM.

The optimum control winding current I_c with respect to the total output power P_T as characterized in Fig. 2.5 is obtained from maximum efficiency measurements as shown in Fig. 2.6. Thus, it is conceivable that information regarding the maximum efficiency of the DFM with respect to the optimal control winding current, I_c^{opt} can also be determined easily from Fig. 2.6. However, this stored information should be updated similar to that of the I_c - P_T curve.

This algorithm is not entirely based on a perturbation (search) approach as the one described in section 2.2.1.1. (blocks 106 to 113 in Fig. 2.4), but uses perturbation for fine adjustments.

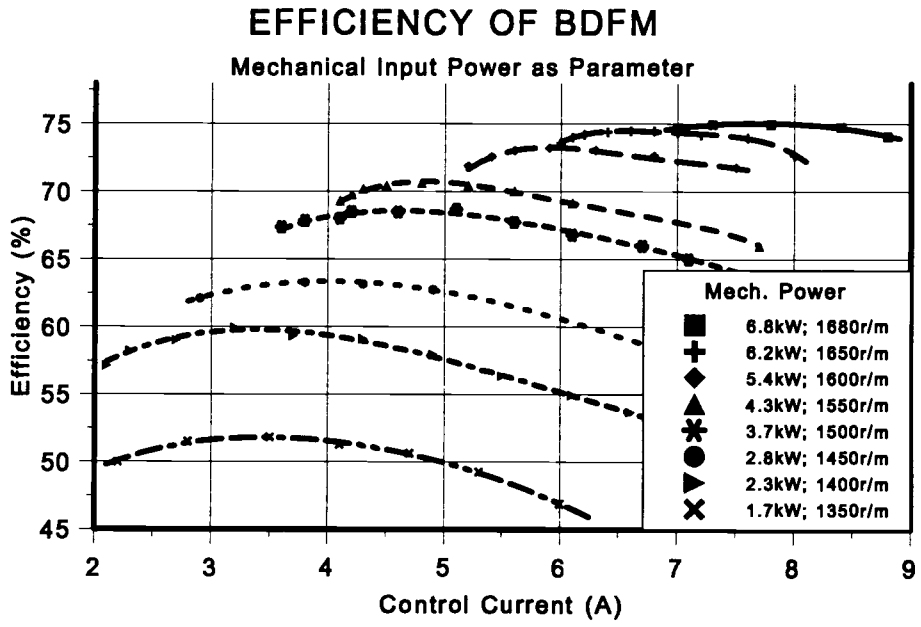


Figure 2.6 Maximum efficiency point characterization of a 230 V, 7.5 kW prototype BDFM with input mechanical power as a parameter.

The following describes the wind speed estimation based algorithm. The simplified flowchart of this controller is illustrated in Fig. 2.7:

1. The total output power P_T of the wind turbine system is measured as indicated in Fig. 2.2.

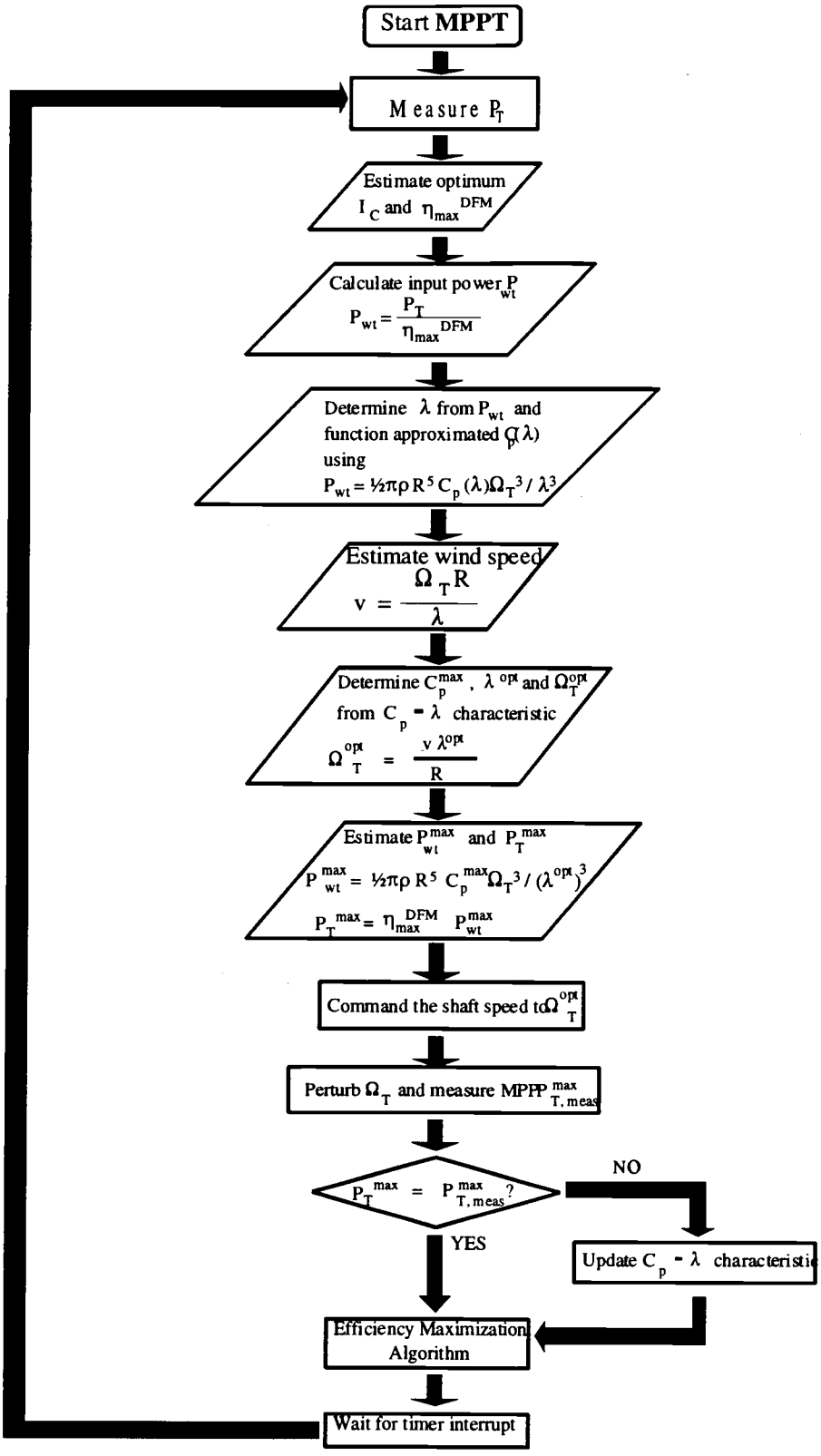


Figure 2.7 Maximum power point tracking based on estimated wind speed.

2. Following the measured power, the optimum control winding current, I_c^{opt}, is commanded based upon information as stored in the $I_c - P_T$ profile similar to as depicted in Fig. 2.5.

3. The maximum DFM efficiency η_{max} is estimated at a particular control current optimized operating point utilizing a stored η -vs- I_c^{opt} characteristic of the generator.

4. DFM input power, P_{wt}, is calculated from the estimated maximum efficiency η_{max} and measured output power P_T as

$$P_{wt} = \frac{P_T}{\eta_{max}} \tag{2.1}$$

5. Based upon information of the shaft speed (from eq. (1.8)) and P_{wt} (which is same as power output of the turbine) the wind speed is estimated employing the following procedure.

   The characteristics of the power coefficient of a wind turbine are normally expressed in terms of the tip-speed-ratio λ, which is defined as:

$$\lambda = \frac{v_p}{v} = \frac{\Omega_T . R}{v}$$
$$\textit{with}: v_p - \textit{tip speed of turbine blade;} \tag{2.2}$$
$$R \quad - \textit{turbine rotor radius;}$$
$$\Omega_T \quad - \textit{rotational angular velocity;}$$
$$v \quad - \textit{wind speed.}$$

   In Fig. 1.1 the power coefficient C_p is plotted as a function of the tip-speed-ratio, λ. C_p depends on the particulars of blade design and can be represented as a function in λ, such as an n^{th} order polynomial:

$$C_P(\lambda) = C_{P0} + C_{P1}\lambda + C_{P2}\lambda^2 + \cdots + C_{Pn}\lambda^n \tag{2.3}$$

Power output of the wind turbine is related to the cube of the upstream wind velocity and can be expressed as

$$P_{wt} = \frac{1}{2}\pi \rho C_p(\lambda) R^2 v^3 \qquad \textbf{(2.4)}$$

where ρ is the specific mass of air; and $C_p(\lambda)$ is the coefficient of power and as described by eq. (2.3)

Substituting for the wind speed, v, from eq.(2.2) in eq.(2.4), the power delivered by the turbine can be expressed in terms of the angular velocity, Ω_T, and the tip-speed ratio, λ, as

$$P_{wt} = \frac{1}{2}\pi \rho C_p(\lambda) R^5 \frac{\Omega_T^3}{\lambda^3} \qquad \textbf{(2.5)}$$

Utilizing an iterative method for determination of the roots of a polynomial, such as Newton-Raphson or bisection method, the roots of the eq. (2.5), λ, can be determined.

Upon further expansion of eq (2.5),

$$F(\lambda) = P_{wt} - \frac{1}{2}\pi \rho R^5 \Omega_T^3 \left[C_{P0}\lambda^{-3} + C_{P1}\lambda^{-2} + C_{P2}\lambda^{-1} + \ldots + C_{Pn}\lambda^{n-3}\right] = 0 \qquad \textbf{(2.6)}$$

and

$$\frac{\partial F(\lambda)}{\partial \lambda} = -\frac{1}{2}\pi \rho R^5 \Omega_T^3 \left[-3C_{P0}\lambda^{-4} - 2C_{P1}\lambda^{-3} - C_{P2}\lambda^{-2} + \cdots + (n-3)C_{Pn}\lambda^{n-4}\right] \qquad \textbf{(2.7)}$$

An iterative method is then used such that

$$\lambda^{(i)} = \lambda^{(i-1)} - \Delta\lambda^{(i-1)} \qquad \textbf{(2.8)}$$

where

$$\Delta\lambda^{(i)} = \left[\frac{\partial F^{(i)}(\lambda)}{\partial \lambda^{(i)}}\right]^{-1} F^{(i)}(\lambda) \qquad \textbf{(2.9)}$$

and the superscripts *(i), (i-1)* represents the i^{th} and $(i-1)^{th}$ iterations.

Only the root that satisfies the range of λ as defined by the C_P - λ curve (Fig. 1.1) is valid and retained for wind speed estimation. Substituting for λ in eq.(2.2), the wind speed, *v,* is estimated. Again, utilizing eq.(2.2) with the estimated wind speed, v, and the optimal tip-speed ratio, λ^{opt}, the desired angular velocity of the turbine is determined as

$$\Omega_T^{opt} = \frac{v\,\lambda^{opt}}{R} \qquad \textbf{(2.10)}$$

6. Using eq.(2.5) and substituting C_P^{max} and λ^{opt} as determined in step 5, the optimal turbine output power P_{wt}^{max} is estimated.

7. The estimated maximum output power of the electrical generator, P_T^{max}, is calculated using the following equation:

$$P_T^{max} = \eta_{max}^{DFM}\, P_{wt}^{max} \qquad \textbf{(2.11)}$$

8. The system is commanded to the desired optimum shaft speed Ω_T^{opt} determined from step 5 above and the total output power P_T is measured.

9. The new measured output power is compared to the estimated maximum, P_T^{max}, as determined in step 7 to update or retain the C_p - λ curve, if required.

10. The shaft speed is perturbed within a small speed range and the total output power, P_T, is measured repeatedly to actually determine and confirm the maximum power point, $P_{T,meas}^{max}$.

11. The estimated P_T^{max} is compared to the measured $P_{T,meas}^{max}$ from step 7.

12. If they do not compare within a predetermined limit or threshold the optimum power coefficient, used in step 4, in the C_p - λ curve, is updated. The power coefficient of the turbine may have changed due to the change in parameters of the turbine over time caused by e.g. wear and tear. The new optimum C_p can be determined by employing P_{wt}^{max} (as determined from (2.11)) in eq.(2.4) and assuming the wind speed to have remained constant and as determined in step 5. This (step 12) is only a possible enhancement to the algorithm which has not been implemented in the laboratory based development system.

13. Efficiency maximization algorithm enabled.

14. The MPPT algorithm is repeated (steps 1-12) upon waiting for a controller interrupt.

Although both the search based and the deterministic MPPT controllers have been presented in this chapter, only the wind speed estimation based MPPT algorithm was implemented in the laboratory VSG system. The faster determination of the MPP by the deterministic method than that of the perturbation method was the contributing factor for this decision. As seen in Fig. 2.7 and in the algorithm outline in this section, perturbation of the rotational speed is only needed for fine tuning purposes to adjust for system parameter changes over time. This is achieved by steps 10-12 above. It should be noted here, that the MPPT controller as implemented in the laboratory VSG system and as detailed in Chapter 4 does not provide for the on-line update of the system parameters, a provision that has been left for future enhancement to the basic MPPT controller.

3. Converter Controller Design

The BDFM requires an ac/ac power conversion interface between the grid and its control winding, which is typically of a reduced rating, but requires four quadrant operational capability. The converter consists of two energy conversion stages with an intermediate dc-energy storage. Due to the requirement of its bi-directional capability, the ac/dc interface requires an active stage unlike in other ac/ac drives where a passive stage with regenerative resistive braking is adequate.

In conventional electronic ac/ac conversion, electrical energy of a certain form is stored in an intermediate dc or ac "link" which can be regarded as an energy reservoir, before it is finally converted to its desired form. In the commonly available version of the ac/ac converters, as in this development, this is usually a capacitor bank in a voltage source converter or an inductor in a current source converter for topologies that use a dc-link. In an ac link a combination of passive devices (capacitors and inductors) with or without the use of active devices (discrete electronic components like thyristors, IGBTs, MOSFETs etc.) is used [14]. It is also possible to convert energy from one form to another form of ac, without the need for intermediate link energy storage. This is typically done in a direct converter topology [14].

The active input stage of the converter is input current controlled. It is connected to the electrical grid through a set of grid interface inductors. The magnitudes of the input currents are determined dynamically by the internal controller from the input/output power flow of the converter with information obtained from a link energy monitor. The input currents are controlled to be sinusoidal with a fundamental frequency (information provided by a phase locked loop) corresponding to that of the grid frequency and the frequency of excitation of the DFM power winding. Since both the magnitude and the phase of the input currents of the converter is actively controlled, it is possible to operate the converter with leading and lagging power factors at the input. This extra demand in

the reactive power should be such that the kVA rating of the converter and the voltage and current ratings of the semiconductor devices and the other components are not exceeded and is controlled as to not interfere with the active power maximization described in this thesis.

The output stage is similar to that of a current controlled conventional dc/ac inverter. Thus, the inverter output currents, i.e. the excitation for the control winding of the BDFM, can be controlled to be sinusoidal with a certain magnitude and fundamental frequency as demanded by the system controller.

3.1. Sensor-based rectifier controller

Active front-end rectifiers with reduced input current harmonics and high input power factor are gaining prominence for almost all utility interfaced applications due to an increasing concern about power quality and harmonic standards such as IEEE 519 and IEC 555. These guidelines necessitate the use of bulky, expensive filters at the input of a passive rectifier stage. These filters are optimized for a certain cut-off frequency and operating or loading condition. This has a negative impact on system dynamic performance and leads to a poor input power factor. Thus, more and more industrial applications, such as variable speed generators and regenerative ac drives, now require ac/ac converters capable of efficient bi-directional power transfer. This has led to the development of various control strategies for high input power factor, sinusoidal input current rectifiers [28-31]. Control techniques found in the literature can be classified as: (a) stationary frame, (b) synchronous frame, (c) space vector control methods. While the stationary frame control strategies, presented so far, are relatively simpler than their synchronous frame and space vector counterparts, they have inherent drawbacks ranging from slow dynamic response to restricted stability regions [28-31]. The methods developed in the rotating frame and the space vector approach can provide for fast dynamic control and stability, but are quite complicated to implement and require significant computational resources. Often their

implementation cannot be justified for industrial applications where such precise control is not needed.

Thus, the development of a simple control algorithm for the active rectifier is appropriate. The controller presented here is based on the stationary frame approach and attempts to address the shortcomings of previous control strategies while retaining the simplicity of the stationary frame [32]. This simple algorithm provides for controllable input power factor and excellent spectral performance of the input line current. Moreover, it has been implemented in a single low-cost fixed-point microprocessor (Intel 80C196KC) which provides an a/d converter with multiplexed 8-channels.

In section 3.2, a model-based rectifier input current controller is presented which proposes the elimination of the current sensors from the load conductance based rectifier to further reduce the cost of the controller [33]. This further reduces the overall cost of the rectifier without sacrificing the advantages of the original sensor-based version of the controller. While the proposed controller is more complex than the sensor based version [32], it can still be accommodated in the same, low-cost fixed-point microprocessor (Intel 80C196KC).

3.1.1. Sensor-based rectifier control algorithm

The proposed system can be broadly classified into its constituent sub-system controllers as shown in Fig. 3.1. The DC-bus regulator and the current controller blocks as implemented are available in traditional stationary frame rectifier controllers, augmented here by equivalent load conductance calculation [34,35]. The overall system conceptualization and implementational details are provided in the following sub-sections.

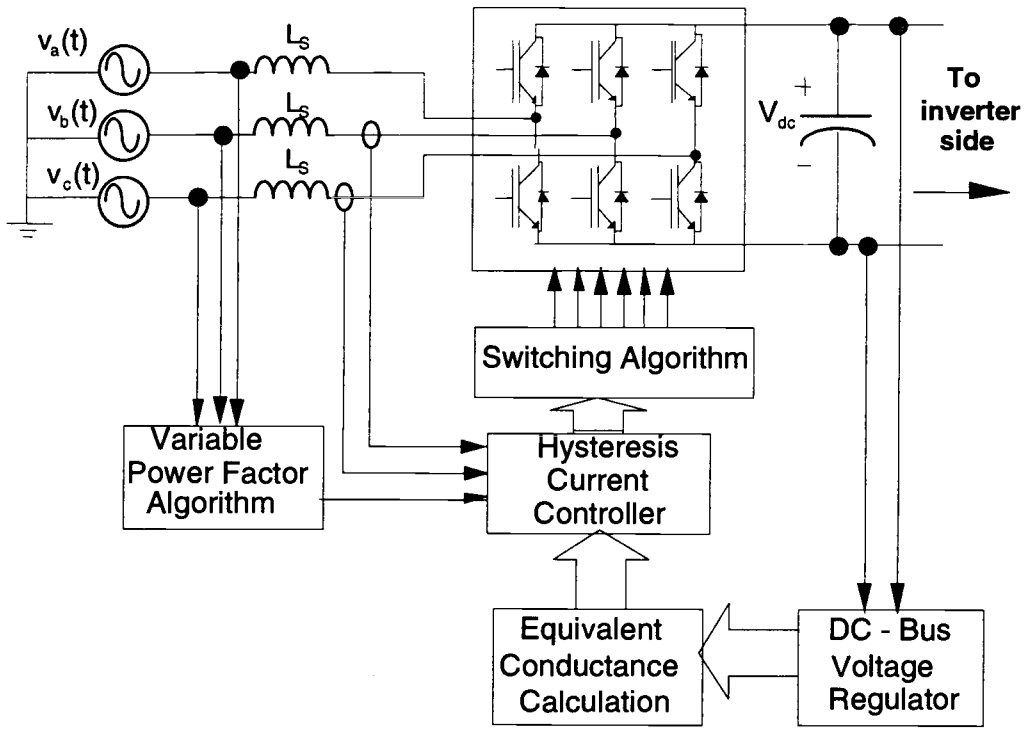


Figure 3.1 Block diagram representation of the sensor-based rectifier controller.

3.1.1.1. DC-bus voltage regulation

For optimal operation and protection of the passive and active components of the rectifier the energy stored in the link capacitor should be kept relatively constant. Also, to avoid saturation of input current regulators and maintain sinusoidal input currents, the dc-bus voltage v_{dc}, must be maintained such that [36]:

$$v_{dc} \geq \frac{2\sqrt{2}}{\sqrt{3}} V_{LL} \tag{3.1}$$

where V_{LL} is the rms line-to-line grid voltage. The dc-bus voltage reference V_{dc}^* is thus set at a minimum, given by eq. (3.1). In practice, V_{dc}^* is set such that:

$$V_{dc}^* = \frac{2\sqrt{2}}{\sqrt{3}} V_{LL} + V_L \tag{3.2}$$

where V_L is the voltage drop across an input line inductor under full-load condition. The link capacitor voltage is measured and averaged over a fixed time window using a low-pass filter with a gain as shown in Fig. 3.2. The output of this filter is proportional to the equivalent rectifier conductance G [35] as explained in section 3.1.1.2. Since the conductance G is proportional to the error of the dc-bus voltage, the rectifier input current increases until the dc-bus voltage error decreases, which in turn reduces the input current.

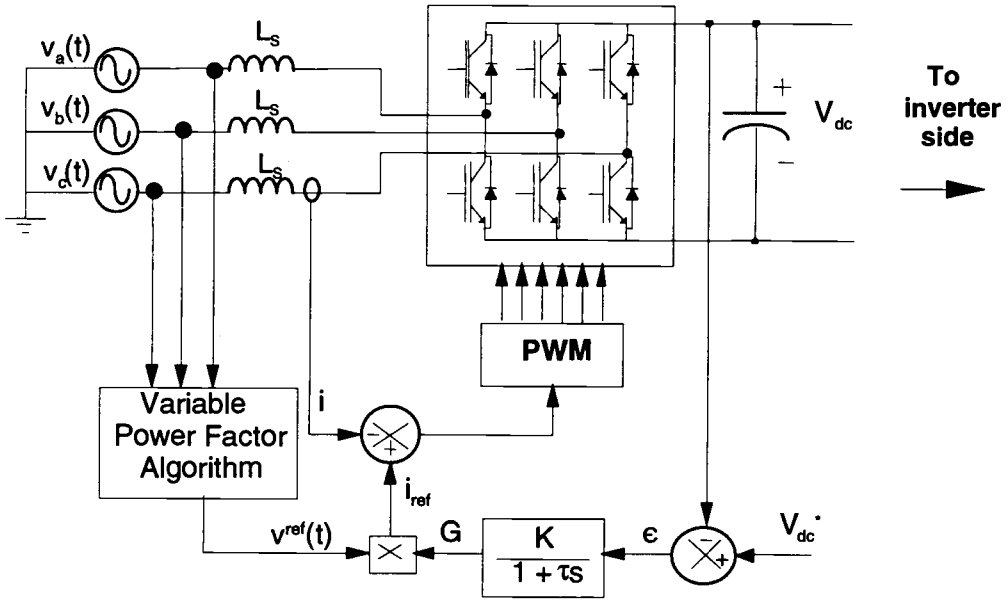


Figure 3.2 Sensor-based controller representation.

3.1.1.2. Equivalent load conductance calculation

The proposed controller is based on an equivalent rectifier conductance calculation scheme [34,35]. By this method, a representation of the information of the real or active power, defined as the average rate of energy transfer from source to load [35], is essential to determine the required rectifier input current.

Under steady state loading conditions, averaging the dc-bus error, $\varepsilon_{dc}(t)$, given by

$$\varepsilon_{dc}(t) = V_{dc}^{*} - v_{dc}(t) \tag{3.3}$$

over a time-window using a low-pass filter gives an estimate of the averaged load conductance, G. The low-pass filter used for conductance calculation uses a gain K (set to 1 for results obtained here) and a relatively long time or low-pass filter constant, τ, as shown in Fig. 3.2. It was observed that while the low-pass filter constant, τ, was critical in smoothing the dc-bus ripple, the filter gain, K, affected the dynamic response time of the system. Once the load conductance is determined the reference current is generated by the hysteresis current controller in conjunction with the variable power factor controller as described in section 3.1.1.3 below.

3.1.1.3. Hysteresis current controller

The instantaneous representation of the active component of the rectifier input current has the same waveform as the voltage and its amplitude depends on the equivalent conductance G; thus a convenient way to determine the reference current is as follows :

$$i_{ref}(t) = G\, v_{ref}(t) \tag{3.4}$$

where $v_{ref}(t)$ is determined by the variable power factor controller which utilizes a user input power factor command and the zero-crossing information of the grid voltages.

The phase current errors of only two phases are directly determined from their corresponding references and measured current magnitudes. In the three-wire system investigated, the current error of the third phase, can, of course, be determined from the two measurements. Details of the switching algorithm are presented in section 3.1.1.4. By maintaining a sufficiently small hysteresis band for the current comparison, and ensuring turn-off of all devices that are on before the next comparison, a fixed switching frequency can be attained. This is, however, only true for full load conditions when current magnitudes are comparable to the designed quantities. For low current levels the switching frequency could vary considerably depending upon the loading condition.

Adjustment of the finite on or off times of the switches based upon the input current rms magnitude can provide substantially improved current waveform at low current levels. Although the controller as implemented could provide for on-line adjustment of the on-time of the switches the control loop required to update the on-time of the switches has not been incorporated. This feature could possibly be exploited in future controller development and enhancement.

3.1.1.4. Switching Algorithm

The switching strategy implemented in the current controller is purely based on the error between the reference current and the measured current as shown in Fig. 3.3 and as explained below. If we ensure balanced current reference generation, by ensuring that the summation of the three phase current references adds up to zero, it can be shown that

$$\varepsilon_{ia} + \varepsilon_{ib} + \varepsilon_{ic} = 0 \tag{3.4}$$

where

$$\varepsilon_{ik} = i_k^{ref} - i_{km} \tag{3.5}$$

i_{km} is the measured phase current and k = a or b or c. From eq. (3.4) it is seen that all three errors cannot have the same sign. Moreover, assuming that the current errors can only be either finite positive or finite negative, i.e. neglecting zero errors, there are only four possible error scenarios as depicted in Table 3.1. At any instant of time, only one of these four error scenarios shown in Table I is prevalent in the rectifier circuit. For a zero error condition, no switching happens and, hence, this can be neglected for the switching algorithm development.

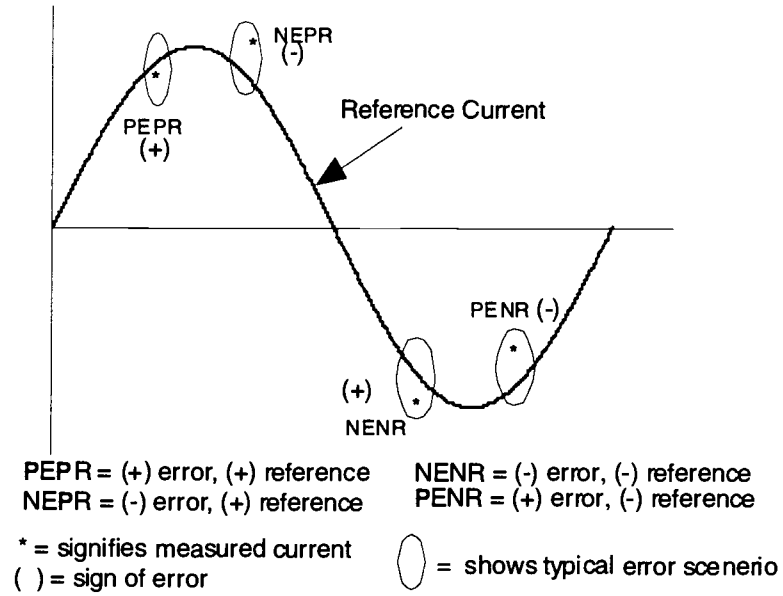


Figure 3.3 Possible current errors in each phase of the rectifier.

Table 3.1 Typical current error scenarios

Typical Scenarios	Error Combination	Mode of operation
Case I	NEPR, PEPR, PENR	Buck, Short, Short
Case II	PEPR, NEPR, NENR	Boost, Short, Short
Case III	NENR, PEPR, PENR	Buck, Short, Short
Case IV	PENR, NEPR, NENR	Boost, Short, Short

Depending upon the error scenario, the mode of operation is determined. A phase is defined to operate in a valid "boost" or "buck" mode if only one of the three lower or upper devices is selected. By the definition and as shown in Table I, valid "boost" or "buck" operations are mutually exclusive and only one phase participates in a valid "boost" or "buck" mode of operation at a time. A "short" is defined to be the operating

condition on the phases which do not participate in a valid boost or buck. It is easily seen that the phases which do not participate in a "buck" or a "boost" operation are shorted via the line reactors, hence the name. The switching algorithm, shown in a flowchart in Fig. 3.4, effectively achieves the modes of operation as shown in Table I.

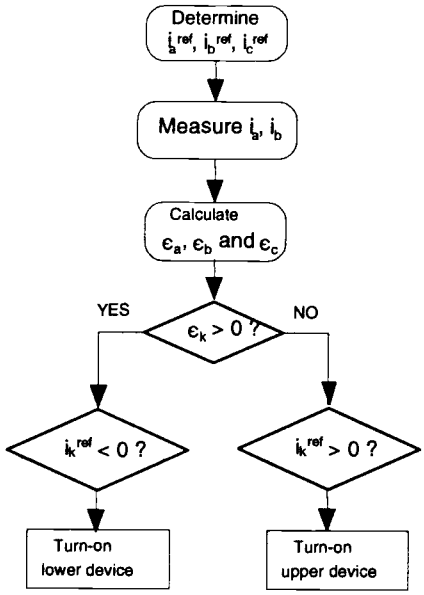


Figure 3.4 Switching logic of the rectifier.

3.1.2. Implementation of sensor-based controller

The microprocessor implementation of the rectifier is illustrated in Fig. 3.5. The sensor-based controller was implemented using an Intel 80C196KC 16-bit embedded microprocessor, which provides an analog to digital converter with multiplexed 8-input channels which is used for the various a/d operations required by the controller algorithm. The high-speed output port, provided by the 80196, is used for interfacing the switching signals to the IGBT drivers. The high-speed input port is used for phase voltage measurement and an ordinary input port is used for detection of any fault signals generated by the drivers, as shown in Fig. 3.5.

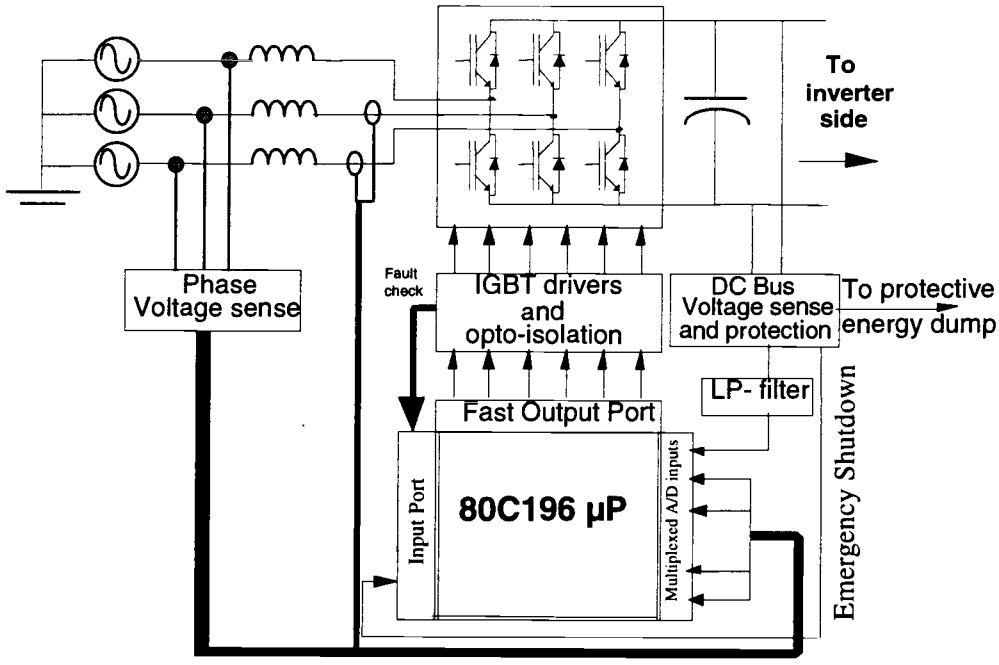


Figure 3.5 Sensor-based rectifier controller implementation

All of the constituent sub-systems, but for the low-pass filter, have been digitally implemented in the microprocessor. Each switching iteration, including the dc-link voltage regulation, presently needs 135 µs providing for a switching frequency of just under 7 kHz. Further optimization of hardware and software will enable an increase in the switching frequency.

3.1.3. Experimental evaluation

The proposed rectifier control system discussed above has been developed on an active input/output IGBT converter. Although identical with respect to the rectifier and inverter controllers this converter was rated at a substantially higher power (35 kVA) than the one (5.5. kVA) that was utilized for the performance optimization system controller evaluation as described in Chapters 4 and 5. Figure 3.6 shows the dynamic performance of the proposed controller when the dc-bus voltage is step-changed from 280 V to 390 V.

Although this scenario is not very likely to occur during the normal operation of the rectifier, it provides for a good opportunity to investigate the dynamic response of the proposed controller. Both the voltage and the load are step-increased by 40 %. It is seen that the rectifier attains steady state within 150 ms.

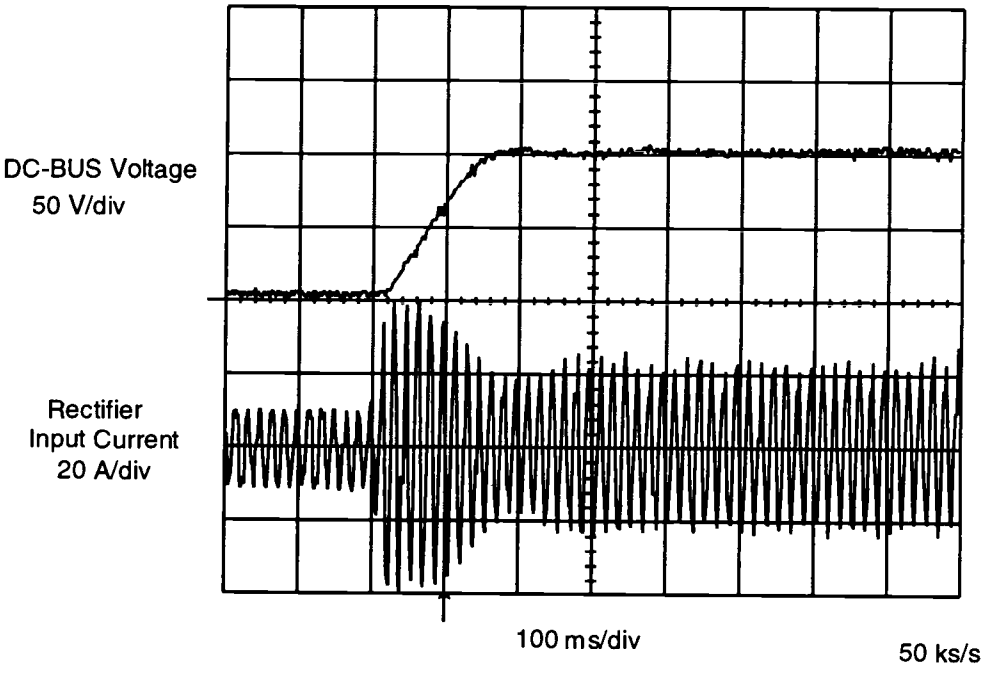


Figure 3.6 Rectifier controller response to a 40% (280 V- 390V) step change in commanded dc-bus voltage. DC-bus capacitance = 2400 μF

The step change in the dc-bus voltage reference results in a big increase in the dc-bus error, $\varepsilon_{dc}(t)$, which results in an increase in the demanded input current reference as seen in Fig. 3.2. Hence, in Fig. 3.6, at the instant of the step change in the dc-bus voltage reference, the input current encounters an overshoot. The resulting overshoot in the current can be attenuated by an proper choice of the low-pass filter gain as explained section 3.1.1.2. However, the attenuation of the overshoot must be traded-off against the dynamic response of the rectifier system. The rectifier system used for controller development employs power switches capable of handling the overshoot current shown in Fig. 3.6. Thus, to obtain the best response of the rectifier system, the overshoot in the input current was allowed.

Figure 3.7 depicts typical unity power factor operation of the rectifier system with a load of 10 kW. Leading power factor operation with similar input/output conditions and dc-bus voltage is shown in Fig. 3.8. Clearly, the rectifier input current waveform has deteriorated in Fig. 3.8. The excess demand of reactive power (5.77 kVAR), with a dc-bus voltage similar to that of the unity p.f. case, saturates the current regulator affecting their capability of maintaining sinusoidal current waveforms.

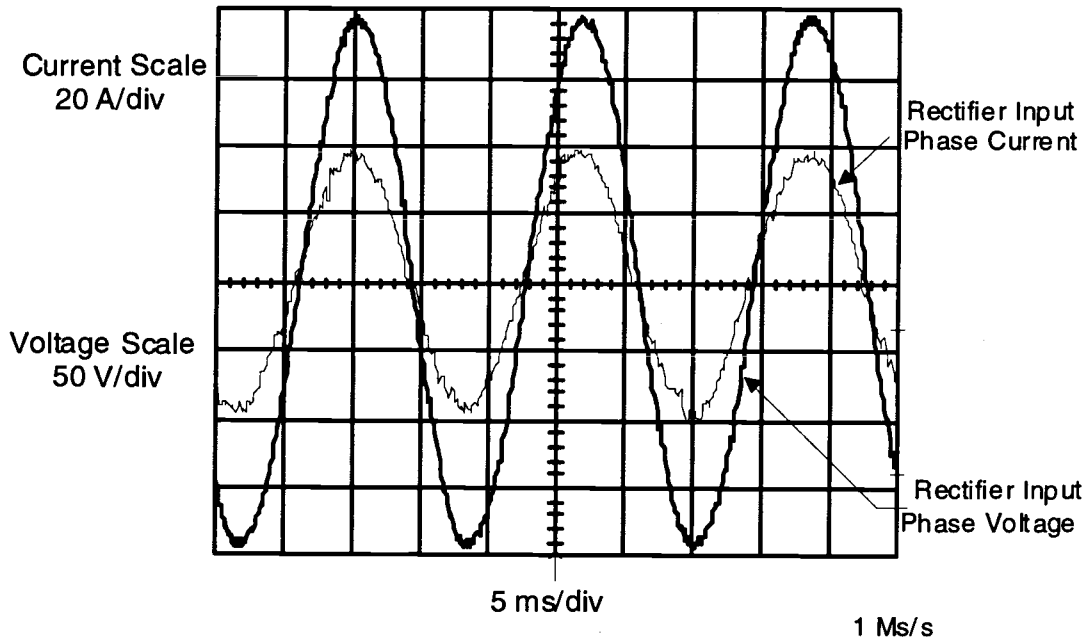


Figure 3.7 Unity power factor operation of the sensor-based rectifier controller.
V_{dc} = 390 V, V_{ac} = 230V, load = 10 kW

Although it may not be a solution for most applications, the saturation problem of the input current regulators is immediately remedied by increasing the dc-bus voltage as shown in Fig. 3.9 for similar input/output conditions. For applications where the dc-bus voltage can not be increased due to voltage stresses on the devices, the demanded reactive power needs to be reduced if spectral performance is desired. The Fourier spectrum for this current waveform shows energy density around 7.5 kHz, the switching frequency. It can also be observed from the spectrum that all of the magnitudes of the major harmonic components, the 3rd, 5th, 7th, 9th, etc. are all less than 0.5% of the fundamental as shown in Fig. 3.10.

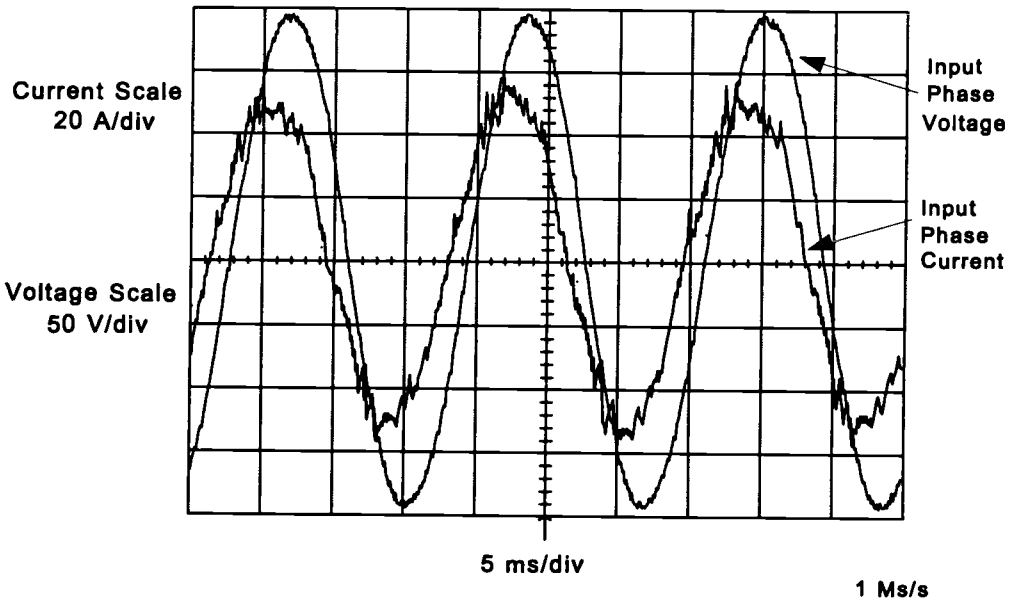


Figure 3.8 Leading 0.85 p.f. operation. V_{dc} = 390 V
V_{ac} = 230 V, load = 10 kW

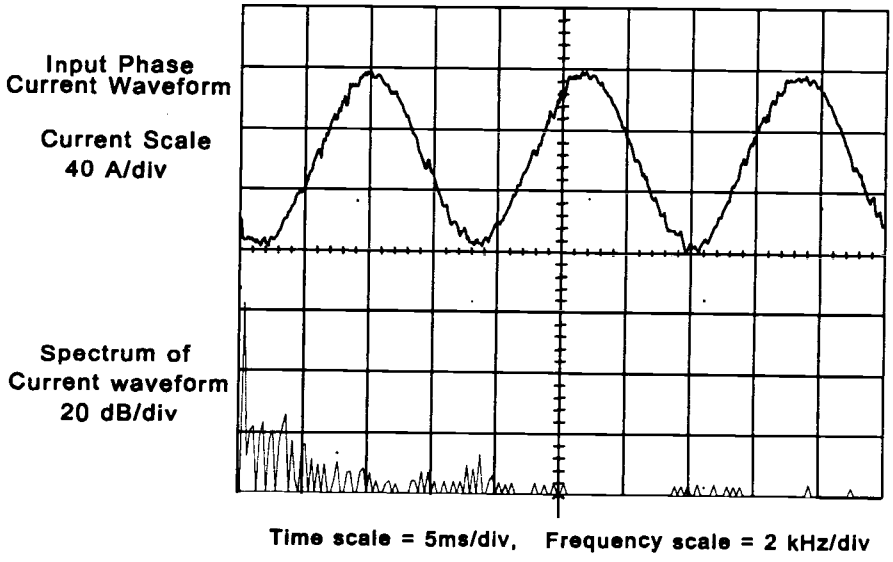


Figure 3.9 Leading 0.85 p.f. operation. V_{dc}= 450 V, V_{ac} = 230 V, load = 10 kW.

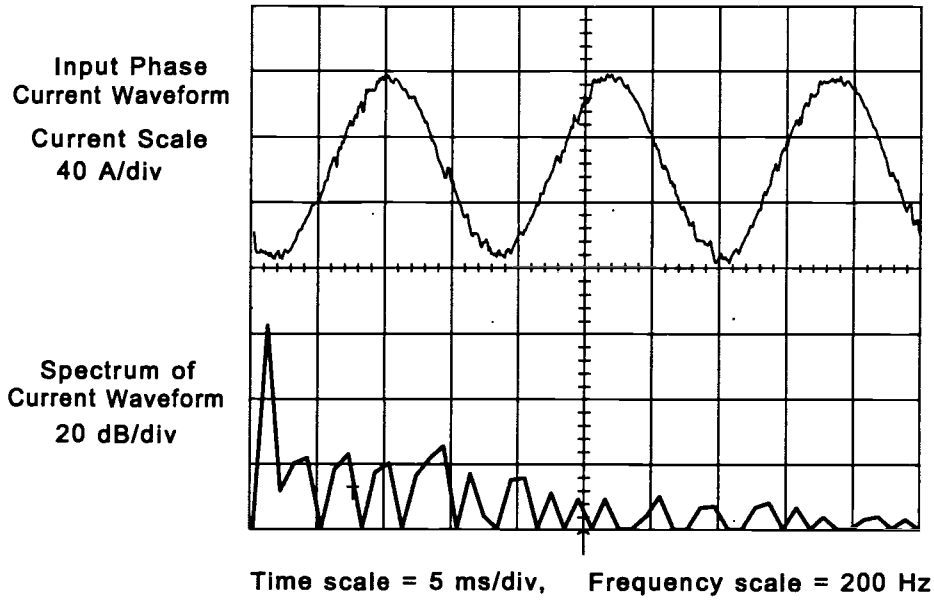


Figure 3.10 Expanded view of the Fourier spectrum as shown in Fig. 3.9.

3.2. Sensorless rectifier controller

In section 3.1 an input current controller, based on load conductance calculation, was presented for an active rectifier which uses current sensors for line current feedback. Due to the availability of the information regarding grid voltages, dc-bus voltage and the switching configuration of the rectifier circuit, a discrete switched rectifier model can be constructed for control purposes. The model can then be used to estimate and predict the input phase currents of the rectifier.

This section investigates the theoretical and practical aspects of the model-based rectifier controller implementation and proposes the elimination of the current sensors from the load conductance based rectifier. This further reduces the overall cost of the rectifier without sacrificing the advantages of the original sensor-based version of the controller. While the proposed controller is more complex than the sensor based version,

it can still be accommodated in the same, low-cost fixed-point microprocessor (Intel 80C196KC).

While the controller has the obvious advantage of removing the current sensors, other benefits include elimination of sensor offsets, resolution problems, non-simultaneous sampling of the phase currents associated with single analog-to-digital (A/D) converter and analog measurement noise problems.

3.2.1. Model-based predictive rectifier controller algorithm

A block diagram of the proposed system is shown in Fig. 3.11. The proposed sensorless controller is built upon the load conductance based rectifier controller as described in section 3.1 [32]. Thus, details regarding most of the constituent blocks of the original sensor-based controller also apply, conceptually though not implementationally, to this sensorless version of the controller. The rectifier model based input current estimator and predictor will be presented in detail in this section. Theoretical validation for the sensorless approach will be provided analytically. Finally, the controller is verified using simulation and laboratory implementation.

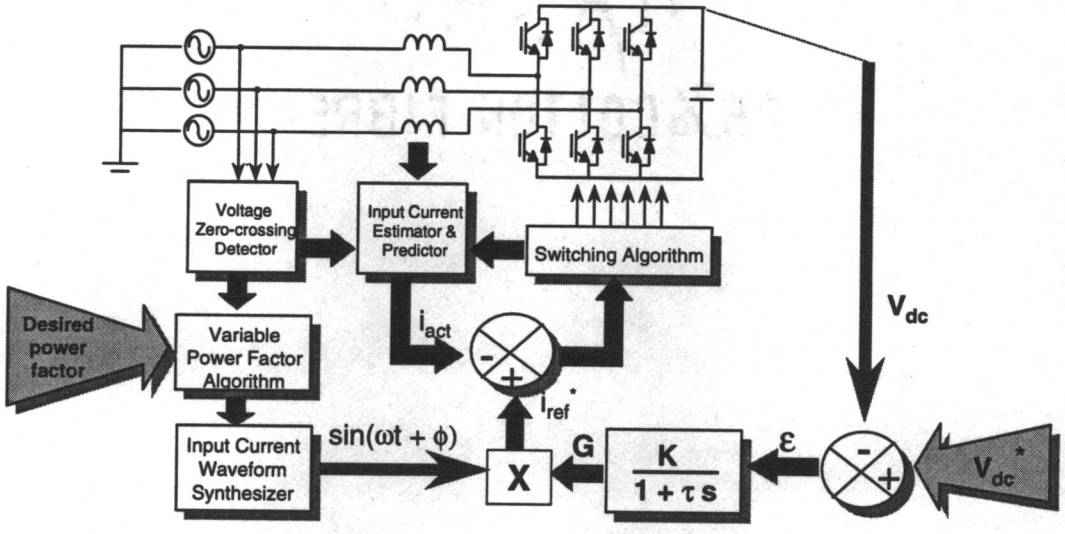


Figure 3.11 Sensorless version of the rectifier controller.

3.2.1.1. Model development

Figure 3.12 depicts the lumped circuit representation of the various components of the rectifier hardware for a particular switching state. For simplicity and ease of model implementation, all stray effects have been neglected.

The three-phase input line reactors are assumed to be balanced. They are also assumed to be linear owing to the presence of an air gap in the flux path. A current regulated inverter stage (DC-AC conversion) which usually follows the link can be represented by a constant current source during steady state operation. Thus, under normal circumstances, the dc bus can be assumed to be loaded by a constant current source.

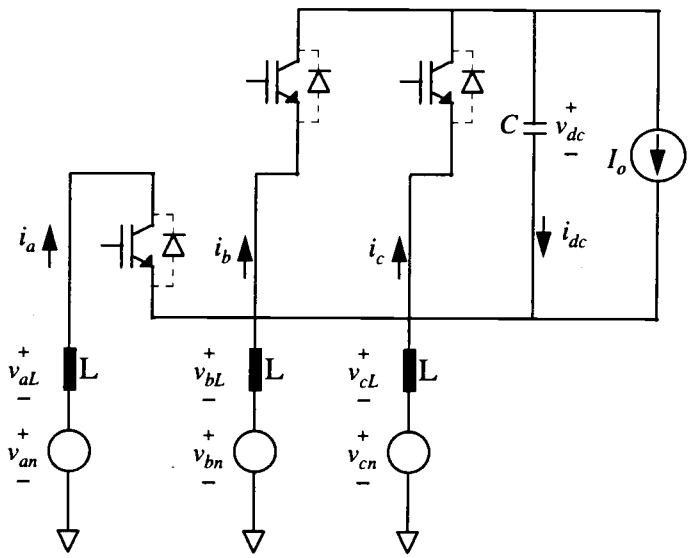


Figure 3.12 Lumped circuit representation of a switching state of the rectifier

The six IGBT-diode combination switches can be classified into the high-side and the low-side switches, forming a conjugate pair per phase. The simultaneous operation of a conjugate pair is disallowed to prevent shoot-through faults, leaving only eight non-destructive switching states. In an effort to develop an observer based current controller, only six valid switching states or scenarios are permitted by the algorithm.

At any instant of time only three of the six switches participate in current conduction. Participation of all three switches of the same category in a switching state is considered to be an illegal state for reasons provided in section 3.2.1.2. The two illegal states of the proposed control algorithm are also known as the zero vector in space vector modulation strategies [37]. A valid switching state consists of the participation of two switches of the same category with the third participating switch from the other category. This switch combination is based upon the possible current errors [5]. It should be noted here that the current errors in the three phases add up to zero as shown in eq.(3.4).

Each switching state is associated with a complementary state. Two states are defined to be a complement of each other if all the participating switches of one state are the conjugate switches of the other state. The discrete pulse modulation (DPM) as implemented in the current controller provides for a finite on-time and off-time of the switches before the initiation of the next state by the current controller. At the end of a valid switching state, during which IGBTs were conducting, its complementary state, consisting of the diodes of the complementary switches, is active during the off-time of the previously participating switches. Thus, a model can be developed which accommodates both on-time and off-time circuit updates.

Assuming negligible conduction drops across the participating switches, the non-reduced model equations are as shown in eqs.(3.6)-(3.9) for the switching state in Fig. 3.12. Similarly, equations can be generated for the remaining valid switching states.

$$\frac{di_a}{dt} = \frac{2}{3L} v_{dc} + \frac{1}{L} v_{an} \tag{3.6}$$

$$\frac{di_b}{dt} = -\frac{1}{3L} v_{dc} + \frac{1}{L} v_{bn} \tag{3.7}$$

$$\frac{di_c}{dt} = -\frac{1}{3L} v_{dc} + \frac{1}{L} v_{cn} \tag{3.8}$$

$$\frac{dv_{dc}}{dt} = -\frac{1}{C} i_a + \frac{1}{C} I_0 \quad \textbf{(3.9)}$$

3.2.1.2. Control Issues

Based upon the following identities for a three phase, three-wire system,

$$i_a + i_b + i_c = 0 \quad \textbf{(3.10)}$$

$$v_{an} + v_{bn} + v_{cn} = 0 \quad \textbf{(3.11)}$$

the four equations (3.6-3.9) reduce to two equations, i.e. (3.6) and (3.9) with states i_a and v_{dc}. Knowledge of one of these states provides information about the other. As implemented in the original sensor-based version, the measurement of the link voltage is essential for the load conductance calculation. Thus, the current information, i_a, is derived from the measurement of the dc- bus voltage, v_{dc} and eq. (3.6).

Assuming linear line inductors, it is seen that at any instant of time, with any switch or diode combination allowed by the six valid switching states, the rectifier circuit is an observable, linear, second order system. Once v_{dc} is measured, the remaining currents i_b, i_c are readily obtained using equations (3.7) and (3.10).

Thus, the current sensors can be eliminated while still maintaining active current waveform shaping. It should be noted that all the conclusions above are only valid for the six legal switching states. The two non-destructive, illegal switching states (zero states) lead to the loss of the property of observability as the dc-bus voltage is no longer a part of the circuit during those switching states.

3.2.1.3. Algorithm Flowchart

The input current estimator uses the grid voltages, dc-bus voltage, switching signals and the magnetic characteristics of the line inductors to determine the current. The switching strategy implemented in the current controller is based on the error between the

reference current and the estimated current and has been described in detail is section 3.1.1.4.

A simplified flowchart representing the current estimation and control is shown in Fig. 3.13. Table 3.2 lists the different switching states and the difference equations as implemented in the controller. The dc-bus voltage regulation and its associated control loop are shown in Fig. 3.11. Both the dc-bus voltage regulation and input current control loops have the same bandwidth and are updated every program execution cycle.

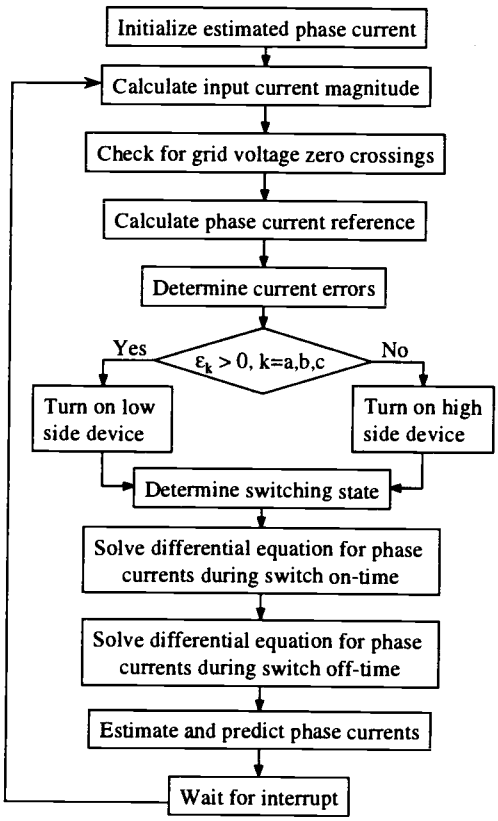
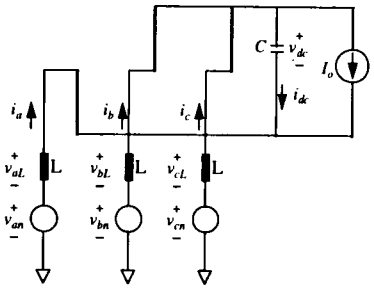
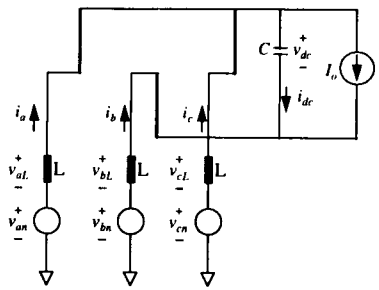
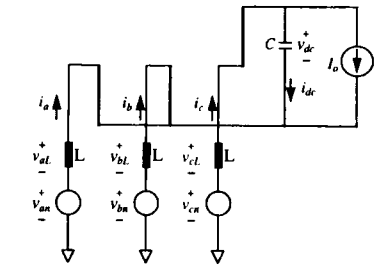
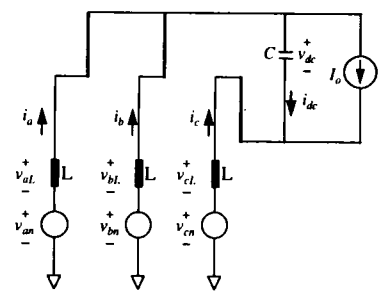
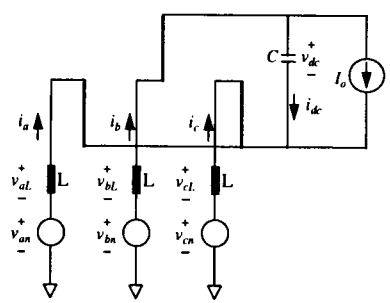
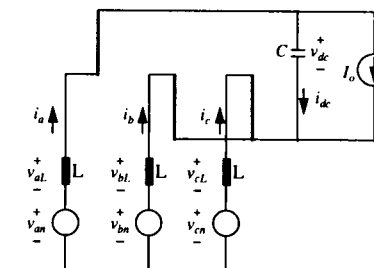


Figure 3.13 Simplified flowchart of the sensorless version of the rectifier controller

Table 3.2 Possible switching states

# 1	# 2	# 3
		
$i_a(n+1) = i_a(n) + [\, v_{an}(n) + \frac{2}{3} v_{dc}(n) \,] \frac{\Delta T}{L}$ $i_b(n+1) = i_b(n) + [\, v_{bn}(n) - \frac{1}{3} v_{dc}(n) \,] \frac{\Delta T}{L}$ $i_c(n+1) = -[\, i_a(n+1) + i_b(n+1) \,]$	$i_a(n+1) = i_a(n) + [\, v_{an}(n) - \frac{1}{3} v_{dc}(n) \,] \frac{\Delta T}{L}$ $i_b(n+1) = i_b(n) + [\, v_{bn}(n) + \frac{2}{3} v_{dc}(n) \,] \frac{\Delta T}{L}$ $i_c(n+1) = -[\, i_a(n+1) + i_b(n+1) \,]$	$i_a(n+1) = i_a(n) + [\, v_{an}(n) + \frac{1}{3} v_{dc}(n) \,] \frac{\Delta T}{L}$ $i_b(n+1) = i_b(n) + [\, v_{bn}(n) + \frac{1}{3} v_{dc}(n) \,] \frac{\Delta T}{L}$ $i_c(n+1) = -[\, i_a(n+1) + i_b(n+1) \,]$
# 4	**# 5**	**# 6**
		
$i_a(n+1) = i_a(n) + [\, v_{an}(n) - \frac{1}{3} v_{dc}(n) \,] \frac{\Delta T}{L}$ $i_b(n+1) = i_b(n) + [\, v_{bn}(n) - \frac{1}{3} v_{dc}(n) \,] \frac{\Delta T}{L}$ $i_c(n+1) = -[\, i_a(n+1) + i_b(n+1) \,]$	$i_a(n+1) = i_a(n) + [\, v_{an}(n) + \frac{1}{3} v_{dc}(n) \,] \frac{\Delta T}{L}$ $i_b(n+1) = i_b(n) + [\, v_{bn}(n) - \frac{2}{3} v_{dc}(n) \,] \frac{\Delta T}{L}$ $i_c(n+1) = -[\, i_a(n+1) + i_b(n+1) \,]$	$i_a(n+1) = i_a(n) + [\, v_{an}(n) - \frac{2}{3} v_{dc}(n) \,] \frac{\Delta T}{L}$ $i_b(n+1) = i_b(n) + [\, v_{bn}(n) + \frac{1}{3} v_{dc}(n) \,] \frac{\Delta T}{L}$ $i_c(n+1) = -[\, i_a(n+1) + i_b(n+1) \,]$

3.2.1.4. Simulation

The proposed current controller was simulated assuming a regulated constant dc-bus voltage and a constant switching frequency. The loading conditions on the dc-bus were simulated using a constant current source for reasons mentioned in section 3.2.1.1. The following conditions were imposed on a C-language based computer generated active rectifier model and controller simulator:

(i) grid voltage = 230 V,

(ii) dc-bus voltage = 380 V,

(iii) switching frequency = 5 kHz

(iv) duty of switches = 90%

(v) switching strategy = DPM

Simulation results are shown in Fig 3.14. As illustrated, the current waveform of Fig. 3.14. shows minimal distortion at the low frequencies compared to a diode-based rectifier and substantial spectral density at higher frequencies. The spectral energy in the submultiple harmonics of the switching frequency is due to the discrete pulse modulation (DPM) strategy in the simulation. Thus, this controller promises to retain the spectral performance of the original sensor based controller implementation.

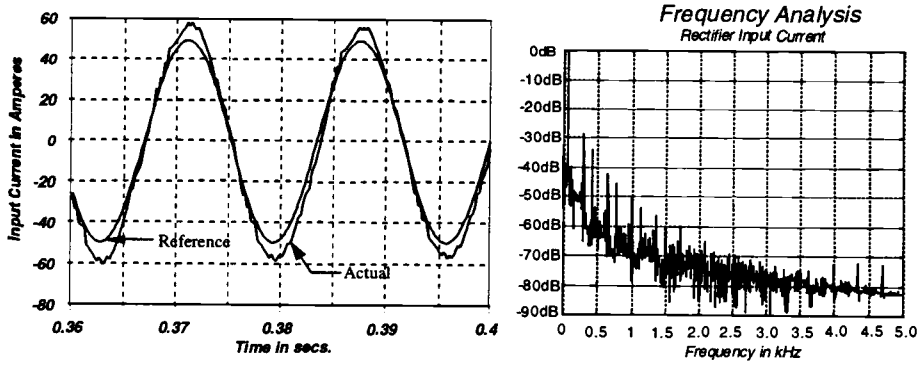


Figure 3.14 Time and frequency domain simulation result of rectifier input current with the proposed model based current controller under unity p.f. condition. f_{switch} = 5 kHz, V_{grid} = 230 V and V_{dc} = 380 V.

3.2.1.5. Implementation of sensorless controller

The microprocessor implementation of the sensorless version of the rectifier algorithm is similar to that as shown in Fig. 3.5 with no current measurement. The original current sensor based controller was modified to bring in the current predictor and estimator and was implemented using an Intel 80C196KC 16-bit embedded microprocessor.

In this version of the implemented controller, the only measured quantity is the dc-bus voltage, v_{dc}. The implementation of the proposed controller assumes balanced three-phase grid voltages. Thus, the information of the zero-crossings on only one of the phase voltages is adequate to determine the three phase voltages required by the current estimator. The zero-crossings of the "A" - phase grid voltage are detected using a zero-crossing detection circuit which incorporates a narrow band-pass filter. The phase shift of the filter is compensated for in the variable power factor controller. The zero-crossings enable the formation of a phase locked loop (PLL) for the current controller, which thus tracks the grid frequency. They also help synthesize the grid phase voltages needed by the estimator block as shown in Fig. 3.11.

All of the constituent sub-systems, but for the low-pass filter, and the zero-crossing detection circuit have been digitally implemented in the microprocessor. Each switching iteration, including the dc-link voltage regulation and input current estimation, presently requires 200 μs, yielding a switching frequency of 5 kHz. Although minimal changes were required in the hardware to accommodate the sensorless current control strategy, there was a 25% reduction in the switching speed compared to that of the sensor-based version due to the increased computational overheads. In the future, most of the expected modifications associated with this controller implementation will be in the software code for the microcontroller.

3.2.1.6. Experimental evaluation

To prove the efficacy of the model based current controller, the sensorless controller is compared to the sensor-based version discussed in section 3.1. Quantitative and comparative analysis are provided for various operational modes as detailed in Table 3.3. While the implemented version of the algorithm will benefit from software code optimization, it compares favorably with the sensor-based version in operational stability and spectral performance. All the results presented here were obtained with the following conditions:

(i) grid voltage = 230 V line-to-line

(ii) type of load = variable resistive load on the dc-bus in the link

(iii) switching strategy = DPM

(iv) duty ratio of switches = 90%

3.2.1.6.1. Sensor-Based Version

The sensor-based controller current waveform and its spectrum under unity power factor condition are shown in Fig. 3.15. The link voltage was maintained at a steady 380 V with 17.4A load current. This constituted a real power consumption of around 6.5 kW. The details of the spectrum are presented in Table 3.3.

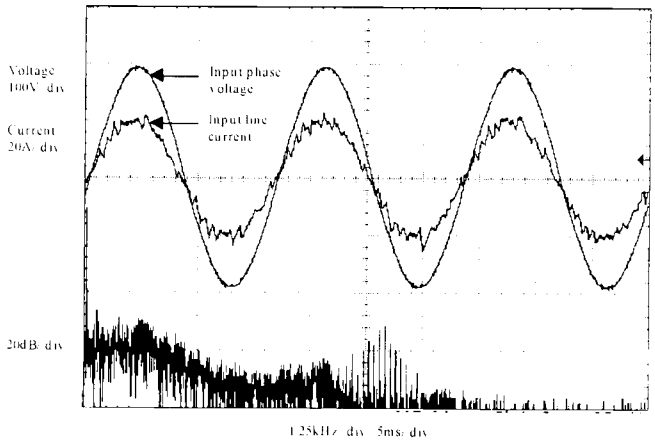


Figure 3.15 Experimental waveforms generated by the sensor-based controller. V_{ac} = 230 V, V_{dc} = 390 V, f_{switch} = 6.67 kHz, load ≈ 6.5 kW

3.2.1.6.2. Sensorless Version

Figure 3.16 shows unity power factor operation of the proposed model based controller. The dc-bus was maintained at 380 V and the resistive load on the dc-bus was equivalent to around 6 kW. The spectrum details are as shown in Table 3.3 and compare favorably with those of Fig. 3.15, though the model-based controller was implemented with a 25% slower switching speed than its original version.

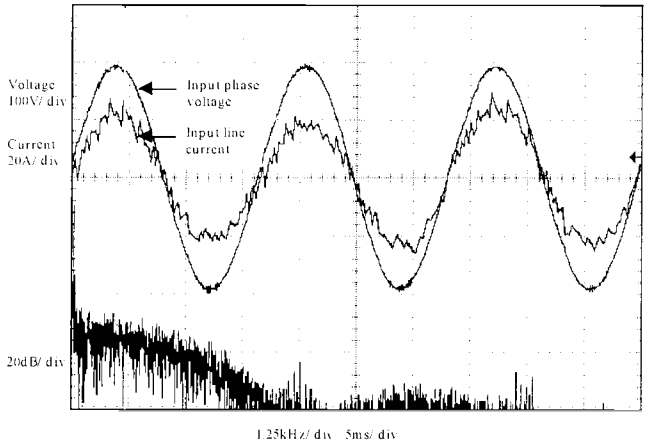


Figure 3.16 Unity power factor operation of the sensorless controller.
V_{ac} = 230 V, V_{dc} = 390 V, f_{switch} = 5 kHz, load ≈ 6 kW

With similar loading conditions, Fig. 3.17 shows a 0.85 leading power factor operation of the active rectifier. It should be noted that the input current magnitude increases marginally to compensate for the increase in reactive power demanded by the rectifier. The amount of reactive power that can be delivered by the rectifier is dependent upon the dc-bus voltage, the voltage drop across the line reactors and the magnitude of the input current.

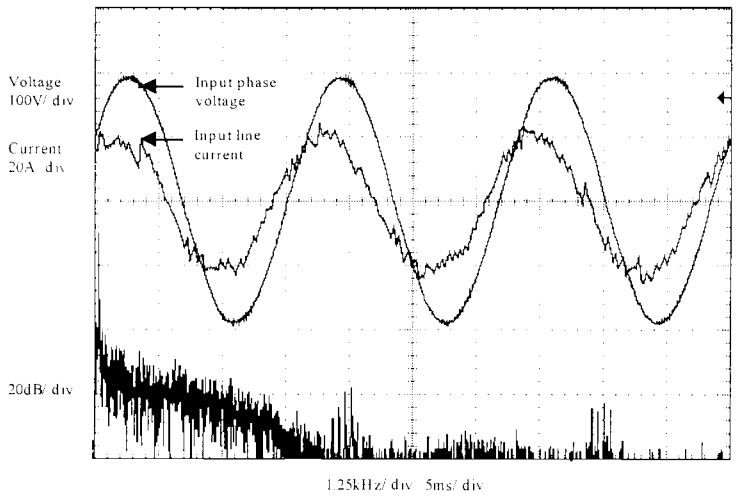


Figure 3.17 Leading power factor operation of the sensorless controller.
V_{ac} = 230 V, V_{dc} = 390 V, f_{switch} = 5 kHz, load ≈ 6 kW

The robustness of the controller with respect to the measured grid voltages was also investigated. As indicated in Fig 3.11, the grid voltage magnitudes used in the input current estimator are constructed from the grid voltage zero-crossing detections in combination with a nominal voltage magnitude. The controller was tested for error conditions arising due to discrepancies in the actual and the set grid voltages. Two types of error conditions, positive and negative errors, were imposed. The grid voltage error is defined to be the algebraic difference of the actual rms grid voltage and the assumed rms grid voltage set in the current observer.

Figure 3.18 shows the current waveform due to an imposed 20% negative voltage error. The grid voltage in the current observer was set at a 230V line-to-line, whereas the actual grid voltage was maintained at 184 V line-to-line. This reduction in the grid voltage is detected by the dc-bus regulator, which demands more input current to compensate for the required power flow into the link and the load. Thus, with loading conditions similar to that of Fig. 3.16, the current magnitude increases as seen in Fig. 3.18. The rectifier can operate with low grid voltages till the current protection, set according to the devices used in the hardware, gets activated.

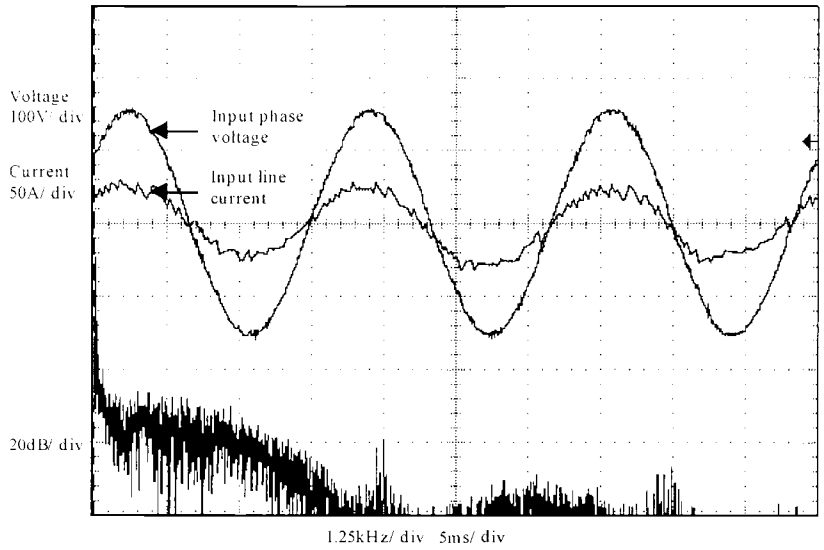


Figure 3.18 Rectifier operation with an (-)20% grid voltage error.
V_{ac} = 184 V, V_{dc} = 390 V, f_{switch} = 5 kHz, load ≈ 6 kW

The controller was also tested for the less likely error scenario, a positive grid voltage error. To evaluate this condition the voltages in the current observer was set at a value (190 V) lower than that of the actual grid voltage (230 V). This type of grid voltage error condition provides for smaller ripple in the estimated current than that in the actual input current, as determined from the equations in Table 3.2. Thus, the current errors are not corrected for as often as required. Hence, it was noticed that upon further decrease in the magnitude of the grid voltage used in the current observer, the number of erroneous switching decisions is increased, finally rendering the controller incapable of active current waveshaping. Figure 3.19 illustrates the deterioration of the input current with a positive grid voltage error. It was experimentally observed that the controller tolerated around 17% of positive voltage errors before becoming unstable.

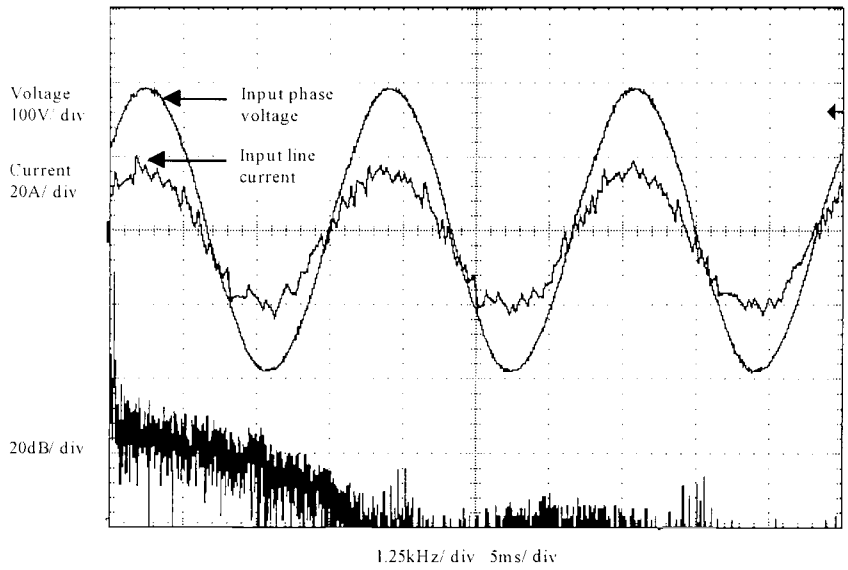


Figure 3.19 Rectifier operation with an imposed (+)17% grid voltage error. V_{ac} = 230 V, V_{dc} = 390 V, f_{switch} = 5 kHz, load ≈ 5.5 kW

The effect of a 21% decrease (under estimation) of the inductance value set in the model (from 7 mH to 5.5 mH) on the rectifier input current waveform is shown in Fig. 3.20. Incorrect switching selections still occur under this condition. The ripple of the estimated current is larger than that of the actual current waveform due to the smaller inductance value chosen, as expected from the equations of Table 3.2. Thus, the current controller switches more often than necessary to correct for the estimated errors which makes the actual input current always smaller than the demanded input current magnitude. The controller will survive a decrease in inductance till the input current limit gets activated. For the opposite error condition, it was experimentally determined that the controller can survive a maximum of 25% increase (over estimation) of the inductance of the line reactors. The effect of any further increase in the inductance value leads to a similar performance degradation as observed with a positive grid voltage error.

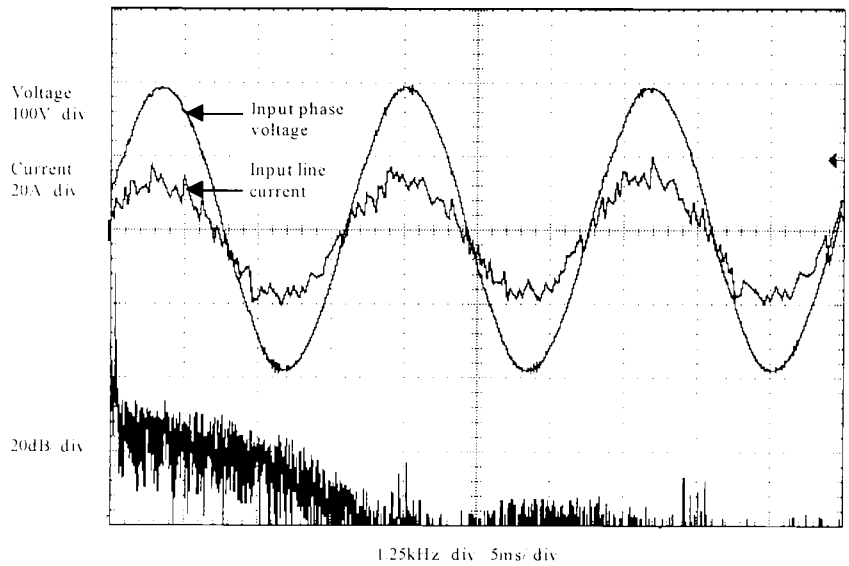


Figure 3.20 Rectifier operation with (-)21% line reactor inductance error.
V_{ac} = 230 V, V_{dc} = 390 V, f_{switch} = 5 kHz, load ≈ 6 kW

Fig. 3.21 depicts balanced three phase operation of the rectifier with a 0.85 leading power factor operation and loading conditions similar to that of Fig. 3.17.

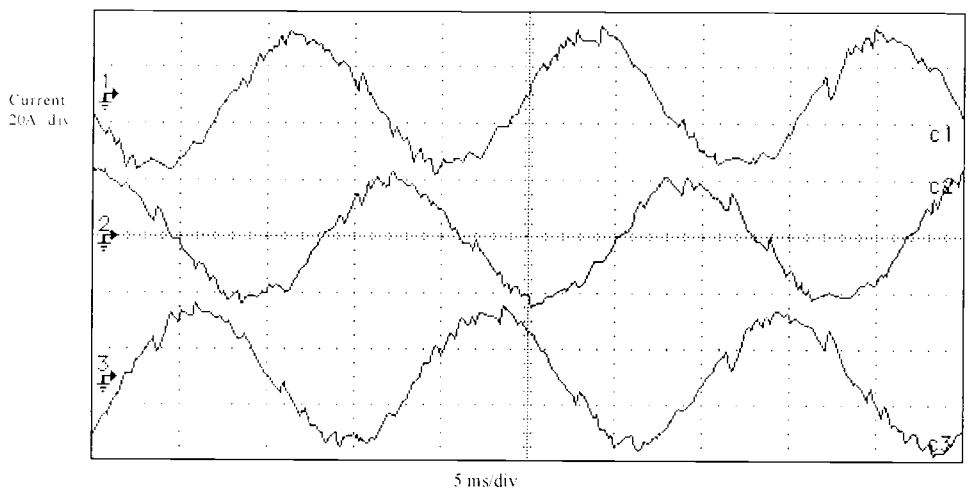


Figure 3.21 Three phase current waveforms for 0.85 leading p.f. operations.
Conditions similar to that of Fig. 3.17.

Table 3.3 Comparison of spectral performance

f_{harm}	Diode-based	Sensor-based	Sensorless Regular	Sensorless (-) 20% Voltage Error	Sensorless (-) 21% Inductance Error
Operational Conditions	V_{dc} = 274Vdc V_{grid} = 230Vac load = 13.7 Adc f_{switch} = N/A Ref. = N/A	V_{dc} = 380Vdc V_{grid} = 230Vac load = 17.4 Adc f_{switch} = 6.67kHz Ref. = Fig. 7	V_{dc} = 380Vdc V_{grid} = 230Vac load = 15.5 Adc f_{switch} = 5.0kHz Ref. = Fig. 8	V_{dc} = 390Vdc V_{grid} = 184Vac load = 15.1 Adc f_{switch} = 5.0kHz Ref. = Fig. 10	V_{dc} = 380Vdc V_{grid} = 230Vac load = 16.0 Adc f_{switch} = 5.0kHz Ref. = Fig. 12
5	-18.0 dB	-36.4 dB	-43.2 dB	-38.4 dB	-41.0 dB
7	-23.2 dB	-44.8 dB	-44.8 dB	-52.7 dB	-39.1 dB
11	-35.2 dB	-41.2 dB	-44.2 dB	-55.4 dB	-42.3 dB
13	-35.2 dB	-50.8 dB	-49.2 dB	-50.6 dB	-42.5 dB
17	-46.4 dB	-50.8 dB	-54.2 dB	-46.3 dB	-47.1 dB
19	-41.6 dB	-49.6 dB	-47.2 dB	-52.9 dB	-44.3 dB

Table 3.3 shows a comparative listing of the low-order harmonics of the input current waveform for the non-current regulated, the sensor based and the model-based current regulated controllers under similar conditions. The error free model-based current regulator compares favorably with the sensor-based version although the sensorless version has a 25% slower switching frequency due to the increase in the computational intensity of the switched rectifier model. Accuracy of switching decisions increases with the increase in the input current magnitude due to an enhanced resolution of the current errors. Thus it is observed in Table 3.3 that the increase in the input current magnitude for the grid voltage error condition (see Fig. 3.18) results in excellent spectral performance, compared to even the non-error scenarios.

3.3. Remarks on the rectifier controllers

The sensor-based controller presented here is remarkably simple compared to existing control strategies for active rectifiers. It provides for a very low component count as most of the algorithm can be implemented in a single low-cost microprocessor, thus increasing

reliability and robustness. The algorithm maintains constant switching frequency at full-load conditions and provides good control of the spectral performance of the input current waveforms. The low-pass filter used for the equivalent load conductance calculation limits the dc-bus voltage dynamic response but helps in averaging the calculated conductance over the selected time window. Thus the proposed algorithm may be a bit slower than the more complex and expensive space vector or synchronous frame based controller, but it is extremely simple and inexpensive to implement.

With no additional hardware, the original sensor based version of the controller was modified to implemented the model-based or sensorless version of the rectifier controller utilizing the same microcontroller. The sensor-based version was implemented with the on-chip A/D which required multiplexing for multiple conversions. Thus simultaneous sampling of the currents in different phases was rendered impossible with the single A/D. For an improved sensor-based performance, off-chip multiple A/Ds or off-chip sample and hold circuitry for all the channels is required. Thus, the model-based current control scheme not only eliminates the need for current sensors, but also A/D converters and the circuits associated with them. While these are direct cost benefits due to the elimination of the current sensors and A/D converters, they also reduce the cost of printed circuit boards due to a significant reduction in size of the boards. Conservation in size reduces cost in layout and fabrication.

Elimination of the current sensors eradicated the offset problems associated with them. Noise problems associated with the analog measurements and A/D converters become non-existent for the current measurements. Thus, it is expected that the sensorless controller would perform better than the sensor-based controller implemented with a single A/D while operating at similar switching frequencies.

Although the estimator requires the parameters of the input line reactors as an input, robustness to parameter variation was demonstrated. Since the active rectifier circuit is a closed loop system through the dc-bus voltage measurement, it can accommodate a

decrease in the inductance value till the controller reaches the maximum input current limit. It was also shown that the controller can accommodate discrepancies between the actual and measured grid voltage by monitoring the dc link voltage. The inaccurate current errors, calculated from the erroneously estimated currents, are compensated dynamically by the dc-bus voltage regulator. There is, however, an upper limit in the magnitude of discrepancies that can be fully compensated for, as discussed in section 3.2.1.6.

In view of its simplicity and ease of implementation using a fixed-point microprocessor the DPM-based switching algorithm was chosen for purposes of this thesis. However, in a commercial implementation where harmonic distortion at high power levels could be a serious concern some form of pulse width modulation (PWM) or space vector modulation (SVM) would be a desirable enhancement to switching controller [37]. While a PWM implementation would require additional analog circuitry to determine the turn-off instants of the switches in conduction, the SVM method might require a floating point microprocessor, for its implementation, capable of providing the required control bandwidth despite the additional computational overheads.

3.4. Inverter Controller

Many efficient inverter control algorithms are available in the literature, most of them variations of the fundamental sinusoidal PWM strategy. For purposes of simplicity of the algorithm with respect to its use of computational resources, discrete pulse modulation strategy as explained in section 3.1.1.4 was chosen. The state machine of the controller executes at a fixed switching frequency, although switching occurs only if the measured current errors are outside a predetermined band. Typically, the error band is set at a value to reflect and ignore the noise floor in the circuitry.

The controller of the output stage (inverter) of the ac/ac converter is a subset of the rectifier controller with respect to its current waveform shaping. Hence, all the principles

of the required switching decisions are similar to that of the rectifier switching algorithm as described in section 3.1.1.4. It should be noted here, though, that the switching logic is reverse of that of the rectifier. Thus, positive errors turn-on the upper device and negative errors turn-on the lower device. A faster switching speed could be attained with the inverter controller than its rectifier counterparts, since the computational overheads of the inverter are significantly relaxed than those of either of the rectifier as described in sections 3.1 and 3.2. Representative waveforms of the inverter are provided in Chapter 5 as part of the report for the overall system performance.

4. System Implementation

The model-based optimization controller, as described in Chapter 2, was developed and implemented in a laboratory VSG wind generation model to verify its efficacy. The laboratory model utilized of a motor-generator test rig consisting of a dc machine and a BDFM mechanically coupled through a flexible coupling. Shaft speed and torque were measured via a strain gauge transducer. The measurements were acquired analog through a multichannel sample/hold circuit and stored digitally in a computer for data processing through a multichannel A/D circuit. The dc machine was set up as a motor with a torque-speed profile similar to that of a wind turbine driving the BDFM utilizing speed and torque measurement signals for closed-loop control. Figure 4.1 illustrates a block representation of the overall system implementation.

A 115 V, 1500 Watts (at 1800 r/min) BDFM was utilized for the experimental evaluation of the performance optimization controller. In order to interface the control winding of the BDFM, a four-quadrant ac/ac power converter was developed. The sensor-based active rectifier controller, as described in Chapter 3 (section 3.2) was utilized to control the rectifier stage (converter 2 in Fig. 4.1) of the converter. Over-voltage and over-current protection for the converter was provided by an inexpensive Atmel AT2051(Intel 8051 compatible) based protection logic. Other dynamic protection requirements were implemented inside the rectifier controller logic for faster responses.

A Texas Instruments TMS320C3X digital signal processor based A/D converter circuit analyzed the total system power. This measured total power was the feedback variable for closing the system control loop. As described in Chapter 2, maximization of the total output power was the objective of the performance optimization controller.

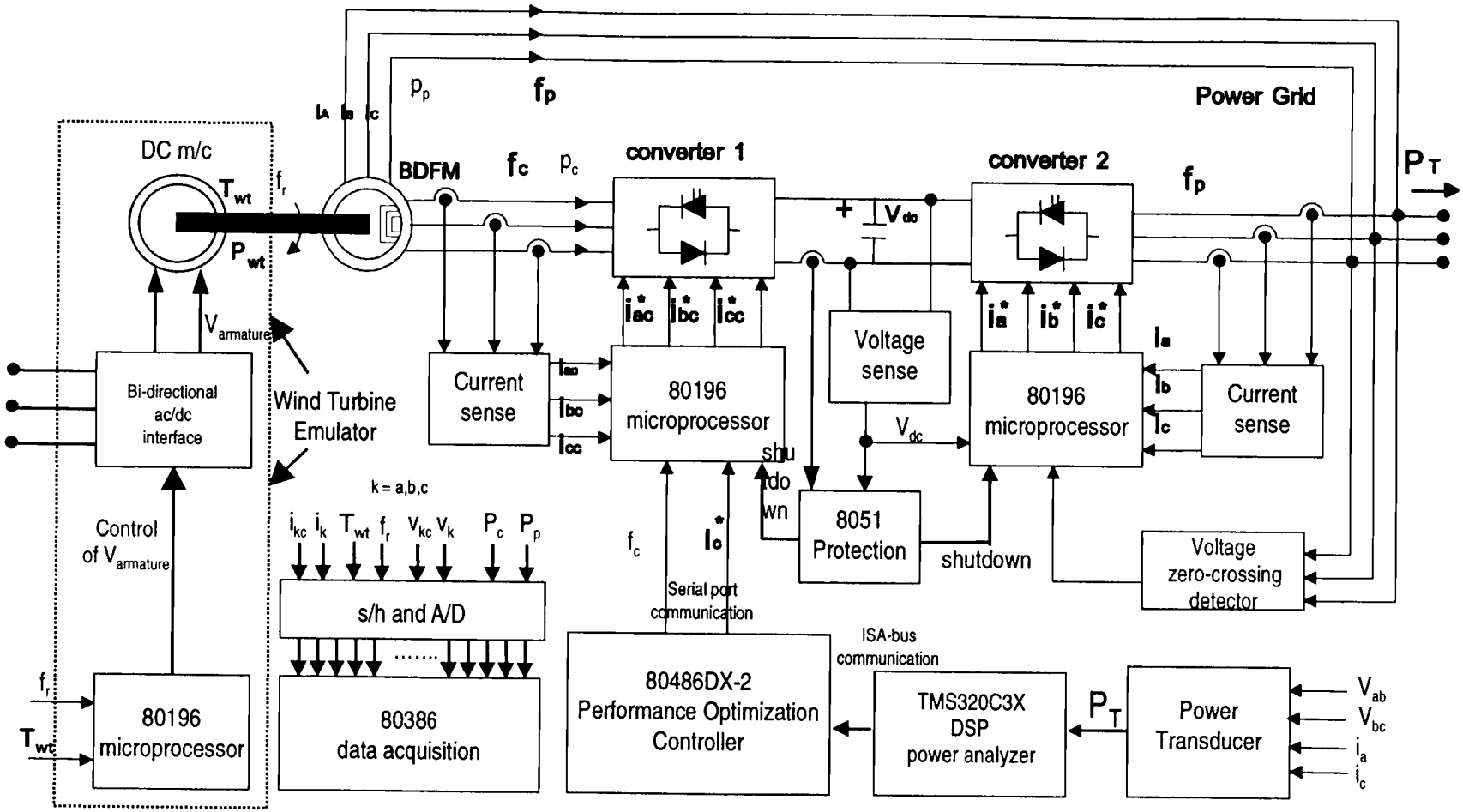


Figure 4.1 Laboratory VSG wind power generation system implementation

The optimization controller was implemented in a high-level computer language (C-language) in a standard 80486-DX2 desktop computer. The availability of the basic graphics libraries for developing the user interface for open-loop and closed-loop modes of operation aided in the decision to utilize the desktop computer. While in the present implementation the power measurement and the controller are located in separate processors both can be adequately programmed utilizing the resources of only the digital signal processor without any negative impact on the control bandwidth.

Proof of performance of the proposed system controller is significantly dependent on the availability of valid experimental data for various system quantities such as currents, voltages, power, shaft torque and speed. Hence, a 80386-desktop based data acquisition system with a 16-channel sample and hold and a multiplexed A/D converter was employed to gather real-time data both for system training and experimental validation. Available rms current, voltage and instantaneous power transducers provided the inputs to the data acquisition system.

4.1. Wind turbine model development

As illustrated in Fig. 4.1 a dc-machine torque controller was developed which could be adapted to effectively model a wind turbine operating under varying wind speeds. The wind turbine model controller utilized the torque and speed signals available from the transducers on the test rig to enable closed loop operation for both torque and speed control.

The dc machine drive employed for the purpose consists of a thyristorized bi-directional ac/dc interface. By controlling the phase angle of the thyristor gating signals, the armature voltage for the dc-machine can be controlled. The drive interface provided an analog interface for commanding the armature voltage of the dc-machine. Hence, the wind turbine model controller regularly updates the control variable, the desired analog armature voltage command, as indicated in Fig. 4.2. While the control loop is updated

every 1.5 ms, the actual dynamics of the dc machine are finally dependent upon the time constants of the inner phase angle controller and those of the machine. Appendix C lists the software code utilized in the implementation of the torque and speed controllers for the dc machine.

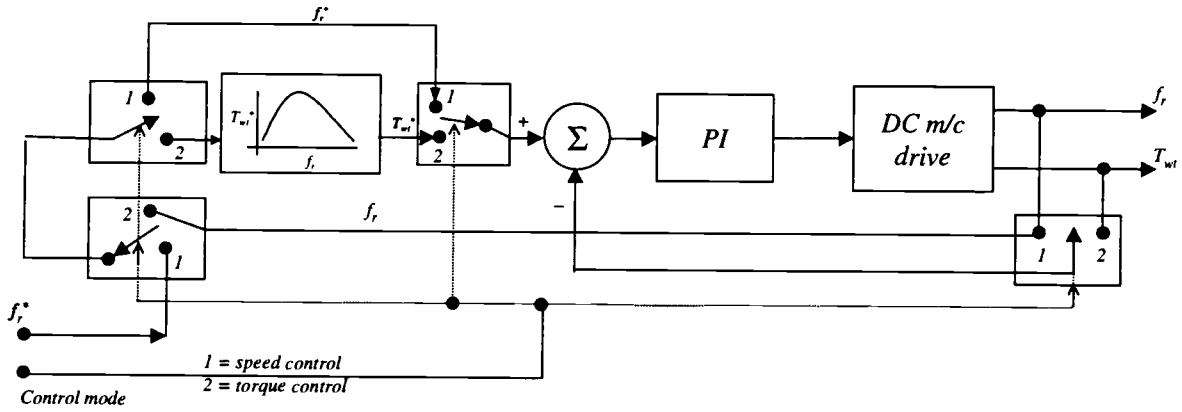


Figure 4.2 DC machine based wind-turbine model controller.

4.1.1. Wind turbine emulator

The speed of the VSG system, f_r, is set by the BDFM operating in the synchronous mode. Hence, the mechanical torque input to the electrical generator is set by the wind turbine which develops an aerodynamic torque by converting kinetic energy in the wind to mechanical energy at the shaft. The mechanical power delivered to the shaft follows the C_p - λ profile of the turbine as shown in Fig. 1.1. The torque-speed (T_{wt} – f_r) curve varies with the wind speed, v, as the net power delivered follows eq.(2.4).

For the system in the laboratory, the design of the BDFM and the mechanical limitations of the coupling constrained the speed range of operation of the VSG system to between 1200 to 1800 r/min with a maximum shaft torque of 70 lb-in. Based on these restrictions and a wind speed of 10 m/s, the power – speed (P_{wt} – f_r) curve developed for the wind turbine is as shown in Fig. 4.3. This power curve was derived from the C_p - λ profile shown in Fig. 4.4, which is similar in shape to that of Fig. 1.1.

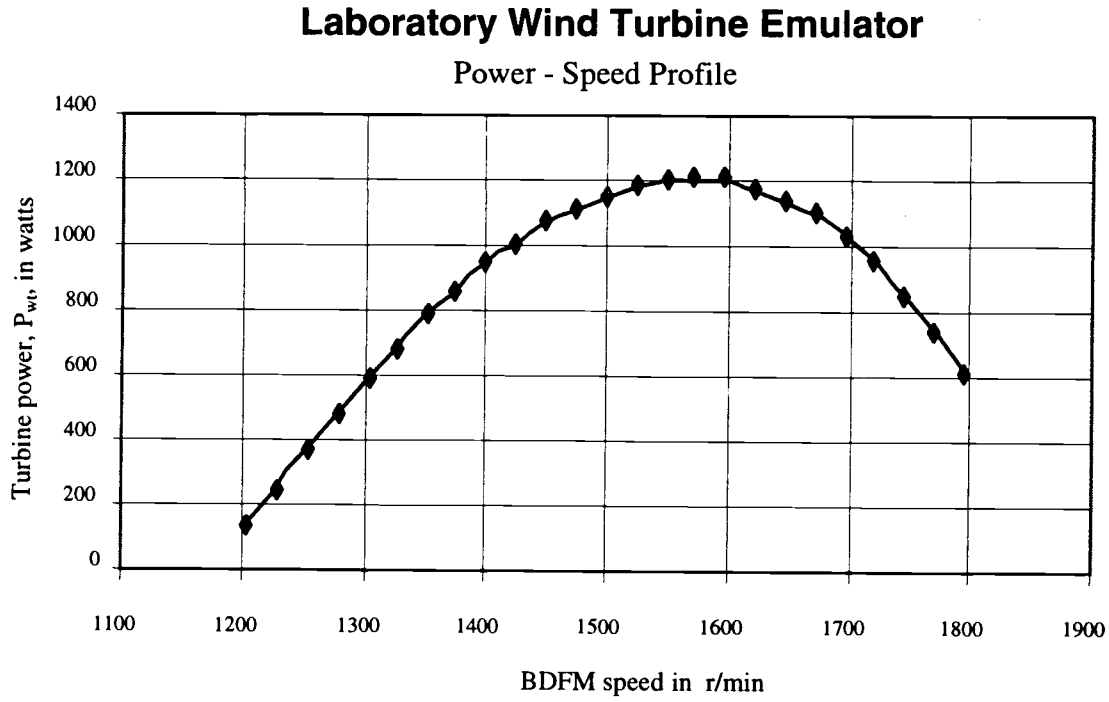


Figure 4.3 Desired mechanical power output of the laboratory wind turbine emulator.

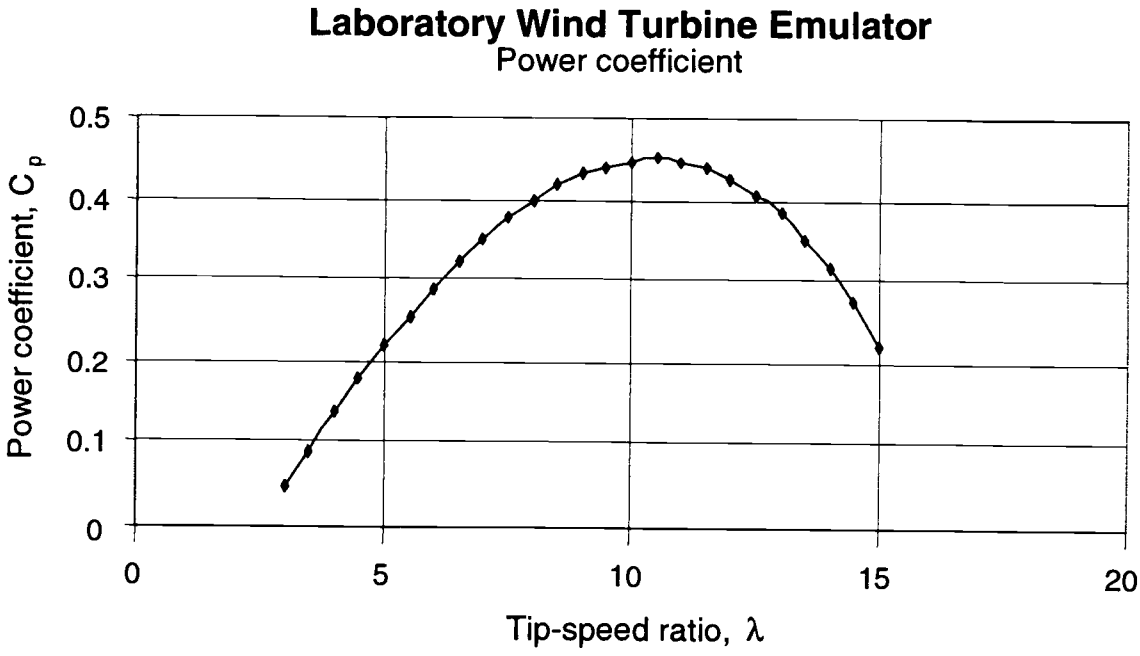


Figure 4.4 Power coefficient of the laboratory wind turbine emulator.

The torque profile for the wind turbine model was derived from the power-speed curve in Fig. 4.3 utilizing

$$T_{wt} = \frac{P_{wt}}{2\pi f_r} \tag{4.1}$$

The desired wind turbine torque is plotted in Fig. 4.5. Regression analysis of the torque as a quadratic polynomial in shaft speed is employed to generate the coefficients for the curve-fitting. This polynomial function of the shaft speed, as shown in Fig. 4.5, is utilized as the torque profiler or the torque reference in the wind turbine model controller. The mechanism is illustrated in Fig 4.2.

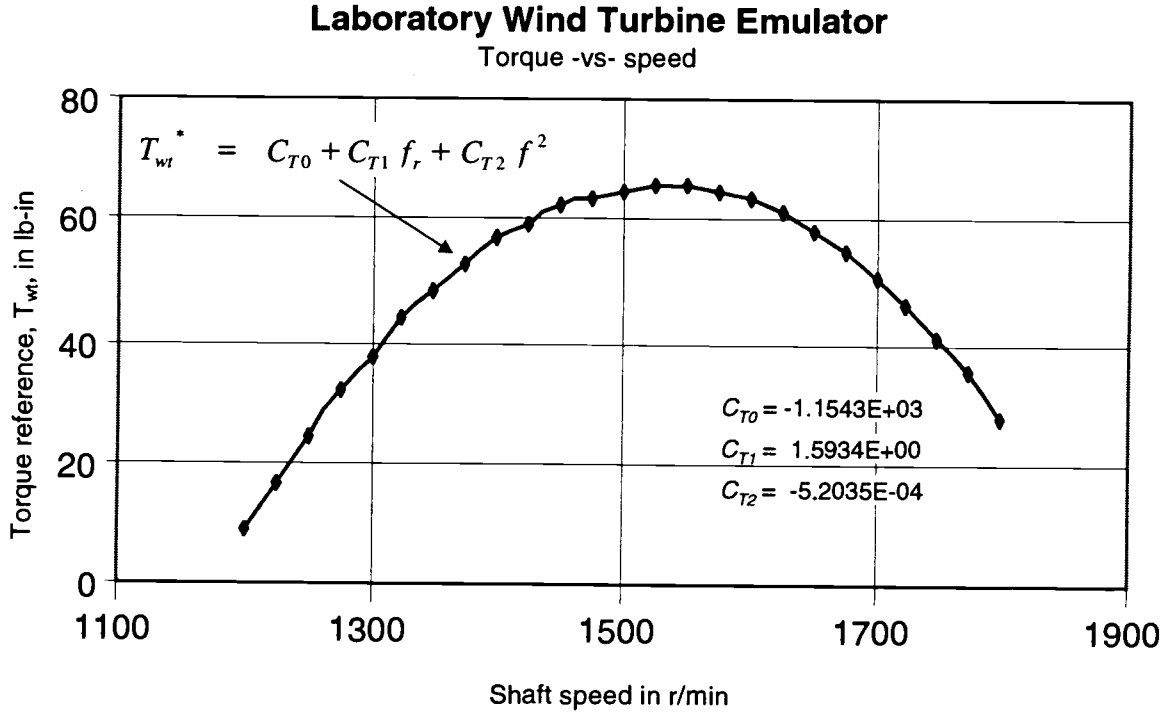


Figure 4.5 Desired torque-speed characteristic of the wind turbine emulator.

Although during normal wind turbine operation the turbine emulator operates in the torque mode, a speed mode for startup operations is also featured. The speed controller is described in section 4.1.2. While performance of the system has been adequately investigated for only one wind speed, the turbine model emulator can be readily adapted for varying wind speed operation.

4.1.2. Speed controller

The turbine model controller is equipped with a speed control mode of operation for system startup purposes as described in section (?). It has been implemented with a proportional-integral control law. In the speed mode, the speed reference, f_r^*, is a user input. The measured error in the dc-machine speed is utilized for generating an armature voltage reference, v^*_{arm}, given by

$$v^*_{arm}(t) = K_P \varepsilon_{fr}(t) + K_I \int \varepsilon_{fr}(t)\,dt \qquad (4.2)$$

implemented digitally as

$$v^*_{arm}(n) = v^*_{arm}(n-1) + K_I \varepsilon_{fr}(n) + K_P\left[\varepsilon_{fr}(n) - \varepsilon_{fr}(n-1)\right] \qquad (4.3)$$

where K_P and K_I are the proportional and integral constants, respectively, and ε_{fr} is the error in the speed of the dc-machine.

4.1.3. Torque controller

This is the normal mode of operation of the turbine emulator. The BDFM sets the speed of operation for the VSG system. Based upon the measured speed, a torque reference, T^*_{wt}, is determined utilizing the profiler as depicted in Fig. 4.5. The error in the dc-machine torque with respect to the commanded torque sets the required armature voltage reference, v^*_{arm}, for the dc machine interface. This is achieved by employing a proportional-integral control law similar to that of the speed controller in section 4.1.2. The result of the performance of the torque controller is presented in Fig. 4.6.

4.2. BDFM Generator

While this thesis merely utilizes a BDFM as a doubly-fed generator to demonstrate as a proof of concept the performance optimization controller, it is relevant to discuss the benefits of a BDFM based VSG system and adjustable speed drive (ASD) as compared to standard induction machine (IM) drives. While the BDFM fundamentals have been introduced in Chapter 1, this section will succinctly review the BDFM, its advantages over other variable speed systems and details of the prototype employed in the experimental evaluation of the system controller.

4.2.1. Features of a BDFM drive

The high cost of all singly-fed VSG systems and adjustable speed drives (ASDs), compared to a direct line connected IM, is an impediment to greater market penetration. If the cost of the power electronic converters needed in ASDs could be reduced substantially, "payback" times would be reduced, enabling wider applications and increased overall energy savings. One approach to lower the cost of inverters, by significantly reducing their rating, is to employ the configuration called the brushless doubly-fed machine (BDFM). The BDFM is a self-cascaded IM with both the control and power windings on the stator and a modified cage rotor. When operated "synchronously" in the doubly-fed mode, the BDFM requires only a fraction of the electrical power to be processed by the power electronic converter. Thus, the overall cost of the system is brought down to a fraction of the cost of that of an ordinary cage rotor IM ASD while providing a robust, adjustable speed drive [26,38-40] with precise speed control.

A single-frame self-cascaded IM, which is capable of operating in both an induction and synchronous mode, has been actively researched since the concept was first developed by Hunt [41,42]. The advent of power electronic converters capable of adjustable frequency, adjustable voltage and bi-directional power flow has made flexible operation of the self-cascaded machine more feasible. This new VSG or ASD configuration is referred to as a BDFM.

To avoid direct transformer coupling, it is essential that the two windings on the stator have different pole numbers. Moreover, to avoid unbalanced magnetic pull, the pole numbers must be separated by more than two. Hence, laboratory prototypes have, so far, concentrated on the most simple configuration of 6-pole power and 2-pole control windings, even though various other pole-number combinations are possible. The rotor has a specialized cage structure with the number of identical sections, or nests, equal to the sum of the pole pairs of the two stator windings. Due to problems of instability and excessive losses arising from internal circulating currents, two isolated 3-phase stator windings are preferred over a single stator winding connection that produces two magnetic fields of different pole numbers.

4.2.2. Generator design

The 115 V BDFM utilized in the experimental evaluation of the system controller had been designed for a proof-of concept VSG wind power application [27]. The 6-pole power winding ($p_p = 3$) and the 2-pole ($p_c = 1$) control winding allow for a generator shaft speed range of 1200 through 2000 r/min. With this configuration, the desired speed range can be attained with an inverter capable of outputs between 20 Hz and 70 Hz. Also, for the pole numbers and speed range of operation, the rating of the inverter can be expected to be less than or equal to 25% [i.e. $\frac{p_c}{p_p + p_c}$]. Details of the machine design can be found in [27] but have been provided here in Tables 4.1 and 4.2 for easy reference.

Table 4.1 BDFM generator design specifications [27]

Frame size	182
Rated voltage	115 V
Rated speed	1800 r/min
Rated output power	1500 watts
Design efficiency	77 %

Table 4.2 Winding specifications [27]

6-pole winding	Double layer 15 turns/coil #16 AWG
2-pole winding	Double layer 10 turns/coil #19 AWG
4-pole Rotor cage	Round copper bars #4 AWG

4.2.3. Characterization of the generator

The performance optimization controller, as discussed in Chapter 2, requires detailed information about the BDFM generated power and its efficiency with respect to the the varying control winding current. The information of the I_c – P_T profile helps in the determination of optimum control winding current settings for maximized efficiency of the BDFM. Concurrently, the efficiency-vs- I_c characteristic of the BDFM is utilized in the model-based MPPT algorithm presented in Chapter 2.

The dc-machine operating as a wind turbine model was utilized to characterize the prototype BDFM. In order to verify the performance of the system controller under conditions different from those during characterization, the programmed torque-speed profile for the open-loop tests was different from that used for closed loop experimental evaluation. Furthermore, the choice of different operating conditions for the open-loop characterization and the closed-loop evaluation would simulate real-life conditions more accurately. Also, the controller would then be evaluated for its adaptive capability for changing operating conditions, namely the wind speed. Figure 4.6 illustrates the difference in the open-loop characterization and closed loop evaluation turbine torque (T_{wt}) –vs– speed profiles.

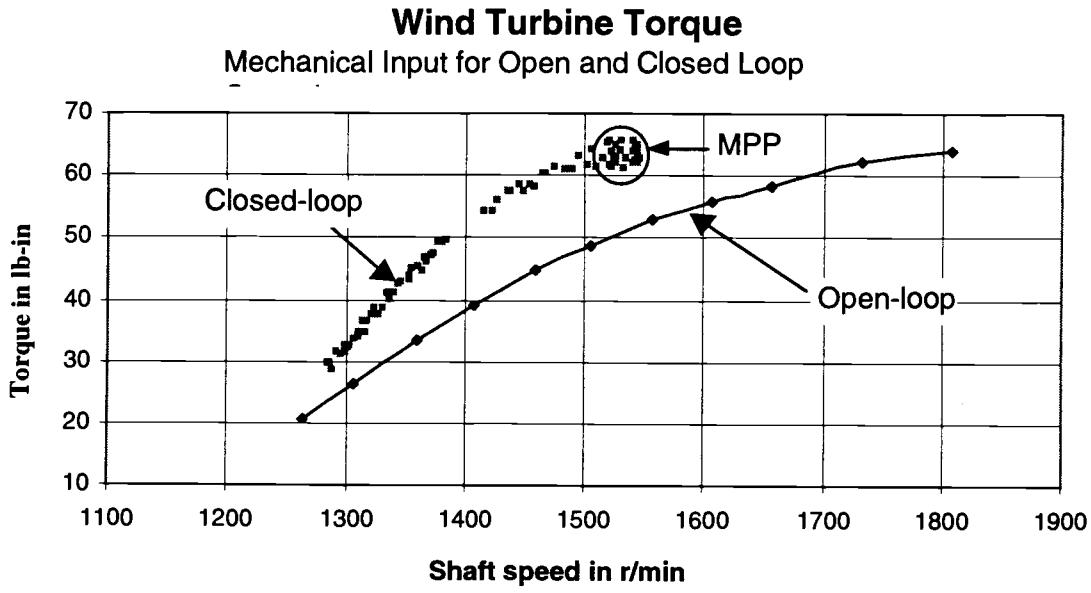


Figure 4.6 Measured turbine model torque. Closed-loop operation illustrates the effectiveness of the optimization controller and turbine convergence at the set MPP.

With the open-loop torque-speed profile in effect, the BDFM was characterized for the its optimum control winding current for maintaining maximum efficiency operation (I_c – P_T profile) at randomly selected operating points. Hence, with the BDFM maintained at certain speed and torque conditions as per Fig. 4.6, the control winding current was adjusted to empirically determine the maximum output power points for different mechanical input power. The experimental characterization obtained for the prototype BDFM is shown in Fig. 4.7.

Coincidentally, for most of the mechanical input power range the prototype 1.5 kW BDFM exhibited maximum efficiency point operation at the lowest possible control winding excitation. Lower current excitation on the control winding drove the BDFM out of synchronism and, hence, could not be represented in the figure. This was, however, not observed for a different 7.5 kW BDFM. Referring to Fig. 2.6, it is seen that the maximum efficiency points are distinctly different from the lowest possible control winding current excitation to maintain synchronism.

Figure 4.7 is utilized to generate the $I_c - P_T$ profile required for the MEPT loop to maintain maximum efficiency operation of the BDFM. This is done by noting the optimum operating points in Fig. 4.7 for varying input mechanical power. The $I_c - P_T$ profile, thus obtained, for the prototype BDFM is depicted in Fig. 4.8. The figure also illustrates the polynomial curve-fit and its coefficients as registered in the MEPT controller for closed-loop operation.

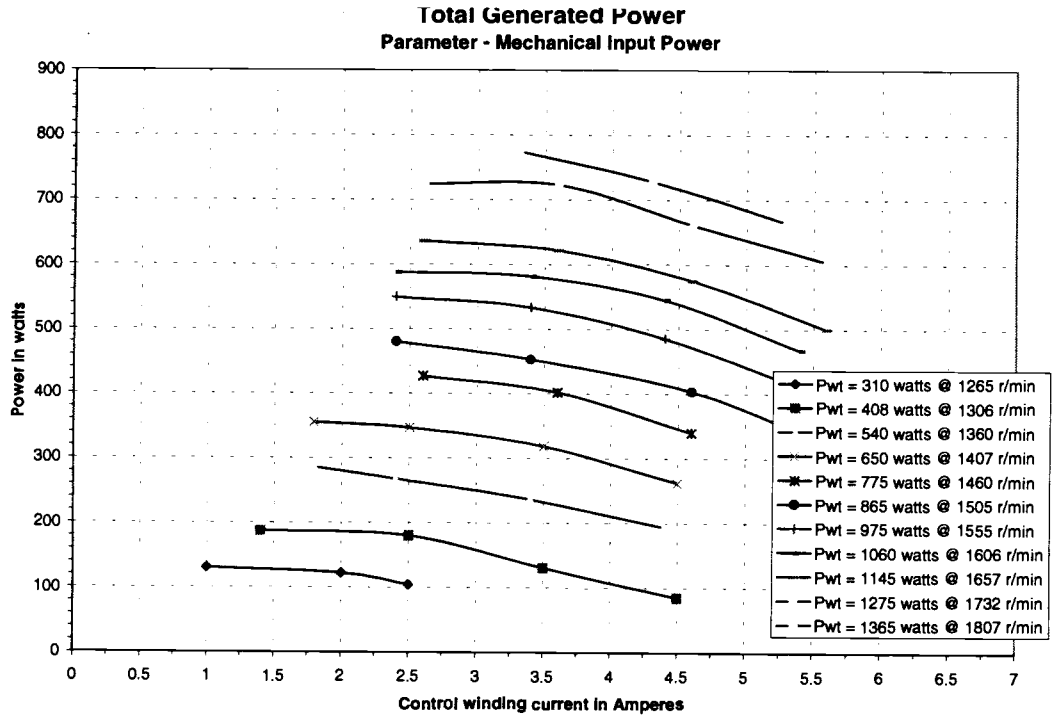


Figure 4.7 Variation of generated power with varying of control winding current for different input mechanical power.

The measured maximum efficiency operating points are as shown in Fig. 4.9. This information is necessary for the wind speed estimation based MPPT algorithm as discussed in Chapter 2 and implemented in the system controller. Again, as in Fig. 4.8, the information required to plot Fig. 4.9 was obtained from the test run that was used for Fig. 4.7.

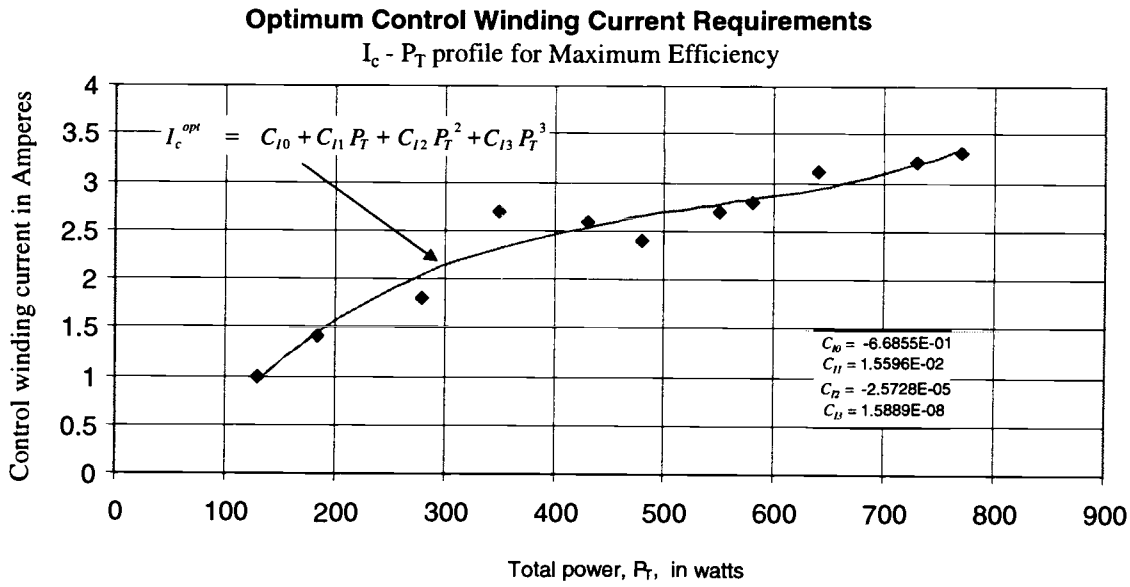


Figure 4.8 Optimum control winding current requirements for maximum efficiency operation of the prototype BDFM. The curve-fit polynomial, as shown, is the MEPT controller realization.

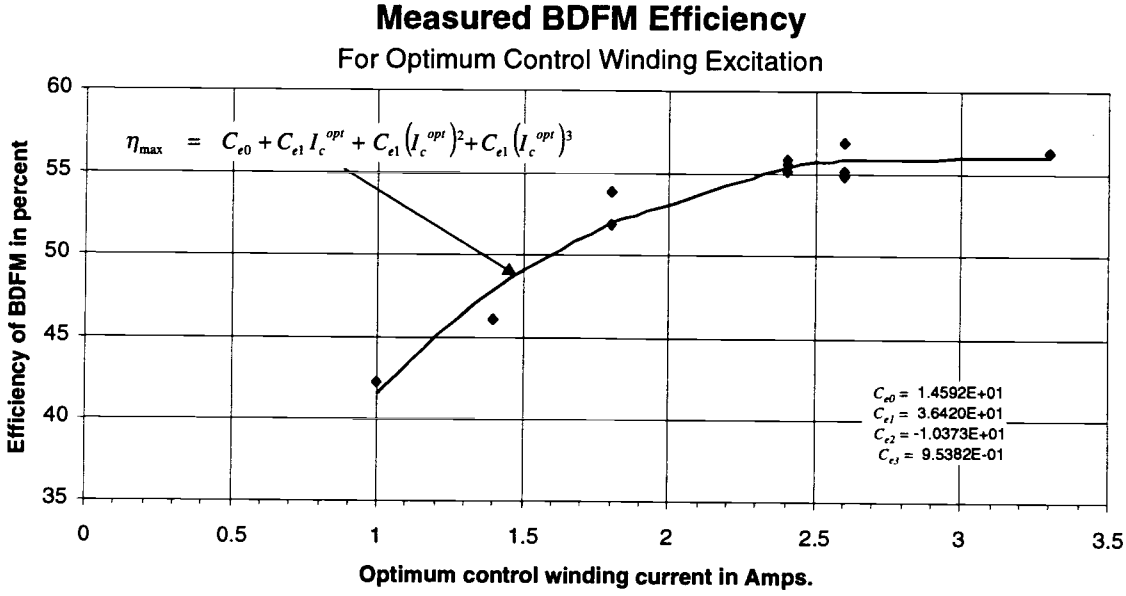


Figure 4.9 $\eta_{max} - I_c^{opt}$ profile of the prototype BDFM as implemented in the adaptive MPPT controller

While higher order polynomials would provide for a closer fit of the experimental data in both Figs 4.8 and 4.9 the curve-fit polynomials were chosen to be only of the 3rd order. This was done deliberately due to two real-life concerns. First, the fewer the number of on-line computations, the faster the controller updates. This also leaves some control bandwidth to bring in advanced control features such as updates on the coefficients of the polynomials to reflect the changes in machine parameters over time. Secondly, the adaptiveness and robustness of the model-based performance controller could be demonstrated by deliberately utilizing crude model approximations. Here again, closer real-life scenarios can be enacted whereby the controller would be called upon to perform adequately under operating conditions which could be significantly different from those during model derivation. Notably, the operating conditions may vary due to different wind speeds, changes in physical parameters of the generator, wear and tear and different weather conditions.

4.3. Power converter implementation

The power converter design issues will be briefly described here. The algorithm and some of the implementation details have already been provided in Chapter 3. The active rectifier was implemented with the sensor-based version, mainly due to the fact that the dynamic performance of the sensorless rectifier controller remains to be investigated. Nonetheless, as mentioned in Chapter 3, there needs to be no hardware modification to implement the sensorless version of the controller.

4.3.1. Converter components

The converter consists of two stages of conventional hard-switched three-phase voltage source topology. Both stages were implemented using Insulated Gate Bipolar Transistors (IGBTs) due to their superior conduction characteristics at high voltages and switching frequencies as compared to power MOSFETs. In order to provide sufficient over capacity in the prototype converter system, IXYS IXGH20N60U1 modules rated at

600 V, 20 A were utilized although this could, potentially, penalize the overall efficiency of the generation system. However, the lower efficiencies measured still could be utilized in verification of the proof-of-concept optimization controller.

While Chapter 3 discusses the general control algorithms and their development for the rectifier and the inverter stages, most of the protection issues will be discussed here. The schematic representation of the protection strategy in this implementation is illustrated in Fig. 4.10. Please refer to Appendix A for the implemented software code.

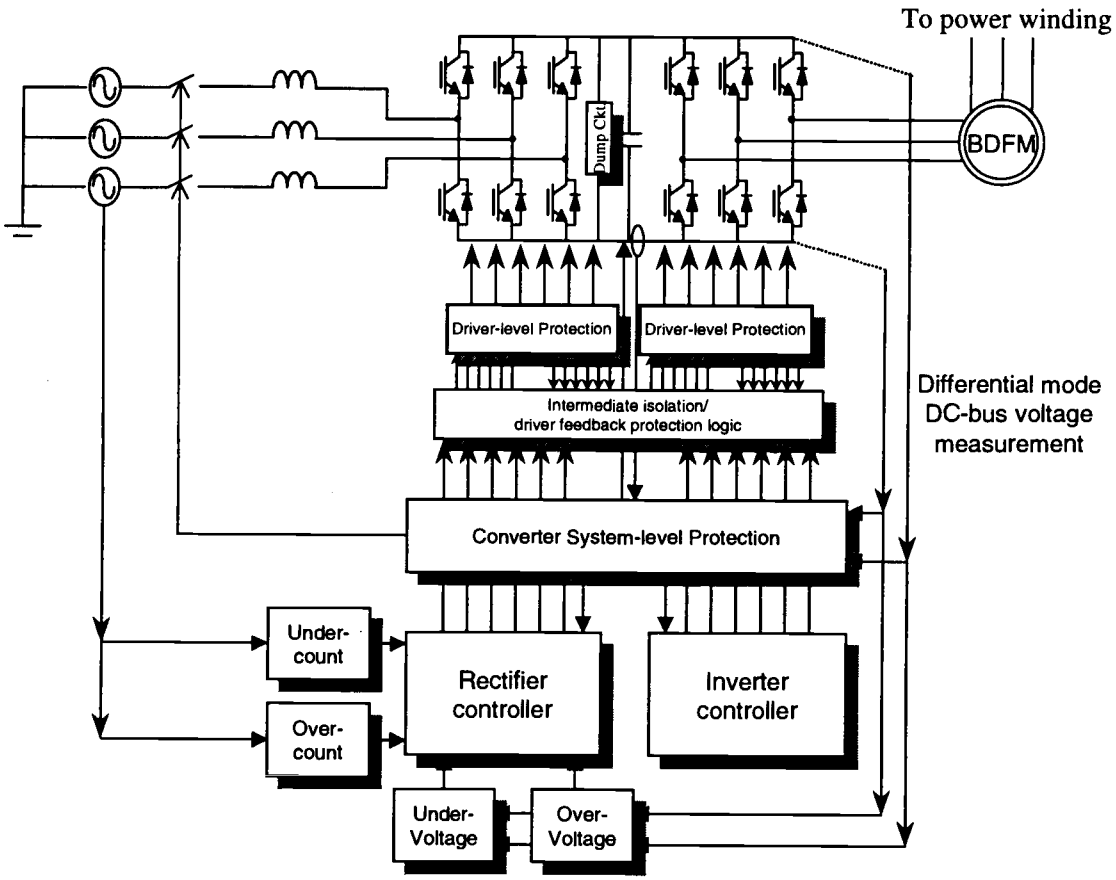


Figure 4.10 Overall converter protection strategy.

4.3.2. Converter protection

Due to the discreteness of the components in this implementation, various levels of protection were necessary. The protection strategy can be broadly classified into the following levels:

(1) Protection at the driver boards for short-circuit currents and control under-voltage;

(2) Protection at the converter system level for over-voltage and over-current; and

(3) Protection in the rectifier controller during every switching cycle.

4.3.2.1. Driver level protection

A desaturation protection circuit is implemented on the device driver modules. This desaturation circuit is based on the principle that the conduction voltage drop or the saturation voltage across the IGBT collector and emitter ($V_{CE,sat}$) increases with increased collector current. The circuit is designed to signal the IGBT driver (IR2121 for the low-side and IR2125 for the high-side) that the threshold $V_{CE,sat}$, has been hit. The drivers go into protection mode by reducing the gate voltage drive and withdrawing the IGBTs from hard saturation. They also provide a feedback signal that can be utilized in cycle-by-cycle shutdown of the system. In the present implementation, the feedback signals of all the drivers for each stage are "OR-ed" together, which shuts down that stage till the short-circuit current is removed. During the time while the IGBTs go out of saturation due to the short-circuit current protection, the freewheeling diodes are left active and current waveshaping is sacrificed.

The drivers are themselves protected for a driver power supply failure. The IGBTs are brought out of saturation by lowering the gate voltages if an error occurs in the driver control power supply.

4.3.2.2. Converter system level protection

A dedicated circuit has been designed to provide over-voltage and over-current protection. While the over-current protection allows the load to to be serviced and normal operation to continue for some time, the over-voltage protection is more critical as the devices are absolutely non-forgiving for over-voltage conditions. Both the error conditions are serviced by an Atmel AT2051 (compatible to Intel 8051, with half the pinout count) microcontroller. The same microcontroller is utilized for turn-on, turn-off of the system contactors and the in-rush protection resistors.

During conditions of sudden regeneration by the BDFM generator, the rectifier controller may not respond to the fast changes in the regenerated power which may result in an over-voltage condition on the dc-bus subjecting the electronic devices to severe stresses. If the situation is not corrected fast enough, it could escalate beyond device ratings and destroy the electronic switches. Designed with a hysterisis, the over-voltage protection turns-ON and OFF a dump IGBT which discharges the dc-bus capacitor for a predetermined period of time (200 μs) through a braking resistor. This typically reduces the dc-bus voltage level sufficiently to restore normalcy and allow the rectifier controller to increase the regenerated power. As a final protection, the protection signals the rectifier and inverter controllers and opens the contactors if "dumping" of energy was not sufficient to reduce the dc-bus voltage to safer levels.

The over-current is based on the dc current measured directly on the dc-bus. As mentioned earlier, it allows normal operation to continue for half a minute before signaling the individual rectifier and inverter controllers to proceed for shutdown.

4.3.2.3. Rectifier code-level protection

The rectifier protects itself from a lack of synchronization with the grid. If it detects too many irregular grid zero-crossings, it shuts down the switching of the devices. This is necessary to prevent disastrous oscillations on the dc-bus resulting in a loss of control.

The rectifier also provides for both dynamic and steady-state under-voltage on the dc-bus. An under-voltage condition, when the controlled voltage is lower than that of the diode-based rectifier, could potentially turn-ON devices when the complementary diodes are conducting. This is a shoot-through condition, whereby the dc-bus is discharged through a phase-arm, destroying the circuit. Although redundant, an over-voltage protection also exists in the rectifier algorithm where it serves to protect the system, in the unlikely event of the failure of the dump-circuit.

4.4. System controller implementation

Figure 4.11 illustrates the block diagram of the performance optimization controller. Both the control loops the MPPT and the MEPT as shown in the Fig. 4.11 can be individually controlled by the user interface. However, the operation of the MPPT loop, as implemented with the wind speed estimation strategy, requires the operation of the MEPT loop. Hence, the optimum control winding current, I_c^{opt}, calculation is common to both the loops.

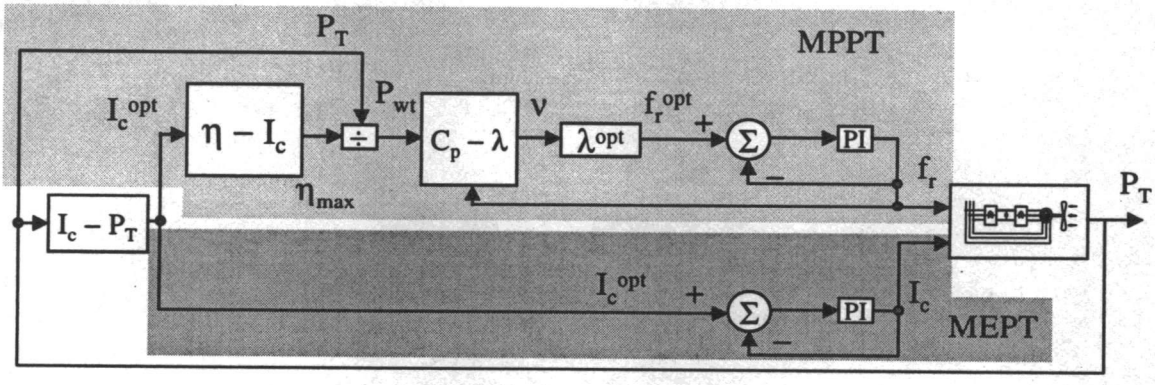


Figure 4.11 Block diagram of the performance optimization system controller as implemented in the laboratory VSG system.

The control blocks of the performance optimization controller of Fig. 4.11 has been implemented in an Intel 80486-DX2 microprocessor based desktop computer, as depicted

in Fig. 4.1. The power measurement and analysis was out-sourced to a Texas Instrument evaluation module consisting of a TMS320C3X DSP. Although all the developmental software and the controller could have been designed based on the TMS, the available graphics libraries for the Intel 80486-DX2 provided a faster developmental turnaround. In a real-life implementation, where extra adaptation of the basic controller would be necessary, a DSP (TMS320C3X) based system would, perhaps, be recommended to implement the total system controller due to the superior computational resources available with it.

The controller was developed in ANSI C and compiled using Borland C++ for DOS. The code development follows the algorithm described in section 2.2.1.2, i.e., the wind speed estimation based power maximization strategy. The program listings are in Appendix B.

4.5. Data acquisition

While the data acquisition for the experimentation of the system is not directly involved with the optimization controller it is an integral part of the validation of the theory. Hence, the software code developed for the multichannel data acquisition based on an Omega A/D hardware system has been presented in Appendix D along with the necessary hardware details.

The graphical user interface (GUI) is the supervisory shell which conducts all file handling, data acquisition, data collection, data processing, data storage and data display. Data is sampled on all 16 channels with a sample and hold circuitry and acquired through a multiplexed A/D through an input port of an Intel 80386 based desktop. Depending on the type of operation of the system, the number of data samples that need to be averaged over a moving time-window can be varied. The data acquired for all experimental results presented in this thesis averaged 2000 samples on each channel over a span of approximately 2 secs. This was necessary to smooth out effects of small perturbations on

the current and voltage measurements always present in the system. The software code of the data acquisition is presented in Appendix D.

5. System Controller Evaluation

With the laboratory based VSG system implemented as discussed in Chapter 4, performance of the optimization controller was verified utilizing the wind turbine emulator. Although performance was verified at a single wind speed, the system controller responded favorably to conditions which were considerably different from those used to characterize the generator model and controller parameters. It should also be noted that the controller estimates the wind speed on-line following the algorithm outlined in Chapter 2. During the experimental evaluation, the controller estimated the wind speed to be around 10.8 m/s. The wind turbine emulator was set to provide an equivalent wind speed of 10 m/s. The difference in the estimated wind speed and the set wind speed in the emulator can be attributed to the fact that, as shown in Fig. 5.1, the wind turbine emulator is not able to closely follow the desired mechanical power input. Hence, the controller is able to optimize the generated power even for varying wind conditions.

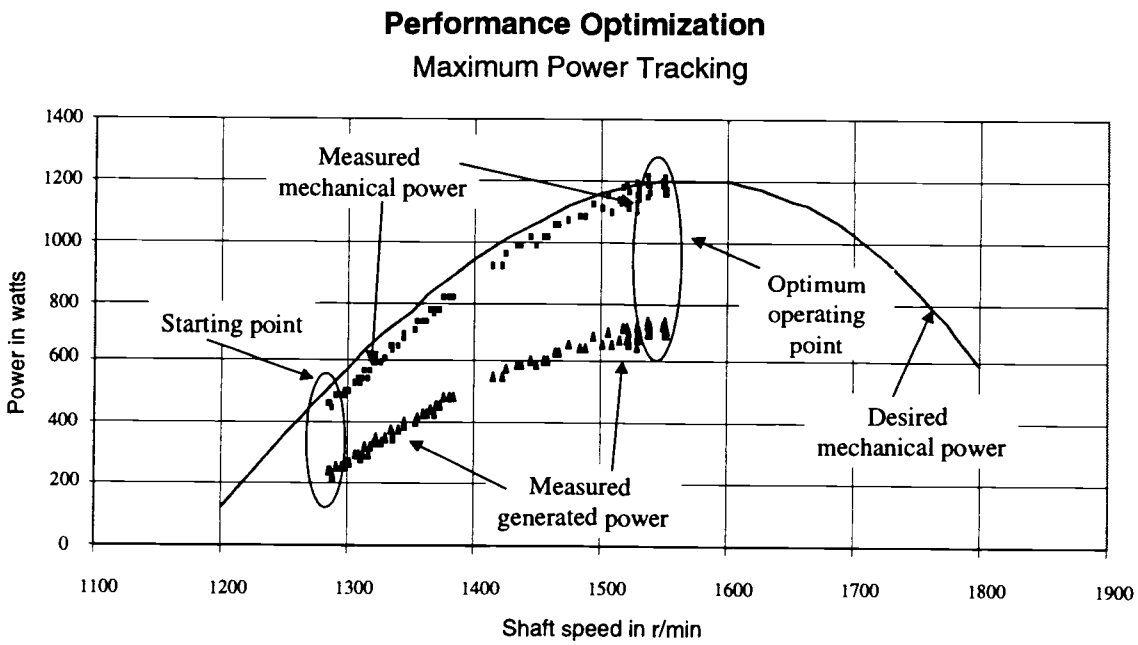


Figure 5.1 Closed loop operation of the performance optimization controller.

5.1. Output power maximization

Figure 5.1 illustrates the operation of the performance optimization controller in the power maximization mode of operation. Both the control loops, namely the MEPT and MPPT loops, were closed at a shaft speed of approximately 1275 r/min. As shown in Fig. 5.1, the controller tracks the maximum power point successfully while maintaining maximum BDFM efficiency.

Figure 5.2 depicts the efficiency trajectory of the BDFM during the maximum power point tracking run of Fig. 5.1. It illustrates the validity of the $\eta_{max} - I_c^{opt}$ model of the BDFM obtained during the characterization of the generator, as mentioned in Chapter 4. The model is discussed in detail in section 4.2.3. The curve representing the estimated BDFM efficiency, as plotted in Fig. 5.2, was calculated off-line by substituting for the measured I_c^{opt} data points into the polynomial expression for η_{max} shown in Fig. 4.9. This, closely reflects the process of on-line estimation of the BDFM efficiency which is required for the wind speed estimation based MPPT algorithm presented in Chapter 2. Although only results of a single run are provided here, the controller was verified for different initial conditions by closing the loop at various other shaft speeds.

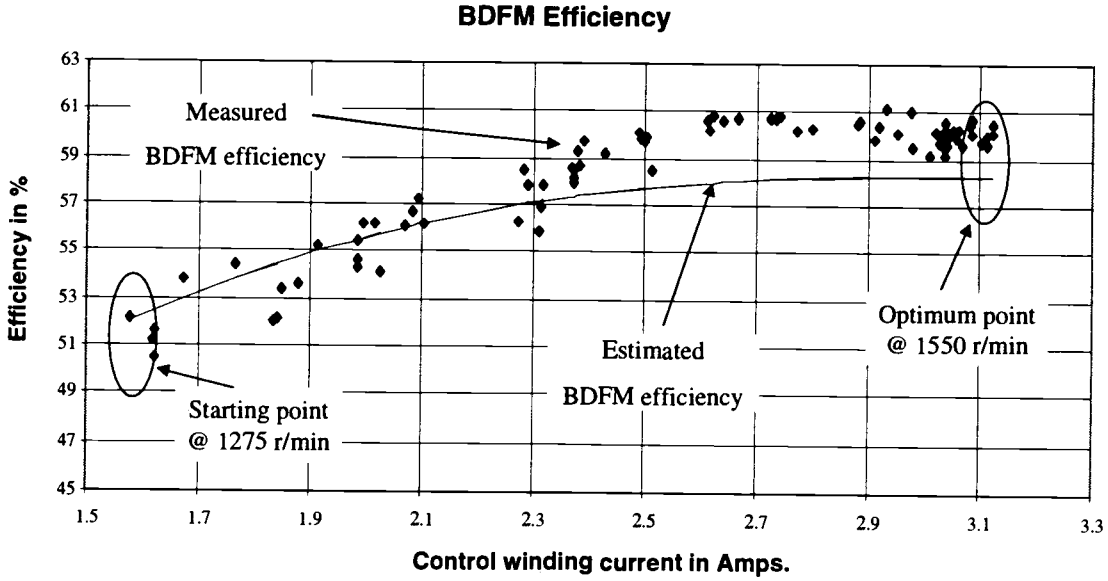


Figure 5.2 Maximum efficiency point tracking by the performance optimization controller.

5.2. Efficiency maximization

It is seen that the measured efficiency points follow the estimated efficiency relatively closely, given the amount of variation in the power measurement at a given operating point due to the noise pickup of the transducers and A/D resolution problems. The estimated efficiency has been derived utilizing the polynomial expression in I_c^{opt}, as shown in Fig. 4.9. The difference in the measured and the estimated efficiencies lies in the use of power transducers of different resolutions, for the data acquisition system and the control-loop feedback signal.

The power transducer employed for acquiring measurement data had a poorer resolution (12 bits = 13.33 kW) than the one utilized for closing the system loop (14 bits = 8 kW). Hence, the accuracy of the power measured, on-line, for the controller operation is more accurate than that of the acquired data. Here, it should be noted that it was the power transducer with the coarser resolution that was employed for conducting the off-line characterization of the BDFM which may explain some of the deviations observed. On-line, the power measurements for the control loop operation were more accurate and hence determined the control winding currents, Fig. 4.8, with a higher resolution. This allowed the implemented scheme to achieve somewhat different efficiencies than expected. However, the closed loop efficiencies were consistently higher than those estimated at higher input mechanical power, as seen in Fig. 5.2.

The BDFM was characterized for a different mechanical power input profile than the one utilized for the closed-loop operation. This is discussed in Chapter 4, section 4.2.3 and illustrated in Fig. 4.6. As seen in Fig. 5.3, the input mechanical power for the closed loop operation is consistently higher, for the shaft speed range of 1300-1550 r/min, than that for the open-loop characterization mode. Due to the low level of excitation maintained on the control winding for both the closed-loop and open-loop operations (for the same speed range), it can be assumed that the losses in the BDFM (core and copper) are similar. Then a higher input power would result in an higher output power, whereby, a higher efficiency could be recorded, as seen in Fig. 5.2.

The "optimum" control winding current, I_c^{opt}, as measured during the closed-loop operation was plotted with respect to the generated power. This is then compared to the estimated control winding current required for maintaining optimized operation utilizing the $I_c - P_T$ profile, obtained during the prototype BDFM characterization, and shown in Fig. 4.8. The estimation of the optimum I_c is done by substituting the measured total power into the polynomial of Fig. 4.8. The comparison of the measured and estimated control winding currents is illustrated in Fig. 5.4. The difference in the two currents is, again, due to the use of power transducers of different resolutions, and hence, accuracy as mentioned before. The estimated current is constructed based on the low-resolution power transducer which is a part of the data acquisition system. On the other hand, the control-loop feedback power, which helped determine the measured current, is of a higher resolution transducer. Still, the two compare favorably as seen in Fig. 5.4.

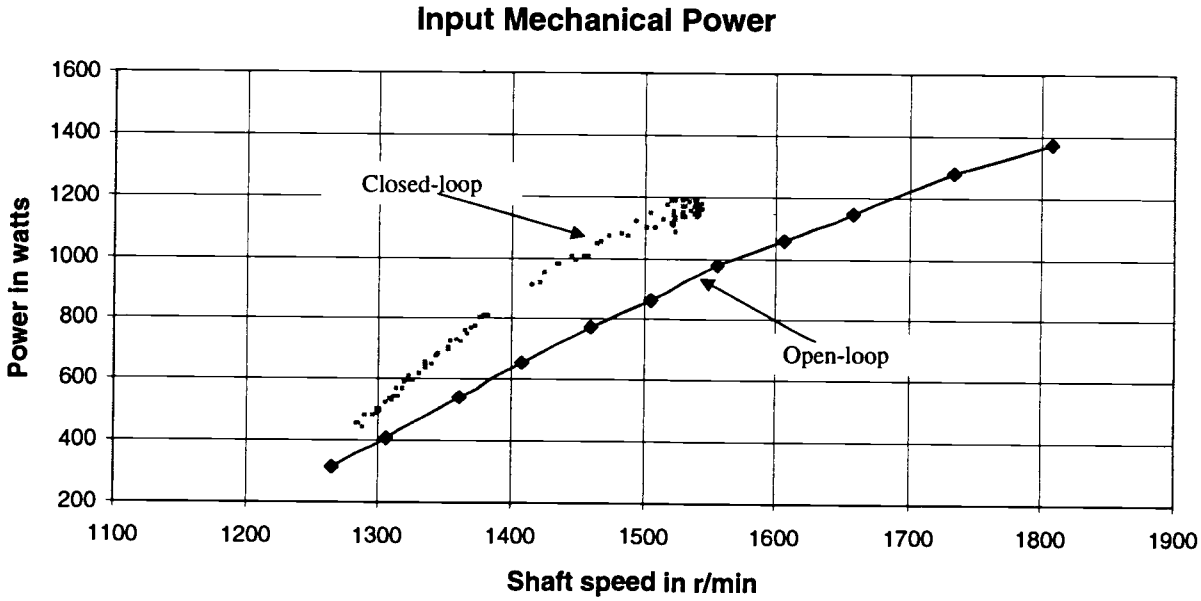


Figure 5.3 Turbine output power for open and closed loop operation.

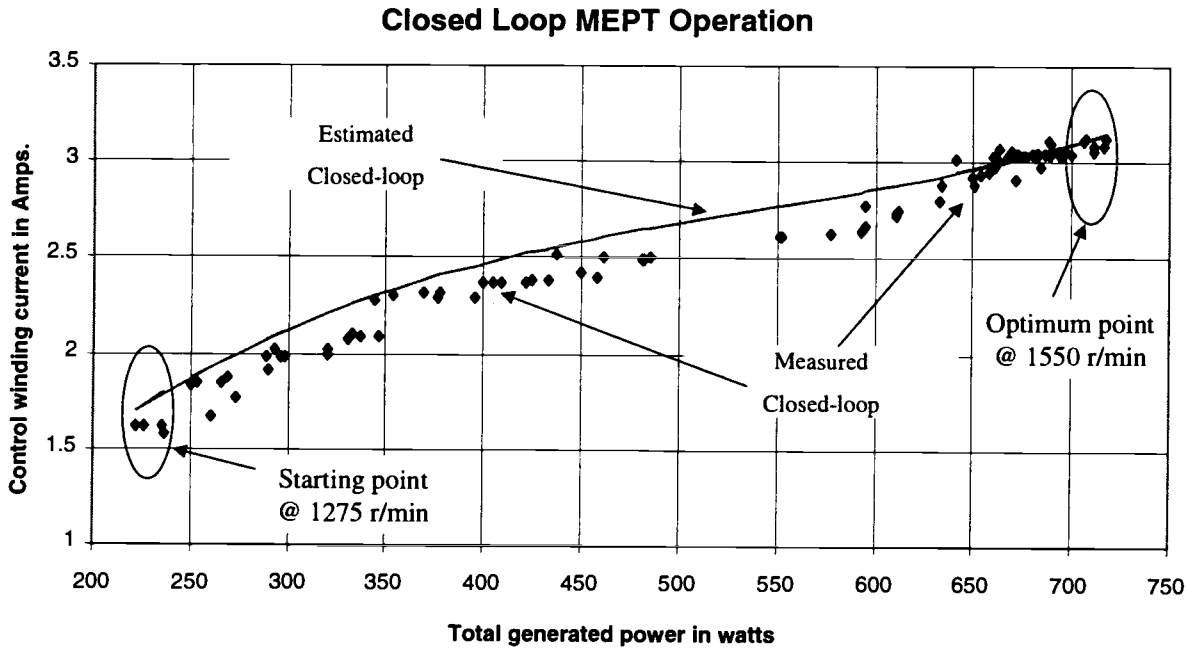


Figure 5.4 Comparison of the measured and estimated control winding current for maintain optimum operation. The current has been estimated by substituting the measured power into the polynomial of Fig. 4.8.

5.3. Power distribution

Figure 5.5 illustrates the distribution of real and apparent power between the two windings under closed-loop control. The voltage and current transducers were present at the two stator terminals of the BDFM. While real power into and out of the two stator windings was directly measured utilizing instantaneous power transducers the apparent and reactive power were calculated from the available voltage, current and real power measurements. The ratio of the control winding (apparent, real, reactive) power to the summation of the control and power winding (apparent, real, reactive) power is then plotted in Fig. 5.5.

It should be pointed out, that the data acquisition system available leads to some inconsistencies in measuring the control winding voltages. The output of the inverter

stage, although current regulated, generates a pulsed voltage waveform whose density and width determines the rms voltage quantity. The voltage transducers in the data acquisition system consistently measured a higher rms voltage. Hence, the calculated (from the three-phase current and voltage measurements) kVA of the converter and, consequently, the kVAR is significantly higher than would normally be. However, the calculated kVA can still be used to determine an upper limit of the converter kVA requirements as a function of the overall system input power.

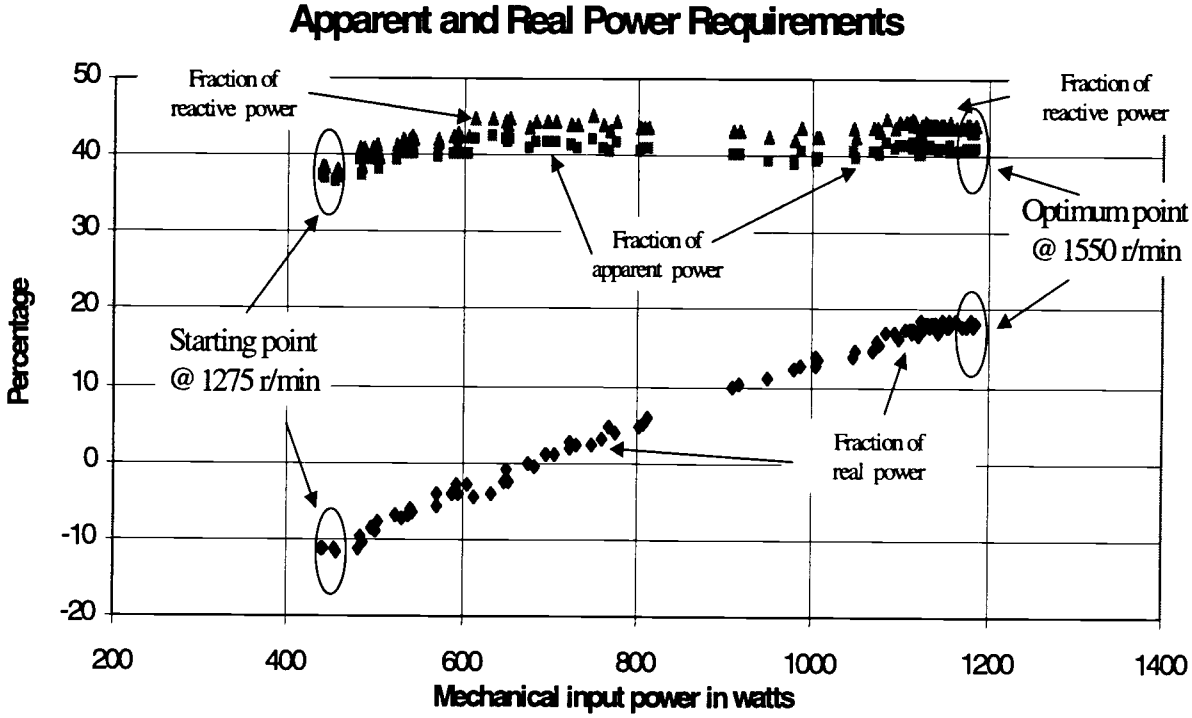


Figure 5.5 Power flow through the inverter as percentage of total apparent, real and reactive power.

It is also observed in Fig. 5.5 that the "optimum" control winding excitation setting provides for either real power flow into or out of the control winding, depending on the mechanical input power. In the "motoring mode" of operation of the control winding, real power flows from the grid to the BDFM through the ac/ac converter. At higher input

power levels, the control winding enters the regeneration mode of operation. Hence, the reversal of sign, in Fig. 5.5, in the percentage of real power processed by the converter.

Figure 5.6 depicts the phase currents of the input and output stages of the converter and the current through the power winding of the BDFM. The low regenerated real power through the control winding mostly accounts for losses in the converter system, resulting in very low current magnitudes at the output of the rectifier stage. At low input current levels on the rectifier, the waveshaping suffers considerably due to the discrete pulse modulation strategy, as implemented in the rectifier controller, with constant on-time of the rectifier switches. This condition could be improved by implementing a pulse-width-modulation based rectifier control strategy, which would most likely be the case for a real-life implementation.

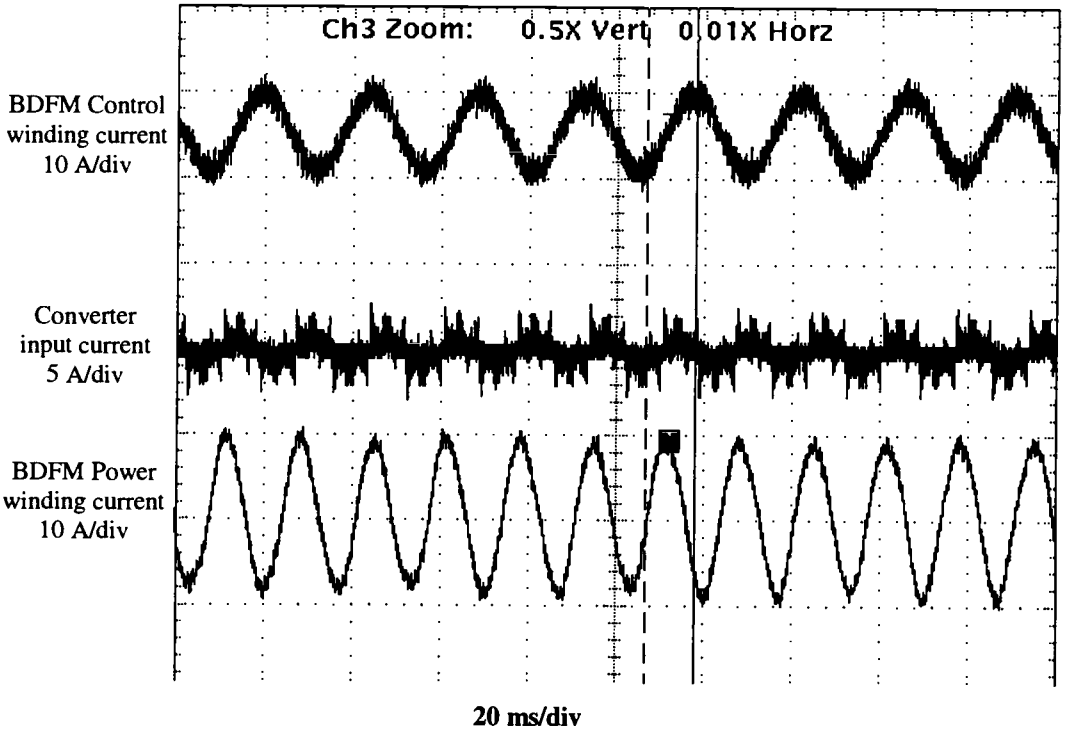


Figure 5.6 Representative current waveforms of the converter and the BDFM approximately at the maximum power point ($P_{wt} \approx 1150$ watts, shaft speed = 1525 r/min).

6. Conclusions and Recommendations

A DFM based VSG system controller is presented. The controller operation and its efficacy was verified in a laboratory based model VSG system utilizing a BDFM for a wind generation application. Although verified in a wind generation application, the control philosophy is applicable for other generation strategies where the resource energy varies significantly. Furthermore, the controller could also be implemented for minimization of power input to a DFM based ASD system. It is also not just restricted to maximization or minimization of power but could, possibly, include any other user defined optimization criteria.

6.1. Salient features of thesis

Since in this implementation a synchronous machine, the BDFM, has been utilized as the electrical generator, no speed sensors were required to achieve variable speed operation. Furthermore, all derivations for the required current and frequency commands of the BDFM are based upon the total output electrical power as the only feedback quantity. Thus, no torque transducer is needed in the present version of the algorithm. Elimination of the expensive mechanical speed and torque sensors, typically required for similar VSG controls, is a beneficial aspect of the controller and has a positive cost impact on real-life implementation. However, for incorporating dynamic inner loop controllers, torque and position sensors may be required.

The wind speed estimation based MPPT, as described in Chapter 2, is faster than a regular search based algorithm [3], with the optimum shaft speed being determined within a few iterations of the MPPT algorithm. The controller developed was verified to be robust to operating conditions substantially different from those during its characterization, as mentioned in Chapter 4. It ensured the use of a power converter rated at only a fraction of the total system rating throughout its operational range. This is

perhaps the single most important reason for the incorporation of a DFM instead of conventional singly-fed VSG or ASD systems.

Besides the additional capital cost, VSG systems usually suffer from extra losses in the power conversion process in the power electronic converter which decreases the net energy capture. Doubly-fed systems, in general, can reduce the power handling requirement of the converter and hence the losses associated with it. Due to maximizing the generator efficiency, as implemented in this thesis, throughout all speed ranges, the energy conversion process from mechanical to electrical is, also, enhanced. Thus, the benefit of the reduction in the power converter size and, hence, its losses, due to the incorporation of a doubly-fed machine as the generator, is not given up during the conversion process in the electromechanical system.

A preliminary cost analysis, on a 100 kW scaled unit, of the BDFM based VSG system was conducted and compared to a conventional induction machine based wind generation system [2]. Utilizing an annual wind speed distribution for a tentative wind power site on the Oregon coast, analysis show substantial increase in energy capture which directly increases the possible annual revenue. While the laboratory system proved the implementational feasibility of such a system, the enhanced energy, and additional revenues thus collected, make this system commercially viable.

In the process of implementing the system controller new strategies for active rectifier control were developed; one utilizing current sensors, the other eliminating those sensors. The sensor-based active rectifier controller has been analyzed under steady state and dynamic loading conditions. Steady state variable power factor operation has been demonstrated as an extension of the algorithm. It is shown that the ease of implementation and the simplicity of the controller are features, which outweigh its relatively slower response to transients compared to the more sophisticated and resource consuming space vector and rotating frame based controllers. However, in support of the dynamic performance improvements for the proposed sensor-based rectifier controller, it

should be pointed out that further fine-tuning of the filter gain in conjunction with an increase in the cutoff frequency of the low-pass filter should improve the dynamic performance of the system. Optimization of the software code along with the hardware implementation will also enable a further increase in the switching frequency and enhance waveform shaping capability.

In an effort to further reduce the cost of implementation of the active rectifier controller, it was realized that a sensorless current controller could be developed which introduces considerable benefits over a controller with sensors. The most important being the elimination of hardware - current sensors and A/D converters - and the associated cost reduction. Problems with offsets in sensors and A/Ds are eradicated and problems introduced by the required multiplexing when using a single A/D are eliminated.

6.2. Recommendations for future work

Although the steady state performance of the VSG controller was verified by developing a laboratory wind turbine-VSG system, dynamic issues such as the tower passing or tower shadow effects of the wind turbine have not been addressed in this thesis. Torque pulsation compensation due to those effects of a wind turbine would have to be accommodated in a faster inner dynamic control loop. This control loop would typically consist of a field oriented control algorithm for the doubly-fed machine [43].

An optimization controller based upon minimization of a quadratic cost function may also be investigated. The cost function could consist of criteria representing response time of the system, penalty for exceeding the torque requirements in trying to achieve fast response times, penalty for being far away from the maximum power point, etc. While the proposed controller directly mimics the DFM from characterization and acceptance testing data, the challenges for the cost function minimization approach would be in the construction of a valid quadratic penalty function. Moreover, issues related to

convergence and uniqueness of the solution of the minimization problem for possible operating conditions would have to be investigated.

A four quadrant ac/ac converter is typically fabricated using power stages with identical power ratings. While the inverter stage needs to source the reactive power requirements of an electric machine, the rectifier under unity power factor operates at only a fraction of the total available kVA. The remaining kVA rating of the rectifier could then be utilized for power factor correction or reactive power control, at the point of common coupling to the grid, for the overall generating system. This controller would probably need to be an outer loop control providing power factor update for the rectifier. It should be mentioned that the present rectifier controller is already equipped to accept power factor update commands from a higher level controller. On similar grounds, the extra kVA in the rectifier could also be utilized for active filtering operations. These peripheral controls should be incorporated without sacrificing the requirements of the performance optimization controller for the chosen optimization criterion.

In a real-life implementation, an outer adaptive control loop would be required to update the parameters of the controller to reflect the changes in the physical parameters of the system. While the rotor time constant changes with temperature, permanent mechanical changes of the system may occur due to the wear and tear of the gears. Finally, inclement climatic conditions such as ice, snow and dirt could also result in the changes in the wind turbine performance. Thus, updating the model over time would, ensure enhanced system performance by mimicking the physical system more closely and, hence, accurately. Implementation of the adaptive parameter update loop could comprise of neural networks, fuzzy logic or recursive identification methods.

In summary, future work for the present controller should address the performance of the controller under real-life dynamic conditions, i.e. torque pulsations due to the tower shadow effect, protection, system starting and stopping issues, tolerance towards wind

gusts, etc. Finally, operation under grid unbalance and voltage variations, an unfortunate possibility in remote wind power generation installations, should also be investigated.

Bibliography

1. R. Spée and S. Bhowmik, "Wind Turbines", Encyclopedia of Electrical Engineering, *John Wiley & Sons*, 1997.

2. R. Spée, S. Bhowmik and J.H.R. Enslin, "Novel control strategies for variable-speed doubly fed wind power generation systems", *Renewable Energy*, **6**: 907-915, 1995.

3. J.H.R. Enslin and J.D. Van Wyk, "A study of a wind power converter with micro-computer based maximal power control utilising an over-synchronous electronic Scherbius cascade", *Renewable Energy*, **2**: 551-562, 1992.

4. L.L. Freris, "Wind energy conversion systems", *Prentice Hall International (UK) Ltd.*, 1990.

5. V. Nelson, W. Pinkerton and R.N. Clark, "Power variation with pitch setting for a horizontal axis wind turbine", *Wind Engineering, 9: 88-94, 1985.*

6. E. Spooner and A.C. Williamson, "Direct coupled, permanent magnet generators for wind turbine applications" *IEE Proceedings of Electrical Power Application,* **143**:1-8, 1996.

7. L. Dessaint, H.L. Nakra and D. Mukhedkar, "Propagation and elimination of torque ripple in a wind energy conversion system", *IEEE Transactions on Energy Conversion*, **1**:104-112, 1986.

8. H.L. Nakra and B. Dubé, "Power recovery induction generators for large vertical axis wind turbines", *IEEE Trans. Energy Conversion,* **3**: 733-737, 1988.

9. R.D. Richardson and W.L. Erdman, *Variable speed wind turbine*. U.S. Patent No. 5,083,039, 1992.

10. L. Xu and Y.Tang, "A novel wind-power generating system using field orientation controlled doubly-excited brushless reluctance machine", *Proceedings of IEEE Industry Applications Society Annual Meeting*, 408-413, 1992.

11. R. Li, R. Spée, A.K. Wallace and G.C. Alexander, "Synchronous drive performance of brushless doubly-fed motors", *IEEE Transactions on Industry Applications*, **30**: 963-970, 1992.

12. D.A. Torrey, "Variable-reluctance generators in wind-energy systems", *Proceedings of IEEE Power Electronics Specialist Conference*, 561-567, 1993.

13. B. Sarlioglu, Y. Zhao and T.A. Lipo, "A novel doubly-salient single phase permanent magnet generator", *Proceedings of IEEE Industry Applications Society Annual Meeting*, 9-15, 1994.

14. S. Bhowmik and R. Spée, "A guide to the application-oriented selection of ac/ac converter topologies", *IEEE Transactions on Power Electronics,* **8**:156-163, 1993.

15. N. Bianchi and A Lorenzoni, "Permanent Magnet Generators for wind power industry: an overall comparison with traditional generators", *Proceedings of International Conference on Opportunities and Advances in International Power Generation,* 49-54, 1996.

16. Y. Shimizu and S. Matsumara, "Rotation speed control of HAWT by tip vane", *JSME International Journal, Series B:Fluids and Thermal Engineering*, **37**: 363-368, 1994.

17. G. Catto and A.C. Williamson, "Direct coupled wind turbine stabilizing using controlled stator oscillations", *Proceedings of the 29th Universities Power Engineering Conference* **1**: 282-285, 1994.

18. E.N Hinrichsen, "Control for variable pitch and turbine generators", *IEEE Transactions on Power Apparatus and Systems* **PAS-103**: 886-892, 1984.

19. A.F. Boehringer, "Struktur und Regelung von Energieversorgungssystemen in Satelliten", *Etz Archiv*, **92(2)**: 114-119, 1971.

20. P. Novak, T. Ekelund, I. Jovik and B. Schmidtbauer, "Modeling and control of variable speed wind-turbine drive-system dynamics", *IEEE Control Systems Magazine*, **15**: 28-38 1995.

21. Y. Tang and L. Xu, "A flexible active and reactive power control strategy for a variable speed constant frequency generating system", *IEEE Transactions on Power Electronics* **10**: 472-478, 1995.

22. H.K. Lauw, "Brushless Doubly-Fed Generator Control System," *U.S. Patent No. 5,028,804*, 1991.

23. M.T. Iqbal, A.H. Cooknick and L.L. Freris, "Dynamic control options for variable speed wind turbines", *Wind Engineering*, **18**: 1-11, 1994.

24. D. Zhou and R. Spée, Field oriented control development for brushless doubly-fed machines, *IEEE IAS Conference Records,* **1**: 304-310, 1996.

25. Cavallo, "High capacity factor wind turbine transmission systems", *ASME SED Wind Energy,* **15**:87-94, 1994.

26. S. Bhowmik, R. Spée, A.K. Wallace and C. Brune, "Comparison testing of equivalent induction motor and brushless doubly-fed motor adjustable speed drives" International Conference on Electrical Machines, Paris 1994.

27. C. Brune, R. Speé and A.K. Wallace, "Experimental Evaluation of a Variable-Speed Doubly-Fed Wind-Power Generation System," *IEEE IAS Annual Meeting Conf.*, pp. 480-487, 1993.

28. A. Busse and J. Holtz, "Multiloop Control of A Unity Power Factor Fast Switching AC to DC Converter", *Conf. Rec. IEEE PESC 1982, pp. 171-179.*

29. J.W. Dixon and B.T. Ooi, "Indirect Control of a Unity-Power Factor Sinusoidal Current Boost Type Three-Phase Rectifier", *IEEE Trans. on Ind. Elec.*, Vol. 35, No. 4, pp. 508-515, Nov. 1988.

30. Rusong Wu, S.B. Dewan and G.R. Slemon, "A PWM AC to DC Converter with Fixed Switching Frequency", *IEEE Power Electronics Specialist Conference*, pp. 706-711, 1989.

31. N.R. Zargari and G. Joos, "Performance Investigation of a Current-Controlled Voltage-Regulated PWM Rectifier in Rotating and Stationary Frames", *IEEE Annual Conf. of the Industrial Electronics Society 1993*, pp. 1193-1197.

32. S. Bhowmik, R. Spée, G.C. Alexander and J.H.R. Enslin, "New simplified control algorithm for synchronous rectifiers", *IEEE IECON* 1995, pp. 494-499.

33. S. Bhowmik, A. van Zyl, R. Spée and J.H.R. Enslin, "Sensorless Current Control for Active Rectifiers", *IEEE Trans. on IAS, May/June*. 1997.

34. S. Fryze, "Wirk-, Blind-, und Scheinleistung in Elektrischen Stromkreisen mit nichtsinusoidalformigen Varlauf von Strom und Spannung", *ETZ* Vol. 53, No. 25, 1932, pp. 596-699, 625-627, 700-702.

35. J.H.R. Enslin and J.D. Van Wyk, "A New Control Philosophy for Power Electronic Converters as Fictitious Power Compensators", *IEEE Trans on PE*, Vol. 5, No. 1, pp. 88-97, Jan. 1990.

36. N. Mohan, T.M. Undeland and W.P. Robbins, *Power Electronics: Converters, Applications and Design*, John Wiley & Sons, 1989, pp. 425-426.

37. H.V.d.Broeck, H. Skundelny and G.V.Stanke, "Analysis and Realization of a Pulsewidth Modulator Based on Voltage Space Vectors", *IEEE Trans. IAS*, vol. IA-24, No.1, 1988.

38. A.K. Wallace, R. Spée and H.K. Lauw, "The Potential of Brushless Doubly-Fed Machines for Adjustable Speed Drives", *Proceedings IEEE IAS Pulp & Paper Industry Conference*, Seattle, June 1990, pp. 45-50.

39. A.K. Wallace, R. Spée and G.C. Alexander, "The Brushless Doubly-Fed Motor as a Limited-Speed-Range Pump Drive", *Proceedings IEEE ISIE*, Budapest, June 1993, pp. 33-37.
40. A.K. Wallace, R. Spée and G.C. Alexander, "Adjustable Speed Drive and Variable Speed Generation Systems with Reduced Power Converter Requirements", ibid., pp. 503-508.

41. L.J. Hunt, "A New Type of Induction Motor", *Journal IEE (London)*, Vol. 39, pp. 648-667, 1907.

42. F. Creedy, "Some Developments in Multispeed Cascade Induction Motors", *Journal IEE (London)*, Vol. 59, 1921, pp. 511-532.

43. D. Zhou, "Dynamic Control for Brushless Doubly-Fed Machine", *Ph.D. dissertation*, Oregon State University 1996.

Appendices

Appendix A. Converter Controller Source Code

The control algorithms for the inverter and the rectifier have been discussed in detail in Chapter 3. As mentioned in Chapter 4 both the inverter and the rectifier stages are controlled by individual Intel 80C196KC microcontrollers. This appendix lists the software code for the rectifier programs. As mentioned in Chapter 3, the inverter code is only a subset of the rectifier code and can be easily contrived from the rectifier code listings presented here. Moreover, this saves space and reduces redundancy. Additionally, the assembly listings of the converter protection logic (utilizing an Atmel 2051) is also provided here.

A.1 Rectifier code

Startup assembly code :

```
;*****************************************************************/
;                                                                 */
;    Active rectifier controller                                  */
;    Copyright 1997 Oregon State University, Corvallis OR         */
;                                                                 */
;    Controller developed by Shibashis Bhowmik                    */
;    Software coded by Shibashis Bhowmik                          */
;                                                                 */
;*****************************************************************/
;    1.19.96
;         communication thru' serial port
;         by Shiba Bhowmik
;    4.20.95
;         total modification made to assembly switching module
;         by Shiba Bhowmik
;    9.03.94
;         major modifications made to existing code
;                  by Shibashis Bhowmik
;    11.15.93
;    adapted to ASM-96 by Shibashis Bhowmik
;         from Brian Wiley's module
;
startup MODULE  main
;
        sp              equ    18h:word
;
; This REG segment is needed so that code does not conflict
; with evaluation board RISM
```

Startup assembly code (continued):

```
        RSEG at 30h
                DSB    9
;
; PORTS 3 & 4
;
        DSEG at 1FFEh
                DSW    1
        CSEG at 2000h
;    hard coded interrupts
                DCW         0FFFFh          ;interrupt 0
                DCW         0FFFFh          ;interrupt 1, A/D conversion complete
                DCW         0FFFFh          ;interrupt 2
                DCW         0FFFFh          ;interrupt 3
                DCW         0FFFFh          ;interrupt 4
                DCW         07000h          ;interrupt 5, HSO software timer
                DCW         0FFFFh          ;interrupt 6
                DCW         0FFFFh          ;interrupt 7
;;
; Comment out the following segment when using the RISM-based eval board
; But needed for the user programed EPROMs
;
        CSEG at 2018h
                DCB         11001111B       ;configuration byte
;;
        CSEG at 2080h
;
;    EXTRN _main
;
                LD     sp, #100h
;
; change "_memoryRemap" to "_main" in the following command line
; when using the RISM-based eval board
;
                BR     _memoryRemap
;
;
;
; Only when using user programed EPROMs
; Just do something @ 1D00h - 1DFFh to remap memory on the eval boards
;
;
        CSEG at 1D00h
;
        EXTRN  _main
;
_memoryRemap:
;
                NOP
                BR     _main
;
        END
```

C-language source code:

main.c

```
/*************************************************************************/
/*                                                                       */
/*      Active rectifier controller                                      */
/*      Copyright 1997 Oregon State University, Corvallis OR             */
/*                                                                       */
/*      Controller developed by Shibashis Bhowmik                        */
/*      Software coded by Shibashis Bhowmik                              */
/*                                                                       */
/*************************************************************************/

#include "proto.h"


void _main()
{
        extern register unsigned char commandChar ;

        AtoD_init() ;
        serial_init();
        initCurrent();

        /* take care that this routine never
          returns to startup.c
        */

        while(1)
        {     commandChar = getChar();
              if ((commandChar == 'E') || (commandChar == 'e'))
              {     putChar(10); putChar(13);
                    begin();
              }
        }
}
```

begin.c

```
/*************************************************************************/
/*                                                                       */
/*      Active rectifier controller                                      */
/*      Copyright 1997 Oregon State University, Corvallis OR             */
/*                                                                       */
/*      Controller developed by Shibashis Bhowmik                        */
/*      Software coded by Shibashis Bhowmik                              */
/*                                                                       */
/*************************************************************************/

#include "proto.h"
#include "80196.h"
#include "global.h"

/* added on Jan 25, 1995 */
extern register signed int saveRemA, oldIndx, oldVdcError, vdcError ;
/************************/
/* added on Feb 17, 1995 about hvRefConst */
extern register signed int hvRef, saveRemImag ;
/************************/
extern register unsigned int incOfIndx, currentLag, cycleFreq, halfCycleFreq ;
extern register unsigned int oldHsiTime, newHsiTime, remA ;
extern register unsigned int countForAPeriod, halfCountForAPeriod ;
extern register unsigned int cycleTime, waitForNext, onTime ;
extern register signed int Imag, oldImag, maxImag, Iam, Ibm, Icm, Ia, Ib, Ic ;
extern register unsigned int Ki, Kp, vdcConst, vdc, vdcDiode, errorBand ;
extern register unsigned char commandChar;
extern register unsigned char i, stop, cycleCount;
extern register unsigned char vChar[3];

/* extern void changeCurrent(void) ;
*/

void begin(void)

{
        int sTempVar;
        unsigned int uTempVar;

        hso_init();
        hsiInit() ;

        IOPORT1 = 0 ;
        IOPORT2 &= 0x3F ;      /*turns on HSO buffer, 244 */
        Iam = Ibm = Icm = 0 ;
        Ia = Ib = Ic = 0 ;
        stop = 0;
```

begin.c (continued)

```
/*************************************************************************
        The code realting to the variable "programCount" added on Jan 31, 1995
        This is believed to help me see what happens after each program loop
        execution. Could possibly give me a better feel for the voltage regu-
        lator, currents and actual voltages.
*************************************************************************/

        countForAPeriod = 16667 ;
        halfCountForAPeriod = countForAPeriod/2 ;

        oldHsiTime = 0 ; /* -16667  */
        newHsiTime = 0 ;
        oldIndx = 0 ;
        incOfIndx = 0 ;
        saveRemA = 0 ;
        remA = 0 ;
        saveRemImag = 0 ;
/* added on Jan 25, 1995 */
        oldVdcError = 0 ;
/*     vdcError = 0 ;
*/
/*************************/

        Imag = 0 ;   /*  corresponds to 2 A  */


/*  Fastest cycleTime = 141 us, ie. waitForNext = 47
*/
        cycleTime = 150 ;   /* remember 110 us for just doing calculations */
        cycleFreq = 6667 ;
        halfCycleFreq = cycleFreq/2 ;
        waitForNext = 50 ;
        onTime = 80 ;
        currentLag = (10 * 25)/3;  /* 10 degrees of lag was 200 Brian 7/3/97 */

/* initialize variables for dc bus voltage regulator loop
*/
/*************************************************************************/

        Ki = 0  ;
        Kp = 0 ;
        cycleCount = 0; /* variable added on 2/05/97 for soft start */
```

begin.c (continued)

```
/*************************************************************************
        Protection from steady bucking
        2/03/97
*************************************************************************/
/* Measure the diode rectifier dc-bus voltage */
        AtoD_start(HV_MONITOR) ;
        vdcDiode = AtoD_read() - 6; /* to account for inaccuracies in measurements */

        WSR = 15;
        TIMER_1 = 0;
        WSR = 0;

/* Do not let the dc-bus reference to be lower than that of the
  diode-based rectifier voltage */
        IOPORT1 = 0;
        hvRef =  (386) * 2 ;
        while (hvRef < vdcDiode)
        {     if (IOPORT1 == 0x00)
                        IOPORT1 = 0x01;
              else
                        IOPORT1 <<= 1;
              while(TIMER_1)
                        ;
        }

/*************************************************************************/

        vdcConst = 60 ;
        errorBand = 0 ;
        oldImag = 0 ;
        maxImag = 50 ; /* ~ 10 A */
/*
        initCurrent() ;
*/
/*********************************************************/

        WSR     = 15 ;
        TIMER_1  = 0 ;
        WSR     = 0 ;

        timer2Init() ;
        setUpSwftInit() ;
        setUpEmgncDown() ;

        WSR     = 15 ;
        HSI_TIME = 0 ;
        WSR     = 0 ;
```

begin.c (continued)

```
        hsiGo() ;

        while (!HSI_TIME)
                ;


/* reset timer1 which will be used as the HSI clock source */

        WSR     = 15 ;
        TIMER_1  = 0 ;
        HSI_TIME = 0 ;
        WSR     = 0 ;
        TIMER_2  = 0xFFFD ;

/*********************************************************************
 second variale intialization incorporated from by A. van Zyl
 necessitated due to noise problems as observed on SPOT
*********************************************************************/

        while(!stop)
        {       commandChar = getChar();
                if ((commandChar == 'n') || (commandChar == 'N'))
                {       putChar(10); putChar(13);
                        stop = 1;
                }
        }

        stop = 0;

        asm ei ;

        while(!stop)
        {
/*********************************************************************
        For soft start purposes 2/05/97
*********************************************************************/

                if (cycleCount < STEPCYCLE)
                        ++cycleCount;
                else
                {       cycleCount = 0;
                        if (Ki < KiMAX)
                                Ki += 2 ;
                        if (Kp < KpMAX)
                                Kp += 100 ;
                }
```

begin.c (continued)

```
/*****************************************************************/

                if (spchk())
                {   commandChar = getBuf();
                        switch(commandChar)
                        {
                                case 'A': case 'a':
                                        putChar(10);putChar(13); putChar('A');
                                        sTempVar = Imag;

                                        if(sTempVar < 0)
                                        {       sTempVar = -sTempVar;
                                                putChar('-');
                                        }
                                        else
                                                putChar('+');

                                        vChar[0] = ((sTempVar * 30 / 19) / 100) + '0';
                                        putChar(vChar[0]);
                                        vChar[1] = (((sTempVar * 30 / 19) % 100) / 10) + '0';
                                        putChar(vChar[1]); putChar('.');
                                        vChar[2] = (((sTempVar * 30 / 19) % 100) % 10) + '0';
                                        putChar(vChar[2]);putChar(10);putChar(13);
                                        break;

                                case 'D': case'd':
                                        asm di;
                                        hsoAllClear();
                                        IOPORT1 = 0x80;
                                        stop = 1;
                                        putChar(10);putChar(13);
                                        break;

                                case 'I': case 'i':
                                        for (i=0; i<3; i++)
                                                vChar[i] = getChar() - '0';
                                        Ki = (vChar[0]*100 + vChar[1]*10 + vChar[2]);
                                        putChar(10);putChar(13);
                                        break;

                                case 'L': case 'l':
                                        for (i=0; i<3; i++)
                                                vChar[i] = getChar() - '0';
                                        currentLag = (vChar[0]*100 + vChar[1]*10 + \
                                                        vChar[2])*25 / 3;
                                        putChar(10);putChar(13);
                                        break;
```

begin.c (continued)

```
/*    can specify a maximum of 40 Amps
        to remain within the overflow limitations */

                case 'M': case 'm':
                        for (i=0; i<3; i++)
                                vChar[i] = getChar() - '0';
                        uTempVar = (vChar[0]*100 + vChar[1]*10 + vChar[2]) * 19 / 30;
                        putChar(10);putChar(13);
                        if (uTempVar < PEAK_CURR)
                        maxImag = (uTempVar*19)/30;
                        break;

                case 'O': case 'o':
                        for (i=0; i<3; i++)
                                vChar[i] = getChar() - '0';
                        uTempVar = (vChar[0]*100 + vChar[1]*10 + vChar[2]);
                        putChar(10);putChar(13);
                        if (uTempVar < (cycleTime - MIN_OFF_TIME))
                        onTime = uTempVar;
                        break;

                case 'P': case 'p':
                        for (i=0; i<3; i++)
                                vChar[i] = getChar() - '0';
                        Kp = (vChar[0]*100 + vChar[1]*10 + vChar[2]);
                        putChar(10);putChar(13);
                        break;

                case 'V': case 'v':
                        for (i=0; i<3; i++)
                                vChar[i] = getChar() - '0';
                        uTempVar = (vChar[0]*100 + vChar[1]*10 + vChar[2])* 2;
                        putChar(10);putChar(13);
                        if (uTempVar < vdcDiode)
                        {       putChar('B');putChar('u');putChar('s');putChar(' ');
                                putChar('r');putChar('e');putChar('f');
                                putChar(' ');putChar('t');putChar('o');
                                putChar('o');putChar(' ');putChar('l');
                                putChar('o');putChar('w');putChar('!');
                                putChar(10);putChar(13);
                        }
                        else
                                hvRef = uTempVar;
                        break;
```

begin.c (continued)

```
                                case 'F': case 'f':
                                        putChar(10);putChar(13);putChar('F');
                                        sTempVar = vdcError/2;
                                        if(sTempVar < 0)
                                        {       sTempVar = -sTempVar;
                                                putChar('-');
                                        }
                                        else
                                                putChar('+');
                                        vChar[0] = (sTempVar/100)+'0';
                                        putChar(vChar[0]);
                                        vChar[1] = ((sTempVar%100)/10)+'0';
                                        putChar(vChar[1]);
                                        vChar[2] = ((sTempVar%100)%10)+'0';
                                        putChar(vChar[2]);putChar(10);putChar(13);
                                        break;
                        }
                }
        }
        hsiStop();
}
```

vdc_iref.c :

```
/*************************************************************************/
/*                                                                       */
/*      Active rectifier controller                                      */
/*      Copyright 1997 Oregon State University, Corvallis OR             */
/*                                                                       */
/*      Controller developed by Shibashis Bhowmik                        */
/*      Software coded by Shibashis Bhowmik                              */
/*                                                                       */
/*************************************************************************/

#include "proto.h"
#include "global.h"
#include "80196.h"


#define AVERAGE 128
#define TIME_OUT      10

extern register signed int Ia0, Ib0, vdc0;
extern register unsigned int cycleTime, waitForNext ;
```

vdc_iref.c (continued):

```
void initCurrent (void)
{

        int i ;

        Ia0 = 0;
        Ib0 = 0;
        vdc0 = 0;

        for (i=0; i<AVERAGE; ++i)
        {
                AtoD_start(I_MONITOR_A);
                Ia0 += AtoD_read()-512;
                AtoD_start(I_MONITOR_B);
                Ib0 += AtoD_read()-512;
                AtoD_start(HV_MONITOR);
                vdc0 += AtoD_read();
        }
        Ia0 = Ia0/AVERAGE;
        Ib0 = Ib0/AVERAGE;
        vdc0 = vdc0/AVERAGE;

        vdc0 = 0;
}


void timer2Init (void)
{

        unsigned char ioc ;

        WSR = 1 ;
        IOC3 = T2_INTERNAL ;

        WSR = 15 ;
        ioc = IOC2 ;
        ioc &= ~T2_FAST_ENABLE ;
        ioc &= ~T2_DOWN_ENABLE ;
        WSR = 0 ;
        IOC2 = ioc ;
}


void setUpEmgncDown (void)
{
/*
        INT_MASK1 |= EXTINT1_MASK ;
*/
        asm di ;
}
```

vdc_iref.c (continued):

```
void setUpSwftInit(void)
{
        unsigned int hsoTime ;
        WSR = 0 ;
        INT_MASK |= SWT_MASK ;
        asm di ;
        HSO_COMMAND = HSO_TIMER_RESET | HSO_TIMER | HSO_CAM_LOCK ;
        HSO_TIME = cycleTime ;
        HSO_COMMAND = HSO_TIMER_0 | HSO_INTERRUPT | HSO_TIMER \
                        HSO_CAM_LOCK ;
        hsoTime = 0 ;
        HSO_TIME = hsoTime ;
}
```

hso.c :

```
/*************************************************************************/
/*                                                                       */
/*      Active rectifier controller                                      */
/*      Copyright 1997 Oregon State University, Corvallis OR             */
/*                                                                       */
/*      Controller developed by Shibashis Bhowmik                        */
/*      Software coded by Shibashis Bhowmik                              */
/*      HSO.C module adapted from Brian Wiley's code                     */
/*                                                                       */
/*************************************************************************/

#include "proto.h"
#include "80196.h"

void hso_init()
{
        char ioc;
        int i;
/* enable all high speed outputs
*/
        WSR = 15;
        ioc = IOC1;
        WSR = 0;
        ioc |= ( HSO_4_ENABLE | HSO_5_ENABLE );
        IOC1 = ioc;
/* enable cam locks and clear
*/
        WSR = 15;
        ioc = IOC2;
        WSR = 0;
        ioc |= HSO_CAM_CLEAR;
```

hso.c (continued):

```
        ioc |= HSO_LOCK_ENABLE;
        IOC2 = ioc;
}


/* hso set          Turn on.
*/
void hso_set( channel )
int channel;
{
        WSR = 0 ;
        while( IOS0 & HSO_HOLDING_FULL );
        HSO_COMMAND = channel | HSO_SET;
        HSO_TIME = TIMER_1 + 5;
}


void hso_clear( channel )
int channel;
{
        WSR = 0 ;
        while( IOS0 & HSO_HOLDING_FULL )
                ;
        HSO_COMMAND = channel;
        HSO_TIME = TIMER_1 + 5;
}

void hsoAllClear()
{
        WSR = 0 ;
        while( IOS0 & HSO_HOLDING_FULL )
                ;
        HSO_COMMAND = HSOALL;
        HSO_TIME = TIMER_1 + 5;
}

void hsoCamClear()
{
        char ioc;

        WSR = 15;
        ioc = IOC2;
        WSR = 0;
        ioc |= HSO_CAM_CLEAR;
        ioc |= HSO_LOCK_ENABLE;
        IOC2 = ioc;
}
```

hso.c (continued):

```
void hso_will_set( channel, t )
int channel;
unsigned int t;
{
        WSR = 0 ;
        HSO_COMMAND = channel | HSO_SET;
        HSO_TIME = t;
}


void hso_will_clear( channel, t )
int channel;
unsigned int t;
{
        WSR = 0 ;
        HSO_COMMAND = channel;
        HSO_TIME = t;
}


void hso_wait(pin)
int pin;
{
        unsigned int bit;

        bit = 1 << pin;

        while ( !(IOS0 & bit) )
                ;

}


void hso_will_AD( channel, t )
unsigned int channel;
unsigned int t;
{
        WSR = 0 ;
        AD_COMMAND = channel;
        HSO_COMMAND = HSO_AD;
        HSO_TIME = t;
}
```

hsi.c:

```
/*************************************************************************/
/*                                                                       */
/*      Active rectifier controller                                      */
/*      Copyright 1997 Oregon State University, Corvallis OR             */
/*                                                                       */
/*      Controller developed by Shibashis Bhowmik                        */
/*      Software coded by Shibashis Bhowmik                              */
/*                                                                       */
/*************************************************************************/

#include "proto.h"
#include "80196.h"

/* hsi initialization
        Configure special function registers for high speed inputs
*/
void hsiInit(void)
{
        WSR     = 0 ;
        HSI_MODE  = HSI0_STATUS ;
}


void hsiGo(void)
{
        WSR     = 0 ;
        IOC0    |= HSI0_ENABLE ;  /* | HSI1_ENABLE ; */
}




void hsiStop(void)
{
        WSR     = 0 ;
        IOC0    &= ~HSI0_ENABLE ;
}
```

serial.c:

```
/*************************************************************************/
/*                                                                       */
/*        Active rectifier controller                                    */
/*        Copyright 1997 Oregon State University, Corvallis OR           */
/*                                                                       */
/*        Controller developed by Shibashis Bhowmik                      */
/*        Software coded by Shibashis Bhowmik                            */
/*        SERIAL.C module originally developed by Brian Wiley            */
/*                                                                       */
/*************************************************************************/
#include "proto.h"
#include "80196.h"
#include "global.h"


/*    serial init
        initialization
*/
void serial_init()
{
        char ioc1;

        WSR = 0;

/*    baud rate = (crystal frequency/(64*(B+1))
        load rate sequentially, low byte first
        most significant bit determines source
*/
        BAUD_RATE = BAUD_LOW;
        BAUD_RATE = BAUD_HIGH | BAUD_XTAL1;

/*    clear 9th data bit
        enable the receiver
        disable parity
        set mode 1 (standard asynchronous)
*/
        SP_CON = SP_REN + SP_MODE_1;


/*    enable the TXD pin
*/
        WSR = 15;
        ioc1 = IOC1;
        WSR = 0;
        IOC1 = ioc1 | TXD_ENABLE;
}
```

serial.c (continued):

```
/* spchk          Check serial port for received character.
*/
char spchk()
{
        WSR = 0;
        return( SP_STAT & SP_RI );
}


/* getBuf         get serial port contents after checking for received char
*/
unsigned char getBuf()
{
        WSR = 0;
        return(SBUF);
}

/* getchar        Get a character from the serial port.
        Polled operation to start with.
*/
unsigned char getChar()
{
/* wait until character received
*/
        WSR = 0;
        while( !(SP_STAT & SP_RI) );
        return(SBUF);
}


/* putchar        Put a character to the serial port.
*/

void putChar( c )
unsigned char c;
{
/*    int x;
*/
/* wait until buffer emtpy
*/
        while( !(SP_STAT & SP_TXE) )
                ;
        SBUF = c;
/*
        for (x=0; x<200; ++x)
                ;
        return(c);
*/
}
```

support.c :

```
/*************************************************************************/
/*                                                                       */
/*      Active rectifier controller                                      */
/*      Copyright 1997 Oregon State University, Corvallis OR             */
/*                                                                       */
/*      Controller developed by Shibashis Bhowmik                        */
/*      Software coded by Shibashis Bhowmik                              */
/*      SUPPORT.C module originally developed by Brian Wiley             */
/*                                                                       */
/*************************************************************************/
#include "proto.h"
#include "80196.h"
#include "ctype.h"

/* delays
        delay by number of timer 1 clocks (us)
*/
void delays( n )
unsigned int n;
{
        unsigned int now;
        unsigned int end;

        now = TIMER_1;
        end = now+n;

        if (end<now) {
                /* wait for rollover first */
                while(TIMER_1 > end)
                        ;
        }
        while(TIMER_1 < end)
                ;
}

/* delayl
        delay by number of 1000 * timer1 clocks (ms)
*/
void delayl( n )
unsigned int n;
{
        while (n--) {
                delays(1000);
        }
}
```

atod.c :

```
/*************************************************************************/
/*                                                                       */
/*      Active rectifier controller                                      */
/*      Copyright 1997 Oregon State University, Corvallis OR             */
/*                                                                       */
/*      Controller developed by Shibashis Bhowmik                        */
/*      Software coded by Shibashis Bhowmik                              */
/*      ATOD.C module originally developed by Brian Wiley                */
/*                                                                       */
/*************************************************************************/
#include "proto.h"
#include "80196.h"
#include "global.h"

#define AVERAGE 100

void AtoD_init()
{
        char ioc ;
        int i;
/*    read options */
        WSR = 15;
        ioc = IOC2;
        WSR = 0;
/* clear bits */
        ioc &= ~AD_TIME_ENABLE ;
        ioc |= AD_NOT_PRESCALE;
        IOC2 = ioc;
}


/* A to D start   initiate conversion
*/
void AtoD_start(channel)
int channel;
{
      WSR = 0 ;
      AD_COMMAND = channel | AD_GO;
}
```

atod.c :

```
/* A to D read     Wait for conversion to finish before reading.
*/
unsigned int AtoD_read()
{
        unsigned int result;

        WSR = 0 ;
        while( AD_LOW & AD_BUSY )
                ;

        result = AD_LOW;
        result += AD_HIGH*256;
        result >>= 6;

        return( result );
}
```

global.c :

```
/*************************************************************************/
/*                                                                       */
/*      Active rectifier controller                                      */
/*      Copyright 1997 Oregon State University, Corvallis OR             */
/*                                                                       */
/*      Controller developed by Shibashis Bhowmik                        */
/*      Software coded by Shibashis Bhowmik                              */
/*                                                                       */
/*************************************************************************/

#include "ctype.h"

const signed int sin [] =
{

            2605,           .           .           .       -26769,
0,          2673,           .           .           .       -26809,
68,         2741,           .           .           .       -26848,
137,        2810,           .           .           .       -26887,
205,        2878,           .           .           .       -26926,
274,        2946,           .           .           .       -26965,
343,        3015,           .           .           .       -27004,
411,        3083,           .           .           .       -27043,
480,        3151,           .           .           .       -27082,
548,        3220,           .           .           .       -27120,
617,        3288,           .        3000           .       -27159,
686,        3356,           .     entries           .       -27197,
754,        3425,           .   (depending          .       -27235,
823,        3493,           .   on length of        .       -27274,
892,        3561,           .   sine table)         .       -27312,
960,        3629,           .           .           .       -27349,
1029,       3697,           .           .           .       -27387,
1097,       3766,           .           .           .       -27425,
1166,       3834,           .           .           .       -27462,
1234,       3902,           .           .           .       -27500,
1303,       3970,           .           .           .       -27537,
1372,       4038,           .           .           .       -27574,
1440,       4106,           .           .           .       -27611,
1509,       4174,           .           .           .       -27648,
1577,       4242,           .           .           .       -27685,
1646,       4310,           .           .      -26204,      -27721,
1714,       4378,           .           .      -26245,      -27758,
1783,       4446,           .           .      -26286,      -27794,
1851,       4514,           .           .      -26327,      -27831,
1920,       4582,           .           .      -26367,      -27867,
1988,       4650,           .           .      -26408,      -27903,
2057,       4718,           .           .      -26449,      -27939,
2125,       4786,           .           .      -26489,      -27974,
2194,       4854,           .           .      -26530,      -28010,
2262,       4922,           .           .      -26570,      -28046,
2331,       4990,           .           .      -26610,      -28081,
2399,       5058,           .           .      -26650,      -28116
2468,       5125,           .           .      -26690,      } ;
2536,       5193,           .           .      -26729,
```

80196 assembly code listings:

hystbipl.a96 :

```
;*****************************************************************/
;                                                                 */
;    Active rectifier controller                                  */
;    Copyright 1997 Oregon State University, Corvallis OR         */
;                                                                 */
;    Controller developed by Shibashis Bhowmik                    */
;    Software coded by Shibashis Bhowmik                          */
;                                                                 */
;*****************************************************************/
hyst1  MODULE  STACKSIZE(8)
;
$ INCLUDE (80196.INC)
$ INCLUDE (MEMORY.INC)
$ INCLUDE (DEFINE.INC)
$ INCLUDE (SYSTEM.INC)
;
; this module is the HSO software timer interrupt service routine
;
PUBLIC  changeCurrent
;
CSEG at 7000h
;
changeCurrent:
                        PUSHF
;
; go to horizontal window 0 (default window)
                        LDB WSR, 0
;
; determine which HSO software timer caused the interrupt
                        STB IOS1, tempIOS1
;
; jump to code for software timer 0
                        JBS tempIOS1, SWTF0_BIT, swft0
;
                        POPF
                        RET
;
swft0:
;
; start a/d conversion for dc bus voltage
                        LDB AD_COMMAND, #HV_MONITOR
;
                        XORB IOPORT1, #00000001b
;
measrGridFreq:
;
; measure grid frequency
```

hystbipl.a96 (continued) :

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; for testing purposes comment these lines,
; but, absolutely essential for actual code implementation
;
;
                    ST HSI_TIME, newHsiTime
;
                    CMP newHsiTime, oldHsiTime
                    JNE mayBe
;
                    LJMP getVdcAd
;
mayBe:
;
                    SUB temp, newHsiTime, oldHsiTime
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; just to check the voltage regulator
;
;           LD temp, #16667
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
                    CMP temp, #lowCount
                    JC checkHighBound
                    LDB IOPORT1, #00001100b
                    LJMP getVdcAd
;
;           LDB HSO_COMMAND, #HSOALL_CLEAR
;           ADD HSO_TIME, TIMER_1, #Await
;           LDB stop, #1
;           POPF
;           DI
;           RET
;
checkHighBound:
                    CMP temp, #highCount
                    JNC newTimePeriod
;
;           LJMP getVdcAd
;
                    LDB IOPORT1, #10110000b
                    LDB HSO_COMMAND, #HSOALL_CLEAR
                    ADD HSO_TIME, TIMER_1, #Await
                    LDB stop, #1
                    POPF
                    DI
```

hystbipl.a96 (continued) :

```
                        RET
;
;
newTimePeriod:
;
                        XORB IOPORT1, #00010000b
;
;
                        ST temp, countForAPeriod
                        ST newHsiTime, oldHsiTime
                        ST countForAPeriod, halfCountForAPeriod
                        SHR halfCountForAPeriod, #1
                        CLR saveRemA
;
; calculate for phase A
; multiply time for an iteration by the length of the sine table
                        LD switchTime+2, 0 ;
                        LD switchTime, #lenOfSinTabl
                        MULU switchTime, cycleTime
;
; divide the product by the time period of the sine waveform to be generated
                        DIVU switchTime, countForAPeriod ;
;
; store the quotient of the division for future use
                        ST switchTime, incOfIndx
;
; store the remainder of the integer division for future use
                        ST switchTime+2, remA
;
;
                        SUB oldIndx, currentLag, incOfIndx
;
; prepare to obtain A/D results
;
getVdcAd:
                        JBS AD_LOW, AD_STATUS_BIT, getVdcAd
;
adVdcdone:
                        ST AD_LOW, temp
                        SHR temp, #6
                        SUB vdc, temp, vdc0
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; to test dc-bus regulator
;
;               ST vdcConst, vdc
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;    Overvoltage and undervoltage protection 2/10/97
```

hystbipl.a96 (continued) :

```
;     Sets PORT1.7-4 high for overvoltage faults
;     Sets PORT1.3-0 high for undervoltage faults
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;     Direct over-voltage protection 2/09/97
;
                    CMP vdc, #MAXHV
                    JNC chkVdcDiode
                    LDB IOPORT1, #11110000b
                    LDB HSO_COMMAND, #HSOALL_CLEAR
                    ADD HSO_TIME, TIMER_1, #Await
                    LDB stop, #1
                    POPF
                    DI
                    RET
;
;     Direct under-voltage protection (2/09/97) to prevent bucking
;     below the doide rectifer dc-bus voltage
;
chkVdcDiode:
                    CMP vdc, vdcDiode
                    JC startADforA
                    LDB IOPORT1, #10001110b
                    LDB HSO_COMMAND, #HSOALL_CLEAR
                    ADD HSO_TIME, TIMER_1, #Await
                    LDB stop, #1
                    POPF
                    DI
                    RET
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; start a/d conversion for Phase A
startADforA:
                    LDB AD_COMMAND, #I_MONITOR_A
;
                    XORB IOPORT1, #00000010b
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
                    SUB vdcError, hvRef, vdc
                    ST vdcError, temp
;
                    JBC temp.15, pstvHvErr
                    NEG temp
;
pstvHvErr:
```

hystbipl.a96 (continued) :

```
                    CMP temp, errorBand
                    JC outOfBounds
                    ST 0, vdcError
                    SJMP smaller
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; Code implemented on Mar 29, 1995.
; Equation :
;
; Imag(n+1) = Imag(n) + (Ki/16) * deltaT * vdcError(n+1)
;
;             + (Kp/1024) * [vdcError(n+1) - vdcError(n)]
;
;
; deltaT = 1/cycleFreq ; if updated every program execution loop, implemented
;
; code modified on 2/13/97 to limit products of multiplications
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
outOfBounds:
;
                    SUB temp, vdcError, oldVdcError
                    MUL switchTime, temp, Kp
; put limit on the product, modified 2/13/97
                    ST switchTime+2, temp
                    JBS temp.15, negVar
                    CMP temp, 0
                    JH limitPosProduct
                    CMP switchTime, #7fffh
                    JNC productOk
                    LD switchTime, #7fffh
                    SJMP productOk
limitPosProduct:
                    LD switchTime, #7fffh
                    SJMP productOk
negVar:
                    CMP temp, #0ffffh
                    JLT limitNegProduct
                    CMP switchTime, #8001h
                    JC productOk
                    LD switchTime, #8001h
                    SJMP productOk
limitNegProduct:
                    LD switchTime, #8001h
;
productOk:
```

hystbipl.a96 (continued) :

```
                        MUL switchTime, cycleFreq
;
                        MUL Ias, oldImag, cycleFreq
                        SHLL Ias, #10
;
                        MUL Ibs, Ki, vdcError
                        SHLL Ibs, #6
;
                        CLRC
                        ADD switchTime, Ias
                        ADDC switchTime+2, Ias+2
                        CLRC
                        ADD switchTime, Ibs
                        ADDC switchTime+2, Ibs+2
                        SHRAL switchTime, #10
                        DIV switchTime, cycleFreq
                        ST switchTime, Imag
;
                        ADD saveRemImag, switchTime+2
                        ST saveRemImag, temp
                        JBS temp.15, negRemImag
                        CMP temp, halfCycleFreq
                        JNH remImagOk
                        INC Imag
                        SUB saveRemImag, cycleFreq
                        SJMP remImagOk
negRemImag:
                        NEG temp
                        CMP temp, halfCycleFreq
                        JNH remImagOk
                        DEC Imag
                        ADD saveRemImag, cycleFreq
;
;
remImagOk:

; put a safety current limiter
                        NEG maxImag
                        CMP Imag, maxImag
                        JGE bigger
;
                        ST maxImag, Imag
                        NEG maxImag
                        LJMP smaller
bigger:
                        NEG maxImag
                        CMP Imag, maxImag
                        JLE smaller
                        ST maxImag, Imag
```

hystbipl.a96 (continued) :

```
smaller:
                        ST Imag, oldImag
                        ST vdcError, oldVdcError
;
;
; A/D for A phase must have been done by now,
; so prepare to obtain A/D results
getAad:
                        JBS AD_LOW, AD_STATUS_BIT, getAad
;
adAdone:
                        ST AD_LOW, Iam
;
; do not waste time, start B-phase conversion immediately
; start a/d conversion for Phase B
;
                        LDB AD_COMMAND, #I_MONITOR_B
;
                        XORB IOPORT1, #00000100b
;
; calculate commanded current value for next phase
currentCommandA:
;
; set Phase A sine table index
                        ADD newIndx, oldIndx, incOfIndx
;
; wrap around index values if more than maximum index of sine table
                        ADD saveRemA, remA
                        CMP saveRemA, halfCountForAPeriod
                        JLT remAok
                        INC newIndx
                        SUB saveRemA, countForAPeriod
;
remAok:
                        ST newIndx, index
;           CMP index, #maxIndxForSinTabl
;           JNH indexAOK
;
;calculate (index modulo #lenOfSinTabl)
                        CLR switchTime+2
                        LD switchTime, index
                        DIVU switchTime, #lenOfSinTabl
                        LD index, switchTime+2
;
indexAOK:
                        ST index, oldIndx
;
; multiply index by 2 to account for word storage in table
                        SHL index, #1
```

hystbipl.a96 (continued) :

```
; multiply magnitude of sine wave reference by appropriate sine table entry
                        MUL Ias, Imag, sin[index]
;
;
;
; multiply magnitude of sine wave reference by appropriate sine table entry
                        MUL Ibs, Imag, sin120[index]
;
; multiply by 2 to compensate for signed multiply (A-phase)
; by another 2 for 2 turns thru' current transducers
                        SHLL Ias, #NUM_TURNS
;
; round off
                        JC phaseANegative
;
; only necessary for positive numbers
                        CLRC
                        ADD Ias, #8000h
                        ADDC Ias+2, 0
;
phaseANegative:
                        LD Ia, Ias+2
;
; multiply by 2 to compensate for signed multiply (B-phase)
; by another 2 for 2 turns thru' current transducers
                        SHLL Ibs, #NUM_TURNS
;
; round off
                        JC phaseBNegative
;
; only necessary for positive numbers
                        CLRC
                        ADD Ibs, #8000h
                        ADDC Ibs+2, 0
;
phaseBNegative:
                        LD Ib, Ibs+2
;
; now that all references have been calculated,
; work on A-phase current error
                        SHR Iam, #6
                        SUB Iam, #512
                        SUB Iam, Ia0
;
; let's assign new meanings to the variable Ia
; Ia (word) will contain the magnitude of the error of the A phase current
;
```

hystbipl.a96 (continued) :

```
; switch A-leg accordingly
compareA:
;
                        SUB errA, Ia, Iam
                        ST errA, Ia
                        JBC Ia.15, getBad
;
; error is negative, so generate the absolute value
                        NEG Ia
;
;
; prepare to obtain A/D results for B phase
getBad:
                        JBS AD_LOW, AD_STATUS_BIT, getBad
;
adBdone:
                        ST AD_LOW, Ibm
                        SHR Ibm, #6
                        SUB Ibm, #512
                        SUB Ibm, Ib0
;
; let's assign new meanings to the variables Ibs, Ib and Ibm
; Ibs (word) will contain the error of the B phase current
; Ib (word) will contain the magnitude of the error of the B phase current

;
; switch B-leg accordingly
compareB:
;
                        SUB errB, Ib, Ibm
                        ST errB, Ib
                        JBC Ib.15, switchC
;
; error is negative, so generate the absolute value
                        NEG Ib
;
;
;
; A and B phases have been switched
; so now switch phase C
; IrefC = -(IrefA +IrefB)
; errC will contain the error in the current waveform for Phase C
; errC = -(errA + errB)
;
switchC:
;
;
                        ADD Ics+2, Ias+2, Ibs+2
                        SUB Ics+2, 0, Ics+2
                        ADD errC, errA, errB
                        SUB errC, 0, errC
```

hystbipl.a96 (continued) :

```
; switch C-leg accordingly
compareC:
                        ST errC, Ic
                        JBC Ic.15, normalA
;
; error is negative, so generate the absolute value
                        NEG Ic
;
;
;
normalA:
;               XORB IOPORT1, #00001000b
                        CMP Ia, #I_HYSTERESIS
                        JNH normalB
;
                        JBS errA.15, checkForNegRefA
;
; positive error, current A too low
;
checkForPosRefA:
                        JBS Ias.31, normalB
overA:
                        JBS IOS0, HSO_HOLDING_BIT, overA
;
                        LDB HSO_COMMAND, #A_LOW_SET
                        ADD temp, TIMER_1, #Await
                        ST temp, HSO_TIME
;
                        SJMP normalB
;
;
; negative error, current A too high
;
checkForNegRefA:
                        JBC Ias.31, normalB
underA:
                        JBS IOS0, HSO_HOLDING_BIT, underA
;
                        LDB HSO_COMMAND, #A_HIGH_SET
                        ADD temp, TIMER_1, #Await
                        ST temp, HSO_TIME
;
;
;
;
normalB:
                        CMP Ib, #I_HYSTERESIS
                        JNH normalC
;
                        JBS errB.15, checkForNegRefB
.
```

hystbipl.a96 (continued) :

```
; positive error, current B too low
;
checkForPosRefB:
                        JBS Ibs.31, normalC
overB:
                        JBS IOS0, HSO_HOLDING_BIT, overB
;
                        LDB HSO_COMMAND, #B_LOW_SET
                        ADD temp, TIMER_1, #Bwait
                        ST temp, HSO_TIME
;
                        SJMP normalC
;
; negative error, current B too high
;
checkForNegRefB:
                        JBC Ibs.31, normalC
underB:
                        JBS IOS0, HSO_HOLDING_BIT, underB
;
                        LDB HSO_COMMAND, #B_HIGH_SET
                        ADD temp, TIMER_1, #Bwait
                        ST temp, HSO_TIME
;
;
normalC:
                        CMP Ic, #I_HYSTERESIS
                        JNH shutOffAll
;
                        JBS errC.15, checkRefForNegC
;
; positive error, current C too low
;
checkRefForPosC:
                        JBS Ics.31, shutOffAll
overC:
                        JBS IOS0, HSO_HOLDING_BIT, overC
;
                        LDB HSO_COMMAND, #C_LOW_SET
                        ADD HSO_TIME, TIMER_1, #sync
;
                        SJMP shutOffAll
;
; negative error, current C too high
;
checkRefForNegC:
                        JBC Ics.31, shutOffAll
underC:
                        JBS IOS0, HSO_HOLDING_BIT, underC
```

hystbipl.a96 (continued) :

```
                        LDB HSO_COMMAND, #C_HIGH_SET
                        ADD HSO_TIME, TIMER_1, #sync
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Turn-off all devices done simultaneously.  Modified 3/02/97  SB
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
shutOffAll:
                        JBS IOS0, HSO_HOLDING_BIT, shutOffAll
;
                        LDB HSO_COMMAND, #HSOALL_CLEAR
                        ADD HSO_TIME, TIMER_1, onTime
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
goBack:
                        POPF
                        RET
;
                END
```

reset.a96 :

```
;*****************************************************************/
;                                                                 */
;   Active rectifier controller                                   */
;   Copyright 1997 Oregon State University, Corvallis OR          */
;                                                                 */
;   Controller developed by Shibashis Bhowmik                     */
;   Software coded by Shibashis Bhowmik                           */
;                                                                 */
;*****************************************************************/
reSet  MODULE  STACKSIZE(0)
; this module is for emergency shutdown
; $INCLUDE (80196.INC)
; $INCLUDE (DEFINE.INC)
;EXTRN      lenOfSinTabl: NULL
;EXTRN      phaseShift: NULL
PUBLIC     rest
;RSEG
CSEG at 6800h
;
rest:
                    RST
              RET
;
;
      END
```

Header files for C-programs:

80196.h :

```
/*************************************************************************/
/*                                                                       */
/*      Active rectifier controller                                      */
/*      Copyright 1997 Oregon State University, Corvallis OR             */
/*                                                                       */
/*      Controller developed by Shibashis Bhowmik                        */
/*      Software coded by Shibashis Bhowmik                              */
/*      80196.H header originally developed by Brian Wiley               */
/*                                                                       */
/*************************************************************************/
/*    80196.h
      special function registers of the 80196
*/

#ifndef _80196
#define _80196

/* Window Select Register
*/
#define WSR                     (* (unsigned char *)( 0x14 ))


/* Software and Hardware protection
*/
#define WATCHDOG                (*(unsigned char *)(0x0A))


/* I/O Status Registers
*/
#define IOS0                (* (unsigned char *)( 0x15 ))
#define HSO_0_STATE             0x01
#define HSO_1_STATE             0x02
#define HSO_2_STATE             0x04
#define HSO_3_STATE             0x08
#define HSO_4_STATE             0x10
#define HSO_5_STATE             0x20
#define HSO_CAM_FULL        0x40
#define HSO_HOLDING_FULL    0x80
/*
#define HSO_0_OFF               0x3E
#define HSO_1_OFF               0x3D
#define HSO_2_OFF               0x3B
#define HSO_3_OFF               0x37
#define HSO_4_OFF               0x2F
#define HSO_5_OFF               0x1F
*/
#define IOS1                (* (unsigned char *)( 0x16 ))
#define HSI_FIFO_FULL       0x40
```

80196.h (continued):

```
#define HSI_READY               0x80

#define IOS2                    (* (unsigned char *)( 0x17 ))
#define HSO_0_EVENT             0x01
#define HSO_1_EVENT             0x02
#define HSO_2_EVENT             0x04
#define HSO_3_EVENT             0x08
#define HSO_4_EVENT             0x10
#define HSO_5_EVENT             0x20
#define HSO_T2_EVENT            0x40
#define HSO_AD_EVENT            0x80


/* I/O Control Registers
*/
#define IOC0                    (* (unsigned char *)( 0x15 ))
/* note: bit 1 always 1 on reads
*/
#define HSI0_ENABLE             0x01
#define SOFT_T2_RESET           0x02
#define HSI1_ENABLE             0x04
#define EXT_T2_RESET            0x08
#define HSI2_ENABLE             0x10
#define HSI0_RESET_SRC          0x20
#define HSI3_ENABLE             0x40
#define HSI1_CLK_SRC            0x80
/**/
#define IOC1                    (* (unsigned char *)( 0x16 ))
#define PWM0_ENABLE             0x01
#define T2_OVFL_INT             0x08
#define HSO_4_ENABLE            0x10
#define TXD_ENABLE              0x20
#define HSO_5_ENABLE            0x40
/**/
#define IOC2                    (* (unsigned char *)( 0x0B ))
/* note: bit 7 always 1 on reads
*/
#define T2_FAST_ENABLE          0x01
#define T2_DOWN_ENABLE          0x02
#define SLOW_PWM                0x04
#define AD_TIME_ENABLE          0x08
#define AD_NOT_PRESCALE         0x10
#define T2_MIDCNT_INT           0x20
#define HSO_LOCK_ENABLE         0x40
#define HSO_CAM_CLEAR           0x80
#define HSOALL                  0x0C
/**/
#define BAUD_RATE               (* (unsigned char *)( 0x0E ))
#define BAUD_XTAL1              0x80
```

80196.h (continued):

```
#define IOC3                    (* (unsigned char *)( 0x0C))
/* note: bits 4-7 reserved, always write as ones
*/
#define T2_INTERNAL             0x01
#define T2_CLOCK_DISABLE        0x02
#define PWM1_SELECT             0x04
#define PWM2_SELECT             0x08
#define IOC3_RESERVED           0xF0
/**/
#define BAUD_RATE               (* (unsigned char *)( 0x0E ))
#define BAUD_XTAL1              0x80

/* Serial Port Registers
*/
/* status
*/
#define SP_STAT                 (* (unsigned char *)( 0x11 ))
#define SP_TXE                  0x08
#define SP_TI                   0x20
#define SP_RI                   0x40
/* control
*/
#define SP_CON                  (* (unsigned char *)( 0x11 ))
#define SP_MODE_0               0x00
#define SP_MODE_1               0x01
#define SP_MODE_2               0x02
#define SP_MODE_3               0x03
#define SP_PEN                  0x04
#define SP_REN                  0x08
#define SP_TB8                  0x10
/* buffers
        same address for reads and writes
*/
#define SBUF                    (* (unsigned char *)( 0x07 ))

/* Timer 1 Register
        note: must be written in window 15
*/
#define TIMER_1                 (* (unsigned int *)( 0x0A ))

/* Timer 2 Register
*/
#define TIMER_2                 (* (unsigned int *)( 0x0C ))

/* High Speed Output Registers
*/
#define HSO_TIME                (* (unsigned int *)   ( 0x04 ))
#define HSO_COMMAND             (* (unsigned char *)( 0x06 ))
#define HSO_TIMER_0             0x08
#define HSO_TIMER_1             0x09
```

80196.h (continued):

```
#define HSO_TIMER_2              0x0A
#define HSO_TIMER_3              0x0B
#define HSO_TIMER_RESET          0x0E
#define HSO_AD                   0x0F
#define HSO_INTERRUPT            0x10
#define HSO_SET                  0x20
#define HSO_CLEAR                0x00
#define HSO_TIMER                0x40        /* 1=timer 2, 0=timer 1 */
#define HSO_CAM_LOCK             0x80

/* High Speed Input Registers
*/
#define HSI_MODE                 (* (unsigned char *)( 0x03 ))
#define HSI_POSITIVE             0x01
#define HSI_NEGATIVE             0x02
#define HSI_TIME                 (* (unsigned int  *)( 0x04 ))
#define HSI_STATUS               (* (unsigned char *)( 0x06 ))

/* positive transitions */

#define HSI0_STATUS              0x01

/* negative transitions */

/* #define HSI0_STATUS           0x02 */



/* A to D converter
*/
#define AD_COMMAND               (* (unsigned char *)( 0x02 ))
#define AD_GO                    0x08
#define AD_8_BITS                0x10
/**/
#define AD_LOW                   (* (unsigned char *)( 0x02 ))
#define AD_BUSY                  0x08
#define AD_HIGH                  (* (unsigned char *)( 0x03 ))
#define AD_BITS                  0xFFC0
#define AD_SEL                   0x02

/* Peripheral Transaction Server
*/

#define PTSSEL                   (* (unsigned int *)( 0x04 ))


/* I/O port 1
        quasi-bidirectional
*/
#define IOPORT1                  (* (unsigned char *)( 0x0F ))
```

80196.h (continued):

```
/* I/O port 2
        shared with special functions
*/
#define IOPORT2             (* (unsigned char *)( 0x10 ))
#define HV_ENABLE             0x020

/* interrupt control
*/
#define INT_MASK              (* (unsigned char *)( 0x08 ))
#define INT_PEND              (* (unsigned char *)( 0x09 ))
#define AD_PEND               0x02


#define TIMER_MASK              0x01
#define AD_MASK               0x02
#define HSIDAT_MASK             0x04
#define HSO_MASK              0x08
#define HSI0_MASK             0x10
#define SWT_MASK              0x20
#define SER_MASK              0x40
#define EXTINT_MASK             0x80


#define INT_MASK1             (* (unsigned char *)( 0x13 ))
#define INT_PEND1             (* (unsigned char *)( 0x12 ))
#define TI_MASK               0x01
#define RI_MASK               0x02
#define HSI4_MASK               0x04
#define T2CAP_MASK              0x08
#define T2OVF_MASK              0x10
#define EXTINT1_MASK          0x20
#define FIFO_MASK               0x40
#define NMI_MASK              0x80


#endif
```

global.h :

```
/*************************************************************************/
/*                                                                       */
/*      Active rectifier controller                                      */
/*      Copyright 1997 Oregon State University, Corvallis OR             */
/*                                                                       */
/*      Controller developed by Shibashis Bhowmik                        */
/*      Software coded by Shibashis Bhowmik                              */
/*      80196.H header originally developed by Brian Wiley               */
/*                                                                       */
/*************************************************************************/
#ifndef _global
#define _global
#define SYSTEM_CLOCK   16.0e6
#define STATE_TIME     (2.0/SYSTEM_CLOCK)
#define TIMER1_PERIOD  (8.0*STATE_TIME)
#define P              10

#define MAXCOUNT_16 0xFFFF

#define BAUD_OUT            9600
/* note: divide by 16 for 80c196kc, not 64
#define BAUD_WORD           (unsigned int)(SYSTEM_CLOCK/(16*BAUD_OUT) - 1)
#define BAUD_LOW            BAUD_WORD%256
#define BAUD_HIGH           BAUD_WORD/256
*/

#define BAUD_LOW            103
#define BAUD_HIGH           0


/* soft start for rectifier */
#define KiMAX         8     /* for a divisor of 16, 500 for a divisor of 1024 */
#define KpMAX         200
#define STEPCYCLE           5     /*  equivalent to 1/40 of a 60 Hz cycle  */

/* limiters 2/27/97
*/

#define PEAK_CURR           150    /* corresponds to 15 Arms on SPHINX */
#define MIN_OFF_TIME   15     /* provides for max. 0.9 duty */
#define MAX_FREQ            900    /* 90.0 Hz */

/* lines per revolution
*/
#define ENCODER_RESOLUTION     4000

/* bits for 5V analog input (full scale)
*/
#define A_D_RESOLUTION  1024
```

global.h (continued) :

```
/* A to D channels
*/
#define BRAKE_COMMAND        0
#define I_MONITOR_A          1
#define I_MONITOR_B          2
#define TORQUE_COMMAND       5
#define DRIVE_MONITOR        4
#define HV_MONITOR           3
/* HV_MONITOR and TORQUE_COMMAND switched for Sybill.
*/
#define I_MONITOR_C          6
#define BATTERY_MONITOR      7


/* high speed output channels
*/
#define A_LOW                0
#define A_HIGH               1
#define B_LOW                2
#define B_HIGH               3
#define C_LOW                4
#define C_HIGH               5

/* self test parameters
*/
#define I_NOISE              5
#define I_LOW_FAULT          15
#define LOW_TIMEOUT          4000
#define I_PULSE_MIN          25
#define I_PULSE_MAX          50

/* fault indicators
*/
#define A_HIGH_FAULT         0x02
#define A_LOW_FAULT          0x01
#define B_HIGH_FAULT         0x04
#define B_LOW_FAULT          0x20
#define C_HIGH_FAULT         0x40
#define C_LOW_FAULT          0x80
#define FAULT_MASK           0xE7

/* voltage levels (bits)
*/
#define LV_MIN               532          /* 14.8 Volts */
#define LV_MAX               546          /* 15.2 Volts */
#define HV_MIN               350          /* 105.6 Volts */
#define HV_MAX               539          /* 150  Volts */
```

global.h (continued) :

```
/* motor types
*/
#define MODIFIED                1
#define REWOUND                 2
#define MOTOR                   MODIFIED


#endif
```

proto.h (continued) :

```
/* #define DOWNLOAD  1 */

#ifndef _proto
#define _proto

void begin( void );

/* vdc_iref */

void initCurrent(void) ;
void timer2Init(void) ;
void setUpEmgncDown(void) ;
void setUpSwftInit(void) ;

/* A to D
*/
void AtoD_init(void);
void AtoD_start( int );
unsigned int AtoD_read( void );

/* HSO
*/
void hso_init( void );
void hso_set( int );
void hso_clear( int );
void hsoAllClear( void );
void hsoCamClear( void );
void hso_will_set( int, unsigned int ) ;
void hso_will_clear( int, unsigned int ) ;
void hso_will_AD (unsigned int, unsigned int) ;
```

proto.h (continued) :

```
/* HSI
*/
void hsiInit(void) ;
void hsiGo(void) ;
void hsiStop(void) ;

/* SERIAL
*/
void serial_init( void );
char spchk( void );
unsigned char getBuf( void );
unsigned char getChar( void );
void putChar( unsigned char );

#endif
```

Assembly code include files:

80196.inc :

```
;*****************************************************************/
;                                                                 */
;    Active rectifier controller                                  */
;    Copyright 1997 Oregon State University, Corvallis OR         */
;                                                                 */
;    Controller developed by Shibashis Bhowmik                    */
;    Software coded by Shibashis Bhowmik                          */
;                                                                 */
;*****************************************************************/
;    - 80196.INC -
;
;
;    Special Function Registers
;
SP          equ   18h
WSR         equ 14h
;
IOS0                equ         15h
HSO0_STATE          equ         01h
HSO1_STATE          equ         02h
HSO2_STATE          equ         04h
HSO3_STATE          equ         08h
HSO4_STATE          equ         10h
HSO5_STATE          equ         20h
HSO_CAM_FULL        equ         40h
```

80196.inc (continued) :

```
HSO_HOLDING_FULL    equ     80h
HSO0_STAT_BIT       equ     0
HSO1_STAT_BIT       equ     1
HSO2_STAT_BIT       equ     2
HSO3_STAT_BIT       equ     3
HSO4_STAT_BIT       equ     4
HSO5_STAT_BIT       equ     5
HSO_HOLDING_BIT     equ     7
;
IOS1                equ     16H
HSI_FIFO_FULL       equ     40H
HSI_READY           equ     80H
SWTF0_BIT           equ     0
SWTF1_BIT       equ     1
SWTF2_BIT       equ     2
SWTF3_BIT       equ     3
;
;
IOS2                equ     17h
HSO_0_EVENT         equ     01h
HSO_1_EVENT         equ     02h
HSO_2_EVENT         equ     04h
HSO_3_EVENT         equ     08h
HSO_4_EVENT         equ     10h
HSO_5_EVENT         equ     20h
HSO_T2_EVENT        equ     40h
HSO_AD_EVENT        equ     80h
;
;
IOC0                equ     15h
; note: bit 1 always 1 on reads
;
HSI0_ENABLE         equ     01h
SOFT_T2_RESET       equ     02h
HSI1_ENABLE         equ     04h
EXT_T2_RESET        equ     08h
HSI2_ENABLE         equ     10h
HSI0_RESET_SRC      equ     20h
HSI3_ENABLE         equ     40h
HSI1_CLK_SRC        equ     80h
;
;
IOC1                equ     16h
PWM0_ENABLE         equ     01h
T2_OVFL_INT         equ     08h
HSO_4_ENABLE        equ     10h
TXD_ENABLE          equ     20h
HSO_5_ENABLE        equ     40h
:
```

80196.inc (continued) :

```
IOC2                    equ         0Bh
; note: bit 7 always 1 on reads
;
T2_FAST_ENABLE      equ         01h
T2_DOWN_ENABLE      equ         02h
SLOW_PWM            equ         04h
AD_TIME_ENABLE      equ         08h
AD_NOT_PRESCALE     equ         10h
T2_MIDCNT_INT       equ         20h
HSO_LOCK_ENABLE     equ         40h
HSO_CAM_CLEAR       equ         80h
;
BAUD_RATE           equ         0Eh
BAUD_XTAL1          equ         80h
;
IOC3                    equ         0Ch
; note: bits 4-7 reserved, always write as ones
;
T2_INTERNAL         equ         01h
T2_CLOCK_DISABLE    equ         02h
PWM1_SELECT         equ         04h
PWM2_SELECT         equ         08h
IOC3_RESERVED       equ         0F0h
;
;
;
SP_STAT             equ         11h
SP_TXE              equ         08h
SP_TI               equ    20h
SP_RI               equ         40h
; control
;
SP_CON              equ         11h
SP_MODE_0           equ         00h
SP_MODE_1           equ         01h
SP_MODE_2           equ         02h
SP_MODE_3           equ         03h
SP_PEN              equ         04h
SP_REN              equ         08h
SP_TB8              equ         10h
; buffers
;       same address for reads and writes
;
SBUF                    equ         07h
;
; Timer 1 Register
;       note: must be written in window 15
;
TIMER_1             equ         0Ah
```

80196.inc (continued) :

```
; Timer 2 Register
;
TIMER_2                 equ         0Ch
;
; High Speed Output Registers
;
HSO_TIME                equ         04h
HSO_COMMAND             equ         06h
HSO_AD                  equ         0Fh
HSO_INTERRUPT           equ         10h
HSO_SET                 equ         20h
HSO_CLEAR               equ         00h
HSO_TIMER               equ         40h     ;   /* 1=timer 2, 0=timer 1 */
HSO_TIMER_RESET         equ         (0Eh OR HSO_TIMER)
HSO_CAM_LOCK            equ         80h
HSO_SWTF0               equ         (08h OR HSO_INTERRUPT)
HSO_SWTF1               equ         (09h OR HSO_INTERRUPT)
HSO_SWTF2               equ         (0Ah OR HSO_INTERRUPT)
HSO_SWTF3               equ         (0Bh OR HSO_INTERRUPT)
;
; High Speed Input Registers
;
HSI_MODE                equ         03h
HSI_POSITIVE            equ         01h
HSI_NEGATIVE            equ         02h
HSI_TIME                equ         04h
HSI_STATUS              equ         06h
HSI0_STATUS             equ         02h
;
;
AD_COMMAND              equ         02h
AD_GO                   equ         08h
AD_8_BITS               equ         10h
AD_LOW                  equ         02h
AD_BUSY                 equ         08h
AD_HIGH                 equ         03H
AD_BITS                 equ         0FFC0h
AD_SEL                  equ         02h
AD_STATUS_BIT           equ         3
;
;
; Peripheral Transaction Server
;
;
PTSSEL                  equ         04h
;
;
; I/O port 1
;     quasi-bidirectional
;
IOPORT1                 equ         0Fh
```

80196.inc (continued) :

```
; I/O port 2
;     shared with special functions
;
IOPORT2             equ     10h
HV_ENABLE             equ     20h
;
; interrupt control
;
INT_MASK            equ     08h
INT_PEND            equ     09h
AD_PEND             equ     02h
;
;
;
TIMER_MASK            equ     01h
AD_MASK             equ     02h
HSIDAT_MASK           equ     04h
HSO_MASK            equ     08
HSI0_MASK           equ     10h
SWT_MASK            equ     20h
SER_MASK            equ     40h
EXTINT_MASK           equ     80h
INT_MASK1           equ     11h
INT_PEND1           equ     12h
TI_MASK             equ     01h
RI_MASK             equ     02h
HSI4_MASK           equ     04h
T2CAP_MASK            equ     08h
T2OVF_MASK            equ     10h
EXTINT1_MASK        equ     20h
FIFO_MASK           equ     40h
NMI_MASK            equ     80h
;
;
WATCHDOG              equ     0Ah
```

define.inc :

```
;*************************************************************************/
;                                                                         */
;    Active rectifier controller                                          */
;    Copyright 1997 Oregon State University, Corvallis OR                 */
;                                                                         */
;    Controller developed by Shibashis Bhowmik                            */
;    Software coded by Shibashis Bhowmik                                  */
;                                                                         */
;*************************************************************************/
; define.inc
;     assembly language definitions
;
;
; sine table
;
lenOfSinTabl        equ         3000
maxIndxForSinTabl   equ         lenOfSinTabl - 1
phaseShift          equ         1000
; lowCount          equ         15900   ; 63 Hz
; highCount         equ         17500   ; 57 Hz
lowCount            equ         16393   ; 61 Hz
highCount           equ         16949   ; 59 Hz
;
; width of hysteresis band
;
;
; A to D channels
;
I_MONITOR_A         equ         (1 OR AD_GO)
I_MONITOR_B         equ         (2 OR AD_GO)
HV_MONITOR          equ         (3 OR AD_GO)
;HV_MONITOR is 5 in Shiba's and SPOT's. Only 3 for Sybill.
I_MONITOR_C         equ         (6 OR AD_GO)
;
;
; high speed output channels
;
A_HIGH              equ         0
A_LOW               equ         1
B_HIGH              equ         2
B_LOW               equ         3
C_HIGH              equ         4
C_LOW               equ         5
;
;
; HSO reference TIMER_1
; positive logic
;
A_LOW_CLEAR             EQU         01
B_LOW_CLEAR             EQU         03
```

define.inc (continued) :

```
A_HIGH_CLEAR        EQU         00
B_HIGH_CLEAR        EQU         02
C_HIGH_CLEAR        EQU         04
HSOALL_CLEAR        EQU         0Ch
A_LOW_SET             EQU         (01 OR HSO_SET)
B_LOW_SET             EQU         (03 OR HSO_SET)
C_LOW_SET             EQU         (05 OR HSO_SET)
A_HIGH_SET            EQU         (00 OR HSO_SET)
B_HIGH_SET            EQU         (02 OR HSO_SET)
C_HIGH_SET            EQU         (04 OR HSO_SET)
;
;
;
sync            equ         2
DEAD_TIME           equ         4
I_HYSTERESIS   equ          2
Await           equ         19
Bwait           equ         11
NUM_TURNS           equ         2     ;two turns thru' current transducers
;
;
; voltage levels (bits)

LV_MIN          equ         532             ; 14.8 Volts
LV_MAX          equ         546             ; 15.2 Volts
HV_MIN          equ         350             ; 105.6 Volts
HV_MAX          equ         539             ; 150  Volts
;
;
; protection code (2/06/97)
;
MAXHV       equ         890    ; corresponds to 420V
```

system.inc :

```
;*****************************************************************/
;                                                                 */
;    Active rectifier controller                                  */
;    Copyright 1997 Oregon State University, Corvallis OR         */
;                                                                 */
;    Controller developed by Shibashis Bhowmik                    */
;    Software coded by Shibashis Bhowmik                          */
;                                                                 */
;*****************************************************************/
; system data
;
sin         equ         2088h
sin120  equ         37F8h
```

memory.inc :

```
;*****************************************************************/
;                                                                 */
;    Active rectifier controller                                  */
;    Copyright 1997 Oregon State University, Corvallis OR         */
;                                                                 */
;    Controller developed by Shibashis Bhowmik                    */
;    Software coded by Shibashis Bhowmik                          */
;                                                                 */
;*****************************************************************/
; memory.inc
;     allocation of on-chip registers
;

RSEG AT 3Ch

; LONG
; double word (long int) aligned
;
;
; Commanded  value of A phase current. A long for calculation from Im and sin.
Ias:            DSL         1
;
; Commanded  value of B phase current.
Ibs:            DSL         1
;
; Commanded  value of B phase current.
Ics:            DSL         1
;
; Variable for frequency adjustment calculations
switchTime:         DSL         1
;
;
;
; INT
; word (int) aligned
;
;
; Commanded phase currents.
Ia:             DSW         1
Ib:             DSW         1
Ic:             DSW         1
;
; Current sensor reading at zero current.
Ia0:        DSW         1
Ib0:        DSW         1
vdc0:       DSW         1
;
; Measured deviation of currents from that commanded.
Iam:        DSW         1
Ibm:        DSW         1
Icm:        DSW         1
;
```

memory.inc (continued) :

```
; Variables needed to keep track of real time.
cycleTime:          DSW         1
waitForNext:   DSW   1
onTime:             DSW         1
oldIndx:            DSW         1
newIndx:            DSW         1
incOfIndx:          DSW         1
currentLag:         DSW         1
;
; Variable for frequency adjustment calculations
countForAPeriod:          DSW         1
halfCountForAPeriod:   DSW         1
saveRemA:                 DSW         1
remA:                     DSW         1
oldHsiTime:               DSW         1
newHsiTime:               DSW         1
;
;
; Magnitude of sinusoidal current
Imag:       DSW         1
;
;
; Temporary storage of index for sine table
index:      DSW         1
;
;
; Temporary variable for calculations
temp:       DSW   1
;
; Registers for dc bus regulator
vdc:              DSW         1
vdcError:         DSW         1
vdcDiode:         DSW         1
hvRef:       DSW         1
Ki:               DSW         1
Kp:               DSW         1
oldImag:          DSW         1
maxImag:          DSW         1
oldVdcError:   DSW         1
;
; error storage
errA:             DSW         1
errB:             DSW         1
errC:             DSW         1
;
; Word storage for counting number of program loops
vdcConst:         DSW         1
errorBand:        DSW         1
;
;
saveRemImag:   DSW         1
cycleFreq:        DSW         1
```

memory.inc (continued) :

```
halfCycleFreq: DSW          1
;
;
; BYTE
; byte (char) aligned
;
; Temporary storage for IOS1 register
;
tempIOS1:           DSB          1
commandChar:   DSB          1
i:                 DSB          1
stop:              DSB          1
cycleCount:        DSB          1
;
;
;
DSEG
vChar  equ    $:NULL
       dsb    3
;
;
;
;
;
;
; EXTERN
;
PUBLIC Ia
PUBLIC Ib
PUBLIC Ic
PUBLIC Ias
PUBLIC Ibs
PUBLIC Ics
PUBLIC Iam
PUBLIC Ibm
PUBLIC Icm
PUBLIC Ia0
PUBLIC Ib0
PUBLIC vdc0
PUBLIC oldIndx
PUBLIC newIndx
PUBLIC incOfIndx
PUBLIC cycleTime
PUBLIC waitForNext
PUBLIC onTime
PUBLIC switchTime
PUBLIC countForAPeriod
PUBLIC halfCountForAPeriod
PUBLIC saveRemA
PUBLIC remA
PUBLIC oldHsiTime
PUBLIC newHsiTime
```

memory.inc (continued) :

```
PUBLIC index
PUBLIC tempIOS1
PUBLIC temp
PUBLIC vdc
PUBLIC vdcError
PUBLIC vdcDiode
PUBLIC errorBand
PUBLIC oldVdcError
PUBLIC hvRef
PUBLIC Ki
PUBLIC Kp
PUBLIC oldImag
PUBLIC maxImag
PUBLIC currentLag
PUBLIC vdcConst
PUBLIC saveRemImag
PUBLIC halfCycleFreq
PUBLIC cycleFreq
PUBLIC commandChar
PUBLIC vChar
PUBLIC i
PUBLIC stop
PUBLIC cycleCount
```

Files for compiling :

link96.bat :

```
\shiba\asm96\rl96 & < start.lnk
```

start.lnk :

```
main.obj, begin.obj, vdc_iref.obj, global.obj, hso.obj, hsi.obj, &
atod.obj, serial.obj, startup.obj, hystbipl.obj, &
reset.obj, c:\shiba\ic96\lib\c96.lib to rec.out &
registers(1AH - 0FFH) list(pl, sb, ln, sm) print windowsize(0)
```

makefile :

```
#
start.out:                      main.obj        \
                                begin.obj       \
                                vdc_iref.obj    \
                                global.obj      \
                                hso.obj         \
                                hsi.obj         \
                                atod.obj        \
                                serial.obj      \
                                startup.obj     \
                                hystbipl.obj    \
                                reset.obj       \


        link96.bat

main.obj:                       main.c proto.h
        \shiba\ic96\bin\ic96  main.c md(kc) co li nore sb xr ot(3)

begin.obj:              begin.c proto.h 80196.h global.h
        \shiba\ic96\bin\ic96  begin.c md(kc)  co li nore sb xr ot(3)

vdc_iref.obj:       vdc_iref.c proto.h 80196.h global.h
        \shiba\ic96\bin\ic96  vdc_iref.c md(kc)  co li nore sb xr ot(3)

global.obj:             global.c ctype.h
        \shiba\ic96\bin\ic96  global.c md(kc)  co li nore sb xr ot(3)

hso.obj:        hso.c proto.h 80196.h global.h
        \shiba\ic96\bin\ic96  hso.c md(kc)  co li nore sb xr ot(3)

hsi.obj:        hsi.c proto.h 80196.h global.h
        \shiba\ic96\bin\ic96  hsi.c md(kc)  co li nore sb xr ot(3)

atod.obj:       atod.c proto.h 80196.h global.h
        \shiba\ic96\bin\ic96  atod.c md(kc)  co li nore sb xr ot(3)

serial.obj:                     serial.c proto.h 80196.h global.h
        \shiba\ic96\bin\ic96    serial.c md(kc)  co li nore sb xr ot(3)

startup.obj:            startup.a96
        \shiba\asm96\asm96  startup.a96

hystbipl.obj:           hystbipl.a96 80196.inc memory.inc define.inc system.inc
        \shiba\asm96\asm96  hystbipl.a96

reset.obj:              reset.a96 80196.inc define.inc
        \shiba\asm96\asm96  reset.a96
```

A.2 Protection logic

over.asm :

```
;overvoltage and overcurrent protection
;algorithm developed by Shibashis Bhowmik
;code developed by Shibashis Bhowmik
;Nov 27, 1996
;
name OVER
;
;
;
STACK           segment DATA
;VARIABLES      segment DATA
FLAGS           segment BIT
;
;
                bseg
VDCLL                   bit     P3.0
VDCGH                   bit     P3.1
USER_ON                 bit     P3.2
ILKOVRLD                bit     P3.3
OVERVM                  bit     P3.4
EXTERNAL1               bit     P3.5
SHUTDOWN                bit     P1.2
DON                     bit     P1.3
GRELAY                  bit     P1.4
RRELAY                  bit     P1.5
OVERVOLT                bit     P1.6
OVERCURR                bit     P1.7
;
public  VDCLL,VDCGH,USER_ON,ILKOVRLD,OVERVM,EXTERNAL1
public  SHUTDOWN,DON,GRELAY,RRELAY,OVERVOLT,OVERCURR
;
;
;
; reserve space for stack
;
                rseg    STACK
                ds      20h             ;reserve 32 bytes for stack
;
;
                dseg    at 0050h
COUNTER:        ds      1
;
public COUNTER
;
;
                rseg    FLAGS
.
```

over.asm (continued) :

```
VDC_CHRGD:  dbit    1
;
;
            cseg    at 0000h
;
            ljmp _main
;
;
            cseg    at 0100h
;
DC_BUS_CHRG_CNT  equ    6
TIMER_BIT16      equ    00000001b
;
public  TIMER_BIT16
extrn NUMBER(initBegin,overvBegin)
;
;
_main:
            mov SP, #STACK-1
            setb DON
            clr GRELAY
            clr RRELAY
            clr SHUTDOWN
            clr OVERVM
            clr IE1                 ;remove previous instances of INT1
            clr EXTERNAL1
            clr OVERVOLT
            clr OVERCURR
;
; GRELAY = !SHUTDOWN
; turn on the grid relay
;
; wait for user to indicate turn-on
;
waitForUserOn:
            clr EA
            clr RRELAY
            clr GRELAY
;

; initialize timer0 as a 16-bit timer
;
            mov TMOD, #TIMER_BIT16
            clr A
            mov TL0, A
            mov TH0, A
;
;
; allow 400 ms for the contactors to open
;
```

over.asm (continued) :

```
                mov COUNTER, #DC_BUS_CHRG_CNT
;
openContactor:
                setb TR0
;
                jnb TF0, $
                clr TR0                 ;stop counter
                clr TF0                 ;reset overflow flag
                djnz COUNTER, openContactor
;
;
; discharge the dc-bus upon exiting due to a normal shutdown
; keep the dump IGBT on for 400 ms.
;
                mov COUNTER, #DC_BUS_CHRG_CNT
;
; set dump ON
;
                clr DON
;
; start (run) timer 0
;
discharge:
                setb TR0
;
                jnb TF0, $
                clr TR0                 ;stop counter
                clr TF0                 ;reset overflow flag
                djnz COUNTER, discharge
;
                setb DON
;
                jnb USER_ON, $
                setb GRELAY
;
; wait for dc bus to charge up using current limiting resistors
; provide for about 200 ms for the resistors to be in circuit
; the RC time constant (R=20, C=2.1m) = 40m
;
                mov COUNTER, #DC_BUS_CHRG_CNT
;
; initialize timer0 as a 16-bit timer
;
                mov TMOD, #TIMER_BIT16
                clr A
                mov TL0, A
                mov TH0, A
;
; start (run) timer 0
;
```

over.asm (continued) :

```
run_ti0:
                setb TR0
;
wait_ti0:
                jnb TF0, wait_ti0
                clr TR0                 ;stop counter
                clr TF0                 ;reset overflow flag
                djnz COUNTER, run_ti0
;
; set flag to indicate dc-bus is charged
;
                setb VDC_CHRGD
;
; short out the resistances
; RRELAY = GRELAY & VDC_CHRGD
;
                setb    RRELAY
;
;
; setup the interrupts
;
                lcall initBegin
;
; wait for overvoltage or overcurrent
; polling for overvoltage and make sure
; (idea of detecting VDCGH implemented from Alex's program for SPOT)
;
waitForOverVolt:
                jnb USER_ON, waitForUserOn
                jb SHUTDOWN, hangThere
                jnb VDCGH, waitForOverVolt
                jnb VDCGH, waitForOverVolt
                lcall overvBegin
                sjmp waitForOverVolt
;
;
;
hangThere:
                clr EA                                  ;no more interrupts
                sjmp $                                  ;remain here, shutdown
;
;
END
```

overv.asm :

```
name OVERV
;
;
extrn BIT(DON,VDCLL,VDCGH,GRELAY,RRELAY,SHUTDOWN,OVERVOLT,OVERVM)
extrn DATA(COUNTER), CODE(TIMER_BIT16)
;
;
OVERV_CODE  segment CODE
;
                rseg    OVERV_CODE
;
DUMP_ON_CNT         equ     1               ;to provide for a maximum of 67 ms of dump at a
time
public  overvBegin
;
overvBegin:
;
; preserve old uC settings
;
                push PSW
;
;
                clr EA          ;disable all interrupts
                clr EX1         ;disable external interrupt 1
                clr DON         ;turn dump ON
;
;
                mov COUNTER, #DUMP_ON_CNT
;
; initialize timer0 as a 16-bit timer
;
                mov TMOD, #TIMER_BIT16
                clr A
                mov TL0, A
                mov TH0, A
;
                nop
                nop
                nop     ;to allow for transients due to dump switch die down
                nop
                nop
                nop
                nop
;
; start (run) timer 0
run_ti0:
                setb TR0
;
lowVolt:
                jb VDCLL, dumpSuccess  ;if vdc is low, dump worked
```

overv.asm (continued):

```
                jb VDCGH, gridOff           ;if dc-bus continues to rise, shutdown
                jnb TF0, lowVolt
                clr TR0
                clr TF0
                djnz COUNTER, run_ti0
gridOff:
                clr GRELAY                  ;open grid contactor
                clr RRELAY                  ;bring in resistors
                clr DON                 ;use dump to dissipate energy in link
                setb SHUTDOWN
                setb OVERVOLT
                clr OVERVM
;               pop PSW
                sjmp $                  ;never return to main code, remain here
;
dumpSuccess:
                setb DON                    ;turn-off dump IGBT
                clr TR0                 ;stop timer 0
                clr OVERVM
;
                pop PSW
;
                ret
END
```

init.asm :

```
name INIT
;
INIT_CODE           segment CODE
;
                rseg INIT_CODE
;
public initBegin
;
initBegin:
                setb EA         ;enable all interrupts, individual activation needed
                setb IT1        ;falling edge interrupt
                setb EX1                ;enable external 1 interrupt
                clr EX0         ;disable external 0 interrupt
                clr ET0         ;disable timer 0 interrupt
;
                ret
;
END
```

ext1.asm :

```
name EXT1
;
;
;
        cseg at 0013H
;
                ljmp ext1Begin
;
;
EXT1_CODE    segment CODE
;
        rseg EXT1_CODE
;
;
extrn DATA(COUNTER)
extrn
BIT(DON,GRELAY,RRELAY,EXTERNAL1,SHUTDOWN,VDCGH,ILKOVRLD,OVERCURR)
extrn CODE(TIMER_BIT16)
;
OVRLD_CNT    equ      150
;
ext1Begin:
                push PSW
                clr EA              ;disable all interrupts
                clr EX1             ;disable external interrupt 1
                setb EXTERNAL1
;
;
                mov COUNTER, #OVRLD_CNT             ;allow 10 secs.
;
; initialize timer0 as a 16-bit timer
;
                mov TMOD, #TIMER_BIT16
                clr A
                mov TL0, A
                mov TH0, A
;
; start (run) timer 0
;
run_ti0:
                setb TR0
;
goAround:
                jb VDCGH, notOvrLd      ;possible overvoltage error, more critical
                jb ILKOVRLD, notOvrLd ;overload ceased
;
                jnb TF0, goAround
                clr TR0
                clr TF0
                djnz COUNTER, run_ti0
;
```

ext1.asm (continued) :

```
ovrLd:
                clr GRELAY                      ;open grid contactor
                clr RRELAY                      ;bring in resistors
                clr DON                 ;use dump to dissipate energy in link
                setb SHUTDOWN
                setb OVERCURR
                clr EXTERNAL1
                pop PSW                         ;disable all interrupts and exit
                reti

;
notOvrLd:
                clr TR0                 ;stop timer 0
                setb EX1
                setb EA
                clr EXTERNAL1
;
                pop PSW
;
                reti
;
END
```

tint0.asm :

```
name TINT0
;
;
        cseg at 000Bh
;
                ljmp tInt0Begin
;
;
TINT0_CODE    segment CODE
;
                rseg      TINT0_CODE
;
;
extrn DATA(COUNTER), BIT(SHUTDOWN,GRELAY,RRELAY,TIM0)
;
;
tInt0Begin:
                push PSW
                clr EA
                setb TIM0
                dinz COUNTER. allowAction
```

tint0.asm (continued) :

```
;
; overvoltage or overcurrent for too long, problems !!!
;
                setb SHUTDOWN
                clr GRELAY
                clr RRELAY
;
allowAction:
                setb EA
                clr TIM0
                pop PSW
                reti
;
END
```

makefile :

```
#
#
#
#
start.out:              over.obj        \
                        init.obj        \
                        overv.obj               \
                        ext1.obj        \



        link51.bat

over.obj:       over.asm
        \shiba\asm51\asm51  over.asm li mr oj pr sb xr

init.obj:       init.asm
        \shiba\asm51\asm51  init.asm li mr oj pr sb xr

overv.obj:              overv.asm
        \shiba\asm51\asm51  overv.asm li mr oj pr sb xr

ext1.obj:       ext1.asm
        \shiba\asm51\asm51  ext1.asm li mr oj pr sb xr
```

Appendix B. Performance Optimization Controller Source Code

The software code for the performance optimization controller was developed in Borland C (compatible to ANSI C) utilizing the Borland C++ compiler and linker. It was compiled for an Intel 80486-DX2 microprocessor. As mentioned in Chapter 4, the availability of the graphics libraries for the Intel processor helped in the decision of utilizing the 486 as the primary computational resource. All the modules required for the optimization controller (i.e. initialization, communication between feedback and control variable updates, MPPT and MEPT loops, etc.) were processed by the Intel 80486.

Power feedback measurement was provided by a TMS320C3X based evaluation module residing on the ISA-bus of the 80486-desktop computer. The evaluation module is manufactured by Texas Instruments and consists of a single channel (±1.5V input) A/D interface. Communication between the desktop and the internal eval module utilized a common register addressable by both the microprocessor. The hardware details can be found in the Texas Instrument publication named "TMS320C3X evaluation module technical reference".

B.1 Optimization controller software

perf_opt.c

```
/*****************************************************************************/
/*                                                                           */
/*      Performance Optimization Controller                                  */
/*      Program developed by Shibashis Bhowmik                               */
/*      (C) 1997 OREGON STATE UNIVERSITY                                     */
/*                                                                           */
/*****************************************************************************/
/*****************************************************************************/
/*                                                                           */
/*                                                                           */
/*      Programming ideas borrowed from the following reference:             */
/*      Digital Signal Processing Applications with the                      */
/*      TMS320C30 Evaluation Module                                          */
/*                                                                           */
/*      OSC_PC.C                                                             */
/*                                                                           */
/*TMS320C30 EVALUATION MODULE DATA ACQUISITION (OSCILLOSCOPE) DEMO           */
/*      :PC PROGRAMS                                                         */
/*                                                                           */
/*                                                                           */
/*****************************************************************************/
#include <stdio.h>    /* INCLUDE NECESSARY STANDARD HEADER FILES      */
#include <stdlib.h>
#include <string.h>
#include <math.h>
```

```
#include <ctype.h>
#include <time.h>
#include <graphics.h>  /* INCLUDE TURBO C SPECIFIC HEADER FILES          */
#include <conio.h>
#include <bios.h>
#include "const115.h"
#include "ser_com.h"
#include "pc_1.h"      /* GENERIC EVM SUPPORT MACROS, STRUCTURES, PROTOTYPES*/
#include "perf_pc.h"   /* MACROS, PROTOTYPES, STRUCTURES, GLOBAL VARIABLES */
#include "perf_cmd.h"  /* SHARED HOST & TMS320C30 COMMAND PASSING MACROS   */
/*=============================================================================
*/
/* MAIN() MAIN PROGRAM LOOP                                                  */
/*=============================================================================
*/
void main(void)
{
        init_evm();       /* INITIALIZE EVALUATION MODULE                    */
                                                /* SETUP SERIAL PORT FOR COMM. WITH
INVERTER                */
        _bios_serialcom(_COM_INIT, comPort, SETTINGS);
        buffer = BLOCK_SIZE;/* SET BLOCK SIZE FOR DATA TRANSFER TO/FROM PC    */
        init_graphics();  /* INITIALIZE GRAPHICS & DRAW PERMANANT PARTS OF DISP*/
        status();         /* DRAW STATUS WINDOW                              */
        scale();          /* UPDATE SCALE                                    */
        iniBiosTime = biostime(0, 0L);      /* RESET TIMER                   */
        for(;;)
        {
                if (kbhit())
                {       if (loopFlag) mainMenuCommandProcess(); /* MAIN MENU COMMANDS*/
                        else invMenuCommandProcess(0);    /* INVERTER MENU COMMANDS*/
                }
                processBiosTime();
                /*-------------------------------------------------------------*/
                /* SET OSCILLOSCOPE DISPLAY INDEX TO LAST BUFFER PLAYED OR
RECORDED */
                /*-------------------------------------------------------------*/
                graph_index = index;
                recording(RECORDING);                   /* GET FRESH DATA FROM
EVM             */
                if(index == MAX_FILE)
                        index = 0;
                if (!meptFlag && !mpptFlag)
                        averagePower();
                graph();
        }
}

/*=============================================================================
*/
/* INIT_GRAPHICS()                                                           */
/* INITIALIZE GRAPHICS AND DRAW PERMANANT PARTS OF THE DISPLAY               */
/*=============================================================================
*/
void init_graphics(void)
{
```

```
char ch ;

GraphDriver = DETECT;
detectgraph(&GraphDriver, &GraphMode);  /* what graphics driver present? */
if ((GraphDriver == EGA) || (GraphDriver == VGA))
{
 GraphMode = EGAHI;
 registerbgidriver(EGAVGA_driver);
}
else if (GraphDriver == CGA)
 {
        printf("Is this a Compaq w/Plasma Display? (Y or N):");
        ch = getche();
        if ( (ch == 'y') || (ch == 'Y') )
        {
          GraphDriver = ATT400; GraphMode = ATT400HI;
          registerbgidriver(ATT_driver);
        }
        else
        {
          displayCheck();
        }
 }
 else
        displayCheck();

if(displayCheck() == grOk)
{       initgraph(&GraphDriver, &GraphMode, "");
        displayCheck();
}

/*---------------------------------------------------------------------*/
/* MAKE SURE DISPLAY HAS ADEQUATE RESOLUTION                           */
/*---------------------------------------------------------------------*/
if (getmaxx() < 639 || getmaxy() < 349)  displayCheck();

/*---------------------------------------------------------------------*/
/* DRAW ALL PERMANANT / NON-CHANGING WINDOWS                           */
/*---------------------------------------------------------------------*/

setcolor(WHITE);          /* SET TEXT AND BORDER COLOR TO WHITE    */

/* Gerry Shoults - modified 9/21/90                                   */
/* If running on Plasma 2-color display, the fill color must be BLACK   */
/* otherwise BLUE is O.K.                                             */

if (GraphDriver == ATT400)
setfillstyle(SOLID_FILL,BLACK); /* SET WINDOW FILL COLOR TO BLACK   */
else
setfillstyle(SOLID_FILL,BLUE);  /* SET WINDOW FILL COLOR TO BLUE    */
/*---------------------------------------------------------------------*/
/* OSCILLOSCOPE DISPLAY WINDOW                                         */
/*---------------------------------------------------------------------*/
setviewport(321,12,639,158,ON);          /* SET WINDOW DIMENSIONS */
bar(0,0,318,146);                  /* FILL BLUE          */
rectangle(0,0,318,146);              /* DRAW BORDER        */
```

```
        moveto(2,2);                        /* HOME CURSOR        */
        gputs("    Instantaneous Real Power");    /* DRAW TITLE         */
        setcolor(WHITE);
        /*-------------------------------------------------------------------*/
        /* SIGNAL WINDOW INSIDE OSCILLOSCOPE DISPLAY                         */
        /*-------------------------------------------------------------------*/
        setviewport(323, 27, 598, 156, ON);      /* SET WINDOW DIMENSIONS */
        clearviewport();                    /* FILL BLUE          */
        rectangle(0,0,274,129);               /* DRAW BORDER        */
        /*-------------------------------------------------------------------*/
        /* TITLE WINDOW                                                      */
        /*-------------------------------------------------------------------*/
        setviewport(0,0,639,11,ON);      /* SET WINDOW DIMENSIONS         */
        bar(0,0,639,11);              /* FILL BLUE                  */
        rectangle(0,0,639,11);          /* DRAW BORDER                */
        moveto(2,2);                  /* MOVE CURSOR TO BEGINNING OF LINE */
        gputs("                BDFM OPTIMIZATION CONTROLLER");
        /*-------------------------------------------------------------------*/
        /* MENU WINDOW                                                       */
        /*-------------------------------------------------------------------*/
        menuWindow(main_menu);
}


/*====================================================================
*/
/* DISPLAYCHECK(): ENSURES FOR PROPER GRAPHICS OPERATION              */
/*====================================================================
*/
int displayCheck(void)
{
        int GraphError;
        /* report any registration errors */
        if ( (GraphError = graphresult()) != grOk )
        {
                printf("Graphics error: %s\n", grapherrormsg(GraphError));
                printf("Press any key to halt:");
                getch();
                exit(1);                                    /* terminate with an error code
*/
        }
        /* otherwise just return */
        return(GraphError);
}


/*====================================================================
*/
/* GPUTS(): GRAPHICS EQUIVALENT OF PUTS()                             */
/*====================================================================
*/
void gputs(char *s)
{
        outtext((char far *) s);              /* PRINT TEXT      */
        moveto(2, gety() + 8);                /* PERFORM LINE FEED */
}
```

```
/*=============================================================================
*/
/* STATUS(): UPDATE AND DISPLAY AIC STATUS                              */
/*=============================================================================
*/
void status(void)
{
/*
        if ((loopFlag && !meptFlag && !mpptFlag) || !loopFlag || \
        (loopFlag && mpptFlag && (maxmPwrLoopCount == LOOP_COUNT)) || \
        (loopFlag && !mpptFlag && meptFlag && (maxmEffiLoopCount == LOOP_COUNT)))
*/
        /*-------------------------------------------------------------------*/
        /* STATUS WINDOW                                                     */
        /*-------------------------------------------------------------------*/
        setviewport(0, 160, 210, 349,ON);          /* DEFINE WINDOW     */
        bar(0,0,210,189);                        /* CLEAR WINDOW      */
        rectangle(0,0,210,189);                    /* DRAW BORDER       */
        moveto(2,2);                             /* HOME CURSOR       */
        gputs("      Status");
        moveto(2, gety() + 10);
        gputs("BDFM");
        if(syncFlag)
                gputs("   Synchronized");
        else
                gputs("   Not Synchronized");
        moveto(2, gety() + 10);
        gputs("CONVERTER");
        if(invFlag)
                gputs("   In Operation");
        else
                gputs("   Not in Operation");
        moveto(2, gety() + 10);
        gputs("OPTIMIZATION CONTROLLER");
        moveto(2, gety() + 5);
        if(loopFlag)
                gputs("   Closed Loop Mode");
        else
                gputs("   Open Loop Mode");
        moveto(2, gety() + 5);
        if(mpptFlag)
                gputs("   MPPT      ON");
        else
                gputs("   MPPT      OFF");
        moveto(2, gety() + 5);
        if(meptFlag)
                gputs("   MEPT      ON");
        else
                gputs("   MEPT      OFF");
        /*-------------------------------------------------------------------*/
        /* DISPLAY WINDOW                                                    */
        /*-------------------------------------------------------------------*/
        setviewport(212, 160, 639, 349, ON);          /* DEFINE WINDOW     */
        clearviewport();
        rectangle(0,0,427,189);                    /* DRAW BORDER       */
```

```
        moveto(2,4);
        gputs("              DISPLAY");
        /*-------------------------------------------------------------------*/
        /* DATA WINDOW INSIDE DISPLAY SCREEN                                 */
        /*-------------------------------------------------------------------*/
        setviewport(222, 180, 629, 339, ON);        /* SET WINDOW DIMENSIONS*/
        bar(0,0,407,159);                          /* CLEAR WINDOW     */
        rectangle(0,0,407,159);                    /* DRAW BORDER      */
        /*-------------------------------------------------------------------*/
        /* PRINT OUTPUT POWER                                                */
        /*-------------------------------------------------------------------*/
        moveto(40,gety()+20);
        sprintf(string,   "Output Power      = %6.1lf watts", power);
        gputs(string);
        /*-------------------------------------------------------------------*/
        /* PRINT INVERTER CURRENT MAGNITUDE                                  */
        /*-------------------------------------------------------------------*/
        if(invFlag)
        {
                moveto(40,gety()+10);
                sprintf(string,"Inverter current   = %6.1lf A", invCurr);
                gputs(string);
                moveto(40,gety()+5);
                if(seqFlag)
                        sprintf(string,"Inverter frequency  = +%5.1lf Hz", invFreq);
                else
                        sprintf(string,"Inverter frequency  = -%5.1lf Hz", invFreq);
                gputs(string);
                moveto(40,gety()+5);
                sprintf(string,"Switch on-time    = %7ld us", invOnTime);
                gputs(string);
        }
        if (syncFlag)
        {
                moveto(40,gety()+10);
                sprintf(string,"BDFM speed       = %7.0lf r/min", speed);
                gputs(string);
        }
        if (mpptFlag)
        {
                moveto(40,gety()+5);
                sprintf(string,"Optimal Ic         = %6.1lf A", invCurrOpt/10);
                gputs(string);
                moveto(40,gety()+5);
                sprintf(string,"Tip-speed ratio     = %4.3lf", lambda);
                gputs(string);
                moveto(40,gety()+5);
                sprintf(string,"Estimated wind speed = %4.3lf m/sec", nu);
                gputs(string);
                moveto(40,gety()+5);
                sprintf(string,"Optimal BDFM speed  = %7.0lf r/min", fROpt*60);
                gputs(string);
        }

}
```

```
/*=============================================================================
*/
/* SCALE(): PRINT OSCILLOSCOPE SCALE                                          */
/*=============================================================================
*/
void scale(void)
{
        setviewport(598,27,638,156,ON);          /* SET WINDOW          */
        bar(0,0,42,129);                         /* BLUE FILL WINDOW    */
        moveto(1,1);                             /* HOME CURSOR         */
        sprintf(string,"%d  ", (64 << magnify) - 1); /* PRINT UPPER SCALE LIMIT */
        outtext(string);
        moveto(1,60);                            /* PRINT ZERO CROSSING */
        outtext("0");
        moveto(1,120);                           /* PRINT LOWER SCALE LIMIT */
        sprintf(string,"%d  ", - 64 << magnify);
        outtext(string);
}


/*=============================================================================
*/
/* GRAPH(): DRAW MOST RECENTLY CAPTURED DATA                                  */
/*=============================================================================
*/
void graph(void)
{
        int i,j;                                 /* TEMPORARY VARIABLES */
        static float old_s_rate;                 /* PREVIOUS SAMPLING RATE */
        setviewport(325, 28, 596, 155, ON);      /* SET WINDOW          */
        /*-------------------------------------------------------------------*/
        /* IF SAMPLING RATE HAS CHANGED CLEAR WINDOW                         */
        /*-------------------------------------------------------------------*/
        if (old_s_rate != sample_rate)
        {
                clearviewport();
                old_s_rate = sample_rate;
        }
        /*-------------------------------------------------------------------*/
        /* ADJUST PLOTTING INTERVAL TO COMPENSATE FOR SAMPLING RATE          */
        /*-------------------------------------------------------------------*/
        j = (sample_rate - 5)/2;
        if (j < 1) j = 1;
        for(i = 0; i < OSC; i += j)              /* DRAW DATA           */
        {
                putpixel(i,y[i],BLACK);          /* ERASE OLD POINT     */
                /*-----------------------------------------------------------*/
                /* CALCULATE NEW POINT TO PLOT                               */
                /*-----------------------------------------------------------*/
                y[i] = 63 - (disk[graph_index + i] >> magnify);
                putpixel(i,y[i],WHITE);          /* PLOT NEW POINT      */
        }
}
```

```
/*=============================================================================
*/
/* MAIN_MENU_COMMAND_PROCESS                                                  */
/*=============================================================================
*/
void mainMenuCommandProcess(void)
{
        char command;           /* COMMAND VALUE                    */
        command = getExtendedCommand();         /* GET COMMAND WORD FROM KEYBOARD
ENTRY */
        fflush(stdin);          /* CLEAR ANY EXTRANEOUS CHARACTERS     */
        switch(command)         /* PARSE COMMAND                    */
        {
                /*-------------------------------------------------------------*/
                /* INITIALIZE EVM FOR CLOSED LOOP OPERATION                    */
                /*-------------------------------------------------------------*/
                case START:
                        if (invFlag)
                                syncFlag = ON;
                        break;
                /*-------------------------------------------------------------*/
                /* TOGGLE CLOSED/OPEN LOOP OPERATION                           */
                /*-------------------------------------------------------------*/
                case LOOP:
                        loopFlag ^= TOGGLE;
                        meptFlag = OFF;
                        mpptFlag = OFF;
                        menuWindow(invCommandMenu);
                        break;
                /*-------------------------------------------------------------*/
                /* TOGGLE MPPT LOOP OPERATION                                  */
                /*-------------------------------------------------------------*/
                case MPPT:
                        if (syncFlag && invFlag && meptFlag)
                                mpptFlag ^= TOGGLE;
                        break;
                /*-------------------------------------------------------------*/
                /* TOGGLE MEPT LOOP OPERATION                                  */
                /*-------------------------------------------------------------*/
                case MEPT:
                        if (syncFlag && invFlag)
                                meptFlag ^= TOGGLE;
                        if (meptFlag == OFF)
                                mpptFlag = OFF;
                        break;
                /*-------------------------------------------------------------*/
                /* STOP INVERTER AND LOOP OPERATION                            */
                /*-------------------------------------------------------------*/
                case STOP:
                        invMenuCommandProcess('d');
/*                      sendCommandToInv((char *)("d"));   */
                        syncFlag= OFF;
                        invFlag = OFF;
                        invCurr         = 0.0;
                        invFreq         = 0.0;
```

```
                invOnTime       = 0;
                mpptFlag= OFF;
                meptFlag= OFF;
                index = 0;              /* RESET DISK STORAGE BUFFER INDEX   */
                break;
        /*-------------------------------------------------------------*/
        /* TOGGLE MAGNIFICATION OF INPUT SIGNAL ON OSCILLOSCOPE          */
        /*-------------------------------------------------------------*/
        case MAGNIFY:
                if (++magnify == 8) magnify = 0;/* TOGGLE MAGNIFICATION      */
                scale();                /* DRAW SCALE                */
                break;
        /*-------------------------------------------------------------*/
        /* QUIT PROGRAM                                                */
        /*-------------------------------------------------------------*/
        case QUIT:
                textmode(LASTMODE);
                clrscr();
                exit(EXIT_SUCCESS);
                break;
        /*-------------------------------------------------------------*/
        /* IF OTHER COMMAND ABORT COMMAND TRANSMISSION                   */
        /*-------------------------------------------------------------*/
        default:
                command = NONE;
                break;
    }
    status();                           /* UPDATE STATUS */
}

/*=============================================================================
*/
/* INVERTER_MENU_COMMAND_PROCESS                                        */
/*=============================================================================
*/
void invMenuCommandProcess(char passCommand)
{
    int i;
    char invMenuCommand;        /* COMMAND VALUE                */
    char *invCommand = NULL;            /* INVERTER COMMAND ARRAY
            */
    const char *paramString = NULL;  /* STRING FOR STORING COMMAND PARAMETER */
    int dec, sign;
    if (passCommand)
            invMenuCommand = passCommand;
    else
            invMenuCommand = getCommand();  /* GET COMMAND WORD FROM
KEYBOARD ENTRY */
    fflush(stdin);          /* CLEAR ANY EXTRANEOUS CHARACTERS    */
    switch(invMenuCommand)      /* PARSE COMMAND                */
    {
            /*-------------------------------------------------------------*/
            /* INITIALIZE INVERTER FOR OPEN LOOP OPERATION                  */
            /*-------------------------------------------------------------*/
            case 'E': case 'e':
```

```
                invCommand      = (char *)("en");/* ENABLE INVERTER
 */
                invFlag         = ON;           /* SHOW INVERTER 'ON' STATUS
 */
                break;
               /*-------------------------------------------------------------*/
               /* STOP INVERTER OPERATION FOR OPEN LOOP OPERATION             */
               /*-------------------------------------------------------------*/
               case 'D': case 'd':
                invCommand      = (char *)("d");  /* DISABLE INVERTER
*/
                invFlag         = OFF;          /* SHOW INVERTER 'OFF' STATUS
 */
                invCurr         = 0.0;
                invFreq         = 0.0;
                invOnTime       = 0;
                break;
               /*-------------------------------------------------------------*/
               /* SET POSITIVE PHASE SEQUENCE                                 */
               /*-------------------------------------------------------------*/
               case '+':
                invCommand      = (char *)("+");  /* DISABLE INVERTER
*/
                seqFlag         = ON;           /* SHOW INVERTER 'OFF' STATUS
 */
                break;
               /*-------------------------------------------------------------*/
               /* SET NEGATIVE PHASE SEQUENCE                                 */
               /*-------------------------------------------------------------*/
               case '-':
                invCommand      = (char *)("-");  /* DISABLE INVERTER
*/
                seqFlag         = OFF;          /* SHOW INVERTER 'OFF' STATUS
 */
                break;
               /*-------------------------------------------------------------*/
               /* INCREASE CURRENT REFERENCE BY 0.1A                          */
               /*-------------------------------------------------------------*/
               case 'W': case 'w':
                invCommand      = (char *)("w");  /* INCREASE CURRENT REF.
*/
                invCurr   += 0.1;
                break;
               /*-------------------------------------------------------------*/
               /* DECREASE CURRENT REFERENCE BY 0.1A                          */
               /*-------------------------------------------------------------*/
               case 'X': case 'x':
                invCommand      = (char *)("x");  /* DECREASE CURRENT REF.
*/
                invCurr   -= 0.1;
                break;
               /*-------------------------------------------------------------*/
               /* INCREASE FREQUENCY REFERENCE BY 0.1Hz                       */
               /*-------------------------------------------------------------*/
               case 'Y': case 'y':
```

```
                invCommand      = (char *)("y");    /* INCREASE FREQUENCY REF.
*/
                invFreq    += 0.1;
                speed = (60 * (PWR_FREQ + invFreq)) / (PWR_POLE + CNTRL_POLE);
                break;
            /*-------------------------------------------------------------*/
            /* DECREASE FREQUENCY REFERENCE BY 0.1Hz                       */
            /*-------------------------------------------------------------*/
            case 'Z': case 'z':
                invCommand      = (char *)("z");    /* DECREASE FREQUENCY REF.
*/
                invFreq    -= 0.1;
                speed = (60 * (PWR_FREQ + invFreq)) / (PWR_POLE + CNTRL_POLE);
                break;
            /*-------------------------------------------------------------*/
            /* SET REFERENCE CURRENT FOR OPEN LOOP OPERATION               */
            /*-------------------------------------------------------------*/
            case 'I': case 'i':
                invCommand = curr;                              /* SET CURRENT
COMMAND                 */
                paramString = (const char *)(commandParamInput('I', PARAM));
                strcat(invCommand, paramString);
                invCurr = strtod(++invCommand, '\0') / 10;
                --invCommand;
                break;
            /*-------------------------------------------------------------*/
            /* SET REFERENCE FREQUENCY FOR OPEN LOOP OPERATION             */
            /*-------------------------------------------------------------*/
            case 'F': case 'f':
                invCommand = freq;                          /* SET FREQUENCY
COMMAND                 */
                paramString = (const char *)(commandParamInput('F', PARAM));
                strcat(invCommand, paramString);
                invFreq = strtod(++invCommand, '\0') / 10;
                speed = (60 * (PWR_FREQ + invFreq)) / (PWR_POLE + CNTRL_POLE);
                fR          = speed/60;
                --invCommand;
                break;
            /*-------------------------------------------------------------*/
            /* SET SWITCH ON-TIME FOR OPEN LOOP OPERATION                  */
            /*-------------------------------------------------------------*/
            case 'O': case 'o':
                invCommand = onTime;            /* SET ONTIME COMMAND
*/
                paramString = (const char *)(commandParamInput('O', PARAM));
                strcat(invCommand, paramString);
                invOnTime = strtoul(++invCommand, '\0', 10);
                --invCommand;
                break;
            /*-------------------------------------------------------------*/
            /* SET BDFM SPEED IN THE OPEN LOOP MODE                        */
            /*-------------------------------------------------------------*/
            case 'S': case 's':
                invCommand = bdfmSpd;  /* SET BDFM SPEED IN OPEN LOOP
OPERATION*/
                paramString = (const char *)(commandParamInput('S', PARAM));
```

```
                        speed = strtod(paramString, '\0');
                        fR = speed/60;
                        invFreq = (speed * (PWR_POLE + CNTRL_POLE) / 60) - PWR_FREQ;
                        invCommand = freq;                          /* SET FREQUENCY
COMMAND                  */
                        paramString = (const char *)(fcvt(invFreq*10,0,&dec,&sign));
                        strcat(invCommand, paramString);
                        break;
                /*-------------------------------------------------------------------*/
                /* TOGGLE CLOSED/OPEN LOOP OPERATION                                 */
                /*-------------------------------------------------------------------*/
                case 'Q': case 'q':
                        invMenuCommand = NONE;
                        loopFlag ^= TOGGLE;
                        menuWindow(main_menu);
                        break;
                /*-------------------------------------------------------------------*/
                /* IF OTHER COMMAND ABORT COMMAND TRANSMISSION                       */
                /*-------------------------------------------------------------------*/
                default:
                        invMenuCommand = NONE;
                        break;
        }
        if (invMenuCommand)     sendCommandToInv(invCommand);  /* SEND COMMAND  */
        if(*++invCommand == 'n')
                *++invCommand = '\0';
        else if((*--invCommand == 'i')|(*invCommand == 'f')|(*invCommand == 'o'))
        {       for(i=0; i<4; ++i)
                        *++invCommand = '\0';
        }
        else *++invCommand = '\0';
        status();                               /* UPDATE STATUS */
}


/*===================================================================
*/
/* MENU WINDOW                                                      */
/*===================================================================
*/
void menuWindow(char **entry)
{
        setviewport(0,12,320,158,ON);       /* SET WINDOW DIMENSIONS        */
        bar(0,0,320,146);                   /* FILL BLUE                    */
        rectangle(0,0,320,146);             /* DRAW BORDER                  */
        moveto(2,2);                        /* MOVE CURSOR TO BEGINNING OF LINE */
        if(entry == main_menu)
                gputs("            Main Menu");/* PRINT DISPLAY TITLE       */
        else
                gputs("        Inverter Command Menu");/* PRINT DISPLAY TITLE   */
        setcolor(WHITE);                    /* DRAW MENU IN WHITE           */
        while(*entry) gputs(*entry++);      /* DRAW ALL MENU ENTRIES        */
}
```

```
/*=============================================================================
*/
/* COMMAND_PARAM_INPUT():CURRENT,FREQUENCY,ON-TIME                          */
/*=============================================================================
*/
char *commandParamInput(char commandLetter, int paramLength)
{
        clock_t i;              /* CLOCK COUNTER                */
        char c = NULL;          /* TEMPORARY CHARACTER          */
        int  j = 0;             /* POINTER TO STRING            */
        stringForParam[0] = NULL;
        setviewport(360,159,639,179,ON); /* SET WINDOW FOR INPUT        */
        rectangle(0,0,279,19);      /* DRAW BORDER                  */
        do
        {
                bar(1,1,278, 18);       /* BLUE FILL WINDOW         */
                moveto(2,2);            /* HOME CURSOR              */
                switch(commandLetter)
                {
                        case 'I':
                                outtext("Enter current: "); /* PROMPT FOR CURRENT INPUT    */
                                break;
                        case 'F':
                                outtext("Enter frequency: "); /* PROMPT FOR FREQUENCY INPUT
*/
                                break;
                        case 'O':
                                outtext("Enter on-time: "); /* PROMPT FOR ON-TIME INPUT    */
                                break;
                        case 'S':
                                outtext("Enter BDFM-speed: "); /* PROMPT FOR ON-TIME INPUT
*/
                                break;
                        default:
                                break;
                }
                do
                {
                        if (kbhit())        /* WAIT FOR KEYBOARD ENTRY      */
                        {
                                c = bioskey(0);    /* GET KEYBOARD INPUT           */
                                if (c)
                                {
                                        /*-------------------------------------------------*/
                                        /* IF INPUT IS BACK SPACE DELETE ONE
CHARACTER       */
                                        /*-------------------------------------------------*/
                                        if (c == '\b' && j) stringForParam[--j] = NULL;
                                        /*-------------------------------------------------*/
                                        /* ELSE IF INPUT IS PRINTABLE APPEND INPUT TO
STRING   */
                                        /*-------------------------------------------------*/
                                        else if (isprint(c))
                                        {
```

```
                                        stringForParam[j] = c;      /* ADD NEW
CHARACTER TO STRING */
                                        stringForParam[++j] = NULL;  /* ADD NEW
NULL TERMINATION   */
                                    }
                                    bar(128,2,224,9);       /* ERASE OLD OUTPUT        */
                                    moveto(140,2);          /* REPRINT UPDATED FILENAME
*/
                                    outtext(stringForParam);
                                }
                            }
                        }
                        /*-------------------------------------------------------------*/
                        /* CONTINUE UNTIL CARRIAGE RETURN IS RECEIVED OR STRING LENGTH
*/
                        /*      EXCEEDS TWELVE                                         */
                        /*-------------------------------------------------------------*/
                        while(c != '\r'&& j < paramLength);
                }
                while(stringForParam[0] == NULL);     /* RETRY UNTIL A VALID FILENAME IS GIVEN */
                clearviewport();            /* REMOVE PROMPT WINDOW            */
                status();                   /* UPDATE STATUS                   */
                return(&stringForParam[0]);
}


/*=============================================================================
*/
/* AVERAGE_POWER():MOVING-WINDOW AVERAGING                                    */
/*=============================================================================
*/
/*
void averagePower(void)
{
        int tag, start, end;
        power = 0;
        tag = graph_index;
        if((start = tag - buffer/2) < 0)
                start += MAX_FILE;
        end = start + buffer;
        for(tag=start; tag<end; tag++)
                power += disk[tag % MAX_FILE];
        power /= (buffer * PWR_SCALER);
}


*/
void averagePower(void)
{
        int tag, start, end, tempBuf;
        power = 0;
        end = graph_index;
        tempBuf = 12*buffer;
        if((start = end - tempBuf) < 0)
        {       start += MAX_FILE;
                end += MAX_FILE;
```

```
        }
        for(tag=start; tag<end; tag++)
                power += disk[tag % MAX_FILE];
        power /= (tempBuf * PWR_SCALER);
}


/*=============================================================================
*/
/* PROCESS_BIOS_TIME(): DETERMINE WHEN TO UPDATE FREQUENCY AND CURRENT     */
/*=============================================================================
*/
void processBiosTime(void)
{
        unsigned long diffBiosTime;
        if(meptFlag || mpptFlag)
                newBiosTime = timerDisplay();
        else
                newBiosTime = oldBiosTime = timerDisplay();
        diffBiosTime = newBiosTime - oldBiosTime;

        if(meptFlag && (diffBiosTime >= MEPT_MIN))
        {       maxmEffiLoop();
                ++maxmEffiLoopCount;
                if(!mpptFlag && (maxmEffiLoopCount == LOOP_COUNT))
                        maxmEffiLoopCount = 0;
        }
        if(mpptFlag && meptFlag && (maxmEffiLoopCount == LOOP_COUNT))
        {       maxmPwrLoop();
                maxmEffiLoopCount = 0;
                ++maxmPwrLoopCount;
                if(maxmPwrLoopCount == LOOP_COUNT)
                        maxmPwrLoopCount = 0;
        }
}


/*=============================================================================
*/
/* TIMER_DISPLAY():DISPLAY TIMER VALUE                                     */
/*=============================================================================
*/
unsigned long timerDisplay(void)
{
        unsigned long biosTime;
        setviewport(360,329,639,349,ON); /* SET WINDOW FOR INPUT           */
        rectangle(0,0,278,18);          /* DRAW BORDER                    */
        bar(1,1,279, 17);               /* BLUE FILL WINDOW               */
        moveto(2,2);
        biosTime = biostime(0, 0L);     /* READ TIMER
         */
        sprintf(string," Time Elasped : %.2f secs.", (biosTime-iniBiosTime)/CLK_TCK);
        outtext(string);
        return(biosTime);
}
```

```
/*=============================================================================
*/
/* MAXM_EFFI_LOOP(): MAXIMUM EFFICIENCY TRACKING                              */
/*=============================================================================
*/
void maxmEffiLoop(void)
{
        double invCurrSave;
        int stringLen = 0;
        char *tempCom2;
        int invCurrInt;
        if ((tempCom2 = (char *) malloc(16)) == NULL)
        {
          printf("Not enough memory to allocate buffer\n");
          exit(1);
        }
        invCurr *= 10;
        invCurrSave = (int)(invCurr);
/*
        timerDisplay();
        graph_index = index;
        recording(RECORDING);
        if(index == MAX_FILE)
                index = 0;
        averagePower();
        graph();
*/
        averagePower();
        /*-------------------------------------------------------------------*/
        /*      MAXIMUM DFM EFFICIENCY UPDATE LOOP
        */
        /*-------------------------------------------------------------------*/
        sqrVar = power * power;
        cubeVar = sqrVar * power;
        invCurrOpt = ((Ci0 + Ci1*power + Ci2*sqrVar + Ci3*cubeVar) * 10) + 5;
        if(invCurrOpt > MAX_INV_CURR)
                invCurrOpt = MAX_INV_CURR;
        else if(invCurrOpt < MIN_INV_CURR)
                invCurrOpt = MIN_INV_CURR;

        invCurrInt = (int)(invCurrOpt);

        if(invCurrInt > invCurrSave) invCurr += 1;
        else invCurr -= 1;

        sprintf(tempCom2,"i%3.0lf", invCurr);
        while (*tempCom2++)
        {
                if (*tempCom2 == ' ')
                        *tempCom2 = '0';
                stringLen++;
        }
        while (stringLen)
        {       tempCom2--;
                stringLen--;
```

```
        }
        tempCom2--;
        sendCommandToInv(tempCom2);
        free(tempCom2);
        oldBiosTime = newBiosTime;
        invCurr /= 10;
}


/*=========================================================================
*/
/* MAXM_PWR_LOOP(): MAXIMUM POWER POINT TRAKING                          */
/*=========================================================================
*/
void maxmPwrLoop(void)
{
        double lambdaOld = lambda;
        int hang;
        double effi_DFM_MAXM;
        /*-------------------------------------------------------------*/
        /*      DETERMINE MAXIMUM DFM EFFICIENCY FROM EFFI-Pt PROFILE
        */
        /*-------------------------------------------------------------*/
        sqrVar = invCurr * invCurr;
        cubeVar = sqrVar * invCurr;
        effi_DFM_MAXM = (Ce0 + Ce1*invCurr + Ce2*sqrVar + Ce3*cubeVar);
        /*-------------------------------------------------------------*/
        /* POWER DELIVERED BY WIND TURBINE
                        */
        /*-------------------------------------------------------------*/
        power_wt = (power/effi_DFM_MAXM)*100;
        /*-------------------------------------------------------------*/
        /* TURBINE SPEED DERIVED FROM DFM SPEED
                        */
        /*-------------------------------------------------------------*/
        fT = Cf0 + Cf1*speed + Cf2*pow(speed,2) + Cf3*pow(speed,3) \
                + Cf4*pow(speed,4) + Cf5*pow(speed,5);
        /*-------------------------------------------------------------*/
        /*      CALCULATE TIP-SPEED RATIO, LAMBDA, ITERATIVELY
        */
        /*-------------------------------------------------------------*/
/*
        *retLambda = 1;
        while ((*retLambda > 0.0001) && iterCount--)
        {
                retLambda = newtonRaphson(lambda);
                lambda = *++retLambda;
                --retLambda;
        }
*/
        lambda = solve(); /* + LAMBDA_0; */
        if((lambda < LAMBDA_0) || (lambda > LAMBDA_MAX))
                lambda = lambdaOld;
        /*-------------------------------------------------------------*/
        /*      ESTIMATE WIND SPEED
                                */
```

```
        /*-------------------------------------------------------------*/
/*      nu = (2*pi*R) * (fR - F_R0) * (LAMBDA_RANGE / (lambda - LAMBDA_0)); */
        nu = 2*pi*R*fT/lambda;
        /*-------------------------------------------------------------*/
        /*      DETERMINE OPTIMUM BDFM SPEED
                        */
        /*-------------------------------------------------------------*/
        fROpt = F_R0 + ((LAMBDA_OPT - LAMBDA_0)/LAMBDA_RANGE)*(nu / (2*pi*R));
        if(fROpt > MAX_FR_OPT)
                fROpt = MAX_FR_OPT;
        else if(fROpt < MIN_FR_OPT)
                fROpt = MIN_FR_OPT;
        /*-------------------------------------------------------------*/
        /*      COMMAND THE REQUIRED FREQUENCY FOR THE INVERTER
                */
        /*-------------------------------------------------------------*/
        invFreqOpt = (fROpt * (PWR_POLE + CNTRL_POLE)) - PWR_FREQ;
        if(invFreqOpt > MAX_INV_FREQ)
                invFreqOpt = MAX_INV_FREQ;
        else if(invFreqOpt < MIN_INV_FREQ)
                invFreqOpt = MIN_INV_FREQ;
        diffInvFreq = (int)((invFreqOpt - invFreq) * 10);
        if(diffInvFreq < 0) directionFlag = OFF;
        else directionFlag = ON;
        diffInvFreq = abs(diffInvFreq);
        /*-------------------------------------------------------------*/
        /*      COMMAND FREQUENCY UPDATE 0.1Hz AT A TIME
        */
        /*-------------------------------------------------------------*/
/*      while (diffInvFreq-- && !kbhit())
        {
*/
                if (directionFlag) invMenuCommandProcess('y');
                else invMenuCommandProcess('z');
/*
                for (hang = 0; hang<10; hang++)
                {       delay(300);
                        maxmEffiLoop();
                }

        }
*/
/*
        sprintf(tempCom,"f%3.0lf", invFreq);
        while (*tempCom++)
        {
                if (*tempCom == ' ')
                        *tempCom = '0';
                stringLen++;
        }
        while (stringLen)
        {       tempCom--;
                stringLen--;
        }
        tempCom--;
        sendCommandToInv(tempCom);
```

```
        oldBiosTime = newBiosTime;
        invFreq /= 10;
*/
        status();
}


/*=============================================================================*
/
/* NEWTON_RAPHSON(): ITERATIVE SOLUTION OF NON-LINEAR TIP-SPEED RATIO      */
/*=============================================================================*
/
double *newtonRaphson(double var)
{
        double F;
        double jacob;
        double deltaVar;
        double invVar;
        double *newtRap;
        invVar = 1/var;
        sqrVar = invVar/var;
        cubeVar = sqrVar/var;
        quadVar = cubeVar/var;
        F = power_wt - K1*(Cp0*cubeVar + Cp1*sqrVar + Cp2*invVar + Cp3)*pow(fR,3);
        jacob = K1*(3*Cp0*quadVar + 2*Cp1*cubeVar + Cp2*sqrVar)*pow(fR,3);
        deltaVar = F/jacob;
        var -= deltaVar;
        *newtRap++     = F;
        *newtRap       = var;
        return(--newtRap);
}


/*=============================================================================*
/
/* SOLVE(): A COMBINATION OF NEWTON-RAPHSON AND BISECTION METHOD          */
/*=============================================================================*
/
double solve(void)
{
        double funcDL, funcDH, *funcRtSafe;
        double rtSafe;
        double temp;
        double dx, dxOld;
        double dF, F;
        double xL, xH;
        double x1 = X1_GUESS;
        double x2 = X2_GUESS;
        int iterCount = MAX_ITER;
/*
        if ((funcDL = (double *) malloc(32)) == NULL)
        {
          printf("Not enough memory to allocate buffer\n");
          exit(1);
        }
        if ((funcDH = (double *) malloc(32)) == NULL)
```

```
        {
          printf("Not enough memory to allocate buffer\n");
          exit(1);
        }
*/
        if ((funcRtSafe = (double *) malloc(32)) == NULL)
        {
          printf("Not enough memory to allocate buffer\n");
          exit(1);
        }
/*
        funcDL = *functionDefine(x1);
        funcDH = *functionDefine(x2);
        if((funcDL)*(funcDH) >= 0) exit(1);
        if(funcDL < 0)
        {
                xL = x1;
                xH = x2;
        }
        else
        {
                xH = x1;
                xL = x2;
        }
*/
        rtSafe = 0.5*(x1 + x2);
        dxOld = fabs(x2 - x1);
        dx = dxOld;
        funcRtSafe = functionDefine(rtSafe);
        F = *(funcRtSafe++);
        dF = *funcRtSafe;
        --funcRtSafe;
        do
        {
/*
                if((((((rtSafe-xH)*dF)-F)*(((rtSafe-xL)*dF)-F)) >= 0) || \
                        (fabs(2*F) > fabs(dxOld*dF)))
                {
                        dxOld = dx;
                        dx = 0.5*(xH-xL);
                        rtSafe = xL + dx;
                        if (fabs(xL-rtSafe) < 1e-05)
                        {
                                free(funcRtSafe);
                                return(rtSafe);
                        }
                }
                else
*/
                {
                        dxOld = dx;
                        dx = F/dF;
                        temp = rtSafe;
                        rtSafe -= dx;
                        if (fabs(temp-rtSafe) < 1e-05)
                        {
```

```
                                free(funcRtSafe);
                                return(rtSafe);
                        }
                }
                if(fabs(dx) < 1e-05)
                {
                        free(funcRtSafe);
                        return(rtSafe);
                }
                funcRtSafe = functionDefine(rtSafe);
                F = *(funcRtSafe++);
                dF = *funcRtSafe;
                --funcRtSafe;
/*
                if(F < 0) xL = rtSafe;
                else            xH = rtSafe;
*/
        }
        while (iterCount--);
        return(0);
}


/*=============================================================================*
/
/* FUNCTION_DEFINE(): RETURNS THE VALUE OF THE FUNCTION AND ITS DERIVATIVE  */
/*=============================================================================*
/
double *functionDefine(double var)
{
        double func;
        double jacob;
        double invVar;
        double *funcD;
        invVar = 1/var;
        sqrVar = invVar/var;
        cubeVar = sqrVar/var;
        quadVar = cubeVar/var;
        func = power_wt - K1*(Cp0*cubeVar + Cp1*sqrVar + Cp2*invVar \
                                        + Cp3 + Cp4*var + Cp5*var*var)*pow(fT,3);
        jacob = K1*(3*Cp0*quadVar + 2*Cp1*cubeVar + Cp2*sqrVar \
                                - Cp4 - 2*Cp5*var)*pow(fT,3);
        *funcD++    = func;
        *funcD      = jacob;
        return(--funcD);
}
```

Header files for perf_opt.c:

const115.h

```
/*************************************************************************/
/*                                                                       */
/*      Performance Optimization Controller                              */
/*                                                                       */
/*      Program developed by Shibashis Bhowmik                           */
/*                                                                       */
/*      (C) 1997 OREGON STATE UNIVERSITY                                 */
/*                                                                       */
/*************************************************************************/
/*=======================================================================*/
/* MACROS                                                                */
/*=======================================================================*/
#define BUFF_NUM            16
#define MAX_FILE    buffer * BUFF_NUM          /* MAX NUMBER OF SAMPLES */
#define LINE_V      1.5     /* AIC VOLTAGE VALUES                       */
#define THREE_V     3
#define OSC         272     /* WIDTH OF OSCILLOSCOPE DISPLAY            */
#define PARAM               12                 /* LENGTH OF INPUT PARAMETERS
                */
#define PWR_SCALER          1.024
#define MAX_INV_CURR        70                 /* SET FOR A MAXIMUM OF 5.0 Arms
        */
#define MIN_INV_CURR 22                        /* SET FOR A MINIMUM OF 2.2 Arms
*/
#define MAX_INV_FREQ        53                 /* SET FOR A MAXIMUM OF 60 Hz
                */
#define MIN_INV_FREQ 25                        /* SET FOR A MINIMUM OF 25 Hz
        */
#define MAX_SPEED           1800
#define MIN_SPEED           1250
#define LOOP_COUNT          10
#define MEPT_MIN            7                  /* ~400 ms for each update */
#define MEPT_MAX            15
#define MPPT_MIN            LOOP_COUNT * MEPT_MIN
#define MPPT_MAX            LOOP_COUNT * MEPT_MAX
#define MAX_ITER    50
#define PWR_FREQ            60
#define PWR_POLE            3
#define CNTRL_POLE          1
#define F_R0                20
#define LAMBDA_0            3
#define LAMBDA_MAX          9
#define LAMBDA_RANGE        6
#define MAX_FR_OPT          33.333
#define MIN_FR_OPT          20.833
#define pi                      3.141592654e+00
#define R                       3.18309886e-01
```

```
#define HALF_PI_ROW          2.643080059e+01        /* PRODUCT OF AIR-DENSITY, PI AND
A HALF */
#define K1                          21.424
#define LAMBDA_OPT           9
#define X1_GUESS             1
#define X2_GUESS             17
/*=============================================================================
*/
/* CONSTANTS FOR Ic-Pt PROFILE APPROXIMATION
                */
/*=============================================================================
*/
#define Ci0        -6.685590975953640e-01
#define Ci1         1.559679433221850e-02
#define Ci2        -2.572805445582490e-05
#define Ci3         1.588912375270070e-08
/*=============================================================================
*/
/* CONSTANTS FOR Effi.-Ic PROFILE APPROXIMATION
                */
/*=============================================================================
*/
#define Ce0         1.459222939136030e+01
#define Ce1         3.642025997231750e+01
#define Ce2        -1.037325774635940e+01
#define Ce3         9.538226457187550e-01
/*=============================================================================
*/
/* CONSTANTS FOR f_t-f_R (r/min) PROFILE APPROXIMATION
                */
/*=============================================================================
*/
#define Cf0        -1.050042869603410e+02
#define Cf1         1.000024213155460e-01
#define Cf2         3.122683836642910e-09
#define Cf3        -2.101814170260100e-12
#define Cf4         7.039221644805220e-16
#define Cf5        -9.385103650988710e-20
/*=============================================================================
*/
/* CONSTANTS FOR Cp-LAMBDA PROFILE APPROXIMATION
                */
/*=============================================================================
*/
#define Cp0        -2.548513356485820e-01
#define Cp1         1.039879368168420e-01
#define Cp2        -5.048169520213590e-04
#define Cp3        -2.873579826562060e-04
#define Cp4        -3.072367130361660e-14
#define Cp5         6.694466999822750e-16
```

ser_com.h

```
/*****************************************************************************/
/*                                                                           */
/*      Performance Optimization Controller                                  */
/*                                                                           */
/*      Program developed by Shibashis Bhowmik                               */
/*                                                                           */
/*      (C) 1997 OREGON STATE UNIVERSITY                                     */
/*                                                                           */
/*****************************************************************************/
/*****************************************************************************/
/*                                                                           */
/*      PC - INVERTER CONTROLLER COMMUNICATION SUPPORT                       */
/*                                                                           */
/*****************************************************************************/
#ifndef  _ser_com
#define _ser_com

#define  COM1                   0
#define COM2                    1
#define DATA_READY              0x100                   /* DATA READY
                        */
#define TRANSMIT_READY  (0x4000 | 0x2000)    /* HOLDING AND SHIFT EMPTY
*/
#define  FRAMING_ERROR          0x800                   /* DEFINE SERIAL PORT
ERRORS          */
#define  PARITY_ERROR           0x400
#define  OVERRUN_ERROR          0x200
#define SETTINGS                (_COM_9600 | _COM_CHR8 | _COM_STOP1 | _COM_NOPARITY)

#endif
```

pc_1.h

```
/*************************************************************************/
/*                                                                       */
/*      Performance Optimization Controller                              */
/*                                                                       */
/*      Program developed by Shibashis Bhowmik                           */
/*                                                                       */
/*      (C) 1997 OREGON STATE UNIVERSITY                                 */
/*                                                                       */
/*************************************************************************/
/*************************************************************************/
/*                                                                       */
/*                                                                       */
/*      Programming ideas borrowed from the following reference:         */
/*                                                                       */
/*      Digital Signal Processing Applications with the                  */
/*                                                                       */
/*      TMS320C30 Evaluation Module                                      */
/*                                                                       */
/*      PC_1.H                                                           */
/*                                                                       */
/*  TMS320C30 EVALUATION MODULE DATA ACQUISITION (OSCILLOSCOPE) DEMO     */
/*        :PC PROGRAMS                                                   */
/*                                                                       */
/*                                                                       */
/*************************************************************************/
#define OFF    0x00
#define ON     0x01
#define TOGGLE 0x01       /* XOR BITFIELD TO TOGGLE ON <-> OFF   */
#define NONE   0x00       /* WORD REPRESENTING ABSENCE OF COMMAND */
/*************************************************************************/
/* FUNCTION PROTOTYPES                                                   */
/*************************************************************************/
void init_evm(void);
void get_iobase(void);
int  getExtendedCommand(void);
char getCommand(void);
void send_command(int i);
void sendCommandToInv(char *commandParam);
void recording(int record_cmd);
/*************************************************************************/
/* COMMUNICATIONS MACROS                                                 */
/*************************************************************************/
/*-----------------------------------------------------------------------*/
/* PC I/O SPACE BASE ADDRESS FOR EVM CONTROL REGISTERS                   */
/*-----------------------------------------------------------------------*/
#define IOBASE_DEFAULT  0x0240

#define CONTROL5      iobase + 0x000A    /* CONTROL REGISTER     */
#define STATUS0       iobase + 0x0400    /* STATUS0 REGISTER     */
```

```
#define COM_CMD       iobase + 0x0800    /* COMMAND REGISTER      */
#define COM_DATA      iobase + 0x0808    /* DATA   REGISTER       */
#define SOFT_RESET    iobase + 0x0818    /* SOFT RESET LOCATION   */
#define MINOR_CMD     iobase + 0x0014    /* MINOR COMMAND REGISTER */

/*-------------------------------------------------------------------*/
/* BIT MASKS FOR READ AND WRITE ACKNOWLEDGE BITS IN STATUS REGISTER  */
/*-------------------------------------------------------------------*/
#define MREAD_ACK     0x0002
#define MWRITE_ACK    0x0004
#define UPDATE_REQ    0x6044

#define WRITE_CMD(x)     outp  (COM_CMD,x)    /* WRITE  8 BIT COMMAND */
#define READ_CMD         inp   (COM_CMD)      /* READ   8 BIT COMMAND */
#define WRITE_DATA(x)    outport(COM_DATA, x) /* WRITE 16 BIT DATA    */
#define READ_DATA        inport (COM_DATA)    /* READ  16 BIT DATA    */
#define RESET_EVM        outport(SOFT_RESET, 0) /* RESET EVM          */
#define RESET_C30        outport(CONTROL5,0x808)/* PLACE 'C30 IN RESET  */
#define RUN_C30          outport(CONTROL5,0x800)/* PULL 'C30 OUT OF RESET*/
#define READ_STATUS0     inport (STATUS0)     /* READ  STATUS WORD    */
#define WRITE_MINOR_CMD(x) outport(MINOR_CMD,x)  /* WRITE STATUS WORD    */

/*-------------------------------------------------------------------*/
/* CHECKS FOR READ OR WRITE ACKNOWLEDGMENT                           */
/*-------------------------------------------------------------------*/
#define IS_READ_ACK    (READ_STATUS0 & MREAD_ACK)
#define IS_WRITE_ACK   (READ_STATUS0 & MWRITE_ACK)
/*-------------------------------------------------------------------*/
/* CLEAR STATUS0 BITS                                                */
/*-------------------------------------------------------------------*/
#define CLR_READ_ACK   WRITE_MINOR_CMD(MREAD_ACK)
#define CLR_WRITE_ACK  WRITE_MINOR_CMD(MWRITE_ACK)
#define UPDATE_STATUS0 WRITE_MINOR_CMD(UPDATE_REQ)
```

perf_pc.h

```
/*************************************************************************/
/*      Performance Optimization Controller                              */
/*                                                                       */
/*      Program developed by Shibashis Bhowmik                           */
/*                                                                       */
/*      (C) 1997 OREGON STATE UNIVERSITY                                 */
/*                                                                       */
/*************************************************************************/
/*************************************************************************/
/*                                                                       */
/*      Programming ideas borrowed from the following reference:         */
/*                                                                       */
/*      Digital Signal Processing Applications with the                  */
/*      TMS320C30 Evaluation Module                                      */
/*                                                                       */
/*      OSC_PC.H                                                         */
/*                                                                       */
/*  TMS320C30 EVALUATION MODULE DATA ACQUISITION (OSCILLOSCOPE) DEMO     */
/*        :PC PROGRAMS                                                   */
/*                                                                       */
/*                                                                       */
/*************************************************************************/
/* FUNCTION PROTOTYPES                                                   */
/*=======================================================================
*/
void  main(void);
void  init_graphics(void);
void  status(void);
void  graph(void);
void  scale(void);
void  gputs(char *s);
void  mainMenuCommandProcess(void);
void  invMenuCommandProcess(char passCommand);
int   displayCheck(void);
void  menuWindow(char **entry);
char  *commandParamInput(char commandLetter, int paramLength);
void  averagePower(void);
void  processBiosTime(void);
unsigned long timerDisplay(void);
void  maxmEffiLoop(void);
void  maxmPwrLoop(void);
double *newtonRaphson(double var);
double solve(void);
double *functionDefine(double var);
/*=======================================================================
*/
/* EXTERNALLY DEFINED GLOBAL VARIABLES (FROM GENERIC EVM SUPPORT PROGRAM)
*/
/*=======================================================================
*/
extern int disk[];    /* DISK STORAGE ARRAY                */
extern int buffer;    /* TRANSFER SIZE VARIABLE            */
```

```
extern int index;      /* INDEX INTO DISK STORAGE ARRAY                */
extern double power;
extern int iobase;     /* I/O BASE ADDRESS FOR EVM/TBC REGISTERS       */
extern int comPort;                  /* COMMUNICATION PORT FOR INVERTER ACCESS
          */
extern delay(unsigned);
/*=============================================================================
*/
/* GLOBAL VARIABLES                                                    */
/*=============================================================================
*/
char string[80];       /* STORAGE ARRAY FOR TEMPORARY CHARACTER STRINGS    */
int y[OSC];            /* STORES PREVIOUS VALUES OF OSCILLOSCOPE PLOTS     */
                                                    /*      FOR EASY ERASE                  */
int graph_index = 0;   /* INDEX INTO THE DISK STORAGE ARRAY FOR DATA PLOTS */
int syncFlag       = OFF;  /* INDICATE BDFM SYNCHRONIZATION
          */
int loopFlag   = ON;   /* CLOSED/OPEN LOOP OPERATION STATUS INDICATOR      */
int mpptFlag   = OFF;  /* MPPT LOOP INDICATOR                              */
int meptFlag   = OFF;  /* MPPT LOOP INDICATOR                              */
int invFlag                  = OFF;  /* INVERTER STATUS INDICATOR
                */
int seqFlag                  = ON;   /* POSITIVE PHASE SEQUENCE ACTIVE
                */
int directionFlag = ON;
int magnify    = 0;    /* SCREEN MAGNIFICATION FOR OSCILLOSCOPE DISPLAY    */
int GraphDriver = EGA; /* EGA DISPLAY DRIVER                               */
int GraphMode  = EGAHI;/* DRIVER IN HIGH RESOLUTION EGA MODE               */

float gain   = THREE_V; /* AIC FULL RANGE INPUT SET FOR LINE-LEVEL         */
float sample_rate = 8.0128;     /* AIC SAMPLING RATE VARIABLE              */
double invCurr = 0;
double invCurrOpt;
double invFreq = 0;
double invFreqOpt = 0;
int diffInvFreq = 0;
double speed   = 0;
double fR            = 0;
double fT            = 0;
unsigned long invOnTime = 0;
unsigned long iniBiosTime, newBiosTime, oldBiosTime = 0;
unsigned int maxmEffiLoopCount = 0;
unsigned int maxmPwrLoopCount = 0;
double power_wt;
double sqrVar;
double cubeVar;
double quadVar;
double fROpt;
double lambda = 4.5;  /*LAMBDA_0;*/
double nu;
/*-----------------------------------------------------------------------------*/
/* MAIN MENU FOR DISPLAY                                               */
/*-----------------------------------------------------------------------------*/
char *main_menu[] =
{
  " ",
```

```
  " ",
  "F1  START",
  "F2  Toggle CLOSED/OPEN control",
  "F3  Enable/disable MPPT",
  "F4  Enable/disable MEPT",
  "F5  STOP ALL",
  " ",
  " ",
  "HOME Toggle Magnification",
  " ",
  "END  Quit",
        NULL
};
/*-------------------------------------------------------------------------*/
/* OPEN LOOP INVERTER COMMAND MENU FOR DISPLAY                             */
/*-------------------------------------------------------------------------*/
char *invCommandMenu[] =
{
  " ",
  "E/e      Enable ",
  "D/d      Disable",
  "+        Set pos. seq.",
  "-        Set neg. seq.",
  "W/w      (Iref+0.1) A",
  "X/x      (Iref-0.1) A",
  "Y/y      (freq+0.1) A",
  "Z/z      (freq-0.1) A",
  " ",
  "I/i[][][]  Set ref. curr.",
  "F/f[][][]  Set ref. freq.",
  "O/o[][][]  Set on-time",
  "S/s[][][][]Set BDFM speed",
  " ",
  "Q/q      Quit Inverter Menu",
        NULL
};
/*-------------------------------------------------------------------------*/
/* OPEN LOOP INVERTER COMMAND ARRAY INITIALIZATION                         */
/*-------------------------------------------------------------------------*/
char enable[]                   = {"EN"};
char disable[]                  = {"D"};
char pos[]              = {"+"};
char neg[]                              = {"-"};
char incCur[]                   = {"w"};
char decCur[]                   = {"x"};
char incfreq[]                  = {"y"};
char decfreq[]                  = {"z"};
char curr[5]                    = {"i"};
char freq[5]                    = {"f"};
char onTime[5]                  = {"o"};
char bdfmSpd[5];
char stringForParam[6];     /* INITIALIZE STRING            */
```

perf_cmd.h

```
#define DEL          83
#define HOME         71
#define END          79
#define F1           59
#define F2           60
#define F3           61
#define F4           62
#define F5           63
#define F6           64
#define F7           65
#define F8           66
#define F9           67
#define F10          68
/*-------------------------------------------------------------------*/
/*     COMMANDS FROM THE PC TO THE EVM                               */
/*-------------------------------------------------------------------*/
#define START        F1     /* GET THE EVM READY FOR LOOP OP.   */
#define LOOP         F2     /* TOGGLE CLOSED/OPEN LOOP OPERATION */
#define MPPT         F3     /* ENABLE/DISABLE MPPT              */
#define MEPT         F4     /* ENABLE/DISABLE MEPT              */
#define STOP         F5     /* STOP INVERTER AND CONTROL        */
#define MAGNIFY      HOME   /* TOGGLE MAGNIFICATION ON SCOPE DISPLAY */
#define QUIT         END    /* EXIT FROM PC CONTROL PROGRAM     */
#define RECORDING    128    /* DMA IN READY CONDITION FOR RECORD   */
#define PLAYING      129    /* DMA IN READY CONDITION FOR PLAYBACK  */

#define BLOCK_SIZE   512    /* BLOCK SIZE FOR TRANSFER TO HOST     */
                                                          /*   THIS VALUE IS STORED
IN buffer    */
```

B.2 TMS320C3X power measurement

The Texas Instruments evaluation module provided an analog interface that was utilized for power measurement feedback as required by the optimization controller. The software code utilized for the purpose can be readily found in a Texas Instrument publication named "Digital signal processing application with the TMS320C3X evaluation module".

Appendix C. Wind Turbine Emulator Source Code

The wind turbine emulator though not a part of the optimization controller helped in the evaluation of the controller simulating wind turbine characteristics. The control of the dc machine to represent a wind turbine or to follow any desired torque-speed profile requires a microcontroller to generate the required armature voltage signal. The 80196KC was utilized to generate a pulse-width-modulated signal at one its high-speed output pins. The output was then filtered by a lowpass filter and fed to the dc machine local controller.

Only the 80196 assembly code for the speed and torque control is presented in this Appendix. The 80196 initialization code was similar to that of the rectifier code and has been presented in Appendix A. Its listing would, hence, be avoided here to reduce redundancy. While the user interface protocol was identical to that of the rectifier and the inverter, the program listing is presented here as the commands for the interface are different.

User-interface and program variable initialization:

begin.c

```
#include "proto.h"
-

-
#include "80196.h"
-

-
#include "global.h"
-

-
#include <fpal96.h>
-

-
```

```
#include <math.h>


extern register unsigned int cycleTime, cycleFreq, halfCycleFreq;

extern register unsigned int onTime, saveRemOnTime ;
extern register unsigned int maxVarm, halfMaxVarm ;
extern register unsigned int Kis, Kps, Kit, Kpt ;
extern register unsigned char commandChar;
extern register unsigned char stop, cycleCount;
extern register unsigned char vChar[3];
extern register unsigned int sync;
extern register signed int speed, speedRef, torqueRef;
extern register signed int trqConst;
extern register unsigned int spdConst;
extern register float C0, C1, C2, C3, C4, C5, Ct, Cs;
extern register signed int spdError, oldSpdError, trqError, oldTrqError;
extern register signed int spdError, oldSpdError, trqError, oldTrqError;
extern register signed int oldSpdCntrlVar, oldTrqCntrlVar;
extern register signed int saveRemSpdCntrlVar, saveRemTrqCntrlVar;
extern register signed int sTempVar;
extern register unsigned int uTempVar;
extern register unsigned char i, tempSP_AND, tempSBUF;

/* extern void changeCurrent(void) ;
*/

void begin(void)

{
        fpinit();           /* initialize FPAL96 */

        hso_init();

/* initialize constants */

        C0 = C0_VAL;
        C1 = C1_VAL;
        C2 = C2_VAL;
        C3 = C3_VAL;
        C4 = C4_VAL;
        C5 = C5_VAL;
        Ct = Ct_VAL;

        stop = 0;
        sync = 0;
        tempSP_AND = 0;

        cycleTime = 1200 ;
        cycleFreq = 833 ;
        maxVarm = 960 ;
        halfCycleFreq = cycleFreq/2 ;
        halfMaxVarm = maxVarm/2;
        saveRemOnTime = 0;
/*
```

```
*/
        spdConst = 725;
        speedRef = 0;
        spdError = 0;
        oldSpdError = 0;
        oldSpdCntrlVar = 0;
        saveRemSpdCntrlVar = 0;
/*
*/
        trqConst = 0;
        torqueRef = 0;
        trqError = 0;
        oldTrqError = 0;
        oldTrqCntrlVar = 0;
        saveRemTrqCntrlVar = 0;
/*
*/
        Kis = KIS_VAL  ;
        Kps = KPS_VAL ;
        Kit = KIT_VAL  ;
        Kpt = KPT_VAL ;
/*
*/
        WSR     = 15 ;
        TIMER_1  = 0 ;
        WSR     = 0 ;

        timer2Init() ;
        setUpSwftInit() ;

        WSR     = 15 ;
        HSI_TIME = 0 ;
        WSR     = 0 ;


/* reset timer1 which will be used as the HSI clock source */

        WSR     = 15 ;
        TIMER_1  = 0 ;
        HSI_TIME = 0 ;
        WSR     = 0 ;
        TIMER_2  = 0xFFFD ;

/*********************************************************************
 second variale intialization incorporated from by A. van Zyl
 necessitated due to noise problems as observed on SPOT
*********************************************************************/

        while(!stop)
        {       getChar();
                commandChar = tempSBUF;
                if ((commandChar == 'n') || (commandChar == 'N'))
                {       putChar(10); putChar(13);
                        stop = 1;
                }
        }
```

```
stop = 0;

asm ei ;

while(!stop)
{
        spchk();
        if (tempSP_AND)                 /* tempSP_AND = spchk() */
        {   getBuf();
                commandChar = tempSBUF;
                switch(commandChar)
                {
                case 'T': case 't':
                                putChar(10);putChar(13); putChar('T');
                                sTempVar = torqueRef;

                                if(sTempVar < 0)
                                {       sTempVar = -sTempVar;
                                        putChar('-');
                                }
                                else
                                        putChar('+');

/* for Ct = 1.7066667, factor here is 35 / 6 */

                                vChar[0] = ((sTempVar * 9) / 1000) + '0';
                                putChar(vChar[0]);
                                vChar[1] = (((sTempVar * 9) % 1000) / 100) + '0';
                                putChar(vChar[1]);
                                vChar[2] = (((sTempVar * 9) % 1000) % 100) / 10 + '0';
                                putChar(vChar[2]);
                                putChar('.');
                                vChar[3] = (((sTempVar * 9) % 1000) % 100) % 10 + '0';
                                putChar(vChar[3]);putChar(10);putChar(13);

                                break;

                        case 'D': case'd':
                                asm di;
                                hsoAllClear();
                                IOPORT1 = 0x80;
                                stop = 1;
                                putChar(10);putChar(13);
                                break;

                        case 'S': case 's':
                                for (i=0; i<4; i++)
                                {       getChar();
                                        vChar[i] = tempSBUF - '0';
                                }
                                uTempVar = (vChar[0]*1000 + vChar[1]*100 + vChar[2]*10
+ vChar[3]) / SPD_FAC;
                                putChar(10);putChar(13);
                                if (uTempVar < PEAK_SPEED)
                                        speedRef = uTempVar;
```

```
                    break;


                case 'I': case 'i':
                    for (i=0; i<3; i++)
                    {       getChar();
                            vChar[i] = tempSBUF - '0';
                    }
                    uTempVar = (vChar[0]*100 + vChar[1]*10 + vChar[2]);
                    if (!sync)
                            Kis = uTempVar ;
                    else
                            Kit = uTempVar ;
                    putChar(10);putChar(13);
                    break;

                case 'P': case 'p':
                    for (i=0; i<3; i++)
                    {       getChar();
                            vChar[i] = tempSBUF - '0';
                    }
                    uTempVar = (vChar[0]*100 + vChar[1]*10 + vChar[2]);
                    if (!sync)
                            Kps = uTempVar ;
                    else
                            Kpt = uTempVar ;
                    putChar(10);putChar(13);
                    break;

                case 'Z': case 'z':
                    putChar(10);putChar(13);
                    if ((sync == 0) & (speed > 625))
                    {       sync = 1;
                            putChar('T');putChar('-');putChar('m');
                            putChar('o');putChar('d');putChar('e');
                    }
                    else if ((sync == 0) & (speed <= 625))
                    {       putChar('S');putChar('-');putChar('m');
                            putChar('o');putChar('d');putChar('e');
                    }
                    else if (sync == 1)
                    {       sync = 0;
                            putChar('S');putChar('-');putChar('m');
                            putChar('o');putChar('d');putChar('e');
                    }
                    putChar(10);putChar(13);
                    break;

            }
        }
    }
}
```

80196 assembly listing:

spd_tor.a96 :

```
sptq   MODULE  STACKSIZE(32)
~
;
~
;
~
                RSEG
~
;
~
        EXTRN       PLMReg: LONG
~
;
~
;
~
$ INCLUDE (MEMORY.INC)
~
;
~
;
~
; this module is the HSO software timer interrupt service routine
~
; it controls the dc-machine either in the speed control mode or
; in the torque control mode
;
;
;
CSEG at 7000h
;
$ INCLUDE (80196.INC)
$ INCLUDE (DEFINE.INC)
;
        PUBLIC      changeArmVolt
;
;
        EXTRN       PIregulator:   ENTRY
        EXTRN       analogOut:         ENTRY
;
; define the FPAL96 procedures that are being used
;
        EXTRN       FpLdInt
        EXTRN       FpSt
        EXTRN       FpMul
        EXTRN       FpAdd
        EXTRN       FpLd
        EXTRN       FpDiv
        EXTRN       FpStInt
```

```
;
changeArmVolt:
                        PUSHF
                        ST SP, saveSP
;
; go to horizontal window 0 (default window)
                        LDB WSR, 0
;
; determine which HSO software timer caused the interrupt
                        STB IOS1, tempIOS1
;
; jump to code for software timer 0
                        JBS tempIOS1, SWTF0_BIT, swft0
;
                        POPF
                        RET
;
swft0:
;
; start a/d conversion for dc bus voltage
                        LDB AD_COMMAND, #SPEED_MONITOR
;
                        XORB IOPORT1, #00000001b
;
getSpdAd:
                        JBS AD_LOW, AD_STATUS_BIT, getSpdAd
;
adSpddone:
                        ST AD_LOW, speed
;
                        SHR speed, #6
                        SUB speed, speed0
;
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; to test speed control regulator
;
;               ST spdConst, speed
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; verify if synchronization is done ?
; if (sync == done), then (torque control), else (speed control)
;
;;
;               CMP speed, #625
;               JC tCntrlRange
;               ST 0, sync
;;
;
tCntrlRange:
                        CMP sync, 0
                        JNE genTrqRef
;
                        SUB spdError, speedRef, speed
;
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; reduce steady state error of the speed regulator
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
                    ST spdError, temp
                    JBC temp.15, posSpdErr
                    NEG temp
posSpdErr:
                    CMP temp, #RECON_SPD_STEP
                    JC okKi
                    MUL longTemp, Kis, #10
                    ST longTemp, KisMod
                    SJMP callSpdReg
okKi:
                    LD KisMod, Kis
;
;
callSpdReg:
                    PUSH spdError
                    PUSH oldSpdError
                    PUSH saveRemSpdCntrlVar
                    PUSH spdCntrlVar
                    PUSH oldSpdCntrlVar
                    PUSH KisMod
                    PUSH Kps
;
                    LCALL PIregulator
;
                    POP saveRemSpdCntrlVar
;
                    LCALL VarmLimit
;
                    POP spdCntrlVar

                    ST spdError, oldSpdError
                    ST spdCntrlVar, oldSpdCntrlVar
;
                    ST spdCntrlVar, oldTrqCntrlVar
;
                    PUSH spdCntrlVar
                    LCALL analogOut
                    SJMP goBack
;
genTrqRef:
                    LDB AD_COMMAND, #TORQUE_MONITOR
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;           LD torqueRef, #170
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
                    ST speed, longTemp
                    ST 0, longTemp+2
                    SHLL longTemp, #1         ; scale to actual value
                    PUSH longTemp+2         ; convert integer speed variable
```

```
                        PUSH longTemp       ; to real
                        LCALL FpLdInt       ; Fp$Acc = spd
;
                        LCALL FpSt
                        ST PLMREG+2, speedInReal+2
                        ST PLMREG, speedInReal  ; speedInReal = (real) speed
;
                        PUSH C1+2
                        PUSH C1
                        LCALL FpMul             ; Fp$Acc = C1 * spd
;
                        PUSH C0+2
                        PUSH C0
                        LCALL FpAdd             ; Fp$Acc = C0+C1*spd
;
                        LCALL FpSt
                        ST PLMREG+2, realTemp+2
                        ST PLMREG, realTemp     ; realTemp = C0+C1*spd
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; C0 + C1*speed + C2*speed^2
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
                        PUSH speedInReal+2
                        PUSH speedInReal
                        LCALL FpLd              ; Fp$Acc = spd
;
                        PUSH speedInReal+2
                        PUSH speedInReal
                        LCALL FpMul             ; Fp$Acc = spd^2
;
                        PUSH C2+2
                        PUSH C2
                        LCALL FpMul             ; Fp$Acc = C2*spd^2
;
                        PUSH realTemp+2
                        PUSH realTemp
                        LCALL FpAdd             ; Fp$Acc = C0+C1*spd+C2*spd^2
;
                        LCALL FpSt
                        ST PLMREG+2, realTemp+2
                        ST PLMREG, realTemp     ; realTemp = C0+C1*spd+C2*spd^2
;
;
; convert torque to measured scale
;
                        PUSH Ct+2
                        PUSH Ct
                        LCALL FpMul             ; Fp$Acc = scaled Torque reference
;
                        LCALL FpStInt
                        ST PLMREG+2, longTemp+2
                        ST PLMREG, longTemp     ; scaled torque to integer
                        ST PLMREG, torqueRef
```

```
;
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
                        ST torqueRef, temp
                        JBC temp.15, getTrqAd
                        ST 0, torqueRef
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
getTrqAd:
                        JBS AD_LOW, AD_STATUS_BIT, getTrqAd
;
adTrqDone:
                        ST AD_LOW, torque
;
                        SHR torque, #6
                        SUB torque, #512
                        SUB torque, torque0
                        NEG torque
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; to test torque control regulator
;
;               ST trqConst, torque
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
                        SUB trqError, torqueRef, torque
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; reduce steady state error of the torque regulator
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
                        ST trqError, temp
                        JBC temp.15, posTrqErr
                        NEG temp
posTrqErr:
                        CMP temp, #RECON_TRQ_STEP
                        JC okKit
                        MUL longTemp, Kit, #2
                        ST longTemp, KitMod
                        SJMP callTrqReg
okKit:
                        LD KitMod, Kit
;
callTrqReg:
                        PUSH trqError
                        PUSH oldTrqError
                        PUSH saveRemTrqCntrlVar
                        PUSH trqCntrlVar
                        PUSH oldTrqCntrlVar
```

```
                    PUSH KitMod
                    PUSH Kpt
;
                    LCALL PIregulator
;
                    POP saveRemTrqCntrlVar
;
                    LCALL VarmLimit
;
                    POP trqCntrlVar

                    ST trqError, oldTrqError
                    ST trqCntrlVar, oldTrqCntrlVar
;
                    ST trqCntrlVar, oldSpdCntrlVar
                    ST speed, speedRef
;
                    PUSH trqCntrlVar
                    LCALL analogOut
;
;
goBack:
                    ST saveSP, SP
                    POPF
                    RET
;
;
;
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; SUBROUTINE TO LIMIT DC-m/c ARMATURE VOLTAGE
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
VarmLimit:
                    POP savePC
                    POP cntrlVar
;
; put a safety voltage limiter
;
                    CMP cntrlVar, 0
                    JGE bigger
;
                    ST 0, cntrlVar
                    LJMP smaller
bigger:
                    CMP cntrlVar, maxVarm
                    JLE smaller
                    ST maxVarm, cntrlVar
smaller:
                    PUSH cntrlVar
                    PUSH savePC
                    RET
;
;
```

```
;
;
;
;
```

END

pireg.a96

```
pireg     MODULE  STACKSIZE(20)
;
;
; this module is the PI-regulator code
;
                    RSEG
;
          EXTRN longTemp:                   LONG
          EXTRN longTempTwo:  LONG
          EXTRN longTempThree: LONG
          EXTRN Ki:                         WORD
          EXTRN Kp:                         WORD
          EXTRN oldCntrlVar:      WORD
          EXTRN cntrlVar:         WORD
          EXTRN saveRemCntrlVar:            WORD
          EXTRN oldInpError:      WORD
          EXTRN inpError:                   WORD
          EXTRN temp:                       WORD
          EXTRN cycleFreq:                  WORD
          EXTRN halfCycleFreq:    WORD
          EXTRN savePC:           WORD
;
;
CSEG at 6000h
;
;
PUBLIC              PIregulator
;
;
PIregulator:
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; Code implemented on Mar 29, 1995.
; Equation :
;
; cntrlVar(n+1) = cntrlVar(n) + (Ki/256) * deltaT * inpError(n+1)
;
;                               + (Kp/256) * [inpError(n+1) - inpError(n)]
;
;
; deltaT = 1/cycleFreq ; if updated every program execution loop,
; implemented code modified on 2/13/97 to limit products of multiplications
; adapted from dc-bus regulator of the active rectifier controller
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
                              POP savePC
                              POP Kp
                              POP Ki
                              POP oldCntrlVar
                              POP cntrlVar
```

```
                        POP saveRemCntrlVar
                        POP oldInpError
                        POP inpError
;
                        SUB temp, inpError, oldInpError
                        MUL longTemp, temp, Kp
; put limit on the product, modified 2/13/97
                        ST longTemp+2, temp
                        JBS temp.15, negVar
                        CMP temp, 0
                        JH limitPosProduct
                        CMP longTemp, #7fffh
                        JNC productOk
                        LD longTemp, #7fffh
                        SJMP productOk
limitPosProduct:
                        LD longTemp, #7fffh
                        SJMP productOk
negVar:
                        CMP temp, #0ffffh
                        JLT limitNegProduct
                        CMP longTemp, #8001h
                        JC productOk
                        LD longTemp, #8001h
                        SJMP productOk
limitNegProduct:
                        LD longTemp, #8001h
;
productOk:
                        MUL longTemp, cycleFreq
;                       SHLL longTemp, #3
;
                        MUL longTempTwo, oldCntrlVar, cycleFreq
                        SHLL longTempTwo, #8
;
                        MUL longTempThree, Ki, inpError
;                       SHLL longTempThree, #10
;
                        CLRC
                        ADD longTemp, longTempTwo
                        ADDC longTemp+2, longTempTwo+2
                        CLRC
                        ADD longTemp, longTempThree
                        ADDC longTemp+2, longTempThree+2
                        SHRAL longTemp, #8
                        DIV longTemp, cycleFreq
                        ST longTemp, cntrlVar
;
; interger arithmetic suck ! 7/5/95
;
                        ADD saveRemCntrlVar, longTemp+2
                        ST saveRemCntrlVar, temp
                        JBS temp.15, negRemCntrlVar
                        CMP temp, halfCycleFreq
                        JNH remCntrlVarOk
                        INC cntrlVar
```

```
                    SUB saveRemCntrlVar, cycleFreq
                    SJMP remCntrlVarOk
negRemCntrlVar:
                    NEG temp
                    CMP temp, halfCycleFreq
                    JNH remCntrlVarOk
                    DEC cntrlVar
                    ADD saveRemCntrlVar, cycleFreq
;
remCntrlVarOk:
                    PUSH 0
                    PUSH 0
                    PUSH 0
                    PUSH 0
                    PUSH 0
                    PUSH cntrlVar
                    PUSH saveRemCntrlVar
                    PUSH savePC
;
;
                    RET
;
;
        END
```

anagout.a96

```
analog MODULE  STACKSIZE(8)
;
;
; this module is generates the HSO pulses required for
; a desired analog output
;
;
                RSEG
;
        EXTRN       longTemp:               LONG
        EXTRN       cntrlVar:               WORD
        EXTRN       cycleTime:              WORD
        EXTRN       maxVarm:                WORD
        EXTRN       halfMaxVarm:    WORD
        EXTRN       onTime:         WORD
        EXTRN       saveRemOnTime:          WORD
        EXTRN       temp:                   WORD
        EXTRN       saveTurnOnTime:         WORD
        EXTRN       savePC:         WORD
;
;
;
```

```
;
CSEG at 6800h
;
;
$ INCLUDE (80196.INC)
$ INCLUDE (DEFINE.INC)
;
;
PUBLIC          analogOut
;
analogOut:
;
                        POP savePC
                        POP cntrlVar
;
; determine duty
;;
;                       MUL longTemp, cntrlVar, cycleTime
;                       DIV longTemp, maxVarm
;                       ST longTemp, onTime
;;
                        ST cntrlVar, onTime
;;
;; interger arithmetic suck !  7/5/95
;;
;                       ADD saveRemOnTime, longTemp+2
;                       ST saveRemOnTime, temp
;                       JBS temp.15, negRemOnTime
;                       CMP temp, halfMaxVarm
;                       JNH remOnTimeOk
;                       INC onTime
;                       SUB saveRemOnTime, maxVarm
;                       SJMP remOnTimeOk
;negRemOnTime:
;                       NEG temp
;                       CMP temp, halfMaxVarm
;                       JNH remOnTimeOk
;                       DEC OnTime
;                       ADD saveRemOnTime, maxVarm
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; Limit on On_Time
;
remOnTimeOk:
                        CMP onTime, #MIN_ON_TIME
                        JC checkHigh
                        LD onTime, #MIN_ON_TIME
                        SJMP armVolt
checkHigh:
                        CMP onTime, #MAX_ON_TIME
                        JNC armVolt
                        LD onTime, #MAX_ON_TIME
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
```

```
;
armVolt:
                    XORB IOPORT1, #00000010b
                    JBS IOS0, HSO_HOLDING_BIT, armVolt
;
                    LDB HSO_COMMAND, #ARM_VOLT_SET
                    ADD HSO_TIME, TIMER_1, #HSO_SYNC
lowLimitOk:
                    JBS IOS0, HSO_HOLDING_BIT, lowLimitOk
;
                    LDB HSO_COMMAND, #ARM_VOLT_CLEAR
                    ADD HSO_TIME, TIMER_1, onTime
;
;
                    PUSH 0
                    PUSH savePC
                    RET
;
;
        END
```

Appendix D. Data Acquisition System Source Code

This appendix lists the source code of the data acquisition system along with its graphical interface. The real-time data acquisition is done by an assembly generated source code (datacq.asm) while all peripheral activities is conducted in C-language. The hardware details of the data acquisition system can be found in the following publications:

1) WB-820 Modular Analog and Digital I/O Board, Omega Systems Inc. 1988.
2) OMX-STB-HL High-Level Voltage Panel, Omega Systems Inc. 1988.
3) Phase 3 of a Brushless Dorbly-Fed Machine System Development Program, Final Technical Report for period Jan 1, 1992 – June 30, 1993, Oregon State University.

C-language modules:

omg_gph.c :

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <bios.h>
#include <time.h>
#include <alloc.h>
#include <conio.h>
#include <string.h>
#define MAX_CH 16
#define CLIP_ON 1  /* activates clipping in viewport */
/*
#define MAX_SAM 10000
*/
/*************************************************************************
        declare global variables
*************************************************************************/

int dim_r, dim_c ;
char filename2[20] ;
char  yes1 ;
float fac_cur, fac_volt, fac_pwr_6, fac_pwr_2, fac_tor, fac_spd ;
FILE *fp2 ;
float tdiff ;
int maxx, maxy, midy, midx, thirdx ;
float pmech, effic ;
char bufr2[300], storque[10];
```

```
float ava[MAX_CH] ;



void chart()

{
        char sspd[10] ;
        char sava0[5], sava1[5], sava2[5], sava3[5], sava4[5], sava5[5] ;
        char spmech[6], seffic[5] ;
        char sava8[5], sava9[5], sava10[5], sava11[5], sava12[5], sava13[5], sava14[10], sava15[10] ;
        int h2ia, h2ib, h2ic, h6ia, h6ib, h6ic ;
        int h2va, h2vb, h2vc, h6va, h6vb, h6vc ;
        int ang_2, ang_6, div_ang ;
        int i ;
        int int_tor ;


        for (i=0; i<14; ++i)
        {  if (i != 6)
                        if (ava[i] <= 0.2)
                                ava[i] = 0.0 ;
        }

/*      for (i=0; i<10; ++i)
                sspd[i] = '\0' ;

        int_tor = (int) ava[6] ;
*/
        if (effic < 0.05)
                effic = 0.0 ;
        else if (effic > 100)
                effic = 100 ;

        if (abs(pmech) < 0.2)
                pmech = 0.0 ;

/********* Clean-up previous diagram *************/
        setcolor(59) ;
        setfillstyle(SOLID_FILL, 59) ;
        bar(41, midy-20-115, 180, midy-20);
/*      bar3d(81, midy-20-100, 120, midy-20, 10, 1);
        bar3d(121, midy-20-100, 160, midy-20, 10, 1);
*/
        bar(461, midy-20-115, 600, midy-20);
/*
        bar3d(501, midy-20-100, 540, midy-20, 10, 1);
        bar3d(541, midy-20-100, 580, midy-20, 10, 1);
*/
        bar(41, maxy-40-115, 180, maxy-40);
/*
        bar3d(81, maxy-40-100, 120, maxy-40, 10, 1);
        bar3d(121, maxy-40-100, 160, maxy-40, 10, 1);
*/
        bar(461, maxy-40-115, 600, maxy-40);
```

```
        setcolor(1) ;
        setfillstyle(SOLID_FILL, 1) ;
        bar(309, 230, 365, 270);
        bar(301, 25, 365, 33);
        bar(301, 160, 365, 168);
        bar(255, 35, 375, 157);


/********* Drawing two-pole currents *************/

        h2ia = (ava[0]/30)*100 ;
        setcolor(8) ;
        setfillstyle(SOLID_FILL, 1) ;
        bar3d(41, midy-20-h2ia, 80, midy-20, 10, 1);
        setcolor(1) ;
        settextstyle(DEFAULT_FONT, HORIZ_DIR, 1) ;
        settextjustify(0,2);
        outtextxy(57, midy-17, "I") ;
        outtextxy(62, midy-15, "a") ;
        gcvt(ava[0], 3, sava0) ;
        setcolor(56) ;
        settextjustify(1,2);
        outtextxy(65, midy-37-h2ia, sava0) ;

        h2ib = (ava[1]/30)*100 ;
        setcolor(8) ;
        setfillstyle(SOLID_FILL, 60) ;
        bar3d(81, midy-20-h2ib, 120, midy-20, 10, 1);
        setcolor(1) ;
        settextjustify(0,2);
        outtextxy(97, midy-17, "I") ;
        outtextxy(102, midy-15, "b") ;
        gcvt(ava[1], 3, sava1) ;
        setcolor(56) ;
        settextjustify(1,2);
        outtextxy(103, midy-37-h2ib, sava1) ;

        h2ic = (ava[2]/30)*100 ;
        setcolor(8) ;
        setfillstyle(SOLID_FILL, 58) ;
        bar3d(121, midy-20-h2ic, 160, midy-20, 10, 1);
        setcolor(1) ;
        settextjustify(0,2);
        outtextxy(137, midy-17, "I") ;
        outtextxy(142, midy-15, "c") ;
        gcvt(ava[2], 3, sava2) ;
        setcolor(56) ;
        settextjustify(1,2);
        outtextxy(150, midy-37-h2ic, sava2) ;


/********* Drawing six-pole currents *************/

        h6ia = (ava[3]/30)*100 ;
```

```
        setcolor(8) ;
        setfillstyle(SOLID_FILL, 2) ;
        bar3d(461, midy-20-h6ia, 500, midy-20, 10, 1);
        setcolor(1) ;
        settextstyle(DEFAULT_FONT, HORIZ_DIR, 1) ;
        settextjustify(0,2);
        outtextxy(477, midy-17, "I") ;
        outtextxy(482, midy-15, "a") ;
        gcvt(ava[3], 3, sava3) ;
        setcolor(56) ;
        settextjustify(1,2);
        outtextxy(485, midy-37-h6ia, sava3) ;

        h6ib = (ava[4]/30)*100 ;
        setcolor(8) ;
        setfillstyle(SOLID_FILL, 4) ;
        bar3d(501, midy-20-h6ib, 540, midy-20, 10, 1);
        setcolor(1) ;
        settextjustify(0,2);
        outtextxy(517, midy-17, "I") ;
        outtextxy(522, midy-15, "b") ;
        gcvt(ava[4], 3, sava4) ;
        setcolor(56) ;
        settextjustify(1,2);
        outtextxy(523, midy-37-h6ib, sava4) ;

        h6ic = (ava[5]/30)*100 ;
        setcolor(8) ;
        setfillstyle(SOLID_FILL, 57) ;
        bar3d(541, midy-20-h6ic, 580, midy-20, 10, 1);
        setcolor(1) ;
        settextjustify(0,2);
        outtextxy(557, midy-17, "I") ;
        outtextxy(562, midy-15, "c") ;
        gcvt(ava[5], 3, sava5) ;
        setcolor(56) ;
        settextjustify(1,2);
        outtextxy(570, midy-37-h6ic, sava5) ;

/********* Drawing two-pole voltages *************/

        h2va = (ava[8]/270)*100 ;
        setcolor(8) ;
        setfillstyle(SOLID_FILL, 57) ;
        bar3d(41, maxy-40-h2va, 80, maxy-40, 10, 1);
        setcolor(1) ;
        settextstyle(DEFAULT_FONT, HORIZ_DIR, 1) ;
        settextjustify(0,2);
        outtextxy(57, maxy-37, "V") ;
        outtextxy(64, maxy-35, "a") ;
        gcvt(ava[8], 3, sava8) ;
        setcolor(56) ;
        settextjustify(1,2);
        outtextxy(65, maxy-55-h2va, sava8) ;

        h2vb = (ava[9]/270)*100 ;
```

```
        setcolor(8) ;
        setfillstyle(SOLID_FILL, 62) ;
        bar3d(81, maxy-40-h2vb, 120, maxy-40, 10, 1);
        setcolor(1) ;
        settextjustify(0,2);
        outtextxy(97, maxy-37, "V") ;
        outtextxy(104, maxy-35, "b") ;
        gcvt(ava[9], 3, sava9) ;
        setcolor(56) ;
        settextjustify(1,2);
        outtextxy(103, maxy-55-h2vb, sava9) ;

        h2vc = (ava[10]/270)*100 ;
        setcolor(8) ;
        setfillstyle(SOLID_FILL, 2) ;
        bar3d(121, maxy-40-h2vc, 160, maxy-40, 10, 1);
        setcolor(1) ;
        settextjustify(0,2);
        outtextxy(137, maxy-37, "V") ;
        outtextxy(144, maxy-35, "c") ;
        gcvt(ava[10], 3, sava10) ;
        setcolor(56) ;
        settextjustify(1,2);
        outtextxy(150, maxy-55-h2vc, sava10) ;


/********* Drawing six-pole voltages *************/

        h6va = (ava[11]/270)*100 ;
        setcolor(8) ;
        setfillstyle(SOLID_FILL, 5) ;
        bar3d(461, maxy-40-h6va, 500, maxy-40, 10, 1);
        setcolor(1) ;
        settextstyle(DEFAULT_FONT, HORIZ_DIR, 1) ;
        settextjustify(0,2);
        outtextxy(477, maxy-37, "V") ;
        outtextxy(484, maxy-35, "a") ;
        gcvt(ava[11], 3, sava11) ;
        setcolor(56) ;
        settextjustify(1,2);
        outtextxy(485, maxy-55-h6va, sava11) ;

        h6vb = (ava[12]/270)*100 ;
        setcolor(8) ;
        setfillstyle(SOLID_FILL, 62) ;
        bar3d(501, maxy-40-h6vb, 540, maxy-40, 10, 1);
        setcolor(1) ;
        settextjustify(0,2);
        outtextxy(517, maxy-37, "V") ;
        outtextxy(524, maxy-35, "b") ;
        gcvt(ava[12], 3, sava12) ;
        setcolor(56) ;
        settextjustify(1,2);
        outtextxy(523, maxy-55-h6vb, sava12) ;

        h6vc = (ava[13]/270)*100 ;
```

```
        setcolor(8) ;
        setfillstyle(SOLID_FILL, 2) ;
        bar3d(541, maxy-40-h6vc, 580, maxy-40, 10, 1);
        setcolor(1) ;
        settextjustify(0,2);
        outtextxy(557, maxy-37, "V") ;
        outtextxy(564, maxy-35, "c") ;
        gcvt(ava[13], 3, sava13) ;
        setcolor(56) ;
        settextjustify(1,2);
        outtextxy(570, maxy-55-h6vc, sava13) ;



        ang_2 = (int)((fabs(ava[14])/(fabs(ava[14])+fabs(ava[15])))*360) ;
        ang_6 = (int)((fabs(ava[15])/(fabs(ava[14])+fabs(ava[15])))*360) ;


        if(ang_2 < ang_6)
        {       div_ang = ang_2/2 ;
                setcolor(58) ;
                pieslice(315, 85, 90-div_ang, 90+div_ang, 60) ;

                setcolor(5) ;
                setfillstyle(SOLID_FILL, 5) ;
                pieslice(315, 105, 90+div_ang, 270, 60) ;
                pieslice(315, 105, 270, 360, 60) ;
                pieslice(315, 105, 0, 90-div_ang, 60) ;
        }
        else
        {       div_ang = ang_6/2 ;
                setcolor(58) ;
                pieslice(315, 105, 270-div_ang, 270+div_ang, 60) ;

                setcolor(5) ;
                setfillstyle(SOLID_FILL, 5) ;
                pieslice(315, 85, 270+div_ang, 360, 60) ;
                pieslice(315, 85, 0, 90, 60) ;
                pieslice(315, 85, 90, 270-div_ang, 60) ;
        }



        gcvt(ava[14], 4, sava14) ;
        gcvt(ava[15], 4, sava15) ;



/*      printf("\nava[6] = %f", ava[6] );

        itoa(int_tor, stor, 10) ;
*/
        gcvt(pmech, 4, spmech) ;
        gcvt(effic, 4, seffic) ;

        setcolor(58) ;
```

```
        settextjustify(2,2);
        outtextxy(365, 230, storque) ;
        outtextxy(365, 250, spmech) ;
        outtextxy(365, 260, seffic) ;

        outtextxy(365, 25, sava14) ;
        outtextxy(365, 160, sava15) ;

        gcvt(ava[7], 4, sspd) ;
        outtextxy(365, 240, sspd) ;

/*
        getch();
        closegraph();
        exit(1);
*/

}

void data_convert(int adc[][MAX_CH], int chann[])
{
        int i, j, torque_index, speed_index, p2_index, p6_index ;
        int key, prex, dim_rc ;

        FILE *fsamp ;
        char filename3[20] ;

        float fac_adc, pelec ;
        float cmp ;
        char  ch1[2], ch2, sdim_rc[10], yes2[2], *ext = ".prn" ;

/*      char    bufr3[300] ; */

        fac_adc = 4.8828125e-03 ;



for (j=0; j<dim_c; ++j)
        {       ava[j] = 0.0 ;
                for (i=0; i<dim_r; ++i)
                        ava[j] += adc[i][j] ;
                ava[j] /= dim_r ;
                if (chann[j] <= 5)
                {       ava[j] = ava[j]*fac_adc*fac_cur ;
                }
                else if (chann[j] == 6)
                {
                        ava[j] = ava[j]*fac_adc*fac_tor ;
                        torque_index = j ;
                }
                else if (chann[j] == 7)
                {
                        ava[j] = ava[j]*fac_adc*fac_spd ;
                        speed_index = j ;
                }
                else if ((chann[j] >= 8) && (chann[j] <= 13))
```

```
                {
                        ava[j] = ava[j]*fac_adc*fac_volt ;
                }
                else if (chann[j] == 14)
                {
                        ava[j] = ava[j]*fac_adc*fac_pwr_6 ;
                        p6_index = j ;
                }
                else
                {
                        ava[j] = ava[j]*fac_adc*fac_pwr_2 ;
                        p2_index = j ;
                }
                if (j == (dim_c-1))
                {
                /* Calculate efficiency
                        if torque is greater than zero then pmech/pelec
                        if torque is less then zero then pelec/pmech
                        added by chris brune 5-30-93
                */
                        pmech = 0.011832 * ava[torque_index]*ava[speed_index] ;
                        pelec = ava[p2_index] + ava[p6_index] ;
                        effic = 0 ;
                        gcvt(ava[6], 4, storque) ;

                        if (pmech < 0)
                        {       effic = (pelec / pmech)*100  ;
                        }
                        else
                        {       effic = (pmech / pelec)*100  ;
                        }

                /*      printf("\t\tpmech\t=\t%8.4f\n",pmech);
                        printf("\t\teffic\t=\t%8.4f\n",effic);

                        printf("\n\n\n") ;
                 */

                }
        }
/*      printf("p_2 = %f", ava[14]) ; */

/*  test data  */
/*
        ava[0] = 30.25 ;
        ava[1] = 10.38 ;
        ava[2] = 20.56 ;
        ava[3] = 20 ;
        ava[4] = 25 ;
        ava[5] = 15 ;
        ava[6] = 100 ;
        ava[7] = 1150 ;
        ava[8] = 270.50 ;
        ava[9] = 100 ;
        ava[10] = 120 ;
        ava[11] = 225 ;
```

```
        ava[12] = 220 ;
        ava[13] = 230 ;
        ava[14] = 1000 ;
        ava[15] = 3000 ;
*/

for (i=0; i<15000; ++i)
        key = kbhit();

        if (key != 0)
                yes1 = getch() ;


/*      if (kbhit())*/

if (yes1 == 's')

{
        for (j=0; j<dim_c; ++j)

        {       if (chann[j] <= 5)
                {       fprintf(fp2,"%6.4f ",ava[j]) ;

                }
                else if (chann[j] == 6)
                {       fprintf(fp2,"%7.4f ",ava[j]) ;

                        torque_index = j ;
                }
                else if (chann[j] == 7)
                {       fprintf(fp2,"%8.4f ",ava[j]) ;

                        speed_index = j ;
                }
                else if ((chann[j] >= 8) && (chann[j] <= 10))
                {       fprintf(fp2,"%7.4f ",ava[j]) ;

                }
                else if ((chann[j] >= 11) && (chann[j] <= 13))
                {       fprintf(fp2,"%7.4f ",ava[j]) ;

                }
                else if (chann[j] == 14)
                {       fprintf(fp2,"%8.4f ",ava[j]) ;

                        p6_index = j ;
                }
                else
                {       fprintf(fp2,"%8.4f ",ava[j]) ;

                        p2_index = j ;
                }
                if (j == (dim_c-1))
                {
                /* Calculate efficiency
```

```
                    if torque is greater than zero then pmech/pelec
                    if torque is less then zero then pelec/pmech
                    added by chris brune 5-30-93
            */
                    pmech = 0.011832 * ava[torque_index]*ava[speed_index] ;
                    pelec = ava[p2_index] + ava[p6_index] ;
                    effic = 0 ;

                    if (pmech < 0)
                    {       effic = (pelec / pmech)*100 ;
                    }
                    else
                    {       effic = (pmech / pelec)*100 ;
                    }
                    fprintf(fp2,"%8.4f",pmech);
                    fprintf(fp2,"%8.4f",effic);

                    fprintf(fp2, "\n") ;
            }
}
fflush(fp2) ;

setfillstyle(SOLID_FILL, 10) ;
bar(0, maxy-17, maxx-thirdx, maxy) ;
setcolor(0) ;
outtextxy(10, maxy-10, "Want to store raw data ? (Y/N)" ) ;


moveto(290, maxy-10) ;

yes2[1] = '\0';             /* end of array */
setcolor(4) ;
while ((ch2 = getch()) != 13)
{
        if (ch2 == 8)
        {       prex = getx() ;
                bar(prex-8, maxy-10, prex, maxy-2) ;
                moveto(prex-8, maxy-10) ;
        }
        else
        {       yes2[0] = ch2 ;
                outtext(yes2) ;
        }
}


if((yes2[0] == 'Y')||(yes2[0] == 'y'))
{ bar(0, maxy-17, maxx-thirdx, maxy) ;
        setcolor(0) ;
        outtextxy(10, maxy-10, "Enter file name (.PRN will be appended) :: " ) ;

        dim_rc = dim_r*dim_c ;
        itoa(dim_rc, sdim_rc, 10) ;

        moveto(354, maxy-10) ;
```

```
                for (i=0; i<20; ++i)
                        filename3[i] = '\0' ;

                i = 0 ;
                ch1[1] = '\0';          /* end of array  */
                setcolor(4) ;
                while ((ch1[0]=getch()) != 13)
                {
                        if (ch1[0] == 8)
                        {       prex = getx() ;
                                bar(prex-8, maxy-10, prex, maxy-2) ;
                                moveto(prex-8, maxy-10) ;
                                i = i-1 ;
                        }
                        else
                        {       filename3[i] = ch1[0] ;
                                ++i ;
                                outtext(ch1) ;
                        }
                }

                strcat(filename3, ext) ;

/*              printf("%s", filename3) ;  */

        if ((fsamp = fopen(filename3, "w+t")) == NULL)
        {       setfillstyle(SOLID_FILL, 10) ;
                bar(0, maxy-17, maxx-thirdx, maxy) ;
                outtextxy(10, maxy-10, "Problem in opening file pointer.") ;
                getch() ;
                exit(1) ;
        }

        if ((setvbuf(fsamp, bufr2, _IOLBF, 300)) != 0)
        {
                setfillstyle(SOLID_FILL, 10) ;
                bar(0, maxy-17, maxx-thirdx, maxy) ;
                outtextxy(10, maxy-10, "Not enough memory to allocate buffer space") ;
                getch() ;
                exit(1) ;
        }


                bar(0, maxy-17, maxx-thirdx, maxy) ;
                setcolor(0) ;
                outtextxy(10, maxy-10, "Please wait, writing      samples.... " ) ;
                setcolor(4) ;
                settextjustify(2,2);
                outtextxy(218, maxy-10, sdim_rc) ;
/*              settextjustify(0,2);  */

                for (i=0; i<dim_r; ++i)
                {       fprintf(fsamp,"%f  ", (tdiff/dim_r)*i) ;
                        for (j=0; j<dim_c; ++j)
                        {       if (chann[j] <= 5)
                                        fprintf(fsamp,"%6.4f ",adc[i][j]*fac_adc*fac_cur) ;
```

```
                                else if (chann[j] == 6)
                                        fprintf(fsamp,"%7.4f ",adc[i][j]*fac_adc*fac_tor) ;
                                else if (chann[j] == 7)
                                        fprintf(fsamp,"%8.4f ",adc[i][j]*fac_adc*fac_spd) ;
                                else if ((chann[j] >= 8) && (chann[j] <= 13))
                                        fprintf(fsamp,"%7.4f ",adc[i][j]*fac_adc*fac_volt) ;
                                else if (chann[j] == 14)
                                        fprintf(fsamp,"%8.4f ",adc[i][j]*fac_adc*fac_pwr_6) ;
                                else
                                        fprintf(fsamp,"%8.4f ",adc[i][j]*fac_adc*fac_pwr_2) ;
                        }
                        fprintf(fsamp, "\n") ;
                }
                fflush(fsamp) ;
                fclose(fsamp) ;
        }


/*
        printf("\n\tWriting averaged data\n") ;
        printf("dim_r = %d", dim_r) ;
        fp5 = fopen("adc2.prn", "w+t") ;
        for (i=0; i<dim_r; ++i)
        {       for (j=0; j<dim_c; ++j)
                        fprintf(fp5,"%d\t",adc[i][j]) ;
                fprintf(fp5,"\n") ;
        }
        fclose(fp5) ;
*/

        yes1 = 'c' ;
}

        free (adc) ;
        chart() ;


}



void enter (int chann_num[])

{
/*
        int data[MAX_SAM][MAX_CH] ;
*/

        int (*data)[MAX_CH] ;
/*      int data[100][MAX_CH] ; */
        int i,j ;

        char *stdiff ;

/*
```

```
        clock_t time, time_d[MAX_SAM] ;
        long time, time_d[MAX_SAM] ;
*/

        clock_t start, end ;

        settextjustify(0,2) ;

        if ((data = farcalloc(2*dim_r,dim_c)) == NULL)
        {
                setfillstyle(SOLID_FILL, 10) ;
                bar(0, maxy-17, maxx-thirdx, maxy) ;
                outtextxy(10, maxy-10, "Not enough memory to allocate data storage space") ;
                getch();
                exit(1) ;
        }

        start = clock() ;
        for (i=0; i<dim_r; ++i)
                for (j=0; j<dim_c; ++j)
                        data[i][j] = dat_acq(chann_num[j]) ;
        end = clock() ;

        tdiff  = (end-start)/CLK_TCK ;
/*      tdiff  = 8.079654 ; */
        gcvt(tdiff, 4, stdiff) ;

        setfillstyle(SOLID_FILL, 10) ;
        bar(0, maxy-17, maxx-thirdx, maxy) ;
        setcolor(0);
        outtextxy(10, maxy-10, "Duration of data acquisition =       secs.") ;
        setcolor(4) ;
        outtextxy(260, maxy-10, stdiff) ;
        data_convert(data, chann_num) ;
}


void display()

{

        int window1[8], window2[8], window3[8] ;
        int rec1[8], rec2[8], rec3[8], rec4[8] ;
        int clearanceX, clearanceY ;
        int i;

        /* set for EGA mode */
        int gdriver = EGA, gmode = EGAHI, errorcode;

        /* initialize graphics and local variables */
        initgraph(&gdriver, &gmode, "c:\shiba");

        /* read result of initialization */
        errorcode = graphresult();
        if (errorcode != grOk)  /* an error occurred */
        {
```

```
            printf("Graphics error: %s\n", grapherrormsg(errorcode));
            printf("Press any key to halt:");
            getch();
            exit(1); /* terminate with an error code */
        }

        maxx = getmaxx() ;
        maxy = getmaxy() ;
        midy = maxy/2 ;
        midx = maxx/2 ;
        thirdx = (maxx-6)/3 ;



        window1[0] = 0 ;
        window1[1] = 10;
        window1[2] = maxx;
        window1[3] = 10;
        window1[4] = maxx;
        window1[5] = maxy-20;
        window1[6] = 0;
        window1[7] = maxy-20;

        clearanceX = 15 ;
        clearanceY = 20 ;

        rec1[0] = clearanceX ;
        rec1[1] = clearanceY ;
        rec1[2] = thirdx-5;
        rec1[3] = clearanceY ;
        rec1[4] = thirdx-5;
        rec1[5] = midy-5;
        rec1[6] = clearanceX ;
        rec1[7] = midy-5;

/*      rec2[0] = thirdx+10 ;
        rec2[1] = clearanceY ;
        rec2[2] = 2*thirdx-5 ;
        rec2[3] = clearanceY ;
        rec2[4] = 2*thirdx-5 ;
        rec2[5] = midy-5;
        rec2[6] = thirdx+10 ;
        rec2[7] = midy-5;
*/
        rec2[0] = maxx-clearanceX ;
        rec2[1] = clearanceY ;
        rec2[2] = maxx-thirdx ;
        rec2[3] = clearanceY ;
        rec2[4] = maxx-thirdx ;
        rec2[5] = midy-5;
        rec2[6] = maxx-clearanceX ;
        rec2[7] = midy-5;

        rec3[0] = clearanceX ;
        rec3[1] = midy+5 ;
        rec3[2] = thirdx-5;
```

```
rec3[3] = midy+5 ;
rec3[4] = thirdx-5;
rec3[5] = maxy-27;
rec3[6] = clearanceX ;
rec3[7] = maxy-27;

rec4[0] = maxx-clearanceX ;
rec4[1] = midy+5 ;
rec4[2] = maxx-thirdx ;
rec4[3] = midy+5 ;
rec4[4] = maxx-thirdx ;
rec4[5] = maxy-27;
rec4[6] = maxx-clearanceX ;
rec4[7] = maxy-27;

window2[0] = 0 ;
window2[1] = maxy-17;
window2[2] = maxx-thirdx;
window2[3] = maxy-17;
window2[4] = maxx-thirdx;
window2[5] = maxy;
window2[6] = 0;
window2[7] = maxy;

window3[0] = maxx-thirdx-5 ;
window3[1] = maxy-17;
window3[2] = maxx;
window3[3] = maxy-17;
window3[4] = maxx;
window3[5] = maxy;
window3[6] = maxx-thirdx-5 ;
window3[7] = maxy;

setfillstyle(SOLID_FILL, 1) ;
setlinestyle(0, 0xFFFF, 2);


/* draw a rectangle */
fillpoly(4, window1);
rectangle(3,13,maxx-3,maxy-23);


setfillstyle(SOLID_FILL, 5) ;
bar(thirdx/2,0,maxx-(thirdx/2),9) ;


setcolor(0);
setfillstyle(SOLID_FILL, 59) ;
fillpoly(4, rec1) ;
fillpoly(4, rec2) ;
fillpoly(4, rec3) ;
fillpoly(4, rec4) ;

setfillstyle(SOLID_FILL, 10) ;
fillpoly(4, window2) ;
```

```
        setfillstyle(SOLID_FILL, 12) ;
        fillpoly(4, window3) ;

        settextjustify(1,1);
        setcolor(0);
        outtextxy(midx+1, 6, "Steady State BDFM Data Aquisition System") ;
        setcolor(10);
        outtextxy(midx, 5, "Steady State BDFM Data Aquisition System") ;

        setcolor(0);
        settextjustify(0,2);
        settextstyle(DEFAULT_FONT, HORIZ_DIR, 1.5) ;
        outtextxy(50, 25, "2-Pole Current");
        line(50, 34, 161, 34) ;

        outtextxy(475, 25, "6-Pole Current");
        line(475, 34, 586, 34) ;

        outtextxy(50, 183, "2-Pole Voltage");
        line(50, 191, 161, 191) ;

        outtextxy(475, 183, "6-Pole Voltage");
        line(475, 191, 586, 191) ;

        setcolor(63) ;
        outtextxy(229, 25, "Power_2 =       watts");
        outtextxy(229, 160, "Power_6 =       watts");

        outtextxy(245, 230, "Torque =       lb-in");
        outtextxy(245, 240, "Speed =       rpm");
        outtextxy(245, 250, "Pmech =       watts");
        outtextxy(245, 260, "Effi. =       %");
        outtextxy(maxx-thirdx+10, maxy-10, "S          Q");
        setcolor(1);
        outtextxy(maxx-thirdx+25, maxy-10, "save       quit");



        /* clean up */
/*      getch();
*/

}


int main()

{
        FILE *fp1 ;
        int i ;
        int chann[20] ;
        float fac[6] ;
        char ch[2] ;
        int prex ;
```

```
        display() ;



        setcolor(0);
        settextstyle(DEFAULT_FONT, HORIZ_DIR, 1) ;

        if ((fp1 = fopen("acqset.up", "r+t"))==NULL)
        {
                outtextxy(10, maxy-10, "Error : file acqset.up could not be opened");
                getch() ;
                exit(1);
                 }
        else
        {
                setfillstyle(SOLID_FILL, 10) ;
                bar(0, maxy-17, maxx-thirdx, maxy) ;
                outtextxy(10, maxy-10, "Opened file acqset.up") ;
        }


        fscanf(fp1,"%d", &dim_c) ;
        fscanf(fp1,"%*[^\n]");
/*
        printf("%d\n",dim_c) ;
*/
        for (i=0; i<dim_c; ++i)
        {       fscanf(fp1,"%d", &chann[i]) ;
/*
                printf("%d\n",chann[i]) ;
*/
        }
        fscanf(fp1,"%*[^\n]") ;
        fscanf(fp1,"%d", &dim_r) ;
        fscanf(fp1,"%*[^\n]") ;
/*
        printf("%f\n",t_dur) ;
*/
        for (i=0; i<6; ++i)
        {       fscanf(fp1,"%f", &fac[i]) ;
                fscanf(fp1,"%*[^\n]") ;
        }
        fac_cur  = fac[0] ;
        fac_volt = fac[1] ;
        fac_pwr_6 = fac[2] ;
        fac_pwr_2 = fac[3] ;
        fac_tor  = fac[4] ;
        fac_spd  = fac[5] ;
        fclose(fp1) ;

/*
        dim_r = (int)(t_dur*19000/dim_c) ;

        l_t_dur= t_dur*CLK_TCK ;
```

```
        printf("l_t_dur = %f\n",l_t_dur) ;
*/

        setfillstyle(SOLID_FILL, 10) ;
        bar(0, maxy-17, maxx-thirdx, maxy) ;
        outtextxy(10, maxy-10, "Averaged data file name :: ") ;

        moveto(maxx-thirdx-202, maxy-10) ;

        for (i=0; i<20; ++i)
                filename2[i] = '\0' ;

        i = 0 ;
        ch[1] = '\0';               /* end of array  */
        setfillstyle(SOLID_FILL, 10) ;
        setcolor(4) ;
        while ((ch[0]=getch()) != 13)
        {
                if (ch[0] == 8)
                {       prex = getx() ;
                        bar(prex-8, maxy-10, prex, maxy-2) ;
                        moveto(prex-8, maxy-10) ;
                        i = i-1 ;
                }
                else
                {       filename2[i] = ch[0] ;
                        ++i ;
                        outtext(ch) ;
                }
        }

/*      scanf("%ls", filename2) ; */

        fp2 = fopen(filename2, "w+t") ;
        setvbuf(fp2, bufr2, _IOFBF, 300) ;


        yes1 = 'C' ;
/*
        if ((yes1 == 'Q')||(yes1 == 'q'))
                fclose(fp2) ;
        else
                enter(chann) ;


        while (!kbhit())
                enter(chann) ;

        if ((yes1 = getch()) == 'q')
                fclose(fp2) ;
*/

        while ((yes1 == 'C')||(yes1 == 'c'))

        {
                enter(chann) ;
```

```
        }
        fclose(fp2) ;
        closegraph();
        system("cls");
        exit(1);

        return (0) ;
}
```

Assembly listing:

datacq.asm

```
                .MODEL large

                .STACK 50h

                .DATA

                .CODE

                PUBLIC _dat_acq
_dat_acq        PROC FAR
                push bp
                mov bp,sp
                mov dx,3e0h
        loop1:
                in  al,dx
                and al,0e3h
                jnz loop1
                mov cl,[bp+6]
        ;setting up the required channel for data acqusition
                mov dx,3e1h
                mov al,cl
                out dx,al
                inc dx
                out dx,al
                mov dx,3e0h
        loop2:
                in al,dx
                test al,40h
                jz loop2
                mov dx,3e3h
                in ax,dx
                pop bp
                ret
_dat_acq        ENDP
                END
```

Sample input parameters:

acqset.up:

```
16              /* # of channels to be sampled */
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 /* the channel numbers that needs to be sampled */
1000    /* # of samples */
10              /* multiplying factor for current transducer */
30              /* multiplying factor for voltage transducer */
2014.6  /* multiplying factor for 6-pole power transducer */
2014.6  /* multiplying factor for 2-pole power transducer */
59.13036        /* multiplying factor for torque transducer */
404.5/* multiplying factor for speed transducer */
```