Ryerson University
Digital Commons @ Ryerson

Theses and dissertations

1-1-2007

Real time wind turbine simulator

Bing Gong
Ryerson University

Follow this and additional works at: http://digitalcommons.ryerson.ca/dissertations

Part of the Electrical and Computer Engineering Commons

Recommended Citation
Gong, Bing, "Real time wind turbine simulator" (2007). *Theses and dissertations.* Paper 378.

This Thesis is brought to you for free and open access by Digital Commons @ Ryerson. It has been accepted for inclusion in Theses and dissertations by an authorized administrator of Digital Commons @ Ryerson. For more information, please contact bcameron@ryerson.ca.

NOTE TO USERS

This reproduction is the best copy available.

UMI®

618194770

TJ
828
.G66
2007

REAL TIME WIND TURBINE SIMULATOR

by

BING GONG
BSc, Shandong University of Technology, China, 1993

A thesis
presented to Ryerson University
in partial fulfillment of the
requirements for the degree of
Master of Applied Science
in the Program of
Electrical and Computer Engineering

Toronto, Canada
© Bing Gong 2007

PROPERTY OF
RYERSON UNIVERSITY LIBRARY

UMI Number: EC54183

INFORMATION TO USERS

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleed-through, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

UMI®

UMI Microform EC54183
Copyright 2009 by ProQuest LLC
All rights reserved. This microform edition is protected against unauthorized copying under Title 17, United States Code.

ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

ABSTRACT

Bing Gong

Real Time Wind Turbine Simulator

Electrical and Computer Engineering

Ryerson University

Toronto 2007

A novel dynamic real-time wind turbine simulator (WTS) is developed in this thesis, which is capable of reproducing dynamic behavior of real wind turbine. The WTS is expected to reduce the system testing cost, provide an indoor test platform for the generator; controller and interface apparatus development. In WTS, the turbine is represented by mathematical model and the output torque is precisely controlled on a dc motor drive. By employing the PWM rectifier with LCL filter, the energy is bidirectional which makes the WTS is universal to different scale turbine/generator configurations. PWM rectifier with LCL filter is detail analyzed with different current sensor positions. The results from different sensor positions are compared. A novel torque sensorless inertia compensation algorithm is developed, which enables the WTS to simulate the real wind turbine not only in steady state but also the dynamic procedures in real-time. Simulation and experiment were carried on the 2kW prototype system. Results from both verify the developed scheme of WTS.

ACKNOWLEDGMENTS

I wish to express my deep gratitude to my supervisor, Professor Dewei Xŭ for his support and the knowledge he shared during my graduate studies at Ryerson University.

I am grateful to Professor Bin Wu, Professor Richard Cheung, Dr. Yunwei Li, and all fellow students at LEDAR for their useful discussions on my research.

I also wish to share my achievements with my families and my wife Yan Yan. I am very grateful for her understanding and support.

TO MY PARENTS

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF PRINCIPAL SYMBOLS

1. LCL-based PWM Rectifier

$e(t)$	Grid side voltage
$v(t)$	Converter output voltage
$v_C(t)$	Filter capacitor voltage
$i_1(t)$	Grid-side current
$i_2(t)$	Converter-side current
$i_{1d}(t)$, $i_{1q}(t)$	Grid-side current in d-q reference frame
$v_{C_fd}(t)$, $v_{C_fq}(t)$	Filter capacitor voltage in d-q reference frame
$i_{2d}(t)$, $i_{2q}(t)$	Converter-side current in d-q reference frame
$v_o(t)$	DC link voltage
$i_o(t)$	DC link current from PWM rectifier
$i_L(t)$	DC link load current
L_g	Grid-side equivalent inductance
L	Inductance of converter-side filter inductor
C_f	Capacitance of filter capacitor
f_{res}	Resonant frequency of LCL filter
i_{cmd}	Reference current
i	Converter-side feedback current
i_g	Grid-side feedback current
T_s, f_s	Sampling period, sampling frequency
K_I	Gain of PI regulator
T_I	Time constant of PI regulator
K_{PWM}	Gain of PWM converter
K_d	Gain of high pass filter

T_d	Time constant of high pass filter
m_a	Modulation index of converter

2. Wind Turbine Simulator

ρ	Air density (kg/m3)
A	Turbine swept area (m2)
v_{wind}	Wind velocity (m/s)
λ	Tip speed ratio of the blade tip speed to wind speed
β	Blade pitch angle (deg)
P_m	Mechanical output power of the turbine (W)
c_p	Performance coefficient of the turbine
P_{m_pu}	Turbine output power in per-unit
c_{P_pu}	Performance coefficient in per-unit at maximum value of c_p
v_{wind_pu}	Wind velocity in per-unit
k_p	Power gain of turbine
T_W	Turbine output torque
T_M	Simulator output torque
T_G	Generator electromagnetic torque
J_M	Moment of inertia of DC motor
J_G	Moment of inertia of generator
J_T	Total moment of inertia of turbine, including blades,rotor and gearbox if applicable
T_f	Coulomb friction torque of simulator
B_m	Viscous friction coefficient of simulator
D	Duty cycle
θ, ω	Rotor position angle, rotating speed
$\ddot{\theta}, \frac{d\omega}{dt}$	Acceleration

Chapter 1 Introduction

1.1 Wind Energy Conversion System

The major components of a typical wind energy conversion system (WECS) include a wind turbine, generator, interconnection apparatus and control systems, as shown in Figure 1-1. Wind turbines can be classified into two types according to turbine installation— vertical axis type and horizontal axis type. Most modern wind turbines use a horizontal axis configuration with two or three blades, operating either down-wind or up-wind.

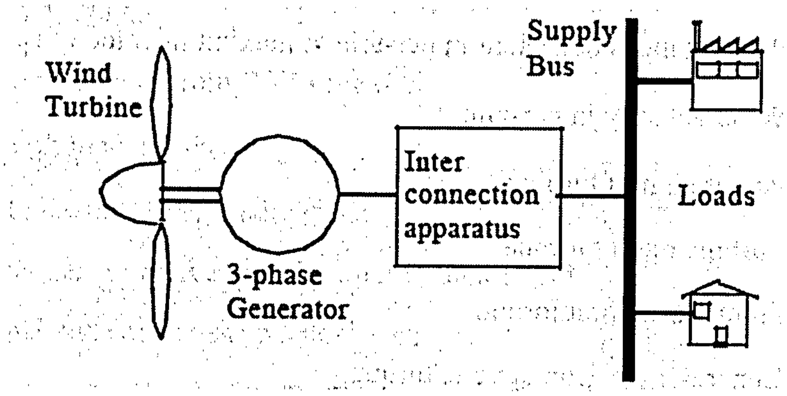


Figure 1-1: Basic components of wind power conversion systems

A WECS can be designed for a constant speed or variable speed operation. A typical constant speed configuration contains a squirrel cage induction generator as shown in Figure 1-2. Gearbox between the low speed turbine rotor and the high speed induction generators is necessary. SCR controlled starter between the generator and the grid aims at reducing the inrush current during connection of the generator to the grid.

Variable speed configurations provide the ability to control the rotor speed and allow the wind turbine system to operate constantly near to its optimum tip-speed ratio [1]. So variable speed wind turbines can produce more energy output as compared to their constant speed counterparts, however, they necessitate power converters to connect the system to the grid. The most common configurations of variable speed WECS are shown in Figure 1-3 [2]. Induction generators, permanent magnet synchronous generators and wound field synchronous generators are widely used in various high power wind turbines. For small to medium power wind turbines, permanent magnet generators and squirrel cage induction generators are more often used because of their reliability and cost advantages. Normally the rotor speed is in the range of 20-40rpm, while the generator rotating at the speed 900-1800rpm. So most of variable speed configurations still need gear box between turbine rotor and generator. Specially designed generator can be used in a direct drive configuration, where a generator is coupled to the rotor of a wind turbine directly, offering high reliability, low maintenance, and possibly low cost for certain turbines.

Interconnection apparatuses are devices to achieve power flow control to the grid. Power electronic converters are often employed to convert and control the variable frequency and variable voltage energy to the fixed grid. Most modern turbine converters are forced commutated PWM converters to provide a fixed voltage and frequency output with a high power capability. Both voltage source voltage controlled converters and voltage source current controlled converters have been applied in wind turbines. For certain high power wind turbines, effective power flow control can be achieved where real power and reactive power are independently controlled.

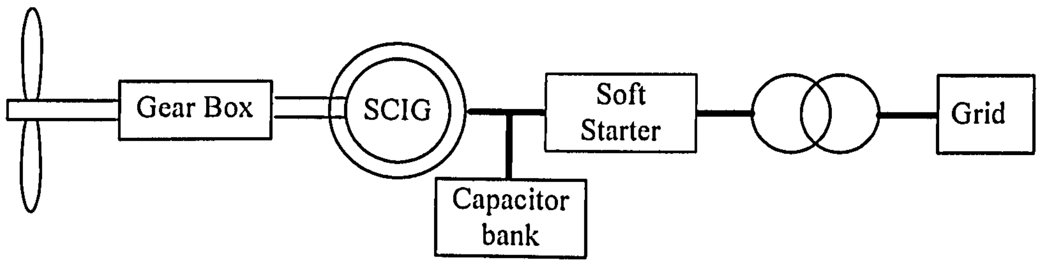
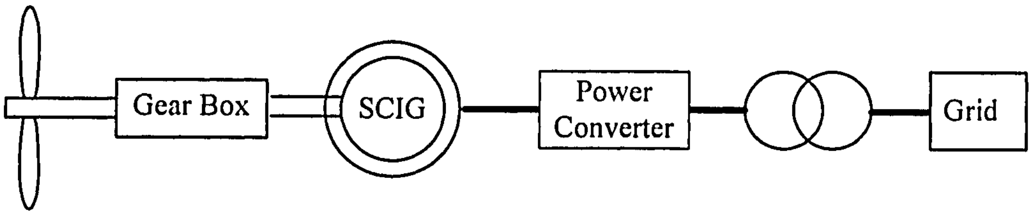
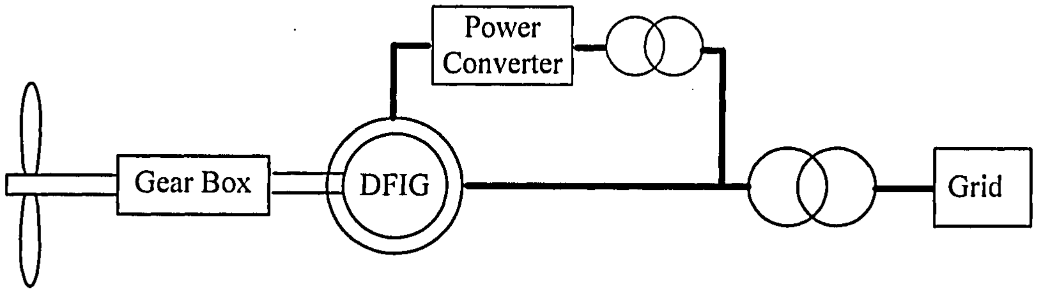


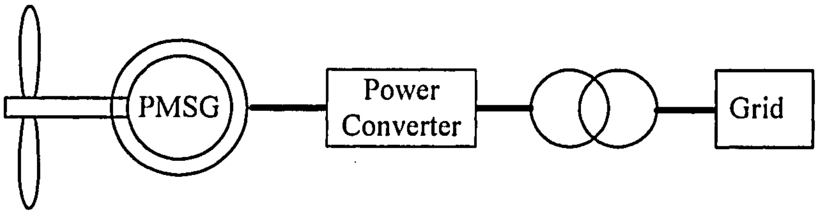
Figure 1-2: Constant-speed wind energy conversion systems



a) WECS using squirrel cage Induction generator □SCIG□



b) WECS using doubly-fed induction generator (DFIG)



c) WECS using direct drive permanent magnet synchronous generator (PMSG)

Figure 1-3: Variable-speed wind energy conversion systems

1.2 Wind Turbine Simulator

To meet the increasing demands for wind power applications, tremendous R&D efforts are needed to develop safe, reliable and cost effective technologies for wind energy conversion. Since the performance of WECS is largely depends on the wind turbine aerodynamics, field test with a real wind turbine would.be very helpful to develop, test and evaluate these wind energy conversion technologies. But a real wind turbine may not always be available and due to the nature of wind, the testing environment is not controllable, which is not very convenient for an early stage development. Although the field test is not avoidable but an alternative, the real-time Wind Turbine Simulator [WTS], which can provide researchers with a controlled in-door test platform for wind turbine generators, inverters and system operations, has been proved to be a very useful tool and has significantly improved R&D effectiveness and efficiency [3].

The wind turbine simulator simulates the mechanical characteristics of a wind turbine at various wind speeds using a controlled motor drive system, as shown in Figure 1-4. For the turbine generator and power electronic converters, the motor drive behaves the same as a variable speed wind turbine. A WTS consists essentially of a torque controlled electric motor and associated controller. A number of WTS have been present in the literature [3-6].

The configuration of developed WTS can be cataloged in following two groups according to the type of motors.

A. DC motor based WTS

A dc motor is the common selection to provide the variable output torque of the wind turbine because the torque is easily estimated and controlled. Thyristor ac-dc converters are usually employed for driving a large dc motor yielding a simple and robust system shown in Figure 1-5. The reference current for the average value of the armature current is calculated as a function of the wind speed and wind turbine speed to simulate the aerodynamic torque of a real wind turbine.

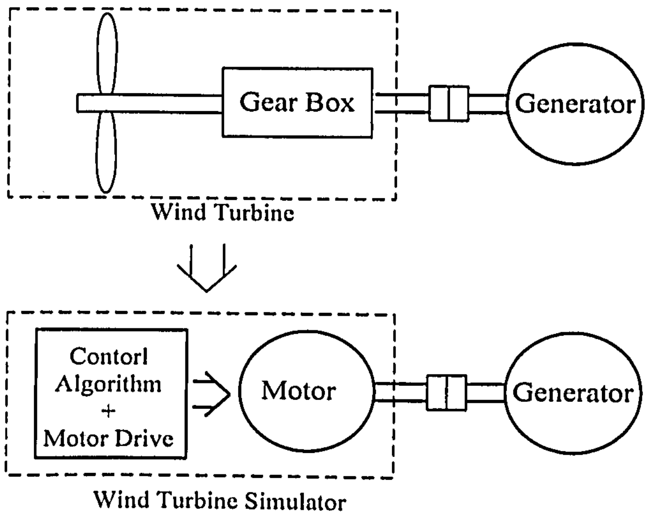


Figure 1-4: Configuration of Wind Turbine Simulator

The disadvantage of this topology is the big armature current ripple and the low bandwidth of the current control loop. A 60 Hz three-phase full-bridge thyristor rectifiers produces a 6th order voltage harmonic that results in a 360 Hz current harmonic in the armature of the dc motor and in a unexpected ripple torque in the WTS. The low bandwidth of the current loop limits the performance of WTS when a fast response is required, especially in the dynamic process.

In order to get a better performance, some developed WTS replaced the thyristor ac-dc converters with a buck converter supplied by a three-phase full-bridge diode rectifier. The block diagram of this configuration is shown in Figure 1-6. A very fast response can be realized due to the PWM buck Converter. In [4], the comparison between this topology and that using SCR was given with respect to the performance as a WTS. The PWM dc-dc converter based WTS was able to reproduce the gradient and the tower shadow effects in an effective manner. It also enables the reproduction of the effects of inertia of the wind turbine and elasticity of the drive train.

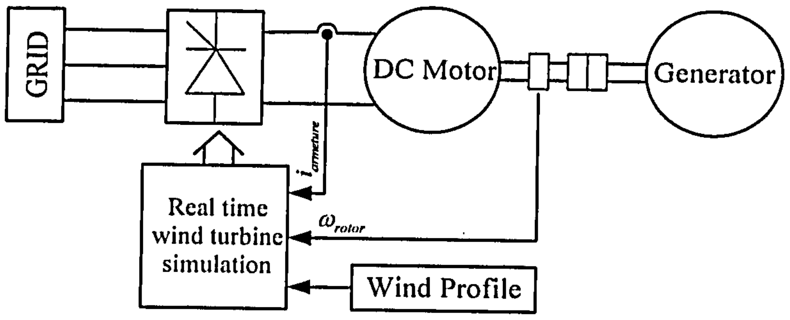


Figure 1-5: DC motor and SCR based WTS

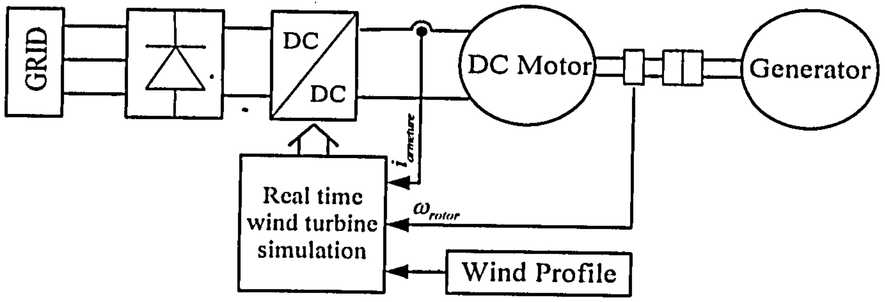


Figure 1-6: DC motor and PWM buck converter based WTS

B. AC motor based WTS

A DC motor, although is ideal from the point of control, but in general is bulky and expensive compare with an AC motor. It needs frequent maintenance due to its commutators and brushes [3]. So AC motor drive with PWM Inverter is also chosen for WTS. A typical block diagram of this configuration is shown in Figure 1-7. A three-phase full-bridge diode rectifier is generally used for the DC supply of PWM Inverter. Both induction machine and synchronous machine can be optional for the AC motor. In [3], an IGBT inverter-controlled squirrel cage induction motor (IM) is used in the WTS. The estimated torque of IM is normally not accurate enough, so the shaft torque transducer is very essential for torque control. In [5], a PMSM (permanent magnet synchronous machine) was employed for WTS due to the merits in terms of power density, inertia, impulse torque response.

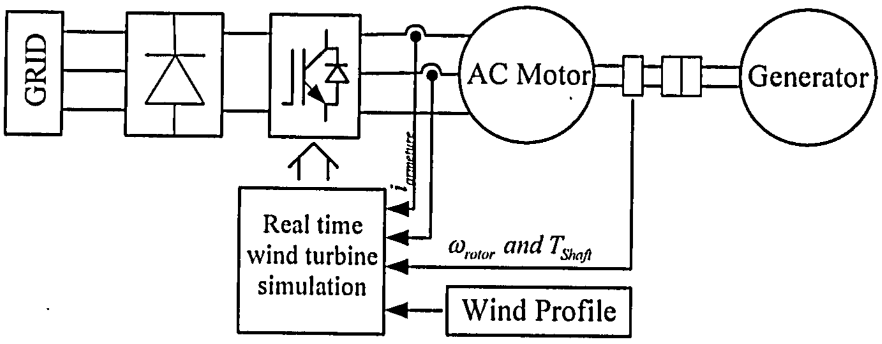

GRID
AC Motor
Generator
$i_{armature}$
Real time wind turbine simulation
ω_{rotor} and T_{Shaft}
Wind Profile

Figure 1-7: Wind Turbine Simulator based on AC motor driven by PWM Inverter

Most of the developed WTS reproduce the mechanical behavior of fixed pitch wind turbines in the steady-state conditions. But only very few of them are capable of dynamic simulations. However the wind, generated by moving air, is not constant. So the real turbine never runs in steady-state. The simulator, which is expected to reproduce the behavior of real turbine, should be able to simulate the turbine mechanical system. The simulator with capability of dynamic simulations generally includes several additional important components: a variable wind speed input, turbine mechanical system real-time simulation and a wind shear and tower shadow model.

A most common approach to get turbine output torque is to use the aerodynamic equation to calculate torque output[5]. The torque is a function of variable wind speed and also the turbine rotor speed when the pitch if fixed. In order to simulate the dynamic process, the model of mechanical system, especially the inertia of turbine-generator should be simulated in real-time. However the analysis of transients due to wind speed or load variations in these systems is sufficient.

The simulator system usually has different inertia of simulated turbine-generator. In order to compensate the difference of simulator inertia and turbine inertia, the torque of the generator, which is mechanically coupled with the motor, must be known. Then a mechanical system simulation can be developed to obtain appropriate torque which is produced by the motor in real-time. In addition, the control bandwidth of the motor torque should be wide enough to accommodate the turbine fast response during transient. Some researches included the effects of the turbine inertia have been done in [4]-[6]. In [6], the generator torque was estimated. The torque together with the simulated turbine torque is used to determine the acceleration on the real turbine. Then the simulator accelerates the generator following the acceleration of real turbine. However this method makes the simulator dependent on the generator side information which sometimes is not accessible. In [5], a torque transducer was mounted on the shaft between the generator and the motor. The generator torque can be derived indirectly from the shaft torque and motor torque. So the simulator can be operated independently. However the installation of the torque sensor makes the system complex and lack of the flexibility.

In [5], the simulator also included the wind shear and tower shadow model, which make the simulator suitable for studying any issues that arise due to the power pulsations resulting from these effects. Topologies with capability of fast response are also necessary for this function which can be proved by the studies in [4].

None of the WTS include effects of the turbine inertia smaller than that of the simulator system. In some applications with small wind turbine systems or low speed direct driven systems, the inertia of the system is possibly smaller than that of the simulator due to the light materials employed.

1.3 Motivation and Objective

WTS is very important for researchers in developing wind energy conversion system. It offers a controllable in-door test environment that allows the evaluation and improvement of control, power converter, interfacing and power system stability, which is hard to achieve with an actual wind turbine and actual grid. So a real-time WTS, which is capable of producing both the static and dynamic torque of a real wind turbine, is necessary. The dynamic effects of both high power large system and low power small should be considered. The simulator should be operated independently and can be easily coupled with different generators. In order to include dynamic effects of the turbine inertia, output torque of WTS should have a fast response. Meanwhile control of the power flow of WTS should be bidirectional because the negative torque may be required during transient when the turbine inertia smaller than that of the simulator system. Further to maintain the simulator performance, especially during long time simulation, the DC link voltage should be maintained no matter the grid side voltage and frequency conditions. The simulator should have friendly connection to the grid, especially for a high-power simulation system. Thus current THD and power factor should be minimized.

Motor speed sensors (encoder) are generally necessary and much easier to be mounted on the shaft than torque sensors. If the torque sensor can be eliminated and an appropriate algorithm of generator torque estimation can be developed, the coupling between the WTS and the generator of the tested system will be much easier. DC motor is still employed for this project due to its excellent performance and simple control scheme since the size and the maintenance of DC machine may not be a big concern for this kind of application.

The objectives of this thesis project include:

1 Construct a lab prototype of WTS. The block diagram of the system is shown in Figure 1-8. PWM rectifier with LCL filter is employed as the front-end of the converter. A DC motor is driven by an H-bridge DC/DC converter, which can implement four-quadrant operation of DC motor drive.

2 Design control schemes for PWM rectifier with LCL-Filter. Designed system should be able to work for a short or long term operation with following merit: bidirectional power flow, controlled DC voltage, fast response, ability to reject grid voltage disturbance, grid-friendly connection with low line current harmonics.

3 Develop algorithm of generator torque estimation and motor torque reference calculation. Implement torque sensorless control of the WTS. The developed WTS can simulate a wind turbine mechanical system in real time, no matter the inertia of real turbine generator system is larger or smaller than that of simulator system.

4 Conduct variety of experiments to verify the performance of the developed WTS system.

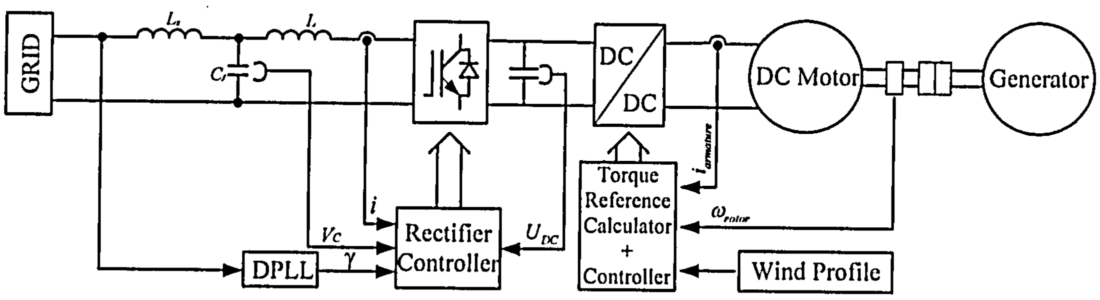


Figure 1-8: The block diagram of the WTS system

1.4 Thesis Organization

This thesis consists of five chapters. The technical background, literature review, and existing wind energy conversion system and wind turbine simulator techniques are presented in this chapter. The motivation and objectives of the thesis are also discussed in section 1.3.

In Chapter 2, control of three-phase PWM rectifier with LCL-filter is proposed. Models of three-phase PWM rectifier with LCL-filter are developed, followed by theoretical analysis of the control scheme with active damping for the LC resonant. The controller design is provided.

In Chapter 3, a novel torque sensorless control scheme for wind turbine simulator is proposed. The WTS system is introduced first. Then the real-time model of the simulated turbine in WTS is developed. At last, the generator torque estimation, which is the key point for the control, is discussed in detail.

Chapter 4 Discrete simulation models for PWM rectifier and WTS are developed respectively. The performance of the system is investigated by simulation results. The experimental verification of the proposed WTS is provided as well. Performance of the PWM rectifier is demonstrated by the test results under different operating conditions including current step response, grid voltage sag and load step change. Then a variety of experiments of the overall WTS system are conducted including simulator steady-state characteristics test, dynamics responses with big turbine inertia and small turbine inertia. The experimental waveforms are analyzed.

Chapter 5 provides the conclusions of the thesis. Other relevant supporting materials are attached in appendices.

Chapter 2 Control of Three-Phase PWM Rectifier with LCL-Filter

2.1 Introduction

The proposed WTS system can simulate the dynamic performance of the real wind turbine, in which the kinetic energy of the real turbine generator should be precisely controlled. Thus the energy is required to be bidirectional in the simulator if the simulated wind turbine system has smaller moment of inertia than the simulator system. A low cost diode rectifier is common in most of the motor drive system. However, the diode rectifier results in unidirectional power flow, and high level of harmonic input currents. Therefore, a three-phase pulse-width-modulated (PWM) rectifier is required due to bidirectional power flow. It also has the advantages including low harmonic distortion of line current; regulated power factor, adjusted of dc-link voltage and reduced dc filter capacitor [7]. The fast and accurate control of motor current also benefits from the regulated dc link voltage.

Traditionally, line inductor (L-type filter) is employed as the interface between the grid and three-phase PWM rectifier. With L-filter, high switching frequency must be used to obtain sufficient attenuation of harmonics caused by PWM. In high power applications, the switch frequency is limited. Then the LC based filter is often used to obtain high attenuation rate. The filter combined with line inductor is also called LCL-filter, which provides the significant improved harmonics performance [8]. However, the LCL-filter's frequency response has resonant peaks which can be internally excited in the control loop if the controller is not properly designed. External loads, disturbances or transients can also excite the resonance, cause voltage oscillations and in the worst case, instability [9]. In order to assure stable operation either a damping resistor or active damping algorithm has to be implemented. Since damping resistor generates additional loss and becomes less efficiency, active damping is preferred.

2.2 Three-phase PWM Rectifier with LCL-filter

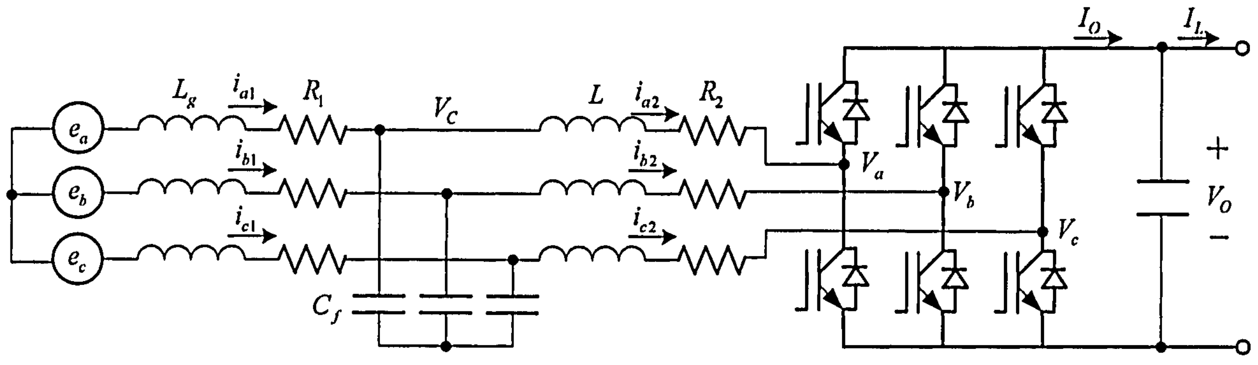


Figure 2-1: Three-phase PWM rectifier with LCL filter

A typical three-phase PWM rectifier with LCL filter is shown in Figure 2-1.

The differential equations can be established for the LCL-filter in the stationary reference frame, which describes the behaviors of the grid side equivalent inductor L_g, filter capacitor C_f and converter side filter inductor L.

$$\frac{di_1(t)}{dt} = \frac{1}{L_g}[-R_1 i_1(t) + e(t) - v_C(t)] \tag{2-1}$$

$$\frac{dv_C(t)}{dt} = \frac{1}{C_f}[i_1(t) - i_2(t)] \tag{2-2}$$

$$\frac{di_2(t)}{dt} = \frac{1}{L}[-R_2 i_2(t) + v_C(t) - v(t)] \tag{2-3}$$

In Eq. 2-1~2-3, $e(t) = \begin{bmatrix} e_a(t) & e_b(t) & e_c(t) \end{bmatrix}^{\mathrm{T}}$ is grid side voltage, $v(t) = \begin{bmatrix} v_a(t) & v_b(t) & v_c(t) \end{bmatrix}^{\mathrm{T}}$ is converter output phase voltage and $v_C(t) = \begin{bmatrix} v_{Ca}(t) & v_{Cb}(t) & v_{Cc}(t) \end{bmatrix}^{\mathrm{T}}$ is filter capacitor voltage.

$i_1(t)=[i_{a1}(t) \;\; i_{b1}(t) \;\; i_{c1}(t)]^{\mathrm{T}}$ and $i_2(t)=[i_{a2}(t) \;\; i_{b2}(t) \;\; i_{c2}(t)]^{\mathrm{T}}$ are grid-side and converter side current respectively.

The above equations can be transferred to selected dq reference frame by applying park-clark transformation. Assume that the three-phase system is balanced three-phase three-wire system, in which zero components can be neglect. By expanding the d and q components, the 6-order differential equations can be written in Eq. 2.4- 2.9, where ω is the rotating speed of reference frame.

$$\frac{di_{1d}(t)}{dt}=\frac{1}{L_g}[-R_1 i_{1d}(t)+e_d(t)-v_{C_f d}(t)]+\omega i_{1q}(t) \tag{2-4}$$

$$\frac{di_{1q}(t)}{dt}=\frac{1}{L_g}[-R_1 i_{1q}(t)+e_q(t)-v_{C_f q}(t)]-\omega i_{1d}(t) \tag{2-5}$$

$$\frac{dv_{C_f d}(t)}{dt}=\frac{1}{C_f}[i_{1d}(t)-i_{2d}(t)]+\omega v_{C_f q}(t) \tag{2-6}$$

$$\frac{dv_{C_f q}(t)}{dt}=\frac{1}{C_f}[i_{1q}(t)-i_{2q}(t)]-\omega v_{C_f d}(t) \tag{2-7}$$

$$\frac{di_{2d}(t)}{dt}=\frac{1}{L}[-R_2 i_{2d}(t)+v_{C_f d}(t)-v_d(t)]+\omega i_{2q}(t) \tag{2-8}$$

$$\frac{di_{2q}(t)}{dt}=\frac{1}{L}[-R_2 i_{2q}(t)+v_{C_f q}(t)-v_q(t)]-\omega i_{2d}(t) \tag{2-9}$$

On the dc link side, the dc link voltage can be described in Eq. 2.10 according to Kirch-Hoff current law.

$$\frac{dv_O}{dt}=\frac{1}{C}(i_O-i_L) \tag{2-10}$$

2.3 Control Scheme for Three-phase PWM Rectifier

2.3.1 Overview

Voltage oriented or virtual flux oriented control are widely used for PWM rectifier of controlling active and reactive power. By proper selecting the synchronous reference frame, the voltage vector is aligned to the d axis. The control of active and reactive power can be decoupled, which generated high dynamic and steady state performance. The controller is usually performed into two loops, current loop and voltage loop on d-q axis respectively. The inner loop (current loop) regulates the output current and the outer loop (voltage loop) controls the dc link voltage. The cascaded structure bases on the fact that dc link has large capacitance. Thus the time constant of dc link voltage is much higher than that of filter or line current. The block diagram of the whole control system is shown in Figure 2-2. It should be noted that in the block diagram, the current feedback comes either the grid side or converter side. The details of different selection will be discussed in section 2.3.2.

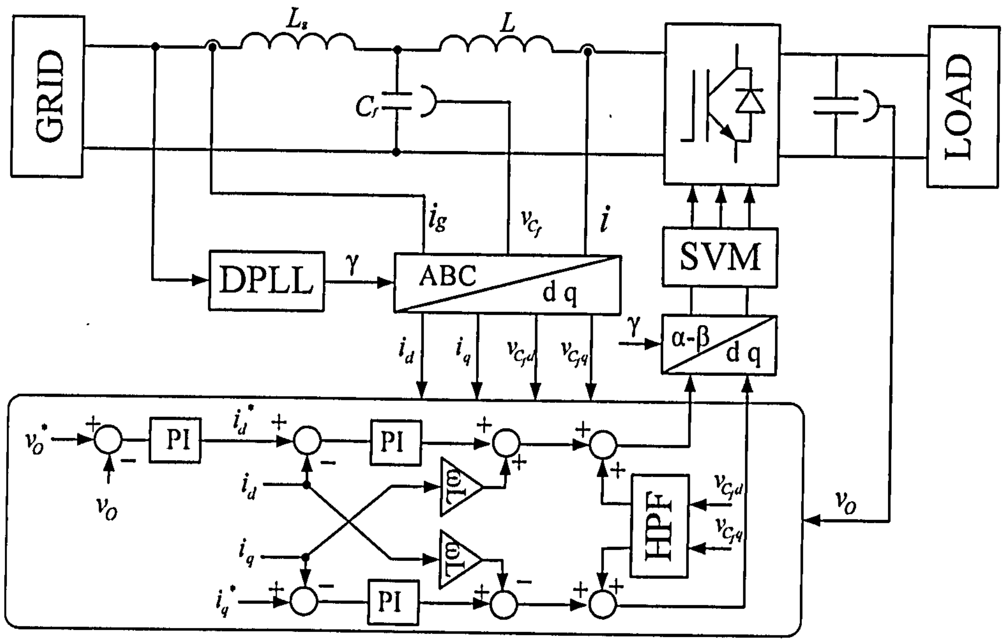


Figure 2-2: Block diagram of the control scheme for PWM rectifier

In Figure 2-2, the reference angle is generated by digital phase-locked-loop (DPLL), which tracks the grid phase angle. When DPLL is locked with grid, the angle γ is the synchronous angle of the reference frame. Transformation blocks receive the synchronous angle and transform the three phase current and filter capacitor voltage into synchronous reference frame.

Two PI regulators are employed to regulate the the d and q component currents. The d-component reference current i_d^* is generated by the dc voltage regulator, which represents the active power flow, while the q-component reference current i_q^* is calculated based on the required reactive power. To improve the tracking performance of the current regulators, the cross-coupling terms $i_d\omega L$ and $i_q\omega L$ between the d- and q-axes are compensated by adding feed forward signals to the outputs. The capacitor voltages are transferred into dq frame and pass the high-pass filters (HPF). Then the high frequency components of capacitor voltage are added to the outputs of the current regulators, which cancel the possible resonant for active damping. The detail will be discussed in section2.3.2.

Normally the current sensors are on the converter side because the sensor can be active in the protection circuit in case of damaged capacitor. It is interesting to be noted that current control loop is much more close to stability by using grid side current sensor instead of converter side. In [10], the design of an LCL-filter based active rectifier is reported which is stable even without damping due to a proper selection of the passive components in LCL-filter and the grid side current sensors. But the comparison of control with these two current sensors positions has not been theoretically studied. In this chapter, comparison based on theoretical analysis is conducted followed by the corresponding control schemes design.

The stability analysis of the control system is based on the conventional method by using root locus and bode plot of the control loop gain in S-domain. Once the transfer function of the open loop gain is derived, the root locus and bode plot can be easily analyzed with the toolboxes in Matlab.

2.3.2 Current control loop design with active damping

According to the differential Eq. 2-1~2-3, single-phase equivalent circuit of the LCL-filter in S-domain can be derived as shown in Figure 2-3.

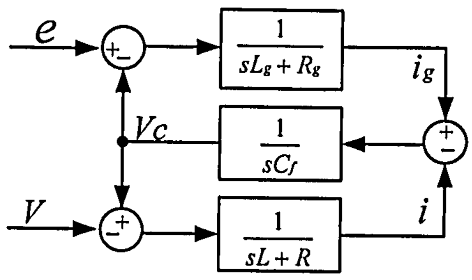


Figure 2-3: Single-phase equivalent circuit of the LCL-filter in S-domain

Combined with the regulators and PWM rectifier, the system can be modeled separately on d and q axis, as shown in Figure 2-4. The cross coupling terms are considered as disturbances, which is taken into account for the regulator design. In the block diagram, assume that the dc link voltage is constant. Then PWM rectifier is treated as a power amplifier with coefficient K_{PWM} and half period (at switching frequency) time delay. The digital control system is running synchronous with the switching frequency and then a sample time Ts delay (processing delay) should be considered, as shown in Figure 2-4. In the block diagram, other time constants including sensors are negligible compared with switching frequency. In Figure 2-4, i^* is the reference current. i_1 and i_2 are current feedback. K_i and T_i are the parameters of regulator. K_d and T_d determine the gain and bandwidth of HPF. Without considering the cross coupling terms, the model for q-axis is the same as the d-axis.

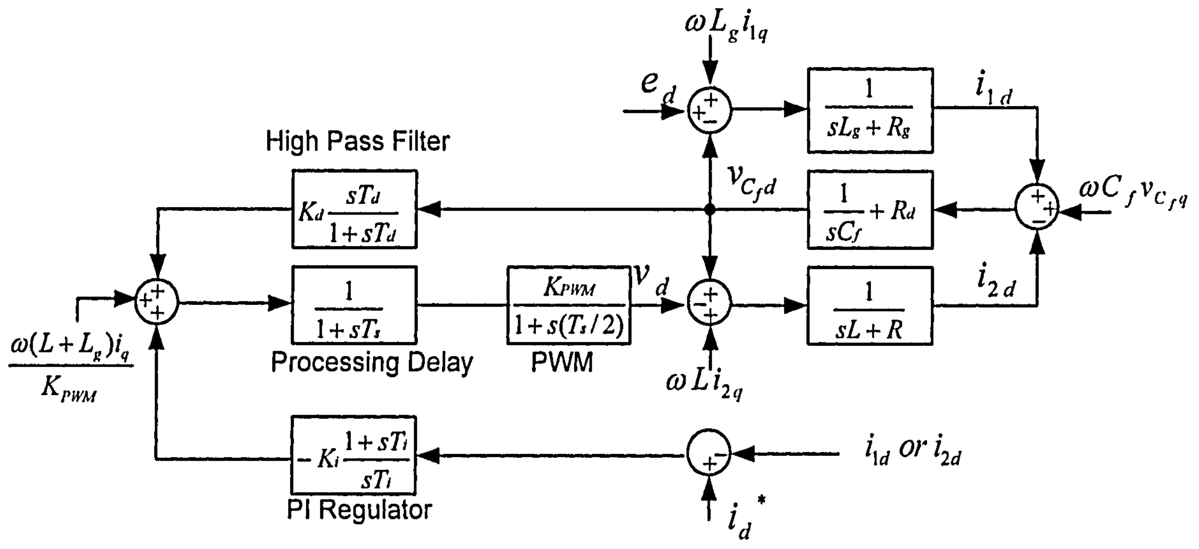


Figure 2-4: d-axis current control system model in the S-domain

Base on the model, the current control plant with different current sensor positions can be expressed in Eq. 2-11 and Eq. 2-12.

When converter side current is sensed, the converter voltage to sensed current transfer function can be written in Eq. 2-11.

$$G(s) = \frac{i_2(s)}{v(s)} = \frac{G_1(s) + G_1(s)G_2(s)G_3(s)}{1 + G_1(s)G_3(s) + G_2(s)G_3(s)} \tag{2-11}$$

When grid side current is sensed, the transfer function of control plant is changed to Eq. 2-12

$$G(s) = \frac{i_1(s)}{v(s)} = \frac{G_1(s)G_2(s)G_3(s)}{1 + G_1(s)G_3(s) + G_2(s)G_3(s)} \tag{2-12}$$

where $G_1 = \frac{1}{Ls+R}, G_2 = \frac{1}{L_g s + R_g}, G_3 = \frac{1}{C_f s}$

The parameters are selected based on the 2kW lab prototype system which is shown in Table2-1. The Bode plots of the plants with converter side current and grid side current are shown in Figure 2-5 and Figure 2-6 respectively. The resonant peak of the LCL-filter can be seen in both situations. Resonant frequency of the LCL-filter can be derived in the Eq. 2-13 [12].

$$f_{res} = \frac{1}{2\pi}\sqrt{\frac{L+L_g}{LL_gC_f}} \tag{2-13}$$

Table 2-1: Parameters of LCL filter

Inductors L_g and L	1mH/20A
Capacitors C_f	$10\,\mu F$

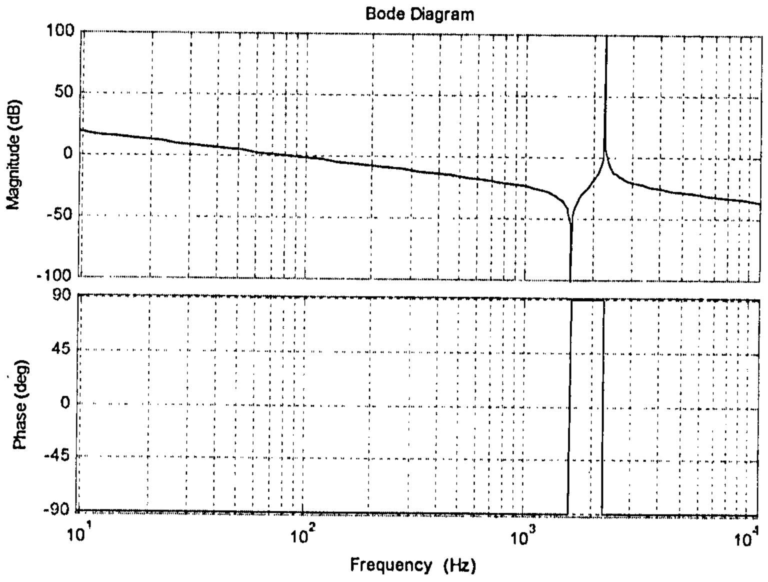


Figure 2-5: The Bode of the plant using converter-side current

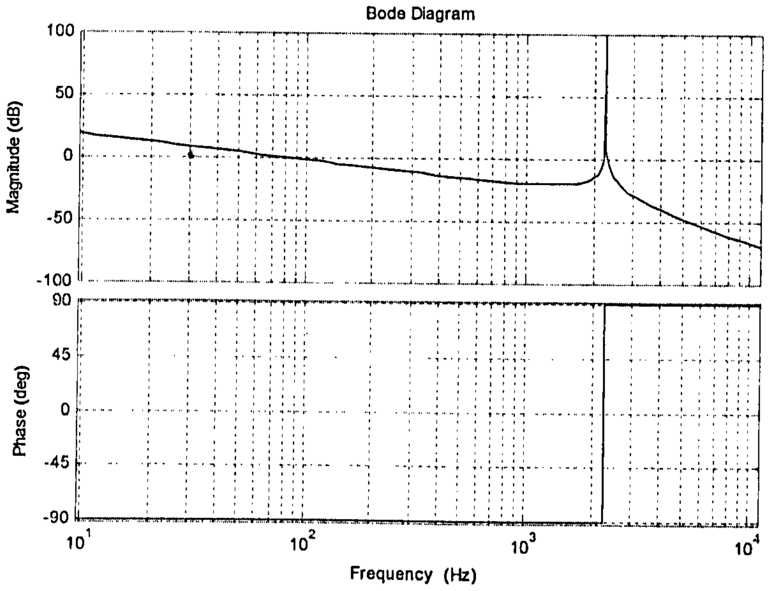


Figure 2-6: The Bode of the plant using grid-side current

At low frequency range, both plants are close to the first order system and have the similar behavior of L-type filter. However the difference is obvious at higher frequency. At frequency range higher than resonant frequency, the converter side current decays at 20 dB per decade while the grid side current decays at 60 dB per decade. This result can explain why the LCL filers have a much better attenuation of high-order harmonics in grid current than L-type filters.

In order to get a satisfied attenuation of high-order harmonics together with other constrains of LCL-filter design [12], the resonant frequency is normally chosen to be lower than half of the switching frequency and avoid the harmonic frequencies of power supply.

In practical design, the processing delay and PWM time delay is exist as dominant delays and can not be neglected. In order to get a higher control bandwidth, the sample frequency should be as higher as possible. In practical, triangle carrier waveform based centre aligned PWM is adopted. The sampling and calculated PWM duty cycles are updated twice during each PWM cycle. In order to avoid the possible switching noise, the sampling often occurs when the carrier waveform reaches the peak value [11]. The processing delay and PWM dead time delay can be modeled in Eq. 2-14 and Eq. 2-15 respectively.

$$G_{prs} = \frac{1}{1+sT_s} \tag{2-14}$$

$$G_{PWM} = \frac{1}{1+s(T_s/2)} \tag{2-15}$$

According to Symmetrical Optimum method described in [11], PI controller parameters are selected. Without active damping (no HPF), the Root Locus and the Open-Loop Bode plots of the d axis can be drawn in Figure 2-7 and Figure 2-8 (the detail parameters are listed in Appendix B). When the converter side current is sensed, the phase at resonant frequency lies in an unstable scope due to the phase shift introduced by delays in the control loop as shown in the Bode plots in Figure 2-7. Then the resonant peak will cause instability. From root locus, a pair of roots located at the right-half plane, which verifies this result. In the other hand, when grid side current is sensed, the current control loop is stable because the phase at resonant frequency is just shifted into a stable range by delays. But the system's stability is very sensitive to the parameters because of the resonant peak, which reduces the system phase margin.

To guarantee stable operation, the resonant peak must be suppressed below 0 dB to provide enough gain margin. Thus an active damping loop is necessary. In this active damping loop [15], the high frequency components of capacitor voltage is reproduced by PWM rectifier. The resonant voltage caused by LC filter will be cancelled. Then the current has no component at resonant frequency. As shown in Figure 2-4, the capacitor voltage passes high-pass filters (HPF) to get high frequency component. HPF can be modeled in Eq (2-16).

$$G_d(s) = K_d \frac{sT_d}{1+sT_d} \tag{2-16}$$

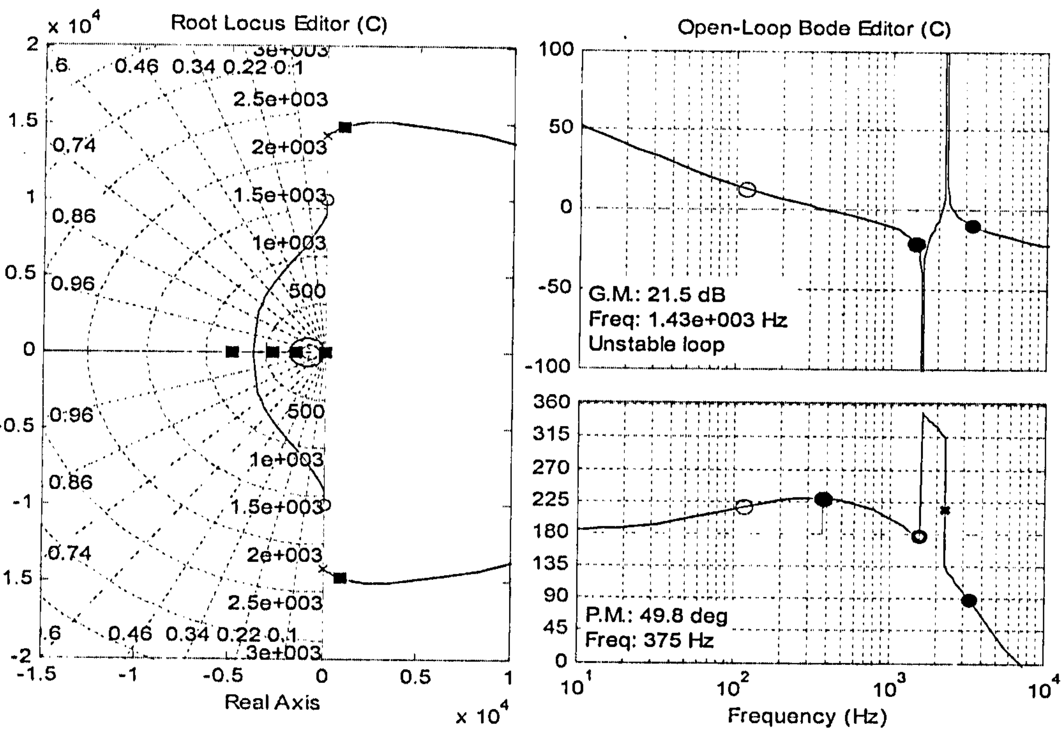


Figure 2-7: Root Locus and Bode Plots of Open-Loop Transfer Function using converter-side current and no damping

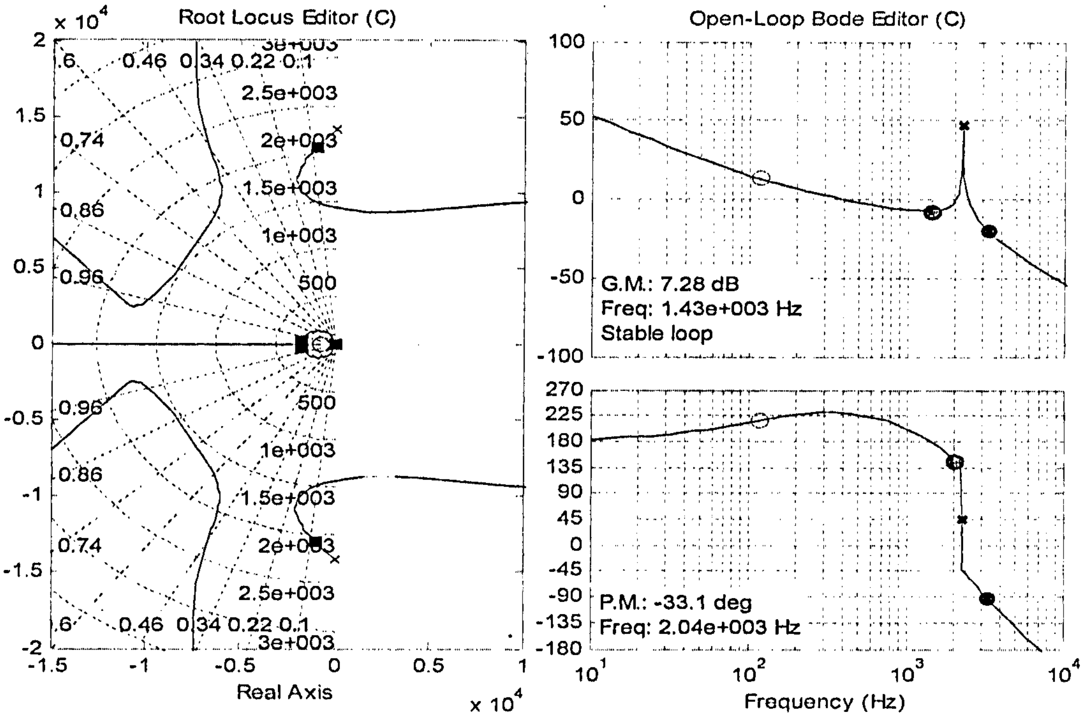


Figure 2-8: Root Locus and Bode Plots of Open-Loop Transfer Function using grid-side current and no damping

Parameters of HPF should be carefully chosen in order to get a satisfied damping effect for the system. Normally K_d is selected to be no more than $1/K_{PWM}$. As long as the resonant peek can be effectively suppressed, T_d is chosen to be a value as small as possible ranging from T_s to $4T_s$. Thus the current control loop within the control bandwidth will not be affected by the damping loop.

With active damping, the Root Locus and the Open-Loop Bode plots of the current control loop can be redrawn in Figure 2-9 and Figure 2-10 respectively. Both converter side current based system and grid side current based system are stable and the suppression of the resonant peak is visible. Compare these figures with that without damping loop, it can be found that the damping loop doesn't affect the Bode plots around corner frequency except the suppression of the resonant peak. Within the control bandwidth, the systems performance is almost keep unchanged.

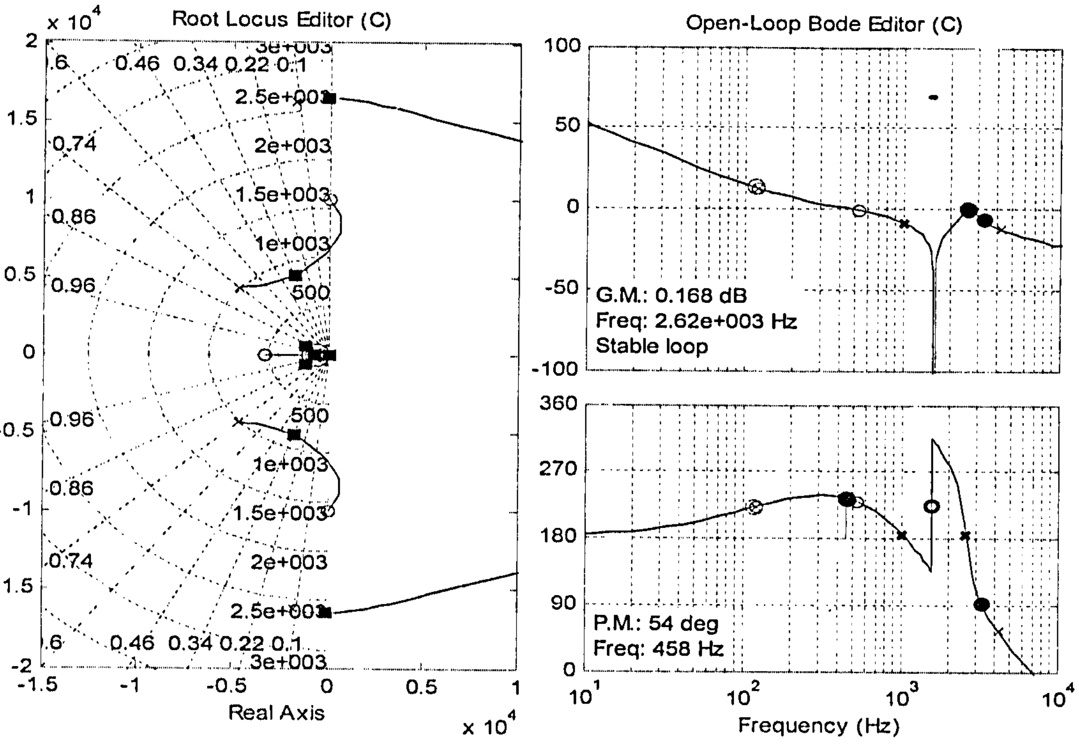


Figure 2-9: Root Locus and Bode Plots of Open-Loop Transfer Function using converter-side current and active damping

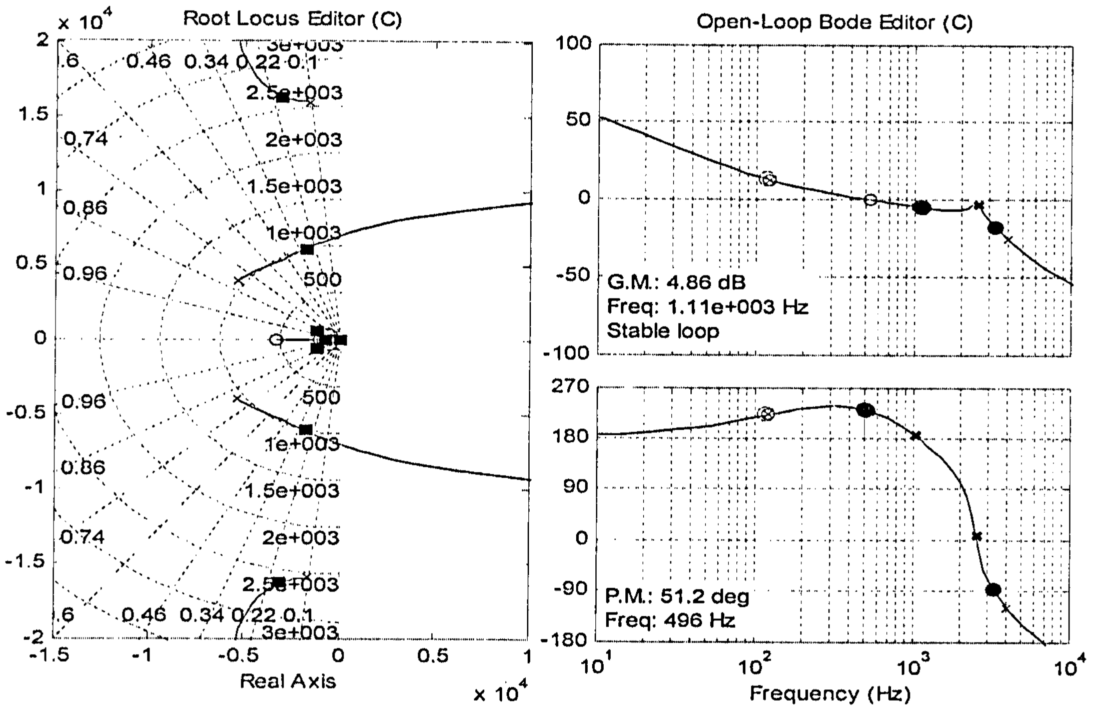


Figure 2-10: Root Locus and Bode Plots of Open-Loop Transfer Function using grid-side current and active damping

In the prototype system, the current control bandwidths of the converter side current based system and grid side current based system are 458Hz and 496Hz respectively. Within the control bandwidth, there is no significant difference between Figure 2-9 and Figure 2-10 because the two systems have quite similar performance under the bandwidth. It should be noted that the q-component reference current for the converter current based system should be set to a certain value, which compensates the capacitor current in order to get a line side unit power factor. The system using grid side current can achieve unit power factor automatically if the q-component reference current is set to zero.

2.3.3 Control Loop Design for DC Link voltage

The voltage loop is the outer loop and should be designed to be much slower than the inner loop according to cascaded system assumption. Generally, the bandwidth of the outer loop is chose to be less than one-fourth of the inner loop, thus the inner loop and the outer loop can be considered decoupled.

The block diagram of voltage control loop is shown in Figure 2-11, including the current loop transfer function and a PI controller, where T_c is the equivalent time constant of the inner loop. The block between DC link current i_o and i_d is derive based on power balance analysis of the rectifier. The selection of PI regulator can be done based on traditional method of bandwidth and phase margin.

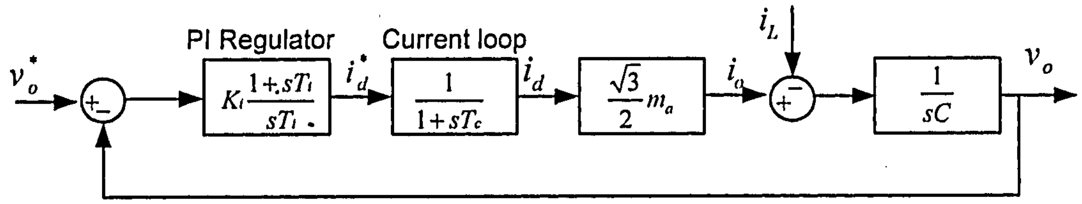


Figure 2-11: voltage control loop in the S-domain

2.4 Conclusion

In this chapter models of PWM rectifier with LCL-filter is presented, which is followed by detail analysis of the system and the control scheme. The effectiveness of the active damping loop for the current control is also theoretically verified. Based on the analysis, the active damping using HPF effectively reduce the peak gain at resonant frequency. Based on the developed loop transfer functions, the current and voltage regulators can be designed based on traditional method. Comparison of control for converter side current based system and grid side current based system is also provided. The two systems will have quite similar performance if the resonant is well damped.

Chapter 3 Wind Turbine Simulator

3.1 Introduction

The proposed wind turbine simulator in this thesis consists of a DC motor supplied by a four quadrant chopper. The chopper is connected to the DC link fed by PWM rectifier. The DC motor with high dynamic performance and accurate control of electromagnetic torque is employed as the major driver. In order to simulate the dynamic behavior of the real wind turbine, the generator side torque is required to exactly produce the acceleration or deceleration of the real wind turbine. The torque can be measured via torque transducers, which is very costly. In this research, a sensorless method is proposed and the generator side torque is estimated by system acceleration and known system moment of inertia. As shown in Figure 3-1. A set of control algorithms including DC motor controller and torque estimator taking into account both the dynamic and static turbine model are developed. First motor relative position angle θ is measured via an incremental encoder and a phase-locked-loop (PLL) block estimates the motor speed ω and acceleration $\frac{d\omega}{dt}$. $\frac{d\omega}{dt}$ is then fed to the torque estimator in which generator side torque T_G is estimated. Wind turbine model produces the reference torque T_M^* for DC motor based on the motor speed ω, generator side torque T_G and wind profile. Obviously, through proper compensation of motor armature reaction, the electromagnetic torque of the motor can be precisely controlled through the armature current. A close loop control of motor armature current is developed and the reference current is calculated from the reference torque T_M^*.

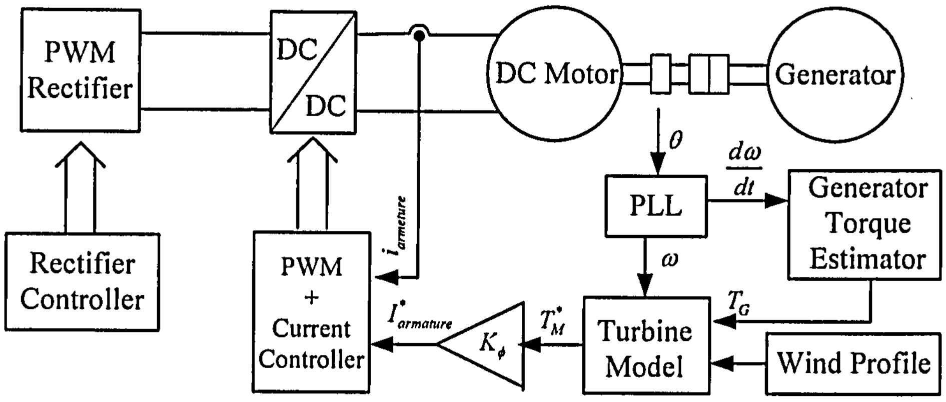


Figure 3-1: Block diagram of Wind Turbine Simulator

3.2 Model of Wind Turbine Simulator

3.2.1 Wind Turbine Aerodynamic Characteristic

Eq3-1 is the turbine aerodynamic characteristic based on the performance coefficient c_P, air density ρ (kg/m3), turbine swept area A (m2), wind velocity v_{wind} (m/s). The performance coefficient is actually a function of tip speed ratio λ and blade pitch angle β (deg). In this equation, the stiffness of the drive train is assumed to be infinite.

$$P_m = c_P(\lambda,\beta)\frac{\rho A}{2}v_{wind}^3 \qquad (3\text{-}1)$$

where P_m is the mechanical output power of the turbine (W)

Eq. 3-1 can be normalized in the per unit (pu) system in Eq. 3-2.

$$P_{m\_pu} = k_p c_{p\_pu} v_{wind\_pu}^3 \qquad (3\text{-}2)$$

where P_{m_pu} is output power in pu for particular values of ρ and A. c_{p_pu} is performance coefficient in pu and v_{wind_pu} is the based wind speed. The base wind speed is usually selected as the annual average expected wind speed in m/s.

A generic equation is used to model performance coefficient $c_P(\lambda,\beta)$. This equation, based on [13], is:

$$c_p(\lambda,\beta) = c_1\left(\frac{c_2}{\lambda_i} - c_3\beta - c_4\right)e^{\frac{-c_5}{\lambda_i}} + c_6\lambda \tag{3-3}$$

with $\frac{1}{\lambda_i} = \frac{1}{\lambda + 0.08\beta} - \frac{0.035}{\beta^3 + 1}$

The coefficients c1 to c6 are: c1 = 0.5176, c2 = 116, c3 = 0.4, c4 = 5, c5 = 21 and c6 = 0.0068. The cp-λ characteristics, for different values of the pitch angle β, are illustrated in Figure 3-2. The maximum value of c_p ($c_{p\_\max}$ = 0.48) is achieved for β = 0 degree and for λ = 8.1. This particular value of λ is defined as the nominal value (λ_nom).

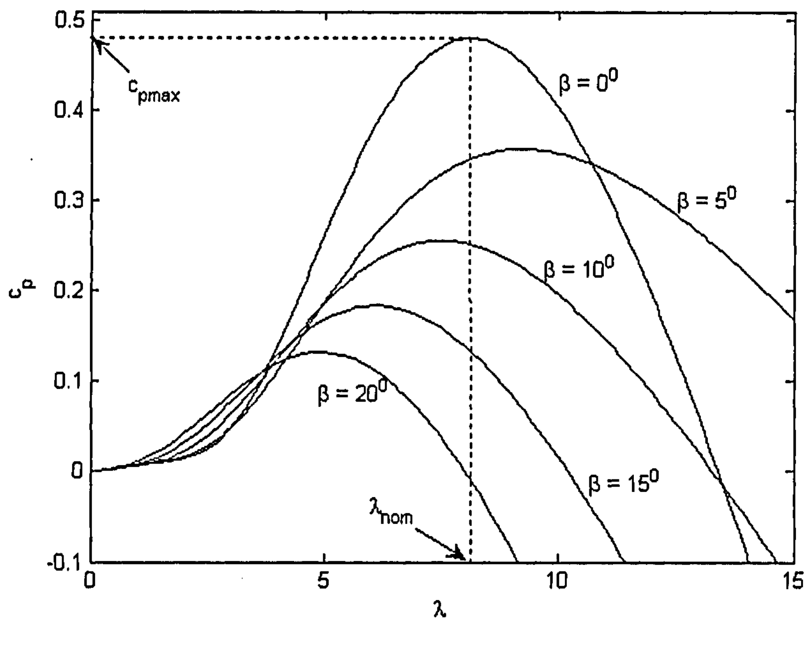


Figure3-2: The cp-λ characteristics of Wind Turbine

Figure 3-3 shows a typical wind turbine power curves with various wind speed and turbine rotor speed. The mechanical power Pm is a function of generator speed, for different wind speeds and fixed blade pitch angle (0 degree). This figure is obtained with the default parameters (base wind speed = 12 m/s, maximum power at base wind speed = 0.73 pu (k_p = 0.73) and base rotational speed = 1.2 pu.

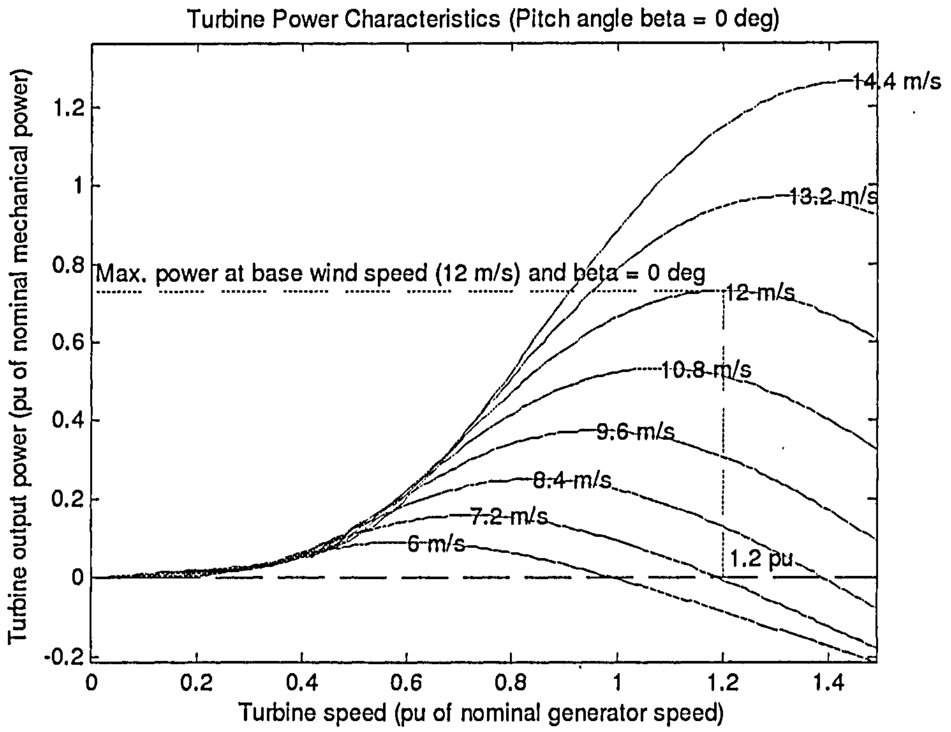


Figure 3-3: Wind Turbine Power Characteristics

In steady state, the aerodynamic turbine rotor torque T_W can be expressed as,

$$T_W = \frac{P_m}{\omega_{rotor}} \tag{3-4}$$

where ω_{rotor} is the turbine rotor speed

3.2.2 Dynamic Model Considering Turbine Moment of Inertia

The dynamic model with turbine inertia is determined by equating the generator acceleration in the field and lab systems. The effect is to alter the turbine torque that the motor will produce in response to a given wind, such that the effect of the turbine rotor inertia is compensated. The model can be written by Eq. 3-5.

$$\frac{T_M - T_F - T_G}{J_M + J_G} = \frac{T_W - T_G}{J_T + J_G} \tag{3-5}$$

where T_M is output torque of DC motor, T_F is the equivalent torque of total mechanical losses of DC motor, T_G is the generator side torque and T_W is aerodynamic turbine rotor torque. J_M is the moment of inertia of DC motor, J_G is the inertia of generator and J_T is the inertia of field turbine rotor. Eq. 3-4 can be rewritten to determine the reference torque (3-6) required for the DC motor.

$$T_M = \frac{J_M + J_G}{J_T + J_G} T_W + \frac{J_T - J_M}{J_T + J_G} T_G + T_F \tag{3-6}$$

Therefore the wind turbine simulator will accurately represent the field wind turbine system if the driving torque T_M is controlled according to (3-6). T_W can be derived directly from the turbine model. The generator torque T_G needs to be measured or estimated.

3.3 Generator Side Torque Estimation

Assume the generator is directly coupled to the DC motor. Generator side torque can be estimated by Eq. 3-7 according to motion equation of the mechanical system.

$$T_G = T_M - \ddot{\theta} \times (J_M + J_G) \tag{3-7}$$

In Eq. 3-7, $J_M + J_G$ is the total moment of inertia of mechanical system and T_M is the net output torque of the dc motor (friction and other mechanical losses are deducted). θ is the motor or generator position angle, which can be measured by either an incremental encoder or resolver. By using DC motor based system, it would be accurate enough to estimate the motor side electromagnetic torque since the torque is proportional to the current. If the losses are known, T_M can be calculated.

In Eq. 3-7, the acceleration of WTS should be derived. Using direct differentiation of measured position is not practical because the noise will be directly amplified. A two state variables observer is developed in Figure 3-4 to observe the state variables of mechanical system. The observer using 2nd order tracking loop tracks the different of rotor position angle from reference model and the measured rotor position angle. θ_{PLL} is the model output and θ_{in} is the measured rotor position. Since design employs the concept of 2nd order phase-locked-loop in order to avoid solving differential equation. The acceleration can be obtained after the P regulator and angular speed can be calculated from the input of last integrator.

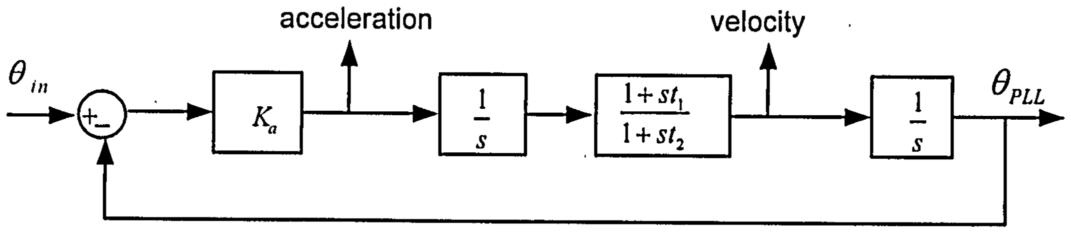


Figure 3-4: 2nd Order Speed and Acceleration Observer

The gain K_a is chosen to get the desired bandwidth of observer. Output K_a gain is the acceleration signal which is also the input of the first integrator. The first integrator outputs the angular speed. In the observer, the lead compensator contains a pole and a zero, which is used to provide necessary phase margin and to reduce the gain of high frequency noise. The second integrator generates reference position angle. The open loop transfer function of observer can be expressed in Eq. 3-8:

$$G(s) = \frac{K_a}{s^2} \times \frac{1+st_1}{1+st_2} \qquad (3\text{-}8)$$

Thus the closed-loop transfer function can be derived in Eq. 3-9:

$$H(s) = \frac{G(s)}{1+G(s)} \qquad (3\text{-}9)$$

Bode plots of $G(s)$ and $H(s)$ for 2kW prototype system are shown in Figure 3-5 and Figure 3-6 respectively. Parameters are chosen to provide appropriate phase margin at selected bandwidth. Then the close loop Bode plots looks like a second-order low-pass filter with corner frequency equal to the bandwidth. The bandwidth should be properly selected to reflect the dynamic torque of turbine system and also as low as possible to suppress the digitalization noise from the encoder.

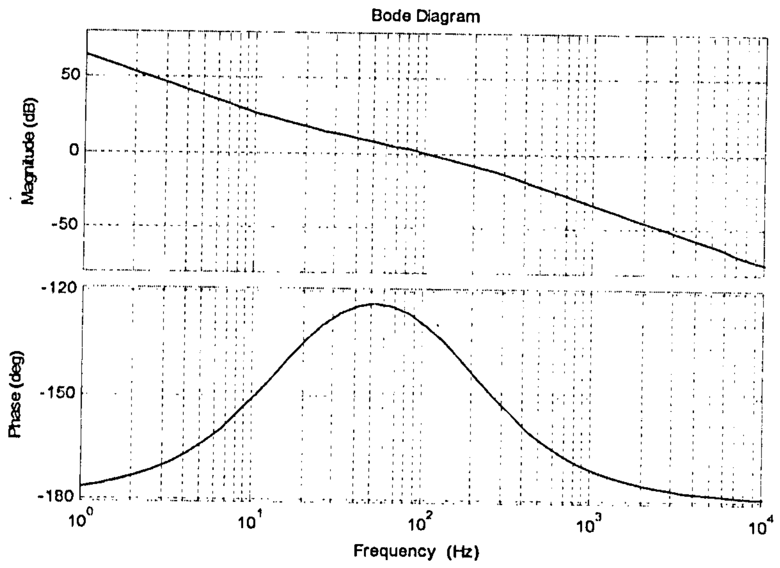


Figure 3-5: Bode plot of observer open loop gain $G(s)$

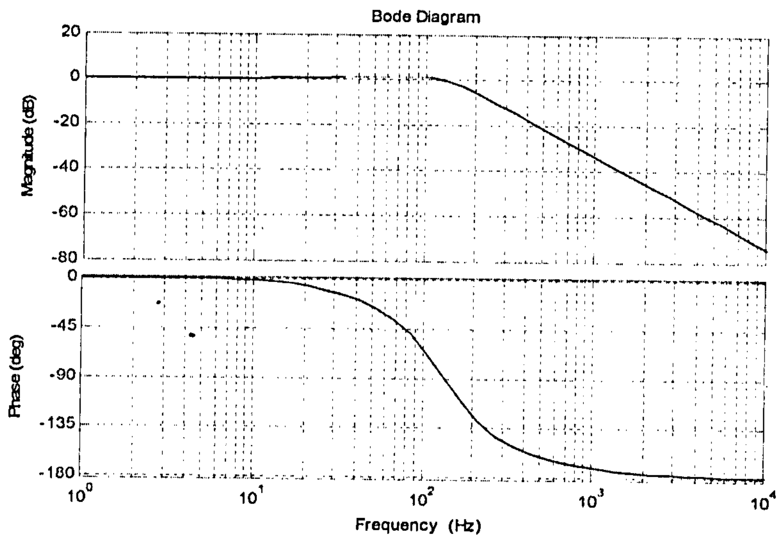


Figure 3-6: Bode plot of observer close loop gain $H(s)$

In order to implement the observer algorithm into DSP, the s-domain transfer functions are converted into z-plane by using bilinear transformation. The observer in z-domain is shown in Figure 3-7.

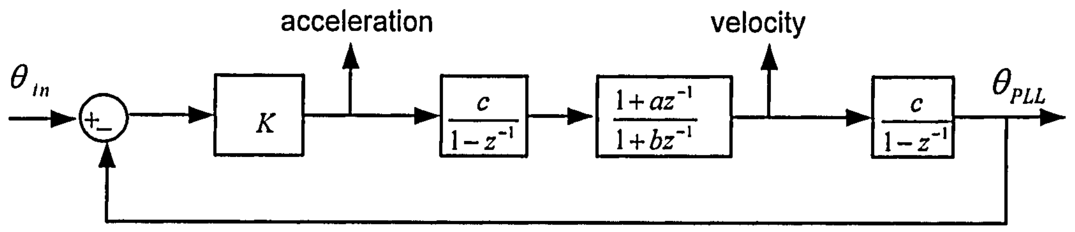


Figure 3-7: The observer in z-domain

The parameters can be selected in Table 3-1

Table 3-1: parameters of observer

Ka	t_1	t_2	f_s	K	a	b	c
74000	0.01	9e-4	9000	$K_a\frac{1-b}{1-a}$	$\frac{0.02f_s-1}{0.02f_s+1}$	$\frac{1.8e-3f_s-1}{1.8e-3f_s+1}$	$\frac{1}{f_s}$

It should be note that the resolution of the speed sensor has significant impact of the observed acceleration because of the big loop gain K_a. In this project a 3600 pulse encoder is employed and θ_{in} is the naturally digitalized by the encoder, while the observer implemented in the DSP has a different sampling frequency of the encoder output. Sampling of encoder output is asynchronous, which could happen at any point between two encoder pulses. The θ_{in} is not updated by the encoder, which is still the previous value. Therefore digitalization error exists and results in big noise in observed acceleration. Using ultra-high resolution encoder or analog position resolver with high speed A/D converter could be possible solutions to this problem but they are costly. In this thesis, an external interpolation method is adopted by measuring the time between last pulse and current sampling, the current position angle can be estimated by the speed and previous non-updated position. The error can be minimized and digitalization noise can be compensated.

3.4 DC Motor Drive

The equivalent circuit of DC motor drive is shown in Figure 3-8. A four quadrant chopper is employed to drive the DC motor in four quadrant operation. Unipolar PWM scheme is used to reduce the current ripple. Then chopper output voltage V_t can be expressed as a function of duty cycle (D) and input voltage (V_d) in Eq. 3-10.

$$V_t = (2D-1)V_d \tag{3-10}$$

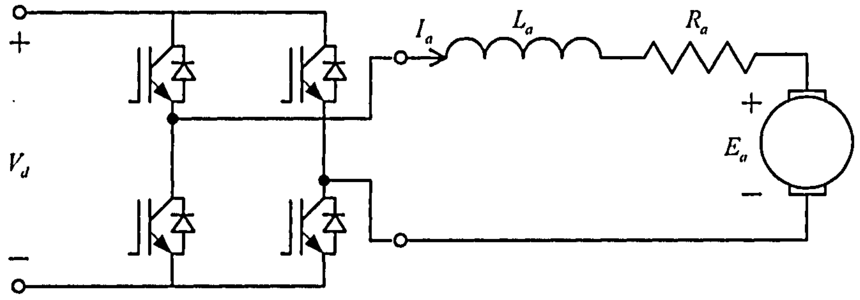


Figure 3-8: Equivalent Circuit of DC Motor Drive

In order to precisely control the electromagnetic torque, a current control loop is developed to track the reference current. The block diagram of the control system in s-domain is shown in Figure 3-9. The armature current is sampled and the control algorithms are executed twice per carrier waveform period, which is similar to control of PWM rectifier. So processing delay and PWM block can be simplified as first-order delays shown in Eq. 3-11 and Eq. 3-12 respectively, where T_s is sampling period and K_{PWM} is the gain of chopper.

$$G_{prs} = \frac{1}{1+sT_s} \tag{3-11}$$

$$G_{PWM} = \frac{K_{PWM}}{1+s(T_s/2)} \tag{3-12}$$

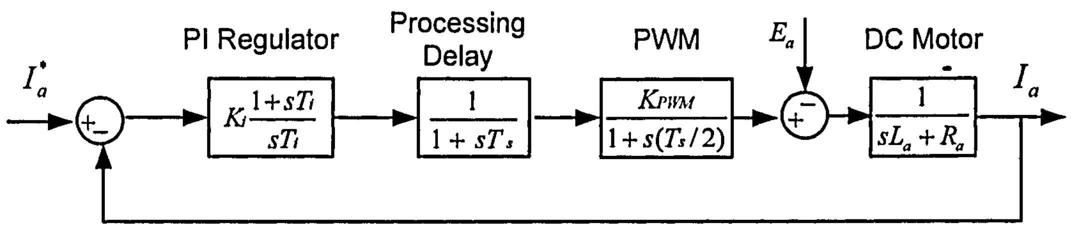


Figure 3-9: Block Diagram of Armature Current Control

PI regulator is designed to be as fast as possible to track with the reference current I_a^*, which is proportional to desired torque of the simulator. Bode plot of the open loop gain of above system is shown in Figure 3-10. The bandwidth is selected to be 443Hz for the prototype system, which is high enough for most of the turbine simulators.

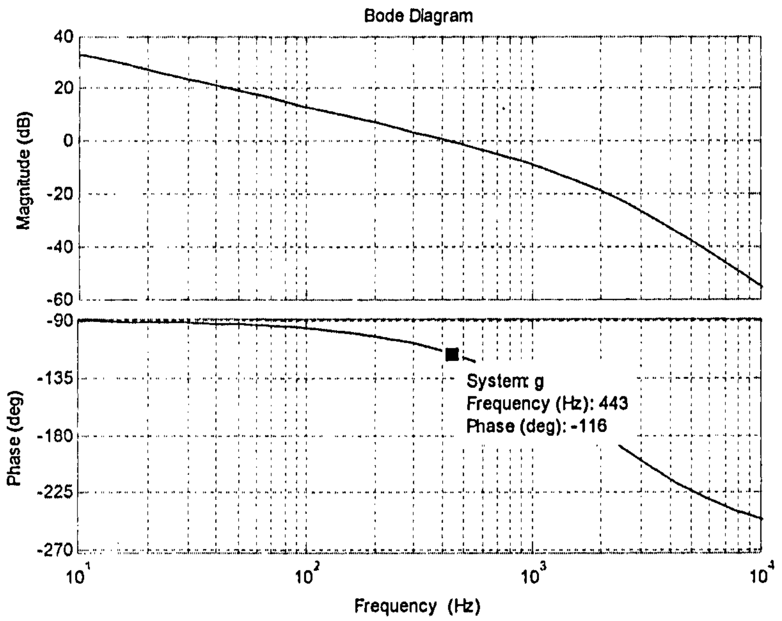


Figure 3-10: Bode plot of Armature Current Control

3.5 Measurement of Moment of Inertia and Mechanical Loss

In Eq. 3-7, the accuracy of the total simulator system inertia ($J_M + J_G$) is very essential in generator side torque estimation. The total inertia varies with different tested generator and coupling. Measurement should be developed either on-line or off-line. In this thesis, traditional dual-slope method is employed to obtain the system inertia.

The motion equation of the simulator system including friction loss can be written as below.

$$T_M - T_G - (T_f + B_m \omega) = \frac{d\omega}{dt}(J_M + J_G) \tag{3-13}$$

where T_f is the coulomb friction torque, B_m is the viscous friction coefficient and ω is the rotating speed of the motor.

Before the measurement, the generator is disconnected from the load (or grid). T_G in Eq. 3-13 is zero. First, apply constant T_M (constant armature current) to the simulator system. Then the motor coupled with generator will free accelerate. At certain speed, power supply to the DC motor is turned off. The motor-generator will free decelerate due to the friction loss. There are two parts in this process: acceleration and deceleration which are illustrated in Figure 3.11. During specified speed range from ω_1 to ω_2, the time is ΔT_1 and ΔT_2 for acceleration process and deceleration process respectively.

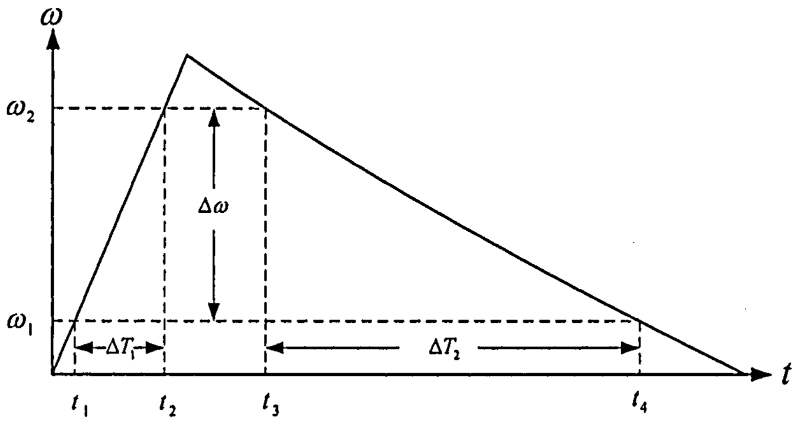


Figure 3-11: Acceleration and deceleration process

During acceleration, Eq. 3-13 can be rewritten in Eq. 3-14:

$$T_M - (T_f + B_m\omega) = \frac{d\omega}{dt}(J_M + J_G) \tag{3-14}$$

Apply integral from t1 to t2 to both sides of Eq. 3-14, we get Eq. 3-15.

$$\begin{aligned}
&\int_{t1}^{t2}(T_M - (T_f + B_m\omega))dt = \int_{t1}^{t2}\frac{d\omega}{dt}(J_M + J_G)dt \\
&\Rightarrow T_M(t_2 - t_1) - \int_{t1}^{t2}(T_f + B_m\omega)dt = \Delta\omega(J_M + J_G) \\
&\Rightarrow T_M\Delta T_1 - (T_f + B_m\frac{\Delta\omega}{2})\Delta T_1 \approx \Delta\omega(J_M + J_G)
\end{aligned} \tag{3-15}$$

Similarly, Eq. 3-13 can be derived into Eq. 3-16 for deceleration.

$$\begin{aligned}
&\int_{t3}^{t4} -(T_f + B_m\omega)dt = \int_{t3}^{t4}\frac{d\omega}{dt}(J_M + J_G)dt \\
&\Rightarrow (T_f + B_m\frac{\Delta\omega}{2})\Delta T_2 \approx \Delta\omega(J_M + J_G) \\
&\Rightarrow T_f + B_m\frac{\Delta\omega}{2} = \frac{\Delta\omega(J_M + J_G)}{\Delta T_2}
\end{aligned} \tag{3-16}$$

Solve (3-15) and (3-17), the inertia can be obtained.

$$J_M + J_G = \frac{T_M}{\Delta\omega} \times \frac{\Delta T_1 \times \Delta T_2}{\Delta T_1 + \Delta T_2} \tag{3-17}$$

Mechanical losses including friction loss of DC machine usually are high and can not be neglected. Measurement of mechanical losses can be done by driving the motor-generator at different speed without load. In steady state, motor torque which is equal to mechanical loss torque is recorded into a table for future use.

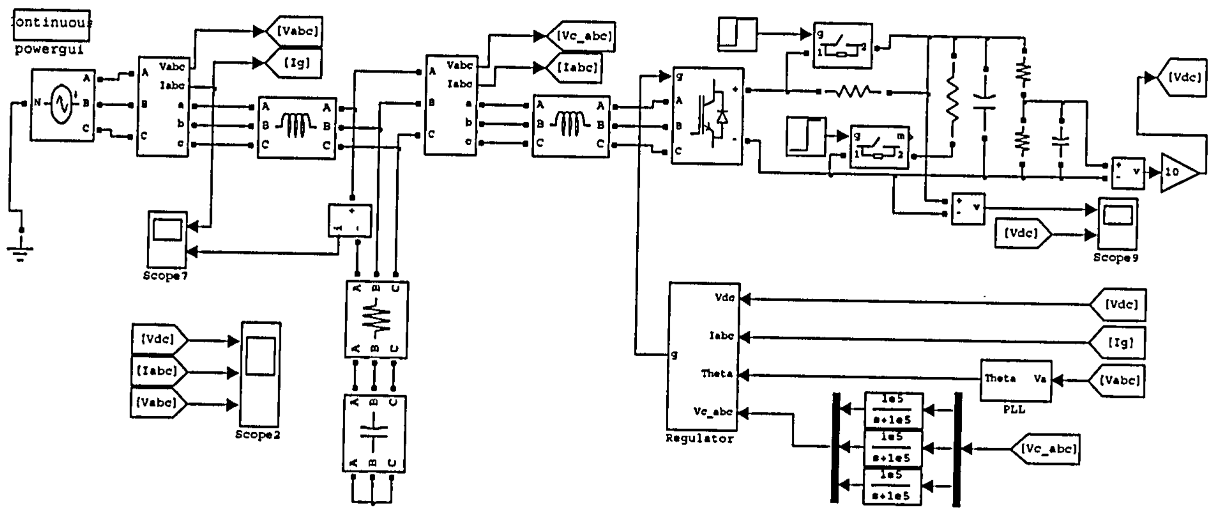
3.6 Conclusion

In this chapter, the simulator is developed based on the aerodynamic characteristic of wind turbine. In order to tracking the dynamic behavior of field turbine generator system, the difference of system inertias is properly compensated. A novel torque sensorless method is developed based on the mechanical system observer, in which the acceleration and speed is observed using reference model. The accuracy of the generator side torque greatly depends on the parameters of mechanical system. The total moment of inertia and friction loss are measure offline on the simulator system. With proposed strategy, the real-time wind turbine simulator can simulate a real wind turbine performance both in steady state and dynamic procedures.

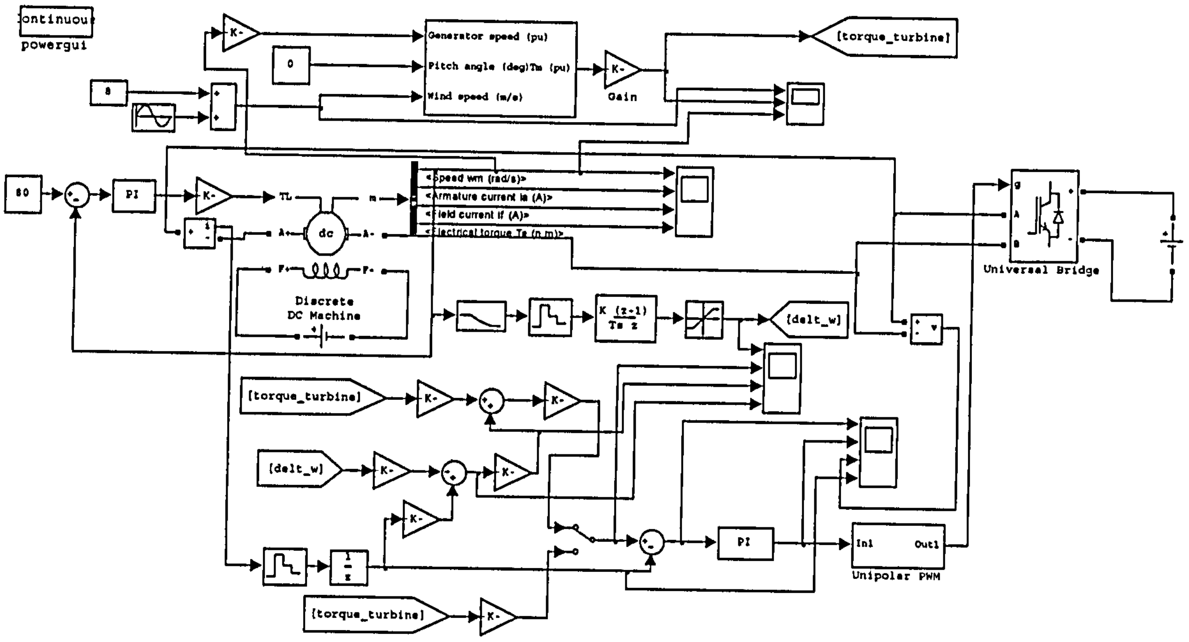
Chapter 4 Simulation and Experimental Results

4.1 Introduction

Both simulation models and experimental setup were built for the verification of the proposed wind turbine simulator. In the simulation, the whole simulator is separated into two parts: PWM rectifier with its control and wind turbine simulator. In the simulation of PWM Rectifier, a resistance load is employed and during simulation of WTS, a constant DC voltage source is placed instead of complex rectifier. The arrangement has little effect to the simulation result since the DC capacitance is large enough and almost fully decouples the two subsystems as shown in Figure 4-1(a) and Figure 4-1(b). It should be noted that in the real system the delays in the control loop is quite common and can be caused by computation time of the digital controller, PWM generation and low pass filters on the feedback signals. These delays have significant effect on the control system, since the PWM is running at low frequency, and close to resonant frequency. Therefore all the control blocks in simulation models are digitized, sampling and calculation delays are added into the control loop to reproduce the real digital control system. All the models were developed under Simulink/MATLAB, which are illustrated in Appendix A.



(a) Simulink model of PWM rectifier



(b) Simulink model of WTS

Figure 4-1: Simulink model of proposed WTS

As shown in Figure 4-2, the experimental set-up consists of a four quadrant power converter, LCL filter, isolation transformer, two DC machines and a DSP-FPGA based digital controller. The multipurpose back-to-back power converter was designed and built for several projects. In this project, one side of the converter works as PWM rectifier while the other side functions as a DC motor drive. Two DC machines are mechanically coupled together. One of them is operated as a motor driven by the converter and another one, as a generator connected to either a resister bank or a controlled converter. The Texas Instruments TMS320F2812 DSP and Altera's Cyclone FPGA were employed as the core of the digital controller. The grid synchronization signal is obtained by the full digital PLL circuit [14] which is embedded in the FPGA. Some experimental variables, such as dq components current or rotor speed ω, are captured from DSP internal memory since these variables are not directly available on the oscilloscopes.

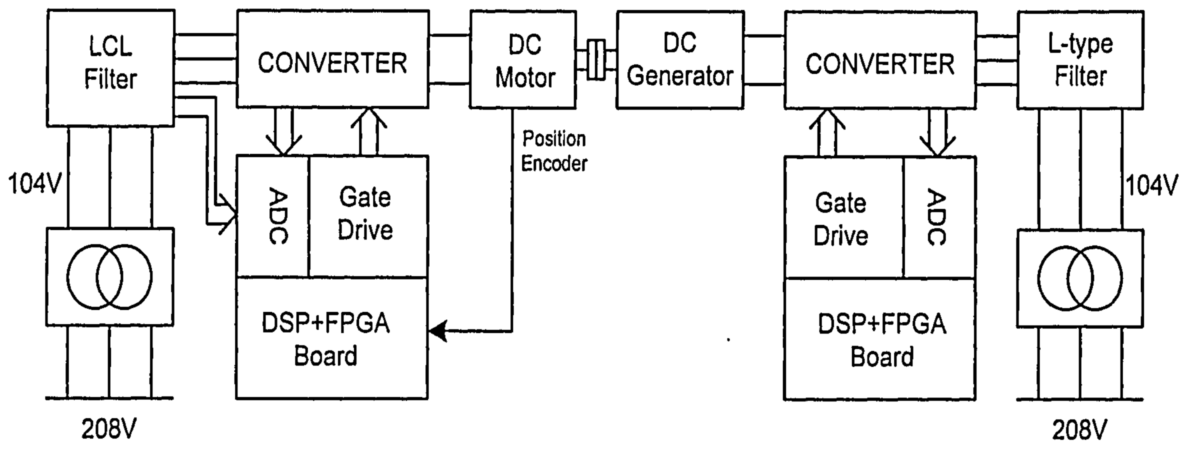


Figure 4-2: Block diagram of experimental set-up

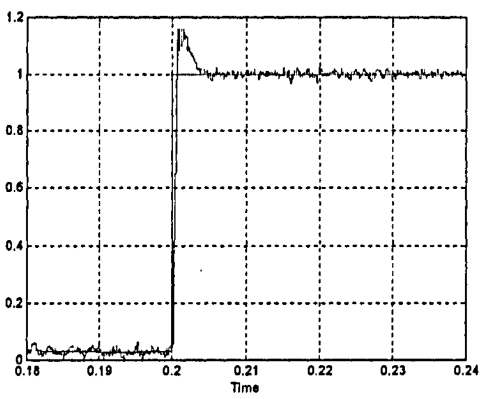
4.2 Three-Phase PWM Rectifier with LCL-filter

The major purpose of the PWM rectifier is to provide a constant DC source to the turbine simulator no matter the load changes or grid voltage changes. So various simulation and experiments under different conditions, including current reference step response, grid voltage sag and load step change, were conducted to explore the performance of the rectifier system. In order to give a fair comparison for verification purpose, the parameters used for the simulation were selected as same as that of the experimental system. Different sensor positions were also investigated in the both simulation and experiment.

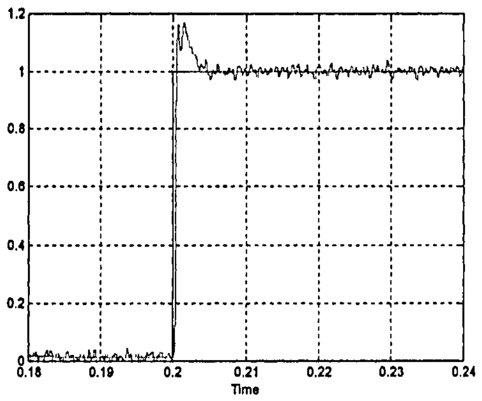
4.2.1 Step Response of dq component currents

Simulated responses of i_d (d-component of sensed current) during a reference current step change are shown in Figure 4-3. Response with converter-side current control in (a) is almost same to that with grid-side current control in (b). This is because of the same bandwidths and same phase margin, which result in same rise time and same overshoot in the step response. System is stable and no resonant due to the active damping which effective suppress the resonant peak.

Experiment results shown in Figure 4-4 closely verified the simulation results, from which the consistency were kept between the real system and simulation models.

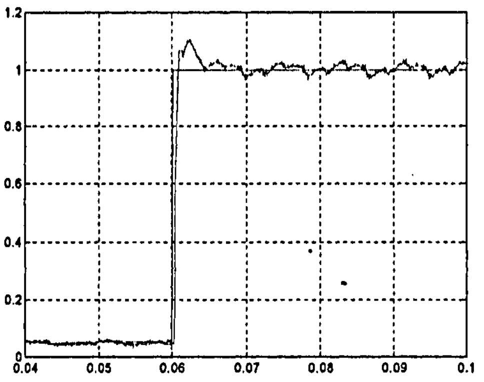


(a) Converter-side current control

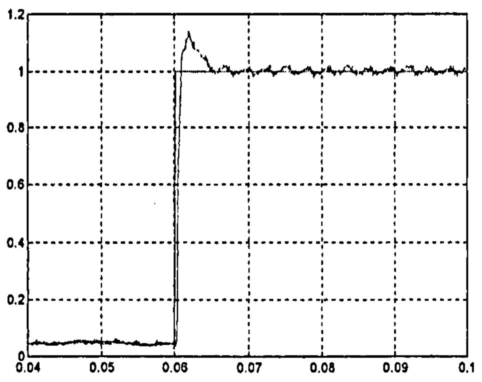


(b) Grid-side current control

Figure 4-3: Simulated d-component current step response



(a) Converter-side current control



(b) Grid-side current control

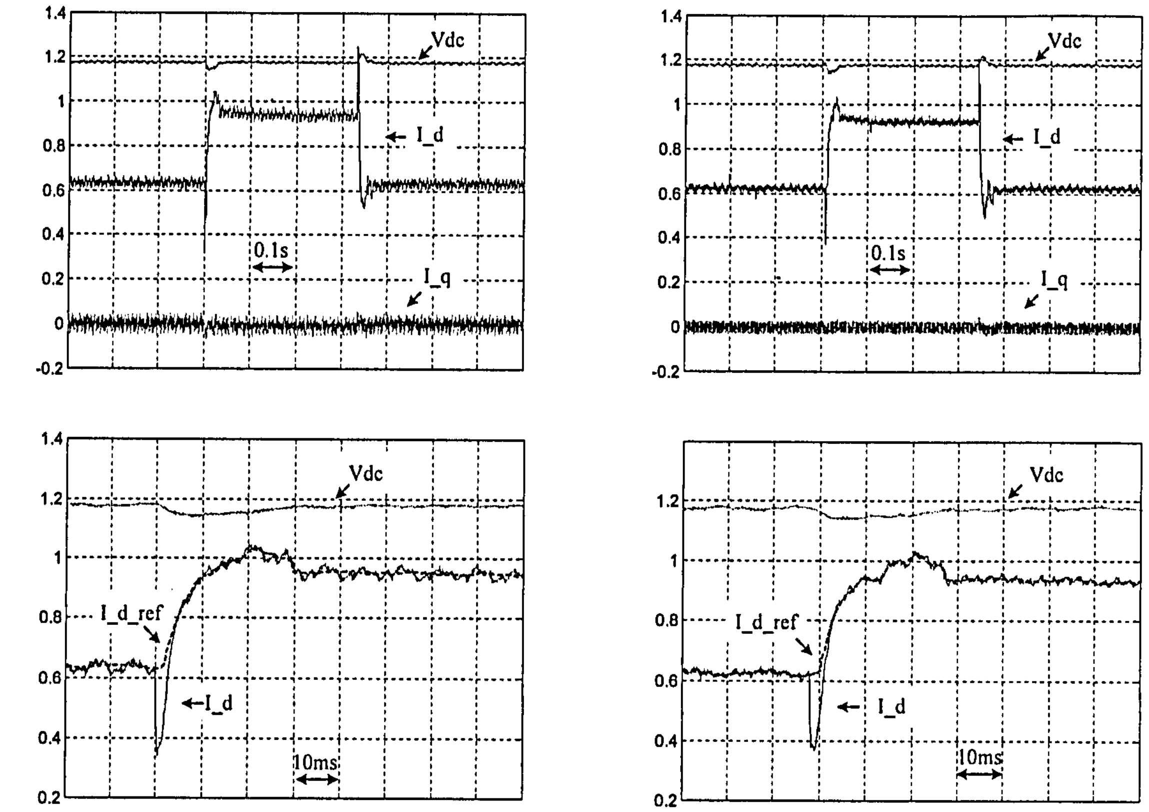
Figure 4-4: Experimental d-component current step response

4.2.2 Grid voltage sag

A programmable power supply was connected and a 43% voltage sag was programmed and supplied to the PWM. The ability of the controller to reject disturbances and its ability to damp the resonance from excitation by external sources were demonstrated in this test. Figure 4-5 and Figure 4-6 show the transient response of the system.

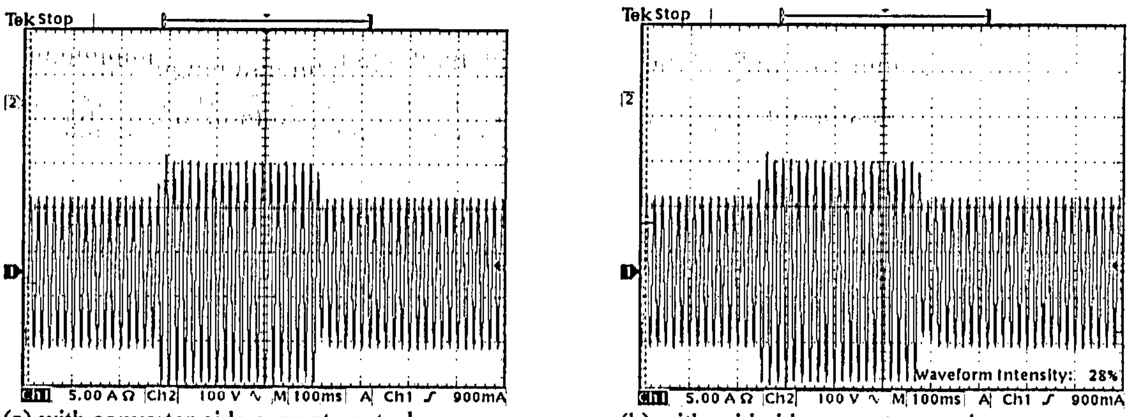
There are two transient responses during grid voltage sag: voltage step down and step up. Each transient response can be approximately divided into two processes: current control dominated process and voltage control dominated process. In Figure 4-5, the captured data from DSP clearly shows the transient procedures during voltage sag. In the time expanded figure in Figure 4-5, it shows that transient response begins with current control dominated process. The current reference remains unchanged but the real current drops suddenly due to the voltage disturbance and then is regulated to be equal to the reference value within 3 ms by current control regulator. Next process is voltage control dominated process during which the current reference is changed due to the action of voltage control loop. Current control dominated process is much shorter than voltage control dominant process and the two cascaded control loops can be considered to be decoupled in the controller design. The response of DC voltage loop is mainly determined by the large DC link capacitor.

The system without active or passive damping will oscillate after the disturbance. The active damping implemented in this system effectively suppresses the resonant and shorten the transient process.



(a) with converter-side current control (b) with grid-side current control

Figure 4-5: dq-axis currents and DC bus voltage during a 43% grid voltage sag



(a) with converter-side current control (b) with grid-side current control

Figure 4-6: Waveforms of grid side voltage (top, 100V/div) and current (bottom, 5A/div) during a 43% grid voltage sag

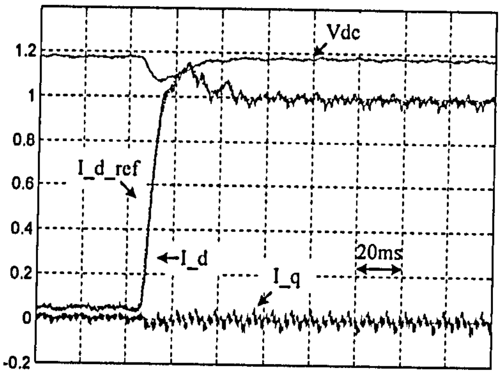
Compare Figure 4-5 (a) with Figure 4-5 (b), very similar responses with converter-side current control and with grid-side current control can be seen. With converter side current control, the sudden change of I_d caused by the disturbance is a slightly bigger than that with grid side current control. This difference is caused by the small difference of the bandwidth of current control loop between these two control schemes. With grid current control, the bandwidth of current control loop is slightly higher.

Figure 4-6 shows the voltage and current waveforms at grid side. The line current increases during voltage sag. With different sensor positions, the responses are very similar. It also should be noted that the dc link voltage has small overshoot/undershoot during the sag. The overshoot/undershoot is around 3%, which has very small effect to the WTS connected on the dc link.

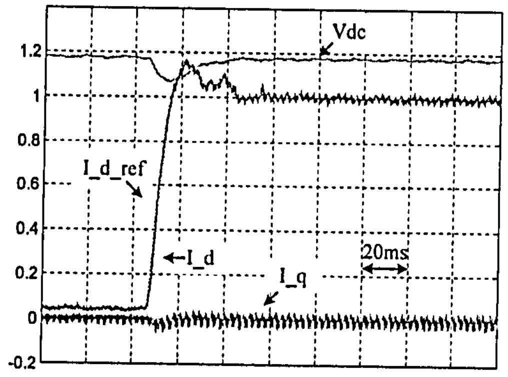
4.2.3 Load Step Change

The performance with load step change is actually determined by the voltage control loop since the dc link capacitor is designed around 4 pu for most of the rectifier system. The current control is much faster than the voltage control The transient responses during a load step change from 0 to 100% are shown in Figure 4-7 and Figure 4-8. The transient responses during a load step change from 100% to 0 are shown in Figure 4-9 and Figure 4-10. Approximately 10% DC voltage changes can be seen during transient and stabled within 30 ms determined by the bandwidth of voltage regulator. In Figure 4-7 and Figure 4-9 d-component current and it's reference current are almost overlapped each other, which means the real current follows the reference very well.

Experimental results also show the same responses with these two different current control schemes.

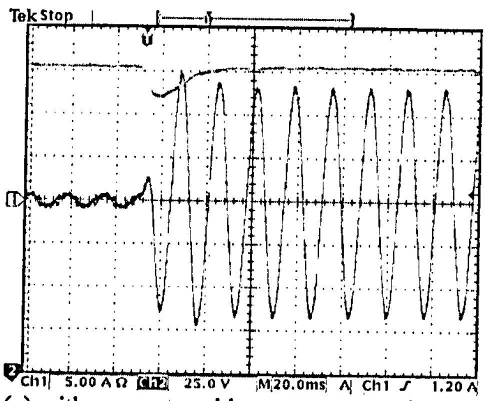


(a) with converter-side current control

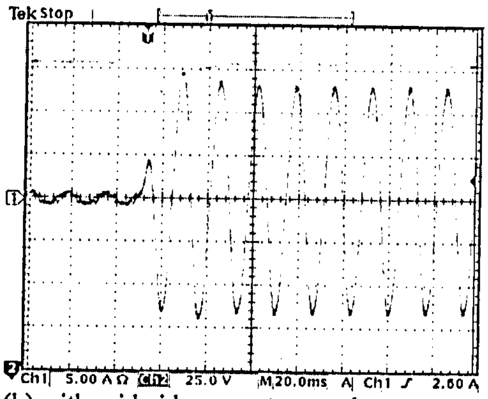


(b) with grid-side current control

Figure 4-7: d and q-axis currents, q-axis current command, and DC bus voltage during a load step change from 0 to 100%

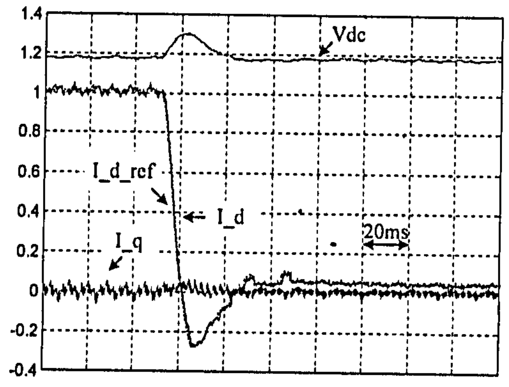


(a) with converter-side current control

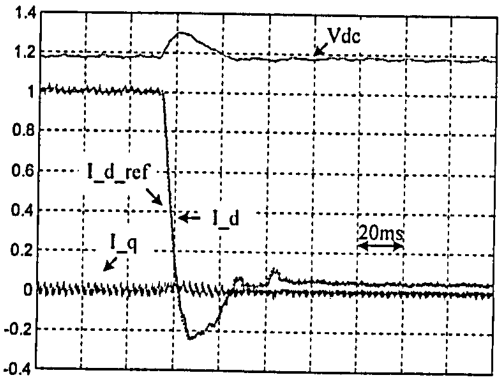


(b) with grid-side current control

Figure 4-8: Waveforms of grid side current and DC bus voltage during a load step change from 0 to 100%

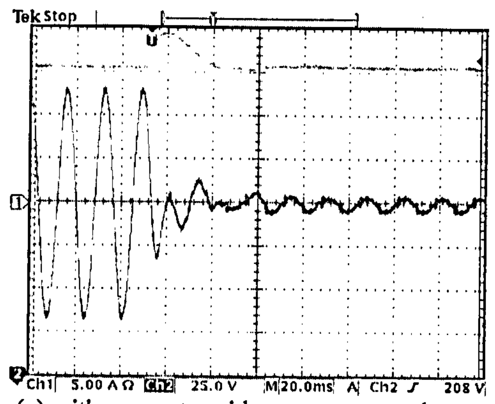


(a) with converter-side current control

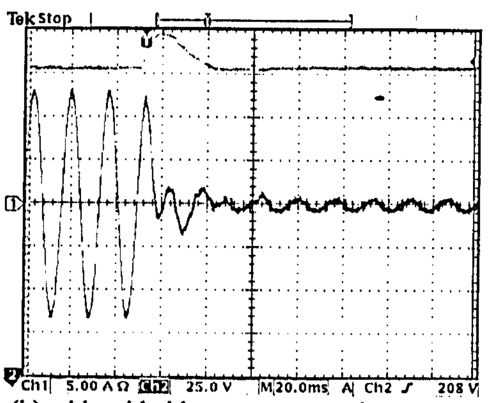


(b) with grid-side current control

Figure 4-9: d and q-axis currents, q-axis current command, and DC bus voltage during a load step change from 100% to 0



(a) with converter-side current control



(b) with grid-side current control

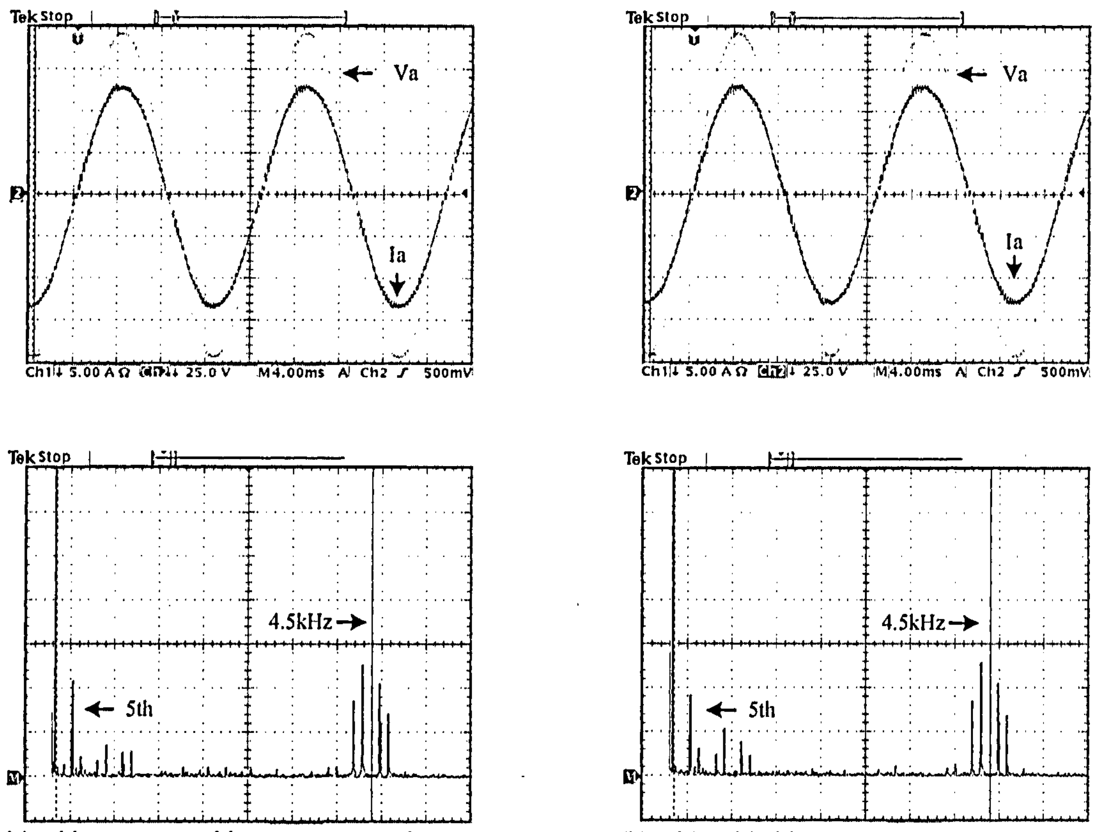
Figure 4-10: Waveforms of grid side current and DC bus voltage during a load step change from 100% to 0

4.2.4 Steady state

Generally, the low total harmonic distortion (THD) of grid-side current and the desired power factor are expected for PWM rectifier.

Grid-side current THD can be divided into two parts: THD of low-order harmonics(less than 50th) and THD of high-order harmonics (around the switching frequency). Because of the bandwidth, current control primarily affects the low-order current harmonics. THD of high-order harmonics is mainly determined by the parameter of LCL filter. If the power supply on the grid side is ideal, the dominant current harmonics are high-order harmonics.

Grid-side current waveforms and FFT of harmonics are shown in Figure 4-11. The dominant current harmonics are high-order harmonics around the switching frequency 4.5 kHz. In addition, 5th order harmonics can be obviously seen in the spectrum. This is primarily caused by the dead band of the switching devices. Grid-side current THD of system with converter-side current control and system with grid-side current control is 3.17% and 3.25% respectively. The LCL filter is very effective to filter the high order harmonics, which has 60db per decade slope compared with 20db/dec L-type filter. With proper compensation of the reactive power, unit power factor can be done on both systems. The grid phase voltage and phase current are in phase in both Figure 4-11(a) and Figure 4-11(b).



(a) with converter-side current control (b) with grid-side current control

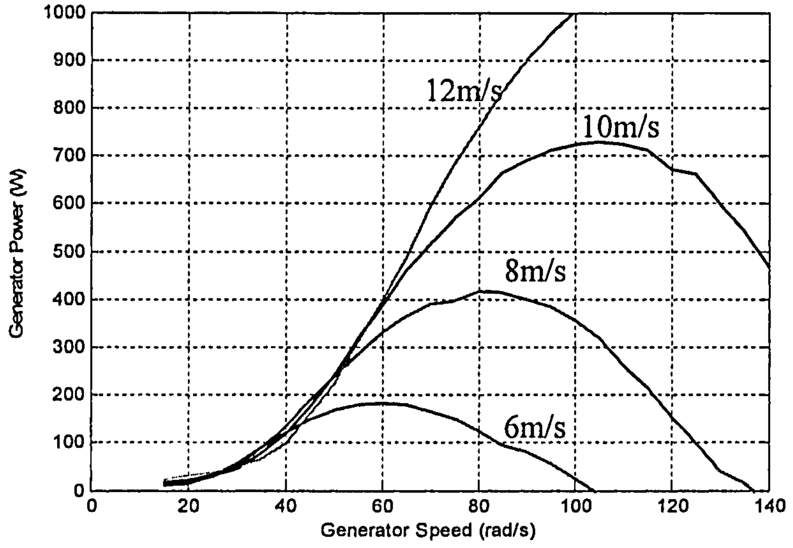
Figure 4-11: Waveforms of grid side current and voltage with spectrum

From above analysis, the designed PWM rectifier with LCL filter effectively provide a constant voltage source to the WTS, which will not be affected grid conditions or power sudden changes. The performance and dynamic simulation of wind turbine can be guaranteed.

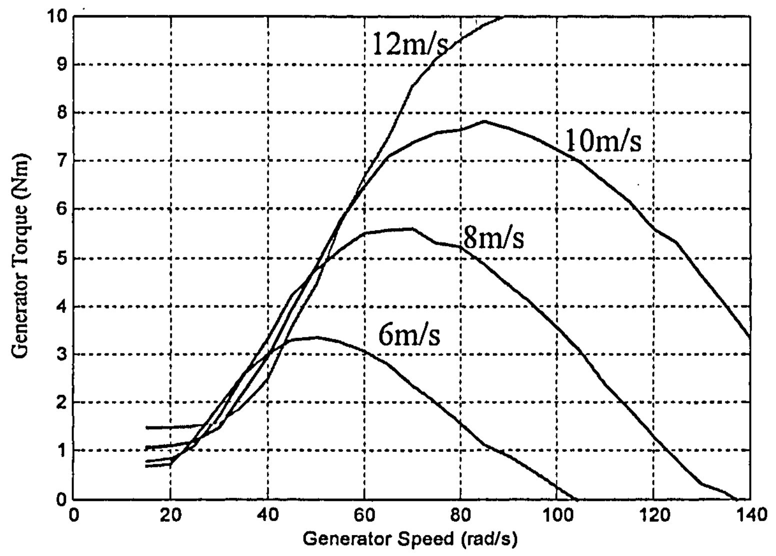
4.3 Wind Turbine Simulator

4.3.1 Steady-state Characteristics

The output power and torque curves of the wind turbine simulator with various wind speed and rotor speed were tested at steady state which is shown in Figure 4-12. The output power and torque were measured from the generator side which is directly coupled with the motor.



(a) Output power of WTS



(b) Output torque of WTS

Figure 4-12: Measured steady-state Characteristics of WTS output

It should be noted that in the Figure 4-12, the power is determined by the motor and converter system. The motor should be selected higher than the rating of generator if the mechanical time constant of simulated turbine is lower the simulator. The prototype system using two DC motors has high loss, which result in low net output power at generator side. The steady-state characteristics measured matches the turbine model in simulator, which verifies that the simulator generates the performance of wind turbine. Actually, the simulator can simulate any mechanical system either as mechanical load or as mechanical driver. The only need to be changed is the model of mechanical load or mechanical drive instead of wind turbine model.

4.3.2 Turbine Dynamics

In order to verify the dynamic simulation caused by difference of simulated turbine inertia and simulator inertia. Two different inertia systems were designed for the test. Assumption of large turbine inertia was considered in the simulator while the inertia of simulator motor-generator system is smaller. In order to verify the algorithm, the experiment was carried with the inertia compensation and without compensation. Rotor speed was measured for comparison purpose.

- Generator Offline

In this test generator is disconnected and offline from the grid. Assume that the turbine outputs a constant torque and when the speed of generator reaches a certain level, the turbine torque is reduced to zero. The generator speed will decrease to zero by friction torque. Figure 4-13 shows the speed traces versus time during test. Trace A is speed waveform of large inertia wind turbine/generator system and trace B is speed waveform of simulator without compensation while trace C is the waveform with inertia compensation. Trace C is very close to trace A, representing the real wind turbine generator.

PROPERTY OF
RYERSON UNIVERSITY LIBRARY

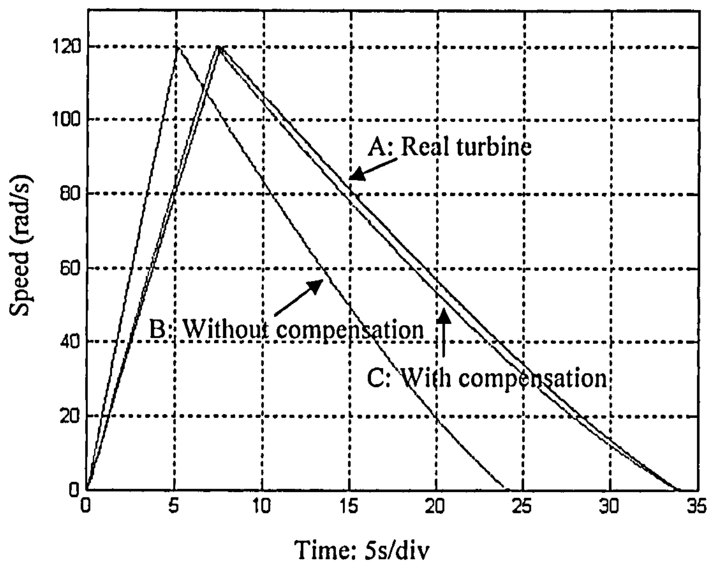


Figure 4-13: Constant electrical torque without load

- Generator Standalone

In this test generator is island operating with fixed resistive load bank. The test procedure is same as the offline test. Speed waveforms shown in Figure 4-14 verify the effectiveness of inertia effect compensation algorithm when load torque exists. The estimated load torque is shown in Figure 4-15. The fluctuation of the estimated load torque is caused by torsional oscillation due to the rubber coupler between motor and generator.

For small inertia wind turbine, inertia compensation algorithm can also get a satisfied result shown in Figure 4-16. Trace A represents the real-turbine system with smaller inertia, and trace B is the speed waveform of simulator without inertia compensation. The speed waveform (trace C) is very close to the real turbine system because of the proper inertia compensation. Although most of the turbine system has pretty large inertia than generator, the small inertia would be possible in the low power system where the generator is usually directly driven and heavily oversized for the low rotating speed.

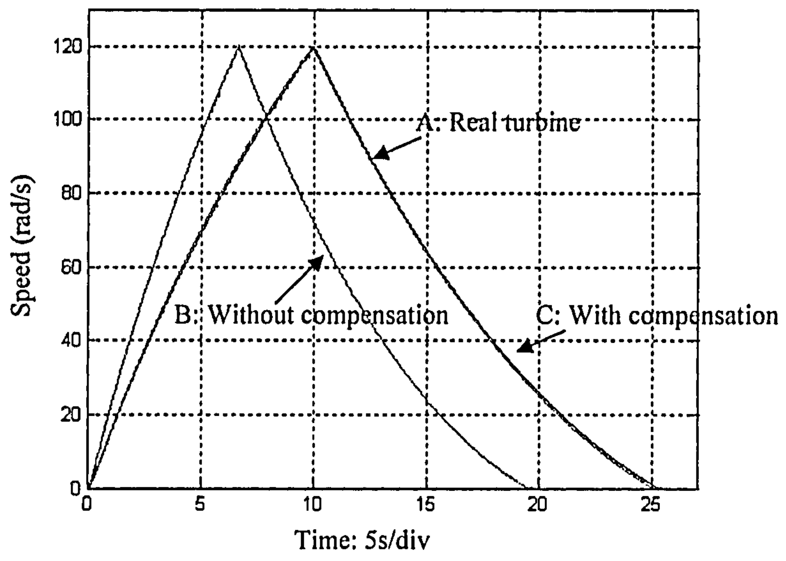


Figure 4-14: Constant electrical torque with load (big turbine inertia)

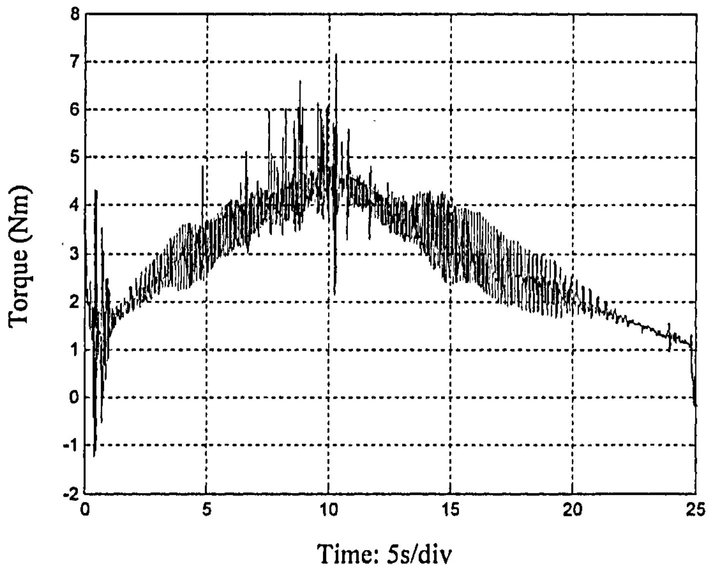


Figure 4-15: Estimated load torque

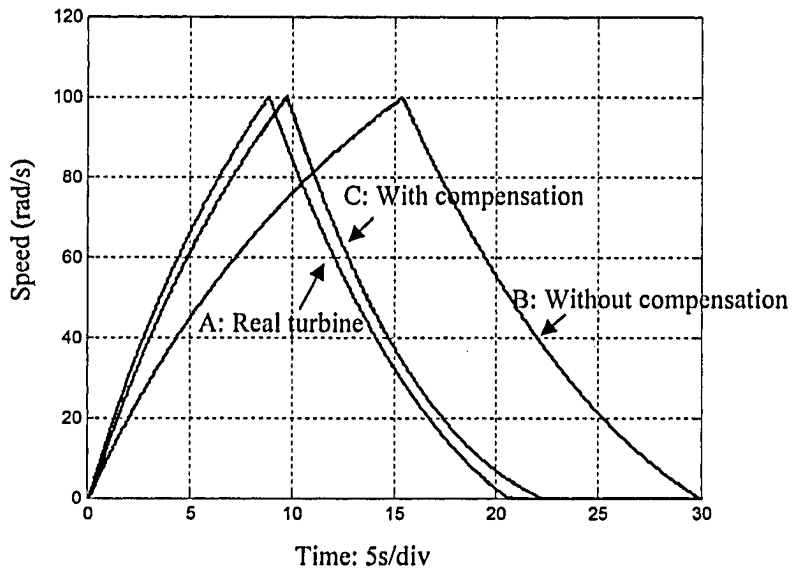
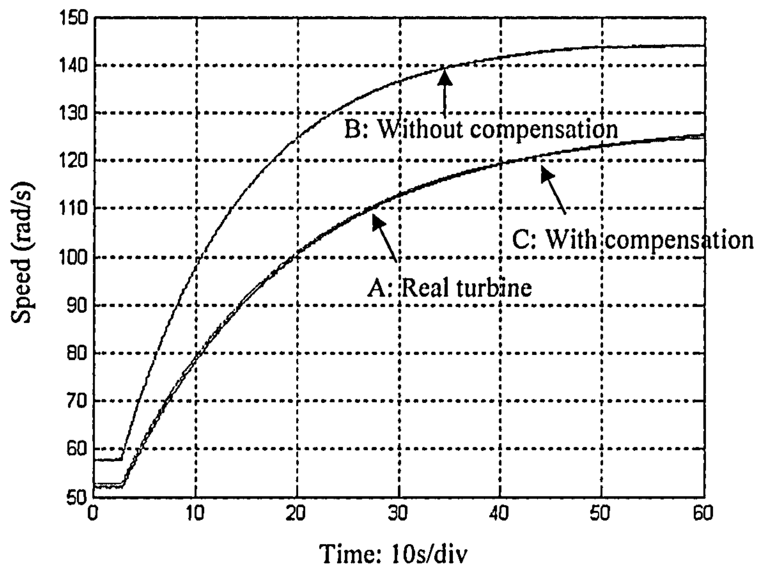


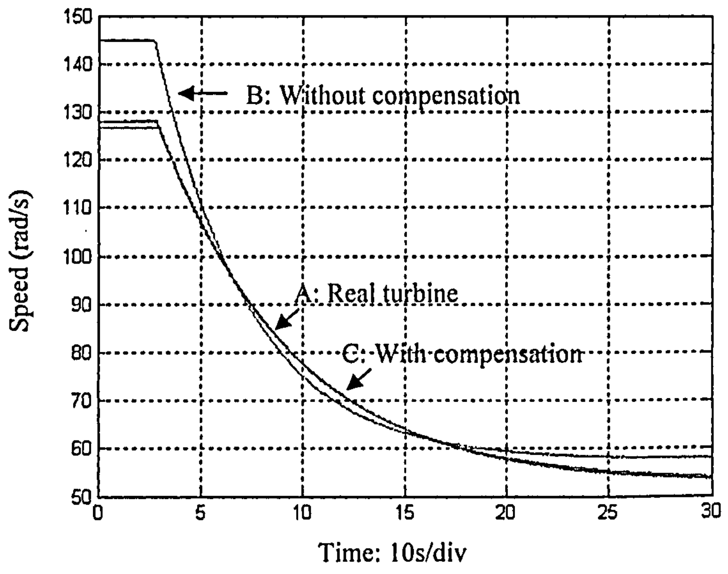
Figure 4-16: Constant electrical torque with load (small turbine inertia)

- Generator Output Step Change

In this test generator was island operating with a resistor load bank. The resistance of load is adjustable to change the load mechanically coupled to the simulator.. Assume that the turbine outputs a constant torque to the generator. With a certain value of resistive load, the system can operate at a certain steady state speed. If the load is suddenly changed, the generator speed will began to increase or decrease until reach to another steady state. The transient responses during this test are shown in Figure 4-17 and Figure 4-18. Test results illustrate that, with inertia compensation, the turbine simulator system can always acts like the turbine system, and no matter the turbine inertia is bigger or smaller than that of simulator system.

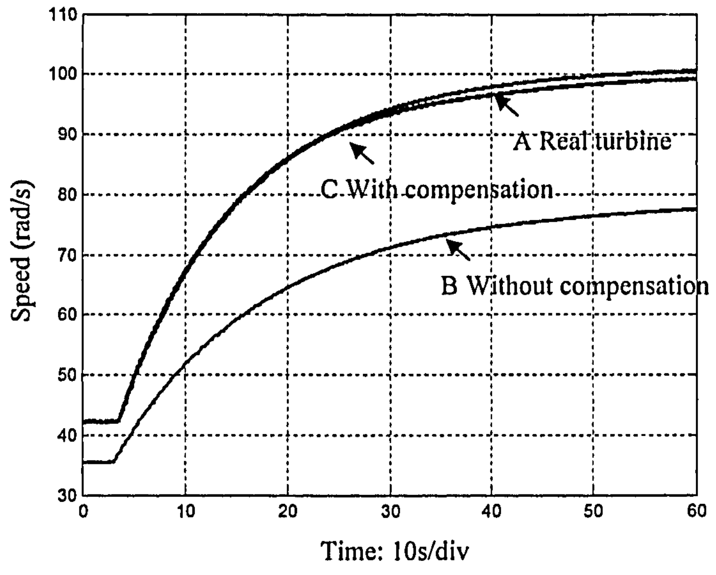


(a) Extra Load is Switched Off

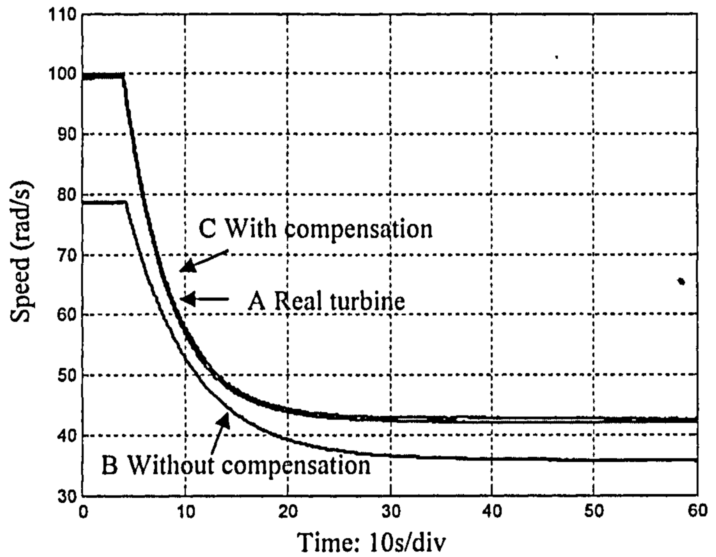


(b) Extra Load is switched On

Figure 4-17: Speed Response with Big Turbine Inertia



(a) Extra Load is switched off

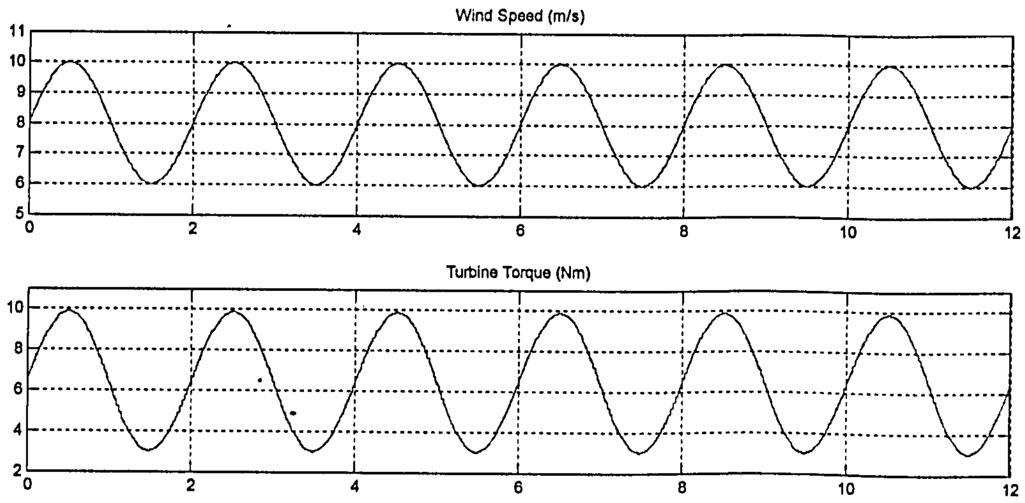


(b) Extra load is switched on

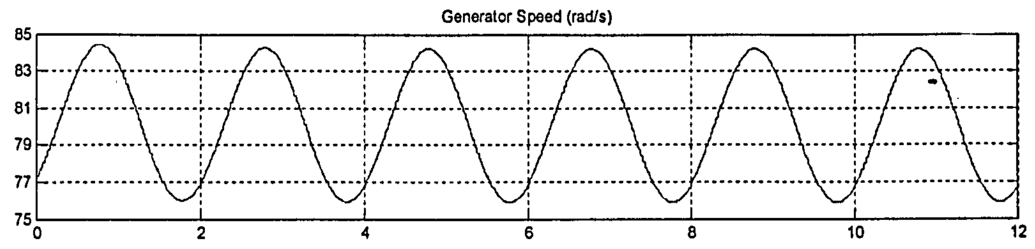
Figure 4-18: Speed Response with Small Turbine Inertia

● Variable Wind Speed

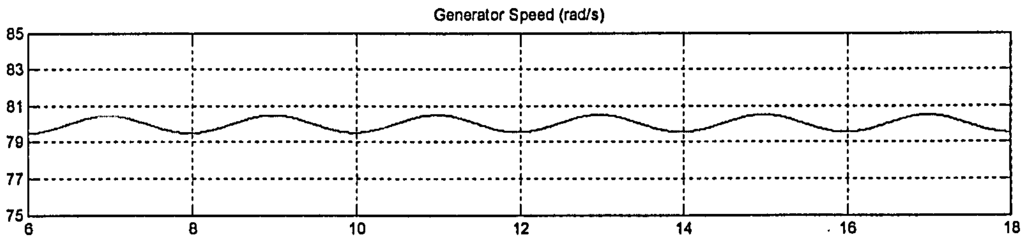
An 0.5Hz and peak-to-peak 4 m/s sinusoidal variables is added to the wind profile. The generator is controlled to operate at a constant speed by a speed regulator. The wind turbine simulator is controlled to simulate a wind turbine system, which has the inertia ten times bigger than the simulator inertia. Simulation results are shown in Figure 4-19. Variable wind speed will result in variable turbine output torque and thus the variable generator speed even the speed is regulated. Without compensate to the turbine inertia, the generator speed varies as shown in Figure 4-19(b). With compensation, simulator will simulate the turbine system with big inertia, resulting in a much smaller ripple of generator speed as shown in Figure 4-19(c). Experimental results shown in Figure 4-20 also verified the simulation results. From Figure 4-19 and Figure 4-20, it shows that the generator side speed controller is designed for large inertia turbine, which is not sufficient to suppress the speed variation in low inertia simulator. But with the inertia compensation, the controller will remain the same and any dynamic response of real wind turbine will be reproduced in real-time.



(a) Small variations in wind speed and variation of turbine output torque

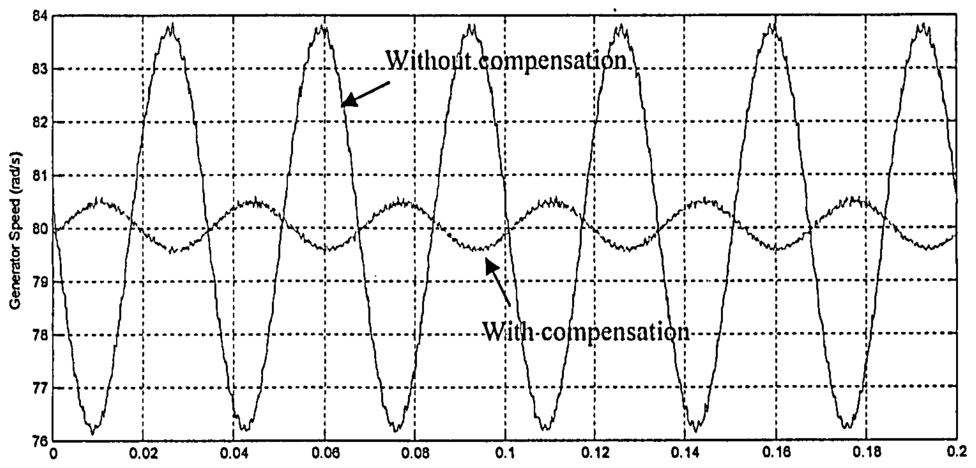


(b) Generator speed without inertia compensation



(c) Generator speed with inertia compensation

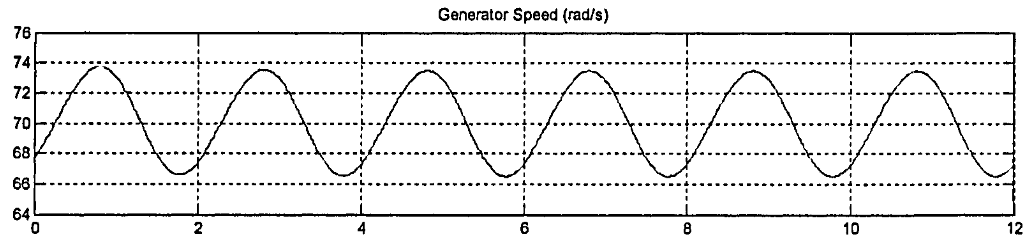
Figure 4-19: Simulation results with big turbine inertia under variable wind speed



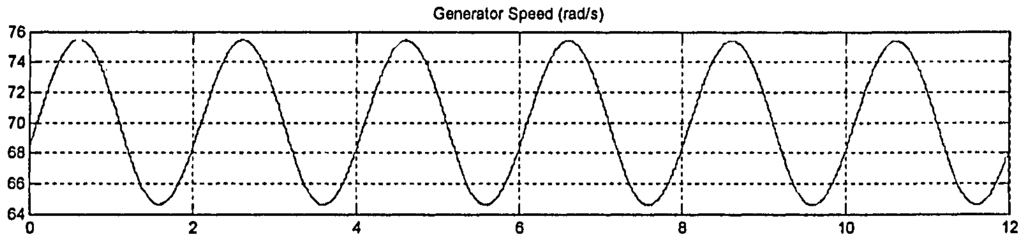
Time: 0.02min/div

Figure 4-20: Experimental Generator Speed with and without Turbine Inertia Compensation

The wind turbine simulator can also simulate a wind turbine system with the smaller inertia Simulation results shown in Figure 4-21 illustrate the generator speed when turbine system inertia is one third of the simulator system inertia. Waveforms (a) and (b) are generator speeds with/without inertia compensation. In this case, the generator speed has large ripple due to small inertia while simulator need to provide extra capacity to absorb or supply the kinetic energy of simulator system, which will speed up the mechanical response limited by mechanical time constant. Experimental results shown in Figure 4-22 also verified the simulation results.



(a) Generator speed without turbine inertia compensation



(b) Generator speed with turbine inertia compensation

Figure 4-21: Simulation results with small turbine inertia under variable wind speed

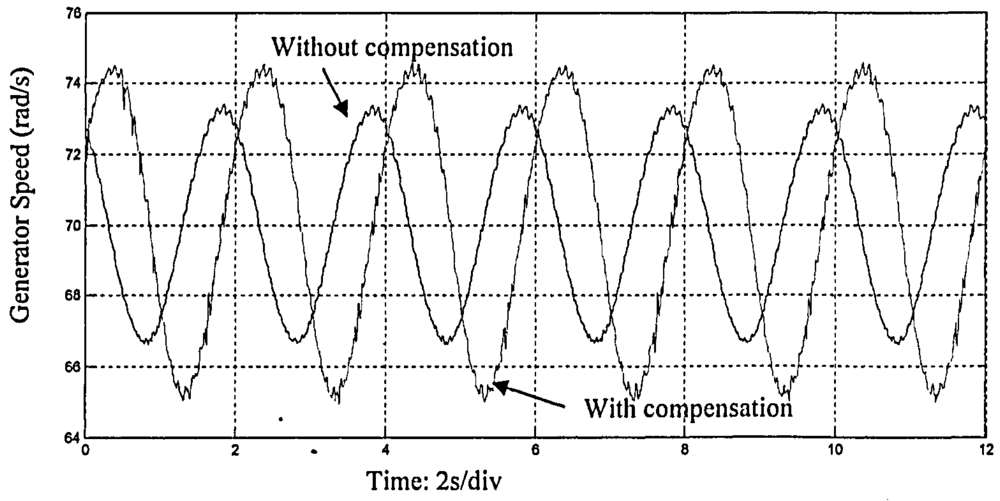


Figure 4-22: Experimental generator speed with small turbine inertia under variable wind speed

4.4 Conclusion

A real time wind turbine simulator was modeled and constructed to reproduce the dynamic behavior of the real turbine system with different moment of inertia. The WTS is capable of simulating a wind turbine system with big or small inertia with the help of bidirectional converters, including PWM rectifier and four-quadrant DC/DC chopper. The generator torque estimation algorithm without torque transducer developed in Chapter 3 is successfully verified by comparing the speed responses of simulator and turbine. PWM rectifier acting as constant DC power supply provides a stable operation for the wind turbine simulator. The energy bidirectional design will enable the WTS multifunction, including any mechanical load simulator or drive simulator with proper load or drive model. The system developed is both steady state and dynamic process.

Chapter 5 Conclusion

5.1 Conclusions

A real-time wind turbine simulator is proposed in this thesis to provide an in-door controllable environment for technology development in wind energy conversion system, including the development of generator, power electronic converter, interfacing apparatus and digital controller. The WTS simulates in real-time the aerodynamic characteristic of real wind turbines as well as the dynamic performance of turbine-generator mechanical systems. The proposed WTS is more close to the field turbine operation with variable wind speed, which is not effectively addressed by other researches. In this thesis, the WTS consists of an energy bidirectional PWM rectifier with LCL filter and a DC motor with bidirectional DC/DC converter. The energy bidirectional configuration is proved to let the WTS universal to different turbine-generator configuration with higher or lower moment of inertia.

PWM rectifier with LCL filter is developed to provide a constant DC link, which stables the operation of DC motor drive in transient. Line side THD is greatly reduced while power factor is successfully maintained at unity. All this guarantees the stable operation of WTS no matter the grid side fluctuation. The control for PWM rectifier with LCL filter is developed. Active damping is developed to wide the bandwidth of current loop, which improves the dynamic performance of WTS system. The controls with current sensor positions at grid side or converter side are investigated. The performances are very similar when the loop bandwidth is maintained.

The DC motor with four quadrant chopper is employed in this thesis to drive the generator because of the accurate electromagnetic torque can be obtained from DC motor. Torque transducers are eliminated. A novel 2nd order observer is developed to obtain the mechanical system state variables, the speed and acceleration. Generator side torque is successfully estimated by using the observed acceleration. The turbine-generator mechanical system is simulated in real time while maintain a same acceleration no matter the WTS has higher or lower inertia. The accuracy of torque estimation depends on the parameters of WTS, including the mechanical system loss and WTS system inertia, which are measured offline by proposed algorithm.

The 2kW prototype WTS system was modeled in MATLAB/Simulink. The simulation result was compared with the experimental result, which successfully verifies the proposed real-time WTS.

5.2 Major Contributions

The major contributions of this thesis are on two sides as following.

1) A novel sensorless wind turbine simulator for real-time dynamic simulation. The turbine aerodynamic characteristic is modeled in WTS and the turbine-generator mechanical system was simulated in real-time. The torque transducers are eliminated by using estimated generator side torque. A 2nd order observer was designed to obtain the state variables of mechanical system. The real-time simulation of mechanical system makes the WTS more close to the field operation since the wind speed is variable.

2) PWM rectifier with LCL filter is employed in proposed WTS, which make the WTS immune to grid voltage disturbance, grid-friendly connecting and capable of simulating real wind turbines with small inertia in real time. In this thesis, investigation has been done on the different control scheme with the feedback of either converter side current or grid side current. The comparison of different control schemes shows similar performance while maintain a same current loop bandwidth. Simulation and experimental results verified the analysis.

5.3 Further Research

In order to precisely simulate the real turbine operation, following work should be done in the future.

1) The model of turbine and turbine mechanical system need to be improved. The turbine model based on the aerodynamic equation empirically describes the turbine output, which is not universal to different turbines. Beside that, the tower effect and wind velocity difference in the swap area should be considered. The improvement also should be done to model the turbine generator mechanical system, including the bearing, gearbox and other mechanical accessories if applicable.

2) Turbine with pitch control should be considered in real-time in the future WTS. This part is simply considered in this thesis as a single input for wind turbine model, which changes the turbine output instantaneously. In the real large wind turbine system, hydraulic pitch control mechanism is employed to rotate the blades. The model based on the pitch control mechanical system should be done in the future.

3) Other types of motors such as permanent magnetic brushless motor (PMBM) or induction motor should be investigated for the real-time WTS. Especially the PMBM has faster torque response.

Appendices

Appendix A

A.1 Control of PWM Rectifier

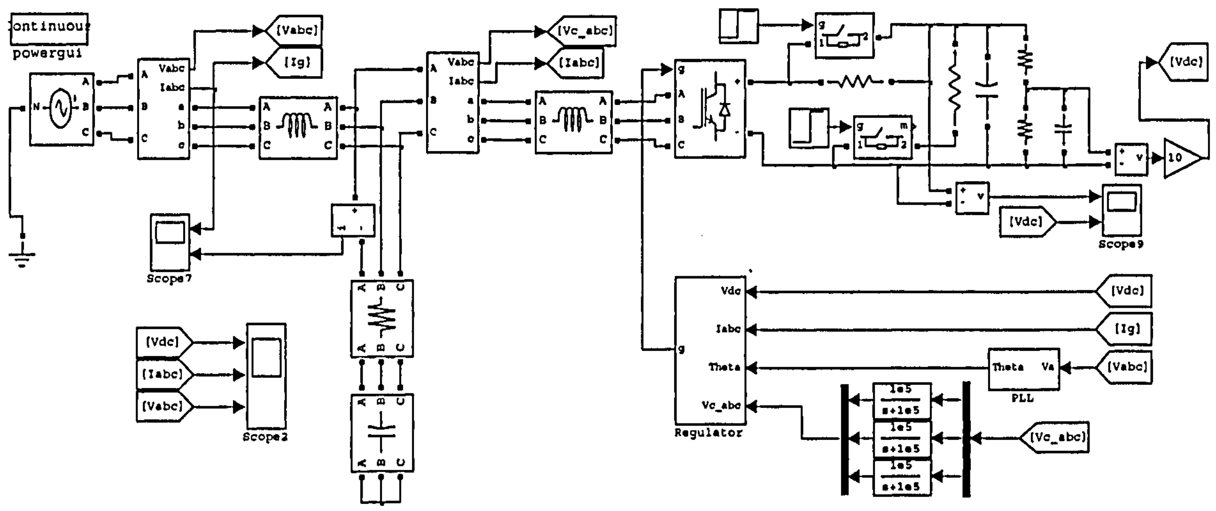


Figure A-1: Simulink model of LCL-filter based PWM Rectifier

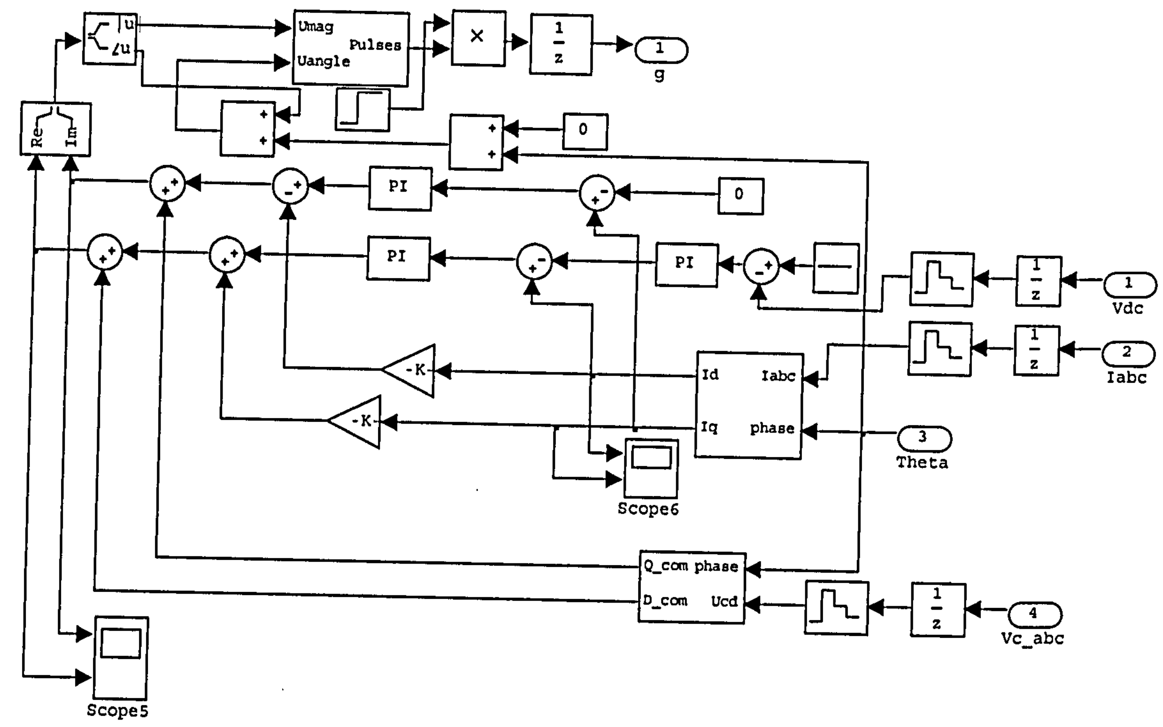


Figure A-2: Simulink model of PWM Rectifier Regulator block

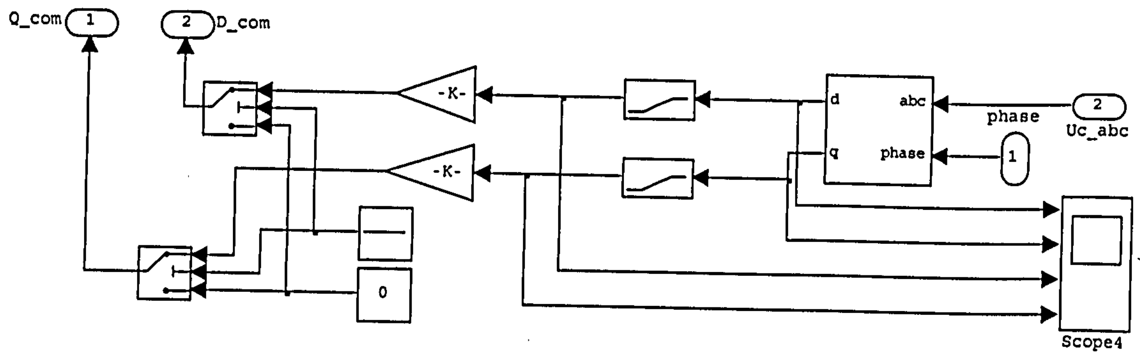


Figure A-3: Simulink model of active damping block

A.2 Wind Turbine Simulator

The Simulink model of the turbine is illustrated in the following figure. The three inputs are the generator speed (ωr_pu) in pu of the nominal speed of the generator, the pitch angle in degrees and the wind speed in m/s. The tip speed ratio λ in pu of λ_nom is obtained by the division of the rational speed in pu of the base rotational speed (defined below) and the wind speed in pu of the base wind speed. The output is the torque applied to the generator shaft.

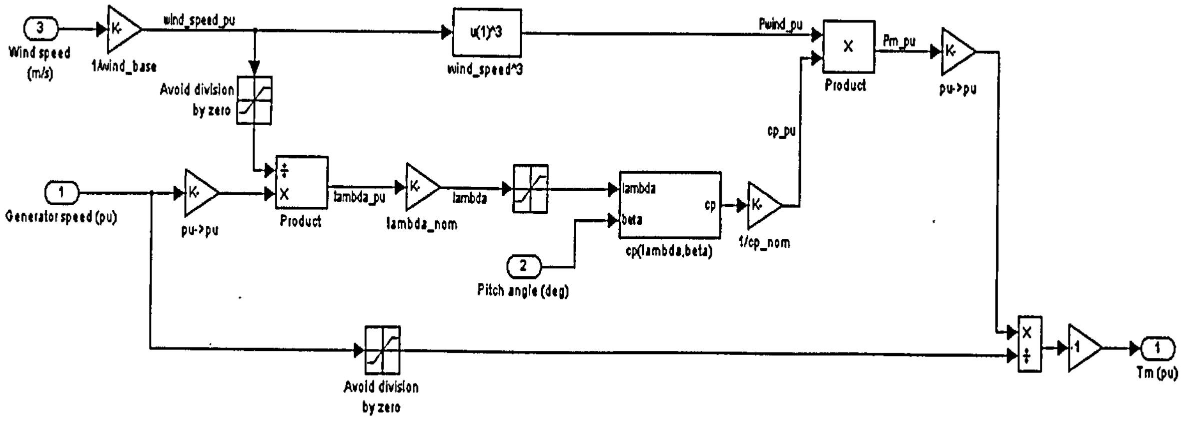


Figure A-4: Simulink model of wind turbine

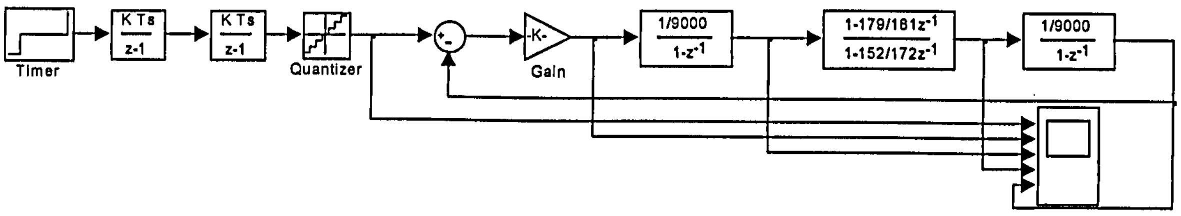


Figure A-5: Simulink model of Speed and Acceleration Observer

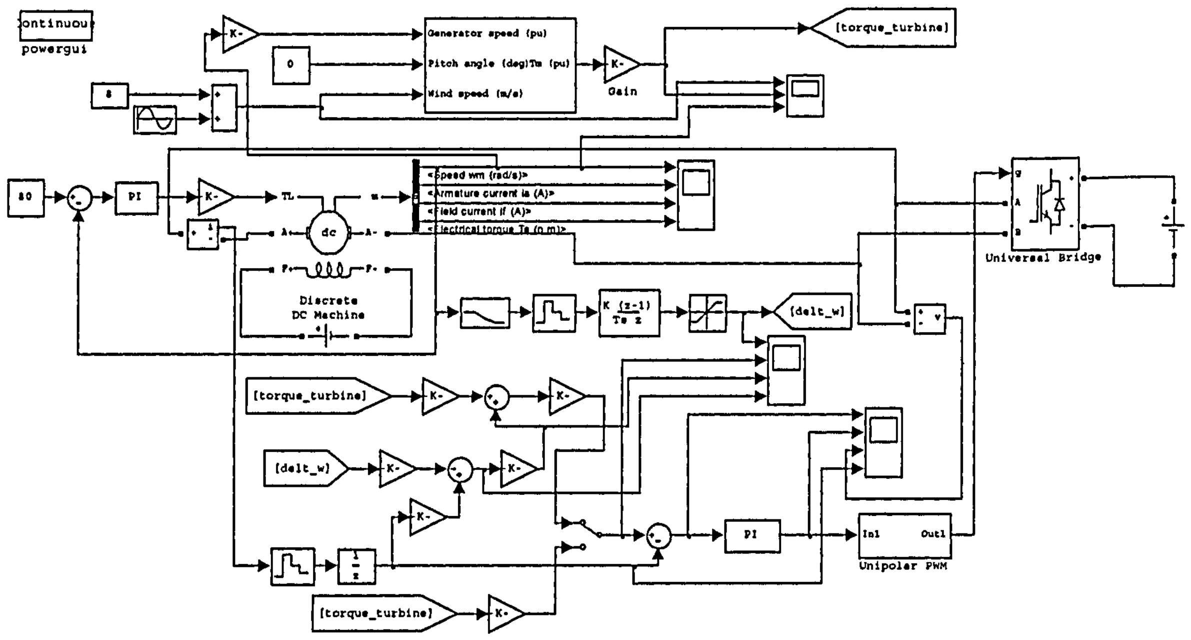


Figure A-6: Simulink model of Wind Turbine Simulator

Appendix B

Parameters for Wind Turbine Simulator system

Table B-1 Parameters of DC machine:

Operating Condition	Motor	Generator
Power	2 kW	1.5 kW
Armature Voltage	120 V	120 V
Shunt Voltage	120V	120V
Armature Current	23 A	12.5 A
Shunt Current	0.81A	1.12A
Speed	1800 rpm	1800 rpm
Armature Inductance	5mH	5mH
Armature Resistance	2Ω	2Ω

Table B-2 Parameters of LCL-filter

Inductors L1 and L2	1mH/20A
Capacitors	$10\,\mu F$

Table B-3 Parameters of controller:

PWM Rectifier	Current control loop	Kp=0.04 , Ki=30
	Voltage control loop	Kp=0.5 , Ki=40
Wind Turbine Simulator	DC motor current control	Kp=0.04 , Ki=16
	PLL control loop	Ka=74000 , t_1=0.01, t_2=9e-4
Generator	Speed control loop	Kp=0.6 , Ki=0.5

Appendix C

M-file for Bode Plot of PWM Rectifier current control loop

```
L=1e-3;R=5000;R1=0;Lf=1.0e-3;Rf=10000;R2=0.0;Cf=1e-5;Rd=0;Kpwm=200/1.732;
K=0.04;Ti=1.3e-3;% PI controller time constant
Ts=1/9e3;%processing delay time constant
Tpwm=1/18e3;
Td=3e-4; %Time constant of  hign_pass filter for active damping
Th=2e-5;% current sensor delay time constant

s=tf('s');
G1=1/(L*s+R1)%+1/R;%(L*s+R)/s*L*R;
G2=1/(Lf*s+R2)%+1/Rf;%(Lf*s+Rf)/s*Lf*Rf;
G3=1/(Cf*s)+Rd;
Gp1=(G1+G1*G2*G3)/(1+G1*G3+G2*G3);
Gp2=G1*G2*G3/(1+G1*G3+G2*G3);
w={60,7e4};
Gpi=(K*(1+s*Ti))/(s*Ti);     %PI controller
[num,den] = pade((Ts+Tpwm),2);
Gdelay=tf(num,den);%approximation of time delay

Gu1=Kpwm*Gpi*Gp1;%open loop gain without active damping(converter side)
Guu1=Kpwm*Gp1*Gdelay;%for SISO plot
Gu1.ioDelay=Ts+Tpwm;
Gu2=Kpwm*Gpi*Gp2;%(grid side)
Guu2=Kpwm*Gp2*Gdelay;%for SISO plot
Gu2.ioDelay=Ts+Tpwm;
%bodeplot(Gu1,Gu2,w);

Gd=(s*Td)/(Kpwm*1.5*(1+s*Td));%High_Pass filter for active damping
Go1=Kpwm*Gpi*(G1+G1*G2*G3)/(1+G1*G3+G2*G3-G1*G3*Gd*Kpwm*Gdelay);%open loop
gain with active damping(converter side)
Goo1=Go1*Gdelay/Gpi;%for SISO plot;
Go1.ioDelay=Ts+Tpwm;
Go2=Kpwm*Gpi*G1*G2*G3/(1+G1*G3+G2*G3-G1*G3*Gd*Kpwm*Gdelay);%open loop gain
with active damping(grid side)
Goo2=Go2*Gdelay/Gpi;%for SISO plot;
Go2.ioDelay=Ts+Tpwm;
Ge1=G1*G2*G3/(1+G1*G3+G2*G3+(G1+G1*G2*G3)*Gpi*Kpwm*Gdelay);
Ge2=(G2+G1*G2*G3)/(1+G1*G3+G2*G3+G1*G2*G3*Gpi*Kpwm*Gdelay);
bodeplot(Gu1,Go1,w);
%bodeplot(Gu2,Ge1,w);
```

Appendix D

DSP Program Code

```
/************************************/
/*File Name:  wts.c                */
/************************************/


#include   "DSP281x_Device.h" // DSP281x Headerfile
Include File
#include   "DSP281x_Examples.h" // DSP281x
Examples Include File
#include   "FPGA_csr.h"
#include   "Display.h"
#include   <stdio.h>
#include   "IQmathLib.h"

#define J_T         0.42 // (N/m^2) Inertia_Turbin
#define J_M          0.28 //(N/m^2) Inertia_Motor
#define BaseWindSpeed 12.0//(m/s)
#define BaseGeneratorSpeed 100.0//(rad/s)
#define BaseRotationalSpeed 1.2//pu of base generator
speed
#define BasePower    2000.0//(w)
#define Kp          0.8//Maximum power at base wind
speed(pu of base power)

#define Speed_min    20//(rad/s)Minimum speed for
inertia test
#define Speed_max    120//(rad)Maximum speed for
inertia test
#define Iarm_test   13.0//(A) Armature current for
inertia test
#define Iarm_test1  15.0//(A) Armature current for
performance test

#define ADC_usDELAY   8000
#define ADC_usDELAY2  40

#define PI          3.1415926
#define sqrt3        1.732051
#define sqrt13      0.577350269
#define Sw_freq     8997.1//sampling frequency
#define AVE_SAMPLE 8
//AVE_SAMPLE=2^AVE_SHFT
#define AVE_SHFT     3
#define AVE_SAMPLE_DC16//
AVE_SAMPLE_DC=2^AVE_SHFT_DC
#define AVE_SHFT_DC  4
#define w_DC        1000.0//cut-off frequency(rad/s) of
low-pass filter for DC voltage
#define w_AD        3000.0//cut-off frequency(rad/s) of
low-pass filter for active damping
#define w_speed     2000.0//cut-off frequency(rad/s) of
low-pass filter for motor speed
#define w_torque    20.0
#define K_res       10.0//gain of resonant regulator

#define Udc_ref     180// DC voltage reference
#define XL         0.00377   //0.03 //Line to Converter
inductance
#define Kf         0.6 // Voltage constant for DC
motor(torque_motor=Kf*Iarm)

#define sen_posi    0//current sensor position,
0:converter side, 1:grid side
#define Kp_Udc      0.5
#define Ki_Udc      40.0//for DC voltage control loop
#define Kp_I       0.04
#define Ki_I       30.0//for current control loop
#define Kp_Iarm     0.05
#define Ki_Iarm     3.0//for motor armature current
control loop
#define K_PLL       778700.0//gain for motor angle
close loop

#pragma DATA_SECTION(graph_data,"graph_data");
#pragma
DATA_SECTION(graph_data1,"graph_data1");
#pragma
DATA_SECTION(graph_data2,"graph_data2");
#pragma
DATA_SECTION(graph_data3,"graph_data3");
#pragma
DATA_SECTION(graph_data4,"graph_data4");
#pragma
DATA_SECTION(graph_data5,"graph_data5");
#pragma
DATA_SECTION(graph_data6,"graph_data6");

#pragma
DATA_SECTION(wind_profile,"wind_profile");

/************************************/
/*VARIABLE DECLARATION             */
/************************************/
```

```
/* ADC */
Uint16 Sample_ch0[AVE_SAMPLE],
Sample_ch1[AVE_SAMPLE],
     Sample_ch2[AVE_SAMPLE],
Sample_ch3[AVE_SAMPLE],
     Sample_ch4[AVE_SAMPLE],
Sample_ch5[AVE_SAMPLE_DC],
     Sample_ch6[AVE_SAMPLE_DC],
Sample_ch7[AVE_SAMPLE],
     Sample_ch8[AVE_SAMPLE_DC],
Sample_ch9[AVE_SAMPLE],
     Sample_ch10[AVE_SAMPLE],
Sample_ch11[AVE_SAMPLE];
     //Sample_ch12[AVE_SAMPLE],
Sample_ch13[AVE_SAMPLE],
     //Sample_ch14[AVE_SAMPLE],
Sample_ch15[AVE_SAMPLE];

Uint16 ConversionCount=0;
Uint16 ConversionCount1=0;
int32 AD_BIAS0, AD_BIAS1, AD_BIAS2, AD_BIAS3,
    AD_BIAS4, AD_BIAS5, AD_BIAS6, AD_BIAS7,
    AD_BIAS8, AD_BIAS9, AD_BIAS10,
AD_BIAS11,
    AD_BIAS12, AD_BIAS13, AD_BIAS14,
AD_BIAS15;

int32 Vdc_P,Vdc_N; //(_iq18 format)
int32 Vab, Vbc, Vca; //(_iq18 format)
int32 Eab, Ebc, Eca; //(_iq18 format)
int32 Ia, Ib, Ic; //(_iq18 format)
int32 Ia_g, Ib_g, Ic_g; //(_iq18 format)
int32 Ia1, Ic1, Ia2, Ic2; //(_iq18 format)
int32 Id_max=0;

/* for testing*/
 Uint32 watch =0;
 Uint16 watch1 =0;
 Uint16 watch2 = 0;
 Uint16 watch3 = 0;
 Uint16 counter_value = 0;
 Uint16 Conter_0 = 0;
 Uint16 Testswitch =0;

/*PLL */
Uint16 Theta_con, Theta_err; //phase and error info in
counter format
Uint16 nmax;
Uint32 Theta_pu; //line voltage angle in per unit value
_iq28 Theta_line;    //line voltage angle in radian value
(0-2pi)
_iq28 Theta_phase; //phase voltage angle in radian value
_iq28 cos_theta, sin_theta;

/*Voltage */
_iq18 Va, Vb, Vc;
_iq18 V_alpha, V_beta;
_iq18 V_d, V_q,V_d0, V_q0,V_d_L, V_q_L,V_d_L0,
V_q_L0,V_d_H, V_q_H;
_iq18 E_d, E_q;

/*Inner loop current control*/
_iq18 I_alpha, I_beta;
_iq18 I_d,I_d_ref, I_d_err,I_d_err1,
upperlimit_Id,lowwerlimit_Id;
_iq18 I_q,I_q_ref, I_q_err,I_q_err1;
_iq18 Ig_d,Ig_q;
_iq18 V_d_O, V_q_O;
_iq18 Va_O,Vb_O,Vc_O;
_iq18 V_alpha_O, V_beta_O;
_iq28 kp_I, ki_I;  //PI controller

/*Outer loop DC voltage control*/
_iq18 Vdc_ref,Vdc,Vdc0,Vdc_L,Vdc_L0,
Vdc_err,Vdc_err1;
_iq28 kp_dc, ki_dc;  //PI controller

/*Low-pass filter*/
_iq28
LPFa,LPFb,LPFa_DC,LPFb_DC,LPFa_speed,LPFb_spe
ed,LPFa_torque,LPFb_torque;

/*Resonant regulator*/
_iq28 Res_6a, Res_6b;
_iq18 I_d_in1,I_d_in2, I_q_in1,I_q_in2;
_iq28 I_d_out,I_d_out1,I_d_out2,
I_q_out,I_q_out1,I_q_out2;

/* SVPWM*/
Uint16 Ts=16672;//PWM period in counter value
Uint16 T1,T2,T0;
int16 section_number;
_iq28 Theta,Theta_sec;
_iq18 Mmag;

/*Inverter*/
_iq18 D_inv,D_inv1;
_iq18 Iarm,Io,Iarm_ref, Iarm_err, Iarm_err1,I_test;
_iq18 If,If0,If_L,If_L0;
_iq28 kp_Iarm, ki_Iarm;  //PI controller

/*Motor position and speed*/
_iq18 Speed_CNT_L,Speed_CNT_L0;
_iq18
Speed_w,Speed_w0,Speed_w_ref,Speed_w_err,Speed_
w_err1,Speed_n,Delta_speed;
_iq18 Te,Tm;
Uint16 Angle_CNT[33],Speed_CNT,Speed_CNT0;
Uint16 Angle_index=0,Angle_index1=0;

/*Motor position and speed close loop control*/
```

```
Uint16
Angle_CNT0,Angle_CNT1,CAP_CNT,CAP_CNT0;
_iq1  Ka_PLL; //open loop gain of the loop
_iq28 A_PLL, B_PLL;//for compensator
_iq28 angle, angle_m, angle_m0, angle_err,angle_com;
_iq18 veloc_x, veloc_x0, veloc_y, veloc_y0;
_iq7  accele_pll;//acceleration
_iq10 accele, accele0, accele_L, accele_L0;

/*Wind turbine simulator*/
int16 wind_speed,beta;
_iq18 inertia;
_iq18 torque_turbine, torque_motor,
torque_motor_ref,torque_G,torque_G0,torque_G_L,torq
ue_G_L0;
_iq18 pitch_angle,speed_test;
_iq28 lambda;//the tip speed ratio
_iq28 Cp_pu;//performance coefficienct in pu of the
maximum value of Cp(0.48)
int32
wind_speed_pu,generator_speed_pu,Pwind,lambda_inv,
e_lambda,torque_turbine_pu,aa,bb;
int16  wind_profile[2048];//wind profile
Uint16 wind_index=0;
Uint16 wind_index1=0;

/*Key input*/
int16 key_value,key_value1, key_monitor,
key_monitor1,key1;

/* Display*/
Uint16 display_counter,display_counter1;

/*Control flags*/
Uint16 FUNCTION, //1:Turbine simulator  2:Inertia test
3:Performance test 4:Speed control
    CONTACTOR, /*0-open, 1-close*/
    INTERRUPT, /*0-t1ufint stopped, 1-Rectifier
running, 2-Inverter*/
          FAULT,    /*0-no fault 1- fault detected*/
          PLL,      /*0-PLL not locked 1-locked*/
          Fault_info_A, /*Result of fault detection from
left (A) side*/
    Fault_info_B, /*Result of fault detection from left
(B) side*/

          SCREEN;   /*0-system status screen0 1-status
screen 1 */
Uint16 Res_flag=0;
Uint16 test_number=0;
Uint16 test_flag=0;//0:acceleration; 1:deceleration
Uint16 test_flag1=0;
Uint16 J_COM_flag=0;//0:without inertia compensation
1: with compensation
Uint16 INERTIA_TEST=0;
Uint16 T_CNT=0;//for inertia test
Uint16 T_CNT0=0;
Uint16 T_CNT1=0;
Uint16 T_CNT2=0;

/*Graph of waveform*/
int16  graph_data[9000];//I_d
int16  graph_data1[9000];//I_d_ref
int16  graph_data2[9000];//V_d
int16  graph_data3[9000];//V_q
int16  graph_data4[9000];//I_q
int16  graph_data5[9000];//Vdc
int16  graph_data6[9000];//Theta_phase

Uint16 graph_index=0;
Uint16 graph_index1=0;
Uint16 graph_index2=0;
Uint16 graph_index_flag=0;
Uint16 data_shift_times=0;

/*************************************/
/*FUNCTION DECLARATION            */
/*************************************/

interrupt void  adc_isr(void);
interrupt void  pdpinta_isr(void);
interrupt void  t3cint_isr(void);
interrupt void EVB_CAPINT4_ISR(void);
interrupt void EVB_CAPINT5_ISR(void);
//interrupt void  t1ufint_isr(void);

void Protection_PLL (void);
void get_theta(void);
void get_adc_V(void);
void get_speed(void);
void angle_closeloop(void);
void abc_dq_transformation(void);
void Regulation(void);
void Turbine_simulator(void);
void Inertia_compensation(void);
void Motor_drive(void);
void Speed_control(void);
void Inertia_test(void);
void performance_test(void);

void SVPWM_7S(void);
void Enable_Rec(void);
void Disable_Rec(void);
void Enable_Inv(void);
void Disable_Inv(void);
void Key_input(void);
void Close_Relay(void);
void Open_Relay(void);
void AD_bias16(void);
void Delay_mili_second (int delay);
void Init_variable(void);
void InitGpio(void);
void InitXintfClocks(void);
void InitPeripherals(void);
void InitEVA(void);
void InitEVB(void);
void InitAdc(void);
void init_fpga(void);
void Buffer_rotating(int16 buffer[9000],Uint16
shift_times);

/**************************************/
/* NAME:      main()              */
/* RETURNS:   void               */
/**************************************/

void main(void)
{
// Step 1. Initialize System Control:
// PLL(PLLCR=0xA), WatchDog(disable), enable
Peripheral Clocks
  InitSysCtrl();

// Step 2. Initialize the Xintf clocks
  InitXintfClocks();

// Step 3. Initalize GPIO:
  InitGpio();

// Step 4. Clear all interrupts and initialize PIE vector
table:
  DINT; // Disable CPU interrupts
  InitPieCtrl(); // Initialize the PIE control registers to
their default state.
           // The default state is all PIE interrupts
disabled and flags
           // are cleared.

  IER = 0x0000; // Disable CPU interrupts
  IFR = 0x0000; // Clear all CPU interrupt flags

  InitPieVectTable(); // Initialize the PIE vector table

// Interrupts  used are re-mapped to ISR functions
  EALLOW;
  PieVectTable.PDPINTA = &pdpinta_isr;
  PieVectTable.ADCINT = &adc_isr;
  PieVectTable.T3CINT = &t3cint_isr;
  PieVectTable.CAPINT4=&EVB_CAPINT4_ISR;
  PieVectTable.CAPINT5=&EVB_CAPINT5_ISR;
  //PieVectTable.T1UFINT = &t1ufint_isr;
  EDIS;

// Step 5. Initialize all the Device Peripherals:
  InitPeripherals();

// Step 6. User specific code, enable interrupts:
  PieCtrlRegs.PIEIER1.bit.INTx1 = 1; //Enable
PDPINTA
  PieCtrlRegs.PIEIER1.bit.INTx6 = 1; // Enable
ADCINT in the PIE: Group 1 interrupt 6
  PieCtrlRegs.PIEIER4.bit.INTx5 = 1; // Enable t3cint in
the PIE: Group 4 interrupt 5
  PieCtrlRegs.PIEIER5.bit.INTx5 = 1; // Enable
CAPINT4 in the PIE: Group 5 interrupt 5
  PieCtrlRegs.PIEIER5.bit.INTx6 = 1; // Enable
CAPINT5 in the PIE: Group 5 interrupt 6
  //PieCtrlRegs.PIEIER2.bit.INTx6 = 1; // Enable t1ufint
in the PIE: Group 2 interrupt 6

  IER |= M_INT1; // Enable CPU INT1:
  IER |= M_INT4; // Enable CPU INT4:
  IER |= M_INT5; // Enable CPU INT5:

  EINT;  // Enable Global interrupt INTM
  ERTM;  // Enable Global realtime interrupt DBGM

  Clear_LED();
  AD_bias16();
  Manu_display();
  AdcRegs.ADCTRL2.bit.INT_ENA_SEQ1 = 1;  //
enable SEQ1 interrupt (every EOS)

// Step 7. Infinite loop
  while(1)
  {
    if (FAULT == 0)Key_input ();
    display_counter++;
           if(display_counter==20000)
           {
            display_counter1++;display_counter=0;
           }
           if (display_counter1==30)
           {
     //Text_LCD_display();
     //key1=1;//for test
     LED_display();
     display_counter1=0;
     display_counter=0;
    }
  }
}  /*End Main*/

/****************************************/
/*  NAME:  adc_isr()
RETURNS: void
 DESCRIPTION: Interrupt for ADC results revieval
*/
/****************************************/

interrupt void  adc_isr(void)
{
```

```
Sample_ch0[ConversionCount]=((AdcRegs.ADCRESU
LT0>>4) );

Sample_ch1[ConversionCount]=((AdcRegs.ADCRESU
LT1>>4) );

Sample_ch2[ConversionCount]=((AdcRegs.ADCRESU
LT2>>4) );

Sample_ch3[ConversionCount]=((AdcRegs.ADCRESU
LT3>>4) );

Sample_ch4[ConversionCount]=((AdcRegs.ADCRESU
LT4>>4) );


Sample_ch7[ConversionCount]=((AdcRegs.ADCRESU
LT7>>4) );


Sample_ch9[ConversionCount]=((AdcRegs.ADCRESU
LT9>>4) );

Sample_ch10[ConversionCount]=((AdcRegs.ADCRESU
LT10>>4) );

Sample_ch11[ConversionCount]=((AdcRegs.ADCRESU
LT11>>4) );
    //
Sample_ch12[ConversionCount]=((AdcRegs.ADCRESU
LT12>>4) );
    //
Sample_ch13[ConversionCount]=((AdcRegs.ADCRESU
LT13>>4) );
    //
Sample_ch14[ConversionCount]=((AdcRegs.ADCRESU
LT14>>4) );
    //
Sample_ch15[ConversionCount]=((AdcRegs.ADCRESU
LT15>>4) );

     if (ConversionCount==AVE_SAMPLE-1)
ConversionCount=0;
     else ConversionCount++;


Sample_ch5[ConversionCount1]=((AdcRegs.ADCRESU
LT5>>4) );//Udc+

Sample_ch6[ConversionCount1]=((AdcRegs.ADCRESU
LT6>>4) );//Udc-

Sample_ch8[ConversionCount1]=((AdcRegs.ADCRESU
LT8>>4) );//If
```

```
     if (ConversionCount1==AVE_SAMPLE_DC-1)
ConversionCount1=0;
     else ConversionCount1++;

  AdcRegs.ADCTRL2.bit.RST_SEQ1 = 1;       // Reset
SEQ1
  AdcRegs.ADCST.bit.INT_SEQ1_CLR = 1;     // Clear
INT SEQ1 bit
  PieCtrlRegs.PIEACK.all = PIEACK_GROUP1; //
Acknowledge interrupt to PIE
} /*End of Adc_isr*/

/***************************************/
/* NAME:      t3cint_isr()
/* DESCRIPTION: Interrupt for computing
*/
/***************************************/
interrupt void  t3cint_isr(void)
{
  Angle_CNT1=Angle_CNT0;
  CAP_CNT=CAP_CNT0;
  EINT;
  watch1= EvbRegs.T3CNT;

  get_theta();
  get_speed();
  get_adc_V();
  angle_closeloop();
  abc_dq_transformation();
  switch(FUNCTION)
  {
   case 0:

   break;
   case 1:
   if (veloc_y>_IQ18(8))
   {Turbine_simulator(); Inertia_compensation();;}
         else Iarm_ref=_IQ18(2.5);
   break;
   case 2:
   Iarm_ref=_IQ18(Iarm_test);//constant torque
   Inertia_test(); //for inertia test
   break;
   case 3:
         performance_test();//for performance test
   break;
   case 4:
   if (INTERRUPT==2) Speed_control();
         else Speed_w_ref=0;
   break;
  }


  switch (INTERRUPT)
  {
    case 1:
         upperlimit_Id=_IQ18(5);
    Regulation();
    SVPWM_7S();
    break;
    case 2:
    if ((Iarm>_IQ18(25))||(Iarm<_IQ18(-25)))
 {Disable_Inv();INTERRUPT = 1;}
    if (veloc_y>_IQ18(190))
 {Disable_Inv();Iarm_ref=0;INTERRUPT=1; }
    upperlimit_Id=_IQ18(30);
    Regulation();
    SVPWM_7S();
    Motor_drive();
    break;
   }
  /*for waveforms display*/
   //graph_data[graph_index]= I_d>>8;
   graph_data1[graph_index]= angle>>18;
   graph_data2[graph_index]= Ia>>8;
   graph_data3[graph_index]= Iarm>>8;
   graph_data4[graph_index]= If_L>>8;
   //graph_data5[graph_index1]= veloc_y>>11;
   graph_data6[graph_index]= accele_L>>3;
   graph_index++;
   if (graph_index== 9000) graph_index=0;
   if (graph_index_flag==1) graph_index1++;
   if (graph_index1==6300) {graph_index--;
graph_index1--;}
  /*sampling data of speed*/ //9000points for 60s
   if (graph_index1== 60)
   {graph_data[graph_index2]= torque_G_L>>8;
    graph_data5[graph_index2]= veloc_y>>11;
    graph_index2++;graph_index1=0;}
  if (graph_index2==9000)
{graph_index1=0;graph_index2=0;graph_index_flag=0;}

  watch2= EvbRegs.T3CNT;
  if (watch2>watch1) {watch3=watch2-watch1;}
  else {watch3=watch1-watch2;}
  EvbRegs.EVBIFRA.bit.T3CINT=1;// Clear INT
  PieCtrlRegs.PIEACK.all = PIEACK_GROUP4;//
Acknowledge interrupt to PIE
} /*End of t3cint_isr*/

/*************************************/
/*NAME:
EVB_CAPINT4_ISR(),EVB_CAPINT4_ISR()*/
/*************************************/
interrupt void EVB_CAPINT4_ISR(void)  // CAP4
{
    CAP_CNT0=EvbRegs.CAP4FBOT;
    Angle_CNT0=EvaRegs.T2CNT;

    EvbRegs.EVBIFRC.all=BIT0;
    PieCtrlRegs.PIEACK.all=PIEACK_GROUP5;
}
interrupt void EVB_CAPINT5_ISR(void) // CAP5
{
    CAP_CNT0=EvbRegs.CAP5FBOT;
    Angle_CNT0=EvaRegs.T2CNT;

    EvbRegs.EVBIFRC.all=BIT1;
    PieCtrlRegs.PIEACK.all=PIEACK_GROUP5;
}

/*************************************/
/* NAME:     pdpinta_isr()
/* DESCRIPTION: Protection interrupt generated from
FPGA              */
/*************************************/

interrupt void pdpinta_isr(void)
{
  Open_Relay ();
  Disable_Rec();
  Disable_Inv();

        INTERRUPT = 0;
        FAULT = 1;
  CONTACTOR = 0;
  Fault_info_A=*FAULT_DETECT_A;
  Fault_info_B=*FAULT_DETECT_B;
  Display_Fault ();
        PieCtrlRegs.PIEACK.all =
PIEACK_GROUP1;// Acknowledge interrupt to PIE
        /*No clear of the interrupt flag*/
} /*End of Pdpinta_isr*/

/*************************************/
/* NAME:     Protection_PLL()              /
/* DESCRIPTION: Protection action when PLL is lost
*/
/*************************************/

void Protection_PLL (void)
{
  Open_Relay ();
  Disable_Rec();
  Disable_Inv();
        INTERRUPT = 0;
        FAULT = 1;
  CONTACTOR = 0;
  Fault_info_A=*FAULT_DETECT_A;
  Fault_info_B=*FAULT_DETECT_B;
  Display_Fault ();
} /*End of Protection_PLL*/

/*************************************/
/* NAME:     get_theta()       */
/* DESCRIPTION: phase for abc-dq transformation
*/
```

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

void get_theta(void)
{
  Theta_con = *PHASE_A;    /*phase info in counter
format*/
  Theta_err = *PHASE_LOCK_A;  /*DPLL phase
error in counter format*/
  nmax = *N_MAX_A + 1;  /* */
        /*PLL phase information*/
  if ((Theta_err>15)&&(Theta_err<480))
        {
   PLL = 0;
   //Protection_PLL();
        }
        else {PLL =1;}
  if (PLL==0) test_number++;
}

/*************************************/
/* NAME:      get_speed()               */
/* DESCRIPTION: motor speed w(rad/s) and speed
acceleration dw/dt        */
/*************************************/
void get_speed(void)
{
  Angle_CNT[Angle_index]=EvaRegs.T2CNT;
  Speed_CNT0=Speed_CNT;

  if (Angle_index==32)
        {
   if (Angle_CNT[32]<Angle_CNT[0])
Speed_CNT=Angle_CNT[32]+14400-Angle_CNT[0];
   else Speed_CNT=Angle_CNT[32]-Angle_CNT[0];
   Angle_index=0;
  }
        else
        {
         if
(Angle_CNT[Angle_index]<Angle_CNT[Angle_index+
1]) Speed_CNT=Angle_CNT[Angle_index]+14400-
Angle_CNT[Angle_index+1];
   else Speed_CNT=Angle_CNT[Angle_index]-
Angle_CNT[Angle_index+1];
   Angle_index++;
        }

  Speed_CNT_L0=Speed_CNT_L;
  Speed_CNT_L=
_IQ18mpyI32(LPFa_speed>>10,(Speed_CNT0+Speed_
CNT));
  Speed_CNT_L+= _IQ18mpyIQX(LPFb_speed,28,
Speed_CNT_L0,18);//low pass filtering
  Speed_n=_IQ18mpy(Speed_CNT_L>>5,
_IQ18(Sw_freq*60/14400));//motor speed in (rpm)
```

```
 if (Angle_index1==31)
 {
  Speed_w0=Speed_w;
  Speed_w=_IQ18mpy(Speed_CNT_L>>5,
_IQ18(Sw_freq*2*PI/14400));//motor speed in (rad/s)
  Delta_speed=_IQ18mpy((Speed_w-
Speed_w0),_IQ18(Sw_freq/32));// dw/dt
  //if (Delta_speed>_IQ18(50)) Delta_speed=_IQ18(50);
  //if (Delta_speed<_IQ18(-50)) Delta_speed=_IQ18(-
50);
  Angle_index1=0;
 }
 else Angle_index1++;

}

/*************************************/
/* NAME:      angle_closeloop()        */
/* DESCRIPTION: motor speed w(rad/s) and speed
acceleration dw/dt tracking */
/*************************************/
void angle_closeloop(void)
{
 int16 a,b;
 a=Ts>>1;
 b=CAP_CNT;
 accele0= accele;
 veloc_x0= veloc_x;
 veloc_y0= veloc_y;
 angle_m0= angle_m;
 accele_L0= accele_L;

 if (b>a) b=b-a;
 else b=a-b;
 angle_com= _IQ28(1.024/150000)*b;

angle_com=_IQ28mpyIQX(angle_com,28,veloc_y0,18);
 angle_com=angle_com>>10;
 if (angle_com>_IQ28(0.0005)) angle_com=0;

 angle=angle_com+_IQ28mpyI32(
_IQ28(2*PI/14400),Angle_CNT1);//motor angle in rad/s
 angle_err= angle-angle_m;
 if (angle_err>=_IQ28(PI)) angle_err=angle_err-
_IQ28(2*PI);
 if (angle_err<_IQ28(-PI))
angle_err=angle_err+_IQ28(2*PI);

 accele_pll= _IQ7mpyIQX(angle_err,28,(K_PLL*2),1);
 accele= _IQ10mpyIQX(angle_err,28,Ka_PLL,1);

 veloc_x=
_IQ18mpyIQX(accele_pll,7,_IQ28(1/Sw_freq),28);
 veloc_x+= veloc_x0;

 veloc_y= veloc_x;
 veloc_y= veloc_y-
_IQ18mpyIQX(veloc_x0,18,A_PLL,28);
 veloc_y+= _IQ18mpyIQX(veloc_y0,18,B_PLL,28);

 angle_m=
_IQ28mpyIQX(veloc_y,18,_IQ28(1/Sw_freq),28);
 angle_m+= angle_m0;
 if (angle_m>=_IQ28(2*PI)) angle_m=angle_m-
_IQ28(2*PI);
 if (angle_m<0) angle_m=angle_m+_IQ28(2*PI);

 accele_L=
_IQ10mpyIQX(LPFa_speed,28,(accele0+accele),10);
 accele_L+= _IQ10mpyIQX(LPFb_speed,28,
accele_L0,10);//low pass filtering
}

/**********************************/
/*  NAME:       get_adc_V()              */
/*  DESCRIPTION: Line voltage and current
conditioning with multi-sampling          */
/**********************************/

void get_adc_V(void)
{
        Uint16 i;

        Uint32 Sum_ch0=0;
  Uint32 Sum_ch1=0;
  Uint32 Sum_ch2=0;
  Uint32 Sum_ch3=0;
  Uint32 Sum_ch4=0;
  Uint32 Sum_ch5=0;
  Uint32 Sum_ch6=0;
  Uint32 Sum_ch7=0;
  Uint32 Sum_ch8=0;
  Uint32 Sum_ch9=0;
  Uint32 Sum_ch10=0;
  Uint32 Sum_ch11=0;
  //Uint32 Sum_ch12=0;
  //Uint32 Sum_ch13=0;
  //Uint32 Sum_ch14=0;
  //Uint32 Sum_ch15=0;

      for(i = 0; i<AVE_SAMPLE; i++)
  {
   Sum_ch0+=Sample_ch0[i];
   Sum_ch1+=Sample_ch1[i];
   Sum_ch2+=Sample_ch2[i];
   Sum_ch3+=Sample_ch3[i];
   Sum_ch4+=Sample_ch4[i];
   Sum_ch7+=Sample_ch7[i];

   Sum_ch9+=Sample_ch9[i];
   Sum_ch10+=Sample_ch10[i];
   Sum_ch11+=Sample_ch11[i];
       //Sum_ch12+=Sample_ch12[i];
       //Sum_ch13+=Sample_ch13[i];
       //Sum_ch14+=Sample_ch14[i];
       //Sum_ch15+=Sample_ch15[i];
      }
    for(i = 0; i<AVE_SAMPLE_DC; i++)
      {
       Sum_ch5+=Sample_ch5[i];
       Sum_ch6+=Sample_ch6[i];
       Sum_ch8+=Sample_ch8[i];
      }
/***************************************/

     Ia2=Ia1;
     Ia1=Ia;
     Ic2=Ic1;
     Ic1=Ic;
   /*Chanel 1*/
     Ia=Sum_ch0 - (AD_BIAS0<<(AVE_SHFT));
     Ia=(Ia<<(18-AVE_SHFT));      //(_iq18 format)
     Ia=Ia/2035; /*2023 for Iag,2035 for Ia*/
     Ia=-Ia*50; /*current sensor 5:1 */

   /*Chanel 2*/
     Ic=Sum_ch1 - (AD_BIAS1<<(AVE_SHFT));
         Ic=(Ic<<(18-AVE_SHFT)); //(_iq18 format)
         Ic=Ic/2050; /*2002 for Icg,2050 for Ic*/
         Ic=-Ic*50; /*current sensor 5:1 */

       /*Chanel 3*/
     Vab=Sum_ch2 - (AD_BIAS2<<AVE_SHFT);
     Vab=(Vab<<(18-AVE_SHFT));          //(_iq18
format)
     Vab=Vab/2100; /*DSP ADC trim: 5V=3159 -
5V=1039*/
     Vab=Vab*300; /*Voltage sensor 30:1 */

       /*Chanel 4*/
     Vbc=Sum_ch3 - (AD_BIAS3<<AVE_SHFT);
     Vbc=(Vbc<<(18-AVE_SHFT)); // (_iq18 format)
     Vbc=Vbc/2098;/*DSP ADC trim: 5V=3156 -
5V=1042*/
     Vbc=Vbc*300; /*Voltage sensor 30:1 */

       /*Chanel 5*/
     Vca=Sum_ch4 - (AD_BIAS4<<AVE_SHFT);
     Vca=(Vca<<(18-AVE_SHFT));          // (_iq18
format)
     Vca=Vca/2132; /*DSP ADC trim: 5V=3170 -
5V=1026*/
     Vca=Vca*300; /*Voltage sensor 30:1*/

       /*Chanel 6*/
     Vdc_P = Sum_ch5 -
(AD_BIAS5<<(AVE_SHFT_DC));
```

```
    Vdc_P = (Vdc_P<<(18-AVE_SHFT_DC)); //
        (_iq18 format)
    Vdc_P=Vdc_P/1927;          /*DSP ADC trim:
5V=1041 -5V=3158*/
    Vdc_P=-Vdc_P*300;    ///* sensor 30:1*/*/

        /*Chanel 7*/
    Vdc_N = Sum_ch6 -
(AD_BIAS6<<(AVE_SHFT_DC));
    Vdc_N = (Vdc_N<<(18-AVE_SHFT_DC)); //(_iq18
format)
    Vdc_N=Vdc_N/2043;          /*DSP ADC trim:
5V=1015 -5V=3130*/
    Vdc_N=-Vdc_N*300;    ///* sensor 30:1*/*/
    Vdc0=Vdc;
    Vdc=Vdc_P+Vdc_N;

  /*Chanel 12*/
    Iarm=Sum_ch7 - (AD_BIAS7<<(AVE_SHFT));
    Iarm=(Iarm<<(18-AVE_SHFT));          //(_iq18
format)
    Iarm=Iarm/2050;
    Iarm=-Iarm*50; /*current sensor 5:1 */

  /*Chanel 13*/
    If=Sum_ch8 - (AD_BIAS8<<(AVE_SHFT_DC));
        If=(If<<(18-AVE_SHFT_DC)); //(_iq18
format)
        If=If/2050;
        If=-If*50; /*current sensor 5:1 */
        If+=_IQ18(0.14);

        /*Chanel 14*/
    Eab=Sum_ch9 - (AD_BIAS9<<AVE_SHFT);
    Eab=(Eab<<(18-AVE_SHFT));            //(_iq18
format)
    Eab=Eab/2100; /*DSP ADC trim: 5V=3159 -
5V=1039*/
    Eab=Eab*300; /*Voltage sensor 30:1 */

        /*Chanel 15*/
    Ebc=Sum_ch10 - (AD_BIAS10<<AVE_SHFT);
    Ebc=(Ebc<<(18-AVE_SHFT)); // (_iq18 format)
    Ebc=Ebc/2100;/*DSP ADC trim: 5V=3156 -
5V=1042*/
    Ebc=Ebc*300; /*Voltage sensor 30:1 */

        /*Chanel 9*/
    Eca=Sum_ch11 - (AD_BIAS11<<AVE_SHFT);
    Eca=(Eca<<(18-AVE_SHFT));// (_iq18 format)
    Eca=Eca/2100; /*DSP ADC trim: 5V=3170 -
5V=1026*/
    Eca=Eca*300; /*Voltage sensor 30:1*/

 /*Low pass filtering for Vdc*/
    Vdc_L0=Vdc_L;
    Vdc_L= _IQ18mpyIQX(LPFa_DC,28,
(Vdc0+Vdc),18);
    Vdc_L= _IQ18mpyIQX(LPFb_DC,28,
Vdc_L0,18)+Vdc_L;

  /*Low pass filtering for If*/
    If_L0=If_L;
    If_L= _IQ18mpyIQX(LPFa_DC,28, (If0+If),18);
    If_L+= _IQ18mpyIQX(LPFb_DC,28, If_L0,18);
        If0=If;
  /*Electrical torque of DC motor*/
    torque_motor=_IQ18mpy((Iarm-Io), _IQ18(Kf));

} /*End of Get_adc_V*/

/*************************************/
/*  NAME:       abc_dq_transformation() */
/*  DESCRIPTION: abc to dq transformation */
/*************************************/

void abc_dq_transformation(void)
{
   Theta_pu = (long)Theta_con<<16;
   Theta_pu = Theta_pu/nmax;
   Theta_pu =_IQ28(1)- (Theta_pu<<12);   /*PLL is in
counting down mode*/
   Theta_line = _IQ28mpy(_IQ28(2*PI), Theta_pu);
/*Va phase angle*/

   Theta_phase = Theta_line- _IQ28(PI/2); /*angle for
DQ transformation */

   if ( Theta_phase > _IQ28(2*PI) )
   Theta_phase = Theta_phase - _IQ28(2*PI);
   if ( Theta_line < 0 )
    Theta_phase = Theta_phase + _IQ28(2*PI);

   sin_theta = _IQ28sin(Theta_phase);
   cos_theta = _IQ28cos(Theta_phase);

   /*Convert voltage from a-b-c to d-q*/
   Va = (Vab - Vca)/3;
   Vb = (Vbc - Vab)/3;
   Vc = (Vca - Vbc)/3;

   V_alpha = Va;
   V_beta = _IQ18mpy(_IQ18(sqrt13), (Vb - Vc));

   V_d0=V_d;
   V_q0=V_q;
   V_d  = _IQ18mpyIQX(cos_theta, 28, V_alpha, 18);
   V_d += _IQ18mpyIQX(sin_theta, 28, V_beta, 18);
   V_q  = _IQ18mpyIQX(-sin_theta, 28, V_alpha, 18);
   V_q += _IQ18mpyIQX(cos_theta, 28, V_beta, 18);

   Va = (Eab - Eca)/3;
   Vb = (Ebc - Eab)/3;
   Vc = (Eca - Ebc)/3;

   V_alpha = Va;
   V_beta = _IQ18mpy(_IQ18(sqrt13), (Vb - Vc));

   E_d  = _IQ18mpyIQX(cos_theta, 28, V_alpha, 18);
   E_d += _IQ18mpyIQX(sin_theta, 28, V_beta, 18);
   E_q  = _IQ18mpyIQX(-sin_theta, 28, V_alpha, 18);
   E_q += _IQ18mpyIQX(cos_theta, 28, V_beta, 18);

   /*High pass filtering for active damping*/
   V_d_L0=V_d_L;
   V_q_L0=V_q_L;
   V_d_L= _IQ18mpyIQX(LPFa,28, ( V_d0+V_d),18);
   V_d_L+= _IQ18mpyIQX(LPFb,28, V_d_L0,18);
   V_q_L= _IQ18mpyIQX(LPFa,28, ( V_q0+V_q),18);
   V_q_L+= _IQ18mpyIQX(LPFb,28, V_q_L0,18);
   V_d_H=V_d-V_d_L;
   V_q_H=V_q-V_q_L;

   V_d_H=_IQ18mpy(_IQ18(0.005), V_d_H);//for
active damping
         V_q_H=_IQ18mpy(_IQ18(0.005), V_q_H);

   /*Convert current from a-b-c to d-q*/
   switch (sen_posi)
   {
    case 0:
    Ib = -Ia-Ic;

    I_alpha = Ia;
    I_beta = _IQ18mpy(_IQ18(sqrt13), (Ib - Ic));

    I_d  = _IQ18mpyIQX(cos_theta, 28, I_alpha, 18);
    I_d += _IQ18mpyIQX(sin_theta, 28, I_beta, 18);
    I_q  = _IQ18mpyIQX(-sin_theta, 28, I_alpha, 18);
    I_q += _IQ18mpyIQX(cos_theta, 28, I_beta, 18);

    Ib_g = -Ia_g-Ic_g;

    I_alpha = Ia_g;
    I_beta = _IQ18mpy(_IQ18(sqrt13), (Ib_g - Ic_g));

    Ig_d = _IQ18mpyIQX(cos_theta, 28, I_alpha, 18);
    Ig_d += _IQ18mpyIQX(sin_theta, 28, I_beta, 18);
    Ig_q = _IQ18mpyIQX(-sin_theta, 28, I_alpha, 18);
    Ig_q += _IQ18mpyIQX(cos_theta, 28, I_beta, 18);
    break;
    case 1:
          Ib_g = -Ia_g-Ic_g;

    I_alpha = Ia_g;
    I_beta = _IQ18mpy(_IQ18(sqrt13), (Ib_g - Ic_g));

    Ig_d = _IQ18mpyIQX(cos_theta, 28, I_alpha, 18);
    Ig_d += _IQ18mpyIQX(sin_theta, 28, I_beta, 18);
    Ig_q = _IQ18mpyIQX(-sin_theta, 28, I_alpha, 18);
    Ig_q += _IQ18mpyIQX(cos_theta, 28, I_beta, 18);
    I_d=Ig_d;
    I_q=Ig_q;
    break;
   }
}

/*****************************************/
/* NAME:       Regulation()           */
/* DESCRIPTION: Converter side control   */
/*****************************************/

void Regulation(void)
{

         /* Outer loop voltage PI control*/
   Vdc_err = Vdc_ref-Vdc_L;
         kp_dc=_IQ28(Kp_Udc);      //kp=0.5
   ki_dc=_IQ28(Ki_Udc/Sw_freq);//ki/fs=45/9k=0.005
   if ((Vdc_err<_IQ18(1.2))&&(Vdc_err>_IQ18(-1.2)))

{kp_dc=_IQ28(Kp_Udc/4);ki_dc=_IQ28(Ki_Udc/Sw_fr
eq/4);Res_flag=1;}
         //if ((Vdc_err<_IQ18(-
5))&&(Vdc_err>_IQ18(5)))
   //{kp_dc=kp_dc<<1;ki_dc=ki_dc<<1;Res_flag=0;}

   Vdc_err1 += _IQ18mpyIQX( ki_dc, 28, Vdc_err, 18);
         if (Vdc_err1>upperlimit_Id)
Vdc_err1=upperlimit_Id;
   if (Vdc_err1<_IQ18(-15)) Vdc_err1=_IQ18(-15);
         I_d_ref= _IQ18mpyIQX( kp_dc, 28, Vdc_err,
18)+Vdc_err1;
   if (test_flag1==1) I_d_ref=_IQ18(13.7);
   if (I_d_ref>upperlimit_Id) I_d_ref=upperlimit_Id;
   if (I_d_ref<_IQ18(-15)) I_d_ref=_IQ18(-15);

         /* Inner loop current PI-Res control*/
   I_d_in2 = I_d_in1;
   I_d_in1 = I_d_err;
   I_d_out2 = I_d_out1;
   I_d_out1 = I_d_out;
   I_d_err = I_d-I_d_ref;
   //if (Res_flag==1)
   //{I_d_out= _IQ28mpyIQX(Res_6a,28, (I_d_err-
I_d_in2),18);
   // I_d_out=I_d_out-I_d_out2-
_IQ28mpyIQX(Res_6b,28, I_d_out1,18);}//resonant
regulator

   I_d_err1 += _IQ18mpyIQX( ki_I, 28, I_d_err, 18);
   if (I_d_err1>_IQ18(0.97)) I_d_err1=_IQ18(0.97);
   if (I_d_err1<_IQ18(0.3)) I_d_err1=_IQ18(0.3);
```

```
        V_d_O = _IQ18mpyIQX(kp_I, 28, I_d_err,
18)+I_d_err1;
   if (V_d_O>_IQ18(0.97)) V_d_O=_IQ18(0.97);
   if (V_d_O<_IQ18(0.3)) V_d_O=_IQ18(0.3);

   V_d_O += _IQ18mpyIQX(_IQ28( XL), 28, I_q, 18);
   //if (Res_flag==1) V_d_O +=I_d_out>>10;//add
resonant regulator output
   //if (display_counter1<27)
   V_d_O += V_d_H;//for active damping
        if (V_d_O>_IQ18(0.97)) V_d_O=_IQ18(0.97);
   if (V_d_O<_IQ18(0.3)) V_d_O=_IQ18(0.3);

   I_q_in2 = I_q_in1;
   I_q_in1 = I_q_err;
   I_q_out2 = I_q_out1;
   I_q_out1 = I_q_out;
   I_q_err = I_q-I_q_ref;
   //if (Res_flag==1)
   //{I_q_out= _IQ28mpyIQX(Res_6a,28, (I_q_err-
I_q_in2),18);
  // I_q_out=I_q_out-I_q_out2-
_IQ28mpyIQX(Res_6b;28, I_q_out1,18);}//resonant
regulator

   I_q_err1 += _IQ18mpyIQX( ki_I, 28, I_q_err, 18);
   if (I_q_err1>_IQ18(0.4)) I_q_err1=_IQ18(0.4);
   if (I_q_err1<_IQ18(-0.4)) I_q_err1=_IQ18(-0.4);
        V_q_O = _IQ18mpyIQX( kp_I, 28, I_q_err,
18)+I_q_err1;
   V_q_O += _IQ18mpyIQX(_IQ28(XL), 28, (-I_d), 18);
   //if (Res_flag==1) V_q_O +=I_q_out>>10;//add
resonant regulator output
   //if (display_counter1<27)
   V_q_O += V_q_H;//for active damping
   if (V_q_O>_IQ18(0.4)) V_q_O=_IQ18(0.4);
   if (V_q_O<_IQ18(-0.4)) V_q_O=_IQ18(-0.4);

        /* for test*/
   //V_d_O=_IQ18(0.8);
        //V_q_O=_IQ18(-0.3);

   //if ((E_d<_IQ18(80))&&(graph_index_flag==0))

//{graph_index_flag=1;data_shift_times=graph_index;}//
trigger for supply voltage sag

} /*End of Regulation*/

/*************************************/
/* NAME:      Turbine_simulator()   */
/* DESCRIPTION: Generate torque according to wind
speed and turbine speed */
/*************************************/

void Turbine_simulator(void)
{
 int16 x,y;
 //int32
wind_speed_pu,generator_speed_pu,Pwind,lambda_inv,
e_lambda,aa,bb;

 //wind_speed=wind_profile[wind_index];
 wind_index1++;
 if (wind_index1==900)
 {
  wind_index++;
  if (wind_index==2048){wind_index=0;}
  wind_index1=0;
 }


wind_speed_pu=(wind_speed*_IQ28(1.0/BaseWindSpee
d));//Q=28
 Pwind=_IQ28mpy(wind_speed_pu,wind_speed_pu);
 Pwind=_IQ28mpy(wind_speed_pu,Pwind);


//generator_speed_pu=_IQ28mpyIQX(veloc_y,18,_IQ28
(1.0/BaseGeneratorSpeed/BaseRotationalSpeed),28);

generator_speed_pu=_IQ28mpyIQX(speed_test,18,_IQ2
8(1.0/BaseGeneratorSpeed/BaseRotationalSpeed),28);//f
or test
 x= wind_speed_pu>>14;
 lambda=generator_speed_pu/x;//Q=14
 lambda=lambda*130;//Q=18

 x=beta*82; //Q=10
 x+=lambda>>8;
 y=beta*beta*beta+1;
 aa=(_IQ28(0.035)/y)>>10;//Q=18
 lambda_inv=_IQ28(1)/x-aa;//Q=18
 aa=lambda_inv*60-(_IQ18(0.20704)*beta)-
_IQ18(2.588);

 bb=lambda_inv*21;
 x=bb>>17;
 switch(x)
 {
  case 0:
  e_lambda=_IQ18(0.6065);
  break;
  case 1:
  e_lambda=_IQ18(0.6065)-_IQ18mpy((bb-
_IQ18(0.5)),_IQ19(0.23862));
  break;
  case 2:
  e_lambda=_IQ18(0.36788)-_IQ18mpy((bb-
_IQ18(1)),_IQ19(0.14475));
  break;
  case 3:
```

```
 e_lambda=_IQ18(0.22313)-_IQ18mpy((bb-
_IQ18(1.5)),_IQ19(0.08779));
 break;
 case 4:
 e_lambda=_IQ18(0.13534)-_IQ18mpy((bb-
_IQ18(2)),_IQ19(0.053255));
 break;
 case 5:
 e_lambda=_IQ18(0.082085)-_IQ18mpy((bb-
_IQ18(2.5)),_IQ19(0.032298));
 break;
 case 6:
 e_lambda=_IQ18(0.049787)-_IQ18mpy((bb-
_IQ18(3)),_IQ19(0.01959));
 break;
 case 7:
 e_lambda=_IQ18(0.030197)-_IQ18mpy((bb-
_IQ18(3.5)),_IQ19(0.011881));
 break;
 case 8:
 e_lambda=_IQ18(0.018316)-_IQ18mpy((bb-
_IQ18(4)),_IQ19(0.007207));
 break;
 case 9:
 e_lambda=_IQ18(0.01111)-_IQ18mpy((bb-
_IQ18(4.5)),_IQ19(0.004371));
 break;
 case 10:
 e_lambda=_IQ18(0.006738)-_IQ18mpy((bb-
_IQ18(5)),_IQ19(0.002649));
 break;
 case 11:
 e_lambda=_IQ18(0.004089)-_IQ18mpy((bb-
_IQ18(5.5)),_IQ19(0.001610));
 break;
}
x=bb>>18;
if (x>=10) e_lambda=0;
switch(x)
{
 case 6:
 e_lambda=_IQ18(0.002479)-_IQ18mpy((bb-
_IQ18(6)),_IQ18(0.001567));
 break;
 case 7:
 e_lambda=_IQ18(0.000912)-_IQ18mpy((bb-
_IQ18(7)),_IQ18(0.000577));
 break;
 case 8:
 e_lambda=_IQ18(0.000335)-_IQ18mpy((bb-
_IQ18(8)),_IQ18(0.000212));
 break;
 case 9:
 e_lambda=_IQ18(0.0001234)-_IQ18mpy((bb-
_IQ18(9)),_IQ18(0.000078));
 break;
```

```
 }

Cp_pu=_IQ18mpy(aa,e_lambda)+_IQ18mpy(_IQ18(0.0
068),lambda);
 Cp_pu=_IQ18mpy(Cp_pu,_IQ18(1/0.48));

 Pwind=_IQ28mpyIQX(Pwind,28,Cp_pu,18);

torque_turbine_pu=Pwind/(generator_speed_pu>>14);//p
u of torque_turbine, Q=14

torque_turbine=_IQ18mpyIQX(torque_turbine_pu,14,_I
Q18(BasePower*Kp/BaseGeneratorSpeed/BaseRotation
alSpeed),18);

}

/*************************************/
/* NAME:      Inertia_compensation() */
/* DESCRIPTION: compensate the inertia of the motor
*/
/*************************************/
void Inertia_compensation(void)
{
 torque_G0=torque_G;
 torque_G=torque_motor-
_IQ18mpyIQX(accele_L,10,_IQ28(J_M),28);

  torque_G_L0=torque_G_L;
  torque_G_L= _IQ18mpyIQX(LPFa_torque,28,
(torque_G0+torque_G),18);
  torque_G_L+= _IQ18mpyIQX(LPFb_torque,28,
torque_G_L0,18);

 torque_turbine+=_IQ18(-0.25)+
_IQ18mpy(veloc_y,_IQ18(-0.0015));//plus friction
torque
 torque_motor_ref=_IQ18mpy(torque_G_L,_IQ18(1-
(J_M/J_T)))+_IQ18mpy(torque_turbine,_IQ18(J_M/J_T)
);

Iarm_ref=_IQ18mpy(torque_motor_ref,_IQ18(1/Kf))+Io
;
}

/*************************************/
/* NAME:      Motor_drive()       */
/* DESCRIPTION: inverter side control */
/*************************************/

void Motor_drive(void)
{
  Uint32 a,b;
  // if (veloc_y>_IQ18(Speed_max)&&
INTERRUPT==2)
{INERTIA_TEST=1;Disable_Inv();Iarm_ref=0;INTERR
UPT=1; }
  /*inner armature current control loop */

  Iarm_err = Iarm_ref-Iarm;
  Iarm_err1 += _IQ18mpyIQX( ki_Iarm, 28, Iarm_err,
18);
  if (Iarm_err1>_IQ18(0.95)) Iarm_err1=_IQ18(0.95);
  if (Iarm_err1<_IQ18(0.05)) Iarm_err1=_IQ18(0.05);
  D_inv = _IQ18mpyIQX(kp_Iarm, 28, Iarm_err,
18)+Iarm_err1;
  if (D_inv>_IQ18(1)) D_inv=_IQ18(1);
  if (D_inv<_IQ18(0.5)) D_inv=_IQ18(0.5);

  a= Ts>>2;
  a= D_inv*a;
  a= a>>16;
  b= (Ts>>1)-2000;//-2000 for If=0.8A
 // b= D_inv1*b;
 // b= b>>16;
  EvaRegs.CMPR3 = a;//Phase A//armature +
  EvaRegs.CMPR2 = Ts-a;//Phase B //armature-
  EvaRegs.CMPR1 = b;//Phase C //field
}

/*************************************/
/*  NAME:      Speed_control()        */
/*  DESCRIPTION:  control the motor speed*/
/*************************************/

void Speed_control(void)
{
 /*outter speed control loop */
  Speed_w_err=Speed_w_ref-veloc_y;
  Speed_w_err1+=_IQ18mpyIQX( _IQ28(0.5/9000), 28,
Speed_w_err, 18);
  if (Speed_w_err1>_IQ18(15))
Speed_w_err1=_IQ18(15);
  if (Speed_w_err1<_IQ18(0.1))
Speed_w_err1=_IQ18(0.1);
  Iarm_ref= _IQ18mpyIQX(_IQ28(0.6), 28,
Speed_w_err, 18)+Speed_w_err1;
  if (Iarm_ref>_IQ18(15)) Iarm_ref=_IQ18(15);
  if (Iarm_ref<_IQ18(0.1)) Iarm_ref=_IQ18(0.1);

}

/*************************************/
/*  NAME:      Inertia_test()         */
/*  DESCRIPTION:  test the inertia of the motor*/
/*************************************/
void Inertia_test(void)
{
 int32 a,b,c,d;

 if (veloc_y>_IQ18(Speed_max)&& INTERRUPT==2)
{INERTIA_TEST=1;Disable_Inv();INTERRUPT=1; }

 switch (INERTIA_TEST)
  {
   case 0:
          if (INTERRUPT==2)
          {
    if (veloc_y>_IQ18(Speed_min)) {T_CNT1++;if
(T_CNT1==0) T_CNT0++; }
          else T_CNT1=0;
   }
   break;
          case 1:
   if (veloc_y>_IQ18(Speed_min))
    {
    T_CNT2++;
          if (T_CNT2==0) T_CNT++;
          }
   else
    {
    a=_IQ28(Iarm_test*Kf);
    b=(T_CNT0*65536+T_CNT1)>>1;
    c=(T_CNT*65536+T_CNT2)>>2;
    c=_IQ29(1)/b+(_IQ28(1)/c);
          d=_IQ10mpyIQX(c,30,_IQ10((Speed_max-
Speed_min)*Sw_freq),10);
    inertia=a/d;

          T_CNT=0;
    T_CNT0=0;
    T_CNT1=0;
    T_CNT2=0;
    INERTIA_TEST=2;
    }

   break;
  }
}

/*************************************/
/* NAME:      Performance_test()    */
/* DESCRIPTION:  test the inertia of the motor*/
/*************************************/
 void performance_test(void)
{

 if (veloc_y>_IQ18(Speed_max)&& INTERRUPT==2)
{test_flag=1; }
 switch (test_flag)
  {
  case 0:
  torque_turbine=_IQ18(Iarm_test1*Kf);
          Inertia_compensation();
  if (veloc_y<_IQ18(1))  Iarm_ref=_IQ18(6);
```

```
   if (J_COM_flag==0) Iarm_ref=_IQ18(Iarm_test1)+Io;
   break;
   case 1:
   torque_turbine=0;
   Inertia_compensation();
   if (J_COM_flag==0) Iarm_ref=0;
   break;
   }
}

/************************************/
/* NAME:      SVPWM_7S()      */
/************************************/

void SVPWM_7S(void)
 {
   _iq28 T1_sin,T2_sin;
   Uint16 comp1,comp2,comp3,comp4;

   Mmag= _IQ18mag(V_q_O,V_d_O);//PU value
          if (Mmag> _IQ18(0.98)) Mmag=_IQ18(0.98);
   Mmag= Mmag*(Ts>>2);//count value of every half
PWM period

   Theta=_IQ28atan2(V_q_O,V_d_O);
   Theta+=Theta_phase;//Phase for SVPWM

   if ( Theta > _IQ28(2*PI) )
   Theta = Theta - _IQ28(2*PI);
   if ( Theta < 0 )
   Theta = Theta + _IQ28(2*PI);

   section_number=Theta/_IQ28(PI/3);
   Theta_sec= Theta-
_IQ28mpyI32(_IQ28(PI/3),(long)(section_number));

          T1_sin= _IQ28sin(_IQ28(PI/3)-Theta_sec);
   T2_sin= _IQ28sin(Theta_sec);
          T1= _IQ18mpyIQX(T1_sin, 28, Mmag,
18)>>16;// T1/2
   T2= _IQ18mpyIQX(T2_sin, 28, Mmag, 18)>>16;//
T2/2
   T0=Ts-T1-T2;//// T0/2
   comp1=T0>>1;
          comp2=comp1+T1;
   comp4=comp1+T2;
          comp3=comp2+T2;
   switch (section_number)
   {
    case 0:

      EvbRegs.CMPR4 =  comp1;
      EvbRegs.CMPR5 =  comp2;
      EvbRegs.CMPR6 =  comp3;
            break;
    case 1:

      EvbRegs.CMPR4 =  comp4;
      EvbRegs.CMPR5 =  comp1;
      EvbRegs.CMPR6 =  comp3;
            break;
    case 2:

      EvbRegs.CMPR4 =  comp3;
      EvbRegs.CMPR5 =  comp1;
      EvbRegs.CMPR6 =  comp2;
            break;
    case 3:

      EvbRegs.CMPR4 =  comp3;
      EvbRegs.CMPR5 =  comp4;
      EvbRegs.CMPR6 =  comp1;
            break;
    case 4:

      EvbRegs.CMPR4 =  comp2;
      EvbRegs.CMPR5 =  comp3;
      EvbRegs.CMPR6 =  comp1;
            break;
    case 5:

      EvbRegs.CMPR4 =  comp1;
      EvbRegs.CMPR5 =  comp3;
      EvbRegs.CMPR6 =  comp4;
            break;
   }
 }

/************************************/
/* NAME:
Enable_Rec(),Disable_Rec(),Enable_Inv(),Disable_Inv()
*/
/* DESCRIPTION: Enable or disable PWM output
*/
/************************************/

 void Enable_Rec(void)
 {
  EvbRegs.COMCONB.bit.FCOMPOE=1;
 } /*End of Enable_Rec*/

 void Disable_Rec(void)
 {
  EvbRegs.COMCONB.bit.FCOMPOE=0;
 }
 void Enable_Inv(void)
 {
  EvaRegs.CMPR1 = 0x0000;
  EvaRegs.CMPR2 = 0x0000;
  EvaRegs.CMPR3 = 0x0000;
  EvaRegs.COMCONA.bit.FCOMPOE=1;
 }
 void Disable_Inv(void)
 {
  EvaRegs.COMCONA.bit.FCOMPOE=0;
 }
/************************************/
/* NAME:      Close_Relay(), Open_Relay*/
/* DESCRIPTION: Open and close relay for protection
*/
/************************************/

void Close_Relay(void)
{
 Uint16 i;
 GpioDataRegs.GPBCLEAR.bit.GPIOB12 = 1;//grid
side charging resistor by-pass relay open
 GpioDataRegs.GPACLEAR.bit.GPIOA7 = 1;//motor
side charging resistor by-pass relay open
 Delay_mili_second(200);
 GpioDataRegs.GPBDAT.bit.GPIOB7 = 1;//grid side
power relay close
          for (i=0; i<10; i++)
          {
          Delay_mili_second(100);
          LED_display();
          }
 GpioDataRegs.GPBDAT.bit.GPIOB12 = 1;//grid side
charging resistor by-pass relay close
 GpioDataRegs.GPADAT.bit.GPIOA7 = 1;//motor side
charging resistor by-pass relay close
 CONTACTOR = 1;
}

void Open_Relay(void)
{
 GpioDataRegs.GPBCLEAR.bit.GPIOB7 = 1;//grid side
power relay open
 CONTACTOR = 0;
}

/*********************************/
/* NAME:      Key_input()         */
/* DESCRIPTION: LCD display and push button
command                */
/*********************************/

void Key_input(void)
{
// key_value = *KEY_IN;

          if(key_value)
          {
//          key_monitor++;
//          if(key_monitor==300)
//          {
//           key_monitor1++;key_monitor=0;
//          }
//          if(key_monitor1==1000)
            {
//             key_monitor1 = 0;
//             key_monitor = 0;
//             *KEY_IN |= 0xf;

    switch (key_value)
         {

            case 1:
       if(CONTACTOR==0)
                  { SCREEN=1;Close_Relay
();Display_Relay_Close();*FAN_CONFIG = 0x1F;}
       else
         switch (INTERRUPT)
                         {
                         case 0:
                         Enable_Rec();
         INTERRUPT = 1;
                         Display_ENB_REC();
         break;
         case 1:
         //while(graph_index!=540){}
                         graph_index_flag=1;
         INERTIA_TEST=0;
         Enable_Inv();INTERRUPT = 2;
         Display_ENB_INV();
         break;
         case 2:
         //while(graph_index!=540){}
         Disable_Inv();
         INTERRUPT = 1;
         Display_DISAB_INV();
         break;
                         }
                break;

            case 2:
       switch (INTERRUPT)
                         {
                         case 0:
                         key1++; if
(key1>9){key1=0;}
         break;
         case 1:
         key1++; if (key1>9){key1=0;}
         break;
         case 2:
         Iarm_ref= Iarm_ref+ _IQ18(0.5);
                         if (Iarm_ref> _IQ18(20))
Iarm_ref= _IQ18(20);
       // Speed_w_ref=Speed_w_ref+_IQ18(10);
       // if (Speed_w_ref> _IQ18(180)) Speed_w_ref=
_IQ18(180);
         break;
                         }
                break;

                case 4:
            switch (INTERRUPT)
                                {
                                case 0:
                                key1--;
                                 if
 (key1<0||key1>9){key1=9;}
            break;
            case 1:
            key1--;
                                 if
 (key1<0||key1>9){key1=9;}
            break;
            case 2:
            Iarm_ref= Iarm_ref- _IQ18(0.5);
                                if (Iarm_ref< _IQ18(-5))
 Iarm_ref= _IQ18(-5);
        // Speed_w_ref=Speed_w_ref-_IQ18(10);
        // if (Speed_w_ref< _IQ18(0.1)) Speed_w_ref=
 _IQ18(0.1);
        break;
                                }
                break;

            case 8:
        if (CONTACTOR==1)
                        {
                        Disable_Rec(); INTERRUPT =
 0;Open_Relay (); SCREEN=0;
                        test_flag1=0;Iarm_ref= 0;
        Delay_mili_second(500);Disable_Inv();
 D_inv=_IQ18(0.5);
                            if (data_shift_times>=2700)
 data_shift_times=data_shift_times-2700;
        if (data_shift_times<2700)
 data_shift_times=data_shift_times+6300;

//Buffer_rotating(graph_data,data_shift_times);
        //Buffer_rotating(graph_data1,data_shift_times);
        //Buffer_rotating(graph_data2,data_shift_times);
        //Buffer_rotating(graph_data3,data_shift_times);
        //Buffer_rotating(graph_data4,data_shift_times);
        //Buffer_rotating(graph_data5,data_shift_times);
        Manu_display();*FAN_CONFIG = 0x1D;

        }
                break;
                }
            }
        }
        key_value=0;//for control without keypad
} /*End of Key_input*/

/****************************************/
/* NAME:      AD_bias()          */
/* DESCRIPTION: ADC dc bias caliberation */
/****************************************/

 int AD_bias (unsigned int channel)
{
        int32 bias_data=0;
        Uint32 i;

        bias_data=0;
        for (i=0;i<65536;i++){
                while
(AdcRegs.ADCST.bit.INT_SEQ1== 0){}
    AdcRegs.ADCST.bit.INT_SEQ1_CLR = 1;        //
Clear INT SEQ1 bit
                switch (channel) {
                        case 0:

        bias_data+=AdcRegs.ADCRESULT0>>4;
                                break;
                        case 1:

        bias_data+=AdcRegs.ADCRESULT1>>4;
                                break;
                        case 2:

        bias_data+=AdcRegs.ADCRESULT2>>4;
                                break;
                        case 3:

        bias_data+=AdcRegs.ADCRESULT3>>4;
                                break;
                        case 4:

        bias_data+=AdcRegs.ADCRESULT4>>4;
                                break;
                        case 5:

        bias_data+=AdcRegs.ADCRESULT5>>4;
                                break;
                        case 6:

        bias_data+=AdcRegs.ADCRESULT6>>4;
                                break;
                        case 7:

        bias_data+=AdcRegs.ADCRESULT7>>4;
                                break;
                        case 8:

        bias_data+=AdcRegs.ADCRESULT8>>4;
                                break;
                        case 9:

        bias_data+=AdcRegs.ADCRESULT9>>4;
                                break;
```

```
                    case 10:

        bias_data+=AdcRegs.ADCRESULT10>>4;
                              break;
                    case 11:

        bias_data+=AdcRegs.ADCRESULT11>>4;
                              break;
                    case 12:

        bias_data+=AdcRegs.ADCRESULT12>>4;
                              break;
                    case 13:

        bias_data+=AdcRegs.ADCRESULT13>>4;
                              break;
                    case 14:

        bias_data+=AdcRegs.ADCRESULT14>>4;
                              break;
                    case 15:
                    default:

        bias_data+=AdcRegs.ADCRESULT15>>4;
                              break;
               }
        }
        bias_data=bias_data>>16;
        return bias_data;
} /*End of AD_bias*/


/****************************************/
/* NAME:      AD_bias16()        */
/* DESCRIPTION: LCD display and ADC dc bias
caliberation          */
/****************************************/

void AD_bias16(void)
{
        Text_LCD_clear_display();
        Text_LCD_printxy(0,1," Test the AD Bias");

        Text_LCD_printxy(0,2," Calculating CH 0");
        AD_BIAS0=AD_bias(0);//2059

        Text_LCD_printxy(0,2," Calculating CH 1");
        AD_BIAS1=AD_bias(1);//2066

        Text_LCD_printxy(0,2," Calculating CH 2");
        AD_BIAS2=AD_bias(2);//2053

        Text_LCD_printxy(0,2," Calculating CH 3");
        AD_BIAS3=AD_bias(3);//2059

        Text_LCD_printxy(0,2," Calculating CH 4");
        AD_BIAS4=AD_bias(4);//2059

        Text_LCD_printxy(0,2," Calculating CH 5");
        AD_BIAS5=2062;//AD_bias(5);

        Text_LCD_printxy(0,2," Calculating CH 6");
        AD_BIAS6=2055;//AD_bias(6);

        Text_LCD_printxy(0,2," Calculating CH 7");
        AD_BIAS7=AD_bias(7);//2047

        Text_LCD_printxy(0,2," Calculating CH 8");
        AD_BIAS8=AD_bias(8);//2050

        Text_LCD_printxy(0,2," Calculating CH 9");
        AD_BIAS9=AD_bias(9);//2055

        Text_LCD_printxy(0,2," Calculating CH 10");
        AD_BIAS10=AD_bias(10);//2051

        Text_LCD_printxy(0,2," Calculating CH 11");
        AD_BIAS11=AD_bias(11);//2047

        /*Text_LCD_printxy(0,2," Calculating CH
12");
        AD_BIAS12=AD_bias(12);

        Text_LCD_printxy(0,2," Calculating CH 13");
        AD_BIAS13=AD_bias(13);

        Text_LCD_printxy(0,2," Calculating CH 14");
        AD_BIAS14=AD_bias(14);

        Text_LCD_printxy(0,2," Calculating CH 15");
        AD_BIAS15=AD_bias(15);*/

        Text_LCD_printxy(0,2," Calculation DONE ");
   Delay_mili_second(100);
} /*End of AD_bias16*/

void Delay_mili_second (int delay)
{
        Uint16 i,j;
        i=delay*30;//150M Hz DSP clock
        for (j=0; j<1000; j++)
        {
         while (i){i--;};
   i=delay*30;
        }
}


/**************************************/
/* NAME:      Init_variable()      */
/* DESCRIPTION: Initialization of system control
variables          */
```

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

 void Init_variable(void)
 {
 /*ADC*/
 ConversionCount=0;

 /*PLL */
 Theta_con = 0;
 Theta_err = 0;
 nmax = 0;
 Theta_pu = 0; //phase in per unit value
 Theta_line = 0;   //phase in radian value (0-2pi)

 /*Low pass filter*/
 LPFa=_IQ28(1-2.0/(2.0+w_AD/Sw_freq));
 LPFb=_IQ28(1)-(LPFa<<1);
 LPFa_DC=_IQ28(1-2.0/(2.0+w_DC/Sw_freq));
 LPFb_DC=_IQ28(1)-(LPFa_DC<<1);
 LPFa_speed=_IQ28(1-2.0/(2.0+w_speed/Sw_freq));
 LPFb_speed=_IQ28(1)-(LPFa_speed<<1);
 LPFa_torque=_IQ28(1-2.0/(2.0+w_torque/Sw_freq));
 LPFb_torque=_IQ28(1)-(LPFa_torque<<1);

 /*PI controller*/
 kp_dc=_IQ28(Kp_Udc);     //kp=0.5
 ki_dc=_IQ28(Ki_Udc/Sw_freq);//ki/fs=45/9k=0.005
 kp_I=_IQ28(Kp_I);   //kp=0.04
 ki_I=_IQ28(Ki_I/Sw_freq);//ki/fs=30/9k=0.0033
 kp_Iarm=_IQ28(Kp_Iarm);   //kp=0.05

ki_Iarm=_IQ28(Ki_Iarm/Sw_freq);//ki/fs=3/9k=0.00033

 Vdc_ref=_IQ18( Udc_ref);
 Vdc_err =0;
 Vdc_err1 =0;

 I_d_ref= 0;
 I_d_err = 0;
 I_d_err1= 0;

 I_q_ref=0;//_IQ18(-0.3);
 I_q_err =0;
 I_q_err1 =0;

 V_d_O=_IQ18(0.8);
 V_q_O=0;

 Iarm_ref= 0;
 Iarm_err = 0;
 Iarm_err1= 0;
 Speed_w_ref=0;
 I_test=0;
 Io=_IQ18(0.2);
 /*Resonant regulator*/

 Res_6a=_IQ28(2*Sw_freq*K_res/(4.0*Sw_freq*Sw_fre
q+(2*PI*360)*(2*PI*360)));
  Res_6b=_IQ28(2*((2*PI*360)*(2*PI*360)-
4.0*Sw_freq*Sw_freq)/(4.0*Sw_freq*Sw_freq+(2*PI*3
60)*(2*PI*360)));

 /* Duty cycle of inverter*/
 D_inv=_IQ18(0.5);
 D_inv1=_IQ18(0.5);

 /*Motor angle close loop*/

//t1=0.01s,t2=0.0009s,G(s)=Ka_PLL*(1/s^2)*(1+s*t1)/(
1+s*t2);
 A_PLL=_IQ28((0.02*Sw_freq-1)/(0.02*Sw_freq+1));
 B_PLL=_IQ28((0.0018*Sw_freq-
1)/(0.0018*Sw_freq+1));

Ka_PLL=2*K_PLL*(0.0018*Sw_freq+1)/(0.02*Sw_fre
q+1);
 accele=0;
 veloc_x=0;
 veloc_y=0;
 angle_m=0;

 /*Turnine simulator*/
 beta=0;
 torque_turbine=0;
 /*for inertia test*/

 /* Key input*/
 key_value = 0;
 key_monitor = 0;
 key_monitor1 = 0;
 key1=0;

 /*system flags*/
 FUNCTION=0;
 J_COM_flag=0;
 CONTACTOR = 0;
 INTERRUPT = 0;
 FAULT = 0;
 PLL = 0;

 SCREEN = 0;
 display_counter=0;
 display_counter1=0;
} /*End of Init_variable*/

/**************************************/
/* NAME:      Gpio_select()       */
/* DESCRIPTION: GPIO configuration initialization
*/
/**************************************/
```

```
void InitGpio(void)
{
   // sets GPIO Muxs as I/Os
    EALLOW;

   GpioMuxRegs.GPAMUX.all= 0x077F;  // Configure
MUXs as digital I/Os or
   GpioMuxRegs.GPBMUX.all= 0x077F;  // peripheral
I/Os
   GpioMuxRegs.GPDMUX.all=0x0000;
   GpioMuxRegs.GPEMUX.all=0x0000;
   GpioMuxRegs.GPFMUX.all=0x0030;
   GpioMuxRegs.GPGMUX.all=0x0030;

   GpioMuxRegs.GPADIR.all=0xF8FF;  //
   GpioMuxRegs.GPBDIR.all=0xF8FF;  // GPIO DIR
select GPIOs as output or input
   GpioMuxRegs.GPDDIR.all=0x0000;
   GpioMuxRegs.GPEDIR.all=0x0000;
   GpioMuxRegs.GPFDIR.all=0x3FDF;
   GpioMuxRegs.GPGDIR.all= 0x0010;

   GpioMuxRegs.GPAQUAL.all=0x0000;  // Set GPIO
input qualifier values
   GpioMuxRegs.GPBQUAL.all=0x0000;
   GpioMuxRegs.GPDQUAL.all=0x000F;
   GpioMuxRegs.GPEQUAL.all=0x000F;

   EDIS;
} /*End of Gpio_select*/


/****************************************/
/*NAME: InitXintfClocks(),InitPeripherals()  */
/* DESCRIPTION: System  configuration   */
/****************************************/

void InitXintfClocks(void)
{
  XintfRegs.XTIMING0.all=0x000358AC;
  XintfRegs.XTIMING6.all=0x0003E746;//3D4A5;
  XintfRegs.XINTCNF2.all=0x00000007;
} /*End of InitXintfClocks*/

void InitPeripherals(void)
{
  //Text_LCD_initiate();
  InitAdc();
  InitEVA();
  InitEVB();
  init_fpga();
  Init_variable();
}

void InitEVA(void)
{
  // Configure EVA
  // EVA Clock is already enabled in InitSysCtrl();

  EvaRegs.EVAIMRA.bit.PDPINTA=1;//PDPINTA
protection enable
 // Enable compare for PWM1-PWM6
  EvaRegs.CMPR1 = 0x0000;
  EvaRegs.CMPR2 = 0x0000;
  EvaRegs.CMPR3 = 0x0000;
  // Compare action control.  Action that takes place
  EvaRegs.ACTRA.all = 0x0999;// active low
  EvaRegs.DBTCONA.all = 0x0FF4; // Enable deadband
3.4uS
  EvaRegs.COMCONA.all = 0xA000;//update twise

/* Initialize QEP circuits*/
  EvaRegs.CAPCONA.all = 0x9004;//disable capture
1,2;select timer2;detect rising edge for capture 3.

/* Initalize EVA Timer1*/
   EvaRegs.T1PR = Ts;//  Timer1 period (for motor side
switching frequency)
   EvaRegs.T1CNT = 0x0000;     // Timer1 counter
   // Timer enable
   EvaRegs.T1CON.all = 0x0800;  // (updowncount
mode) //input clock(HSPCLK)/1
   //EvaRegs.EVAIMRA.bit.T1UFINT=1; //Enable
Timer1 underflow interrupt

 /* Initalize EVA Timer2*/
   EvaRegs.T2CNT = 0x0000;
   EvaRegs.T2PR =14399; // period register (for 3600*4
pules every cycle of speed sensor)
   EvaRegs.T2CON.all = 0x1870; //(directional-
up/downcount mode)input clock from QEP

   EvaRegs.T1CON.bit.TENABLE=1;//Timer enable
}

void InitEVB(void)
{
  // Configure EVA
  // EVA Clock is already enabled in InitSysCtrl();
  EvbRegs.EVBIMRA.bit.PDPINTB=1;//PDPINTA
protection enable
 // Enable compare for PWM7-PWM12

  EvbRegs.CMPR4 = 0x0000;
  EvbRegs.CMPR5 = 0x0000;
  EvbRegs.CMPR6 = 0x0000;
  // Compare action control.  Action that takes place
  // on a cmpare event
  EvbRegs.ACTRB.all = 0x0666;//active high
  EvbRegs.DBTCONB.all = 0x0FF4; // Enable deadband
3.4uS
  //EvbRegs.COMCONB.all = 0x9000;//five segments
  EvbRegs.COMCONB.all = 0xA000;//0xA000;//seven
segments and update twice

/* Initalize the timers*/
/* Initalize EVB Timer3*/

  EvbRegs.T3PR =Ts;// Timer3 PWM period(for
computing fre of 8997Hz which is twice of rectifier
switching frequency)
  EvbRegs.T3CMPR = Ts>>1;   // Timer3 compare
  EvbRegs.T3CNT = 0x0000;   // Timer3 counter
  // TMODE = continuous up/down
  // Timer enable
  EvbRegs.T3CON.all = 0x0802; // (updowncount
mode) //input clock(HSPCLK)/1//timer compare enable
  EvbRegs.EVBIMRA.bit.T3CINT=1; //Enable Timer3
compare interrupt

 /* Initalize EVB Timer4*/
  EvbRegs.T4CNT = 0x0000;
  EvbRegs.T4PR =Ts>>4; // Setup period register (for
sampling feq of 8997Hz*8)
  EvbRegs.GPTCONB.bit.T4TOADC = 1;    // Enable
EVASOC in EVB (underflow starts ADC)
  EvbRegs.T4CON.all = 0x0880;   // (updowncount
mode) //input clock(HSPCLK)/1//start with T3

  EvbRegs.T3CON.bit.TENABLE=1;//Timer enable

  /* Initalize EVB Capture4&5*/
  EvbRegs.CAPCONB.all=0xA6F0;
  EvbRegs.EVBIMRC.all=0x0003;
  EvbRegs.EVBIFRC.all=0x0007;
}

/****************************************/
/* NAME:    InitAdc()        */
/* DESCRIPTION: ADC configuration, use Event
Manager A to start ADC     */
/****************************************/

void InitAdc(void)
{
        /*Start ADC*/
        AdcRegs.ADCTRL3.bit.ADCBGRFDN = 0x3;
        // Power up bandgap/reference circuitry
        Delay_micro_second (ADC_usDELAY);
// Delay before powering up rest of ADC
        AdcRegs.ADCTRL3.bit.ADCPWDN = 1;
        // Power up rest of ADC
        Delay_micro_second (ADC_usDELAY2);

 /*Configure ADC*/
  AdcRegs.ADCTRL1.bit.ACQ_PS = 0x1;//S/H width in
ADC module periods = 2 ADC clocks
  AdcRegs.ADCTRL3.bit.ADCCLKPS = 0x5; //ADC
module clock = HSPCLK/5 = 25MHz
  AdcRegs.ADCTRL1.bit.SEQ_CASC = 1;      // 1
Cascaded mode
  AdcRegs.ADCTRL1.bit.CONT_RUN = 0; //not
Continous running
 // AdcRegs.ADCTRL1.bit.SEQ_OVRD = 1;     //
Enable Sequencer override feature (bit 5)

  AdcRegs.ADCMAXCONV.all = 0x000B; // Setup 16
conv's on SEQ1
  AdcRegs.ADCCHSELSEQ1.bit.CONV00 = 0x1;
  AdcRegs.ADCCHSELSEQ1.bit.CONV01 = 0x2;
  AdcRegs.ADCCHSELSEQ1.bit.CONV02 = 0x3;
  AdcRegs.ADCCHSELSEQ1.bit.CONV03 = 0x4;
  AdcRegs.ADCCHSELSEQ2.bit.CONV04 = 0x5;
  AdcRegs.ADCCHSELSEQ2.bit.CONV05 = 0x6;
  AdcRegs.ADCCHSELSEQ2.bit.CONV06 = 0x7;
  AdcRegs.ADCCHSELSEQ2.bit.CONV07 = 0xC;
  AdcRegs.ADCCHSELSEQ3.bit.CONV08 = 0xD;
  AdcRegs.ADCCHSELSEQ3.bit.CONV09 = 0xE;
  AdcRegs.ADCCHSELSEQ3.bit.CONV10 = 0xF;
  AdcRegs.ADCCHSELSEQ3.bit.CONV11 = 0x9;
  //AdcRegs.ADCCHSELSEQ4.bit.CONV12 = 0x8;
  //AdcRegs.ADCCHSELSEQ4.bit.CONV13 = 0xE;
  //AdcRegs.ADCCHSELSEQ4.bit.CONV14 = 0xF;
  //AdcRegs.ADCCHSELSEQ4.bit.CONV15 = 0x9;

  //AdcRegs.ADCTRL2.bit.INT_ENA_SEQ1 = 1; //
enable SEQ1 interrupt (every EOS)
  AdcRegs.ADCTRL2.bit.EVB_SOC_SEQ = 1;       //
enable EVBSOC to start SEQ1

} /*End of InitAdc*/

/*************************************/
/* NAME:    init_fpga()       */
/* DESCRIPTION: Initialization of FPGA registers
*/
/*************************************/

void init_fpga(void)
{
  /*Hardware Protection*/
        *FAULT_CLEAR_A = 0x0;
        *FAULT_ENABLE_A = 0x1ff;

        *FAULT_CLEAR_B = 0x0;
        *FAULT_ENABLE_B = 0x1ff;

        /*Thermal Protection (disabled)*/
        *FAULT_THERMO_ENABLE = 0x0;

        *FAN_PERIOD = 14999;
        *FAN_COUNT = 0x0;
        *FAN_COMPARE_A = 0x0;
	*FAN_COMPARE_B = 12180;
	//*FAN_CONFIG = 0x1F;

	/*Fault signal distribution*/
	*FAULT_DISTRIBUTE = 0xffff;

	/*Zero-crossing detection (Zero-crossing starts
automatically)*/
   *PHASE_SHIFT_A = 0x4DDC;   // 0x4c2c ///*shift
input to 90degree delay*/
                             // 0xb090   /*shift input to
180degree delay*/
   //*PHASE_SHIFT_CONFIG_A = 0x0118;///*
divider=24+1,DPLL freq=Fclock/(divider+1); input
Vbc*/
   *PHASE_SHIFT_CONFIG_A =
0x0118;//divider=24+1,DPLL
freq=Fclock/(divider+1);Vbc //0x0418 input Vvw
	/*IO pin configuration*/
	*GPIOA6_CONFIG = 0x0A; /*A side PLL
output*/
	*GPIOB6_CONFIG = 0x0B; /*A side SYN
signal*/

	/*XINT interrupt configuration*/
	*XINT1_CONFIG = 0x0; /*Disable XINT1,
2*/
	*XINT2_CONFIG = 0x0;

	/*Key input Text LCD displace interface*/
	*KEY_IN = 0xf;//clear all the registered key
input

	*SWITCH_IN = 0x00;

	/*FPGA LED*/
	*FPGA_LED = 0x0;
 } /*End of Init_fpga*/

void Buffer_rotating(int16 buffer[9000],Uint16
shift_times)
 {
 int16 a;
 Uint16 i,j;

 for (i=shift_times;i>0;i--)
 {
  a=buffer[0];
  for (j=0; j<9000; j++)
  {buffer[j]=buffer[j+1];}
  buffer[8999]=a;
 }
 } /*End of Buffer_rotating*/


// End of WTS.c
```

References

[1] J. Marques, H. Pinheiro, H. A. Gründling, J. R. Pinheiro and H. L. Hey, 'A Survey on Variable-Speed Wind Turbine System', 2003, CE. 7° Congresso Brasileiro de Eletrônica de Potência - COBEP'03, Fortaleza, Brazil, 1, s. 732-738, 2003.

[2] L. H. Hansen, L., Helle, F. Blaabjerg, E. Ritchie, S. Munk-Nielsen, H. Bindner, P. Sørensen and B. Bak-Jensen, 'Conceptual survey of Generators and Power Electronics for Wind Turbines', Risø National Laboratory, Roskilde, Denmark, December 2001.

[3] L. Chang, R. Doraiswami, T. Boutot and H. Kojabadi, 'Development of a Wind Turbine Simulator for Wind Energy Conversion Systems Using an Inverter-Controlled Induction Motor', Energy Conversion, IEEE Transactions, Vol.9, No.3, September 2004.

[4] Luiz A. C. Lopes, Josselin Lhuilier*, Avishek Mukherjee and Mohammad F. Khokhar, 'A Wind Turbine Emulator that Represents the Dynamics of the Wind Turbine Rotor and Drive Train ,' Power Electronics Specialists Conference, 2005. PESC '05. IEEE 36th. 2005 Page(s): 2092-2097.

[5] Dale S. L. Dolan, P. W. Lehn, 'Real-Time Wind Turbine Emulator Suitable for Power Quality and Dynamic Control Studies', International Conference on Power Systems Transients (IPST'05), Montreal, Canada June 19-23, 2005, No. IPST05-074.

[6] Farret, F.A.; Gules, R.; Marian, J.; 'Micro-turbine simulator based on speed and torque of a DC motor to drive actually loaded generators' Devices, Circuits and Systems, 1995. Proceedings of the 1995 First IEEE International Caracas Conference on12-14 Dec. 1995 Page(s):89 - 93.

[7] Mariusz Malinowski, Marek Jasinski, Marian P. Kazmierkowski, " Simple Direct Power Control of Three-Phase PWM Recifier Using Space-Vector Modulation(DPC-SVM)", IEEE Transactions on industrial electronics, vol.51, No.2, April 2004, pp447-453.

[8] Guoqiao Shen, Dehong Xu, Danji Xi, Xiaoming Yuan, "An Improved Control Strategy for Grid-Connected Voltage Source Inverters with a LCL Filter", Applied Power

Electronics Conference and Exposition, 2006. APEC '06. Twenty-First Annual IEEE, 19-23 March 2006, pp7.-

[9] Eric Wu, Peter W.Lehn, “Digital Current Control of Voltage Source Converter with Active Damping of LCL Resonance” IEEE TRANSACTIONS ON POWER ELECTRONICS, VOL. 21, NO. 5, SEPTEMBER 2006, pp1364-1373.

[10] Remus Teodorescu, Frede Blaabjerg, Marco Liserre , Antonio Dell’Aquila, “A stable three-phase LCL-filter based active rectifier without damping”, Conference Record of the Industry Applications Conference, 2003. 38th IAS Annual Meeting, pp1552-1557.

[11] Vladimir Blasko, Vikram Kaura, “A Novel Control to Actively Damp Resonance in Input LC Filter of Tree-Phase Voltage Source Converter”, IEEE Transactions on industry applications, vol.33, NO.2, March/April 1997, pp542-550.

[12] Marco Liserre, Frede Blaabjerg, Steffan Hansen, “Design and Control an LCL-Filter-Based Tree-Phase Active Rectifier”, IEEE Transactions on industry applications, vol.41, No.5, September/October 2005, pp1281-1290.

[13] Siegfried Heier, "Grid Integration of Wind Energy Conversion Systems," John Wiley & Sons Ltd, 1998, ISBN 0-471-97143-X

[14] D.Xu, Y.W. Li, B. Wu, “Direct PWM Synchronization Using an All Digital Phase-Locked Loop for High Power Grid-Interfacing Converters”, in Proceedings Applied Power Electronics Conference, February 2007.

[15] Mariusz Malinowski, Marian P. Kazmierkowski, Steffen Bernet, “New Simple Active Damping of Resonance in Three-Phase PWM Converter with LCL Filter”, IEEE International Conference on Industrial technology 2005, pp542-550.

[16] AD2S1200: 12-Bit R/D Converter with Reference Oscillator Data Sheet

[17] Marco Liserre, Frede Blaabjerg, Antonio Dell’Aquila, “Design and Control of a Tree-Phase Active Rectifier Under Non-ideal Operating Conditions”, Industry Applications Conference, 2002. 37th IAS Annual Meeting, Volume 2, 13-18 Oct. 2002 Page(s):1181 - 1188 vol.2

[18] Mariusz Malinowski, Marian P. Kazmierkowski, Andrzej M. Trzynadlowski, “A comparative study of control techniques for PWM rectifiers in AC adjustable speed drives”, IEEE Transactions on Power Electronics Volume 18, Issue 6, Nov. 2003 Page(s):1390 - 1396.

[19] Malinowski, M.; Kazmierkowski, M.P.; Hansen, S.; Blaabjerg, F.; Marques, G.D, "Virtual-flux-based direct power control of three-phase PWM rectifiers" IEEE Transactions on Industry Applications, Volume 37, Issue 4, July-Aug. 2001 Page(s):1019 – 1027

[20] Mokadem, M.E.; Nichita, C.; Reghem, P.; Dakyo, B, "Development of a maximum power tracking unit integrated in wind turbine simulator" 2005 European Conference on Power Electronics and Applications, 11-14 Sept. 2005 Page(s):10 pp.

[21] Arifujjaman, Md.; Iqbal, M.T.; Quaicoe, John E, "Maximum Power Extraction from a Small Wind Turbine Emulator using a DC - DC Converter Controlled by a Microcontroller" International Conference on Electrical and Computer Engineering, 2006. ICECE '06, Dec. 2006 Page(s):213 – 216

[illegible]