PROTECTION OF WAVE POWER GENERATING SYSTEM

A Thesis Submitted for the Research Degree of Master of Engineering
in Electrical Engineering

Tak yin Taky Chan

B.Eng. (Hons)

School of Electrical Engineering
Faculty of Science, Engineering and Technology
Victoria University of Technology
P.O. Box 14428, MCMC, Melbourne
Victoria 8001, Australia

September 2004

Abstract

Ocean Power Technology Project is an innovative technology for generating green energy at low cost, with minimum pollution and producing electrical power from enormous quantities of dependable renewable energy which is available throughout the world in abundance in the form of ocean waves.

An experimental 20kW power system project using Ocean Wave Technology has been build by Powercor Australia at Portland Victoria. This experimental power system consists of a permanent magnet generator, step up and down transformers, submarine cables and a set of inverters and together it supplies 20kW for the loads that is mainly connected to the remote area power system.

This research project has been generated to analyse and develop the protection of variable frequency AC systems. Design and simulations are carried out on a computer using MATLAB algorithmic language software package and Power System Blockset. The mathematical model comprises of permanent magnet generator SIMULINK model, transmission lines, transformers, rectifiers, load buses, etc. The simulation made is to investigate the behaviour of the system during steady state, low and high load simulation is made to investigate the maximum possible output of the generators under overload conditions and fault studies. Consequently, the simulation is obtained by using a set of characteristic of parameters of protection system in order to implement it on a microprocessor based circuit breaker.

Acknowledgements

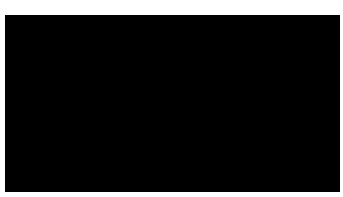
I take this opportunity to express my appreciation to my supervisor Professor Akhtar Kalam the Deputy Dean of the Faculty of Science, Engineering and Technology, Victoria University of Technology who has guided me through this research project. I am grateful to him for his encouragement, support and advice. Also thank to him for providing the opportunity to attend the grid connected energy systems short course. I wish to think Associate Professor Aladin Zayegh Head of School in Electrical Engineering, Dr. Qin Jiang and Dr. Mike Wingate for their support during this research.

I am thankful to Foster, Hanh, Les and Maria from the Electrical Engineering staff, I greatly enjoyed the support and friendship provided by them.

Finally, I wish to dedicate this thesis to my parents. They have brought me up with all the love, understanding and wisdom for which I am ever thankful. I would also like to express my gratitude to my wife Carman Cheung for her encouragement and support to accomplish my education.

Statement of Originality

The work contained in this thesis has not been previously submitted for any degree or diploma at any university and that, to the best of my knowledge, the research described herein is the result of my own work, this thesis contains no material previously published or written by another person except where reference stated in the text of the thesis. It is submitted in fulfilment of the candidature for the degree of Master of Engineering, Victoria University of Technology, Australia.



Tak yin Taky Chan

September 2004

Table of Contents *Page*

Abbreviation

AC	Alternating current
Batt1	Control battery on buoy
Batt2	Main energy storage battery on-shore
BC	Battery Charge for buoy control battery
CT	Current transformer
Cont	On-shore rectifier/energy storage controller
CPU	Center processor unit
DC	Directive Current
DLR	Dummy load resistor
G	Generator
H_s	Wave height
Hz	Frequency unit
HSV	Hydraulic shut-off valve
Hyd M	Hydraulic motor
IGBT	Insulated Gate Bipolar Transistors
k	Generator constant value
k_E	Line to line voltage of generator per radian per second
L_{ls}	stator winding leakage resistance
L_{lkd}	d-axis damper winding leakage inductance
L_{lkq}	q-axis damper winding leakage inductance
L_{md}	d-axis stator magnetizing inductance
L_{mq}	q-axis stator magnetizing inductance
fc	Frequency
f_{sh}	Frequency of shaft speed
T_s	Wave period
kW	Kilo watts
kWh	Network connection energy meter
MATLAB	High-performance language for technical computing

n_{pp}	number of pole pairs
OPT	Ocean Power Technology
OVL	Generator overvoltage limiting resistors
RTW	Real Time Workshop
RTWT	Reak Time Windows Target
rms	Root mean square
rps	Radian per second
r_o	Conversion factor
r_s	Stator winding resistance
r_{kd}	d-axis damper winding resistance
r_{kq}	q-axis damper winding resistance
Sp	Stroke piston
PSB	Power System Blockset
P1	Protection relay 1: Buoy transformer and cable
P2	Protection relay 2: Buoy generator and dummy load
P3	Protection relay 3: On-shore equipment
P4	Protection relay 4: Network connection

List of Figures *Page*

List of Tables *Page*

Chapter 1 INTRODUCTION

1.1 Scope of the Research

Powercor Australia is one of the electricity distribution companies in the state of Victoria. This is a collaborative research project with Powercor Australia Ltd. and Victoria University of Technology. Ocean Power Technology project is an innovative technology for generating green energy at low cost, with minimum pollution and producing electrical power from enormous quantities of dependable renewable energy which is available throughout the world in the form of ocean waves.

An experimental 20kW power system project using Ocean Wave Technology has been build by Powercor Australia at Portland, Victoria. This experimental power system consists of a permanent magnet generator, step up and step down transformers, submarine cables and a set of inverters and together it supplies 20kW for the loads that mainly connect to the remote area power system.

This research project has been generated to analyse and develop the protection of variable frequency AC systems. Design and simulations are carried out on a computer using MATLAB algorithmic language software package and Power System Blockset. The mathematical model comprises of permanent magnet generator SIMULNIK model, transmission lines, transformers, rectifiers, load buses, etc. The simulation made is to investigate the behaviour of the system during steady state. Low and high load simulation is made to investigate the maximum possible output of the generators under overload conditions and fault studies. Consequently, the simulations obtained by using a set of characteristic of parameters of protection system in order to implement it on a microprocessor based circuit breaker.

Finally, the simulated fault signals are generated by the MATLAB real time workshop compiler programmed in C language; through the parallel port interface power converter to test the function of a microprocessor based circuit breaker.

1.2 Aims of the Research

The objective of this research is to design and simulate the protection of variable frequency AC system using algorithmic language MATLAB, implement protection strategy experimentally and verify with the use of a microprocessor based protection.

The specific aims of this research are listed below:

- Literature Review: to understand the requirements and practices of protection system design and application using microprocessor based relays.
- Analyse and identify the characteristics of permanent magnet generator in terms of the wave power generation using practical tested results.
- Design, simulate and analyse the overall power system under the maximum load flow and fault study based on theory of symmetrical component using MATLAB.
- Implement and experiment using a microprocessor based protection relay.
- Explore feasible protection strategy for various power generation conditions.

1.3 Organisation of the Thesis

This thesis consists of six chapters.

Chapter 1 introduces the scope of research and aims, organisation of the thesis and the contribution to the project are given.

Chapter 2 reviews microprocessor based protection, discusses the relevant detail of the Ocean Power Technology and feasibility of the research and methodology.

Chapter 3 identifies the components of the power system in order to simulate analysis and describes the simulation work for the performance of the wave power generating system using the MATLAB / SIMULNIK and Power System Blockset.

Chapter 4 describes the hardware design and development to facilitate the testing. A high power converter has been built to experiment the simulation results, develop and link the MATLAB to compile with C language for experimental testing.

The hardware and software developed in Chapters 3 and 4 are tested and verified experimentally, the results are presented in the next chapter.

Chapter 5 describes and displays the simulation results under various situations such as transient and steady state analysis. The practical testing results also display the results to determine the setting for microprocessor based circuit breaker.

Chapter 6 the final conclusion of the thesis highlights the achievements throughout the thesis and the significance of the study. Suggestions for further work are also given.

The thesis has four appendices. Appendix A shows the electrical block diagram of Ocean Power Technology and also displays some related electrical diagrams. Appendix B shows the hardware design schematics for the power converter and associated schematics. Appendix C

displays the time curves of characteristics of the microprocessor based circuit breaker. Appendix D gives the listings of C language program. These programs are generated from MATLAB real time workshop compiler, they are used for analysis and implementation of power converter to test the microprocessor based circuit breaker.

1.4 Contribution of the Thesis

Major contributions of this thesis are:

1. To develop a completed simulation results for Ocean Power Technology and preform the steady state and transient fault analysis.

2. To experiment the power system by simulation of the fault signal in order to obtain a set of characteristic of parameters for protection system in order to implement it on a microprocessor based circuit breaker.

3. To design and construct a SIMULNIK based real time controlled three phase converter. The purpose of this converter is to provide an existing power from the DC–AC converter by the simulation result.

Chapter 2 LITERATURE REVIEW

2.1 Introduction

Power system occasionally experiences faults and abnormal operating conditions. To avoid damage to the equipment of the utilities, protective relays are used to take suitable corrective actions. In early developments of power systems, protection functions were performed by electro-mechanical relays and many such relays are still used in power system. Solid-state relays were introduced in early seventies and the past thirty years has seen the development in digital relaying techniques. Accurate algorithms have enhanced the performance of power system protection. The review of some literatures for microprocessor based relays, current transformer and protection system is described in this chapter. Also this chapter reviews and analyses the details of the Ocean Power Technology project and the specific microprocessor based circuit breaker.

2.2 Review of Microprocessor Based Relays

Computer based relay utilising digital processor has been developed to in the last half century. In 1969 a comprehensive paper by Rockefeller [1] outlined the feasibility of protecting with a computer all the equipment in power system. The problems associated with the use of a digital computer for performing all the protection functions are clearly recognised in that paper. Though the concept of using a single computer along with its backup has since been discarded, in favour of using individual relaying microprocessors for each major protection function. However, many investigations reported in the paper are still valid and useful.

Carr and Jackson [2] investigated the use of Fourier Transform to design a relay with coordinated analogue and digital filter designs. Digitised fault data was used to test and demonstrate its feasibility. Sachdev and Baribeau [3,4] developed a least square error approach for extracting the fundamental frequency components of voltages and currents from raw data. The real and imaginary components of voltage and current phasors obtained in this manner were used to calculate impedances as seen from a relay location. The advantage of this approach is that the decaying dc component is explicitly filtered out from the input data without pre-specifying the X/R ratio of the system.

Digital techniques for detecting faults in generator have also been investigated. Sachdev and Wind [5, 6] compared instantaneous values of the fault currents (measured as a difference between the currents at the neutral and line ends of the windings) and the through currents. The difference and through currents were pre-processed using analogue summing circuits to alleviate analogue to digital conversion problems and to reduce CPU time requirements. Malik et al. [7] also used the cross-correlation approach to determine the fundamental and second harmonic frequency components in the primary and secondary currents of a transformer.

Schweitzer et al. [8] and Larson et al. [9] used even and odd square waves to extract information regarding the fundamental frequency and second harmonic components from the signals representing currents in the transformer primary and secondary windings. The unique feature of this work consists of recognising the inrush phenomenon from the real and imaginary components of the fundamental frequency and second harmonic currents without calculating their peak or rms values. The proposed approach was implemented on a Motorola MC6800 microprocessor and tested in a laboratory environment.

Researchers and designers have made substantial progress using microprocessor for power system protection during the last ten years. Phadke and Thorp [10] reported the development of a harmonic restraint algorithm for three winding transformers. Later, Phadke and Thorp [11] reported on a computer based flux restrained current differential relay for power

transformers. Again Rahman et al. [12] have reported the comparison of digital techniques used for differential protection of transformers.

In recent years, strong interests in utilising digital methods to protective relaying have been developed for power system protection by Kramer and Elmore [13]. They described the availability of a microprocessor based inverse time over-current relay having selectable characteristic that will greatly relieve the application difficulties that have been associated historically with their fixed-characteristic electromechanical counterparts. Sidhu et al. [14] reported design, implementation and testing of a microprocessor based relay for detecting transformer winding faults by using 16-bit microprocessor.

Extensive research effort in the field of digital relays and systems is worthwhile because the digital technology has several advantages over the analogue one. Properly designed microprocessor relays and systems are in several ways superior to the electro-mechanical and solid-state relays and are being increasingly accepted for general use in the utility [15]. The advantages of digital technology using microprocessor based relaying system are as follows:

- The characteristics of digital components do not drift with temperature, supply voltage changes or ageing.
- The performances of digital components do not change from part to part (as long as numerical value, say 2 stored in memory remains 2 irrespective of where the number is stored).
- The equipment designs based on digital technology use fewer parts and connections.
- The resolutions of the solution provided by digital devices depend on the number of bits per word used in the arithmetic calculations.
- The digital devices are not required to be tuned individually to obtain consistent results.
- Most design changes can be made by changing the software only.
- A digital device can perform both logic and arithmetic functions while controlling a process.
- The data recorded by a digital device is not corrupted except when an equipment failure is experienced.

2.3 Review of Current Transformers

Protective relays demand a reasonably accurate replica of the primary current and voltage, in particular when faults occur on the power system. For this reason current transformer (CTs) are employed to provide a reduction of the primary current for the relays. However, most conventional iron-cored CTs are not ideal because of their nonlinear excitation characteristic and their ability to retain large flux levels in their cores, known as remanent flux, thus they are prone to saturation. Many studies on the analysis of the steady state and transient behavior of iron-cored CTs have been reported [16-19].

Kang et al. [20, 21] proposed a novel compensating algorithm for accurate measurement of the CT secondary current; accurately estimated the secondary current corresponding to the CT ratio, in particular when CT is saturated. The proposed algorithm has a number of significant attributes that can improve the sensitivity of relays to low level internal faults, maximize the stability of relays for external faults, and make a reduction in the required CT core cross section possible. Marti et al. [22] described a fast and non-iterative solution method for obtaining the secondary current from known primary current in the network system. Locci and Muscas [23] proposed the compensation method for enhancing the accuracy of low quality transformers. The main improvement is that it is also able to significantly reduce hysteresis effects. It assumes sinusoidal steady state condition for the primary current, but yield good results even in the presence of slightly distorted current waveforms.

2.4 Ocean Power Technology

2.4.1 Introduction

Ocean Power Technology (OPT) has developed and installed an initial 20kW wave-energy generation system, to be moored to the seabed several kilometres off the Victorian coast near Portland. This is supported by a $230,000 grant from the Renewable Energy Industry Program. The wave energy converter, called the PowerBuoy™ shown in Figure 2.1 is fully-submerged and resembles a large ocean buoy. It can generate renewable electricity for transmission via submerged cable to the grid, where it can be sold to retail customers.

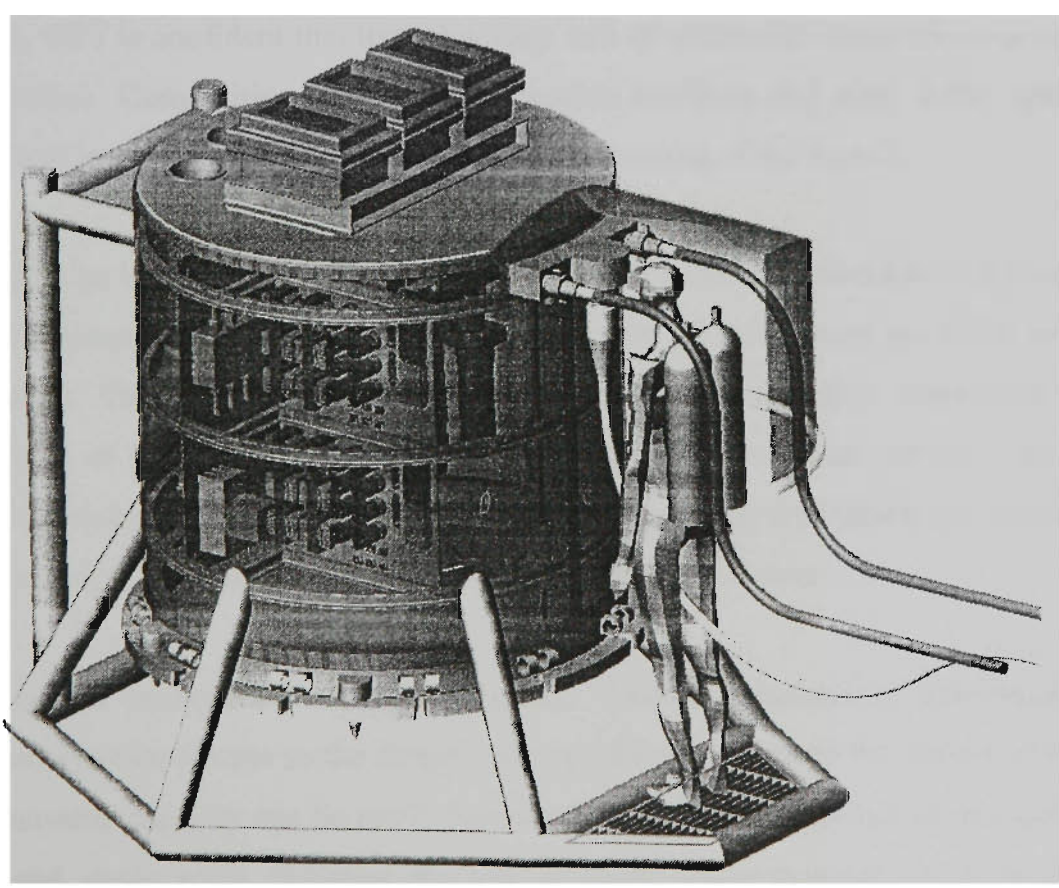


Figure 2.1 The wave energy converter PowerBuoy™

The competitive advantages of OPT's technology over other wave-energy systems include its proprietary computerised control system which is housed in a watertight canister attached to the buoy, plus the in-ocean experience of the PowerBuoy™, the simplicity of its offshore system design, and its marine survivability. The up-and-down motion of ocean waves is harvested by the piston-like PowerBuoy™, driving a hydraulic cylinder and the power generation equipment. This power can then be transmitted to shore via the underwater cable. OPT has held extensive consultations with the Marine Board of Victoria, the Department of Natural Resources and Environment, Environment Australia, Mirimbiak Nations Aboriginal Corporation and fishing groups, and the systems is fixed to the ocean floor to minimise the risk of interfering with migrating whales.

The PowerBuoy™ system is designed to be economically competitive with fossil fuels and it is likely that initial equipment sales will come from isolated diesel markets. In the longer term, though, OPT is confident that its technology will be competitive against large coal-fired power generation. Competitive local steel fabrication facilities and other local vendors for deployment and maintenance are utilised to minimise the cost of the system.

Electricity from an installation of multiple PowerBuoy™ units is expected to cost around 7 to 12 cents per kilowatt hour (kWh) to run compared to the 25 to 50 cents per kWh for diesel-fired generation. This also compares favourably with other renewable sources of energy. Another benefit of OPT's technology is its durability. As it is based around a number of existing proven technologies such as ocean buoys, marine-quality hydraulics and conventional moorings, the unit is estimated to have a useful life of at least 30 years.

The OPT system incorporates special mooring, flotation, anchoring, data-logging and navigation-aid systems for use in the future scale-up of the system. As the system is modular, the total generating capacity can be easily increased by installing more buoys, using the same production and deployment facilities for use in either grid-connected or isolated power networks. The major components of the OPT system and their specific functions are shown in subsequent sections.

2.4.2 Wave Energy Converter

Wave Energy Converter produces linear motion of a water piston in response to an incident ocean wave. Actual piston response is determined by the solution of a non-linear differential equation. In general, the motion is proportional to wave height H_s and falls off with wave period T_s.

2.4.3 Hydraulic Motor

Hydraulic Motor converts linear piston motion to rotary motion of the Permanent Magnet (PM) generator shaft, the conversion factor is a constant r_o meter per radian, i.e., r_o meters of piston motion produce radian of angular shaft rotation. The specific system to be used at Portland has a r_o of 1.124e-3 meters per radian.

2.4.4 Permanent Magnet (PM) Generator

Permanent Magnet Generator produces a 3 phases AC voltage whose *rms* value is directly proportional to shaft speed. The proportional constant k_E is specified in line to line *rms* voltage per *rps*. For the Portland system, the generator k_E is 5.0 voltage *rms* per *rps*.

2.4.5 Generator Voltage, Frequency and Output Power

The *rms* output voltage (open circuit) is directly proportional to shaft speed F_{sh}. The electrical output frequency is equal to $n_{pp} * f_{sh}$ where n_{pp} is the number of pole pairs in the permanent magnet (n_{pp} = 3 for this generator). The output power for a resistive load is proportional to

$Voc^2 = k * f_{sh}^2$. Tracing this back through the hydraulic motor and water piston, it can be seen that the power output is proportional to wave height square H_s^2, therefore, the available wave power. This yields the significant result that the efficiency of the OPT system is independent of H_s.

OPT has gathered extensive experimental data on this motor operating as generator and has developed models which enable the prediction of output power as a function of shaft speed and load resistance. Figures 2.2 and 2.3 show line to line voltage and power output via shaft speed for the expected wave environment at Portland.

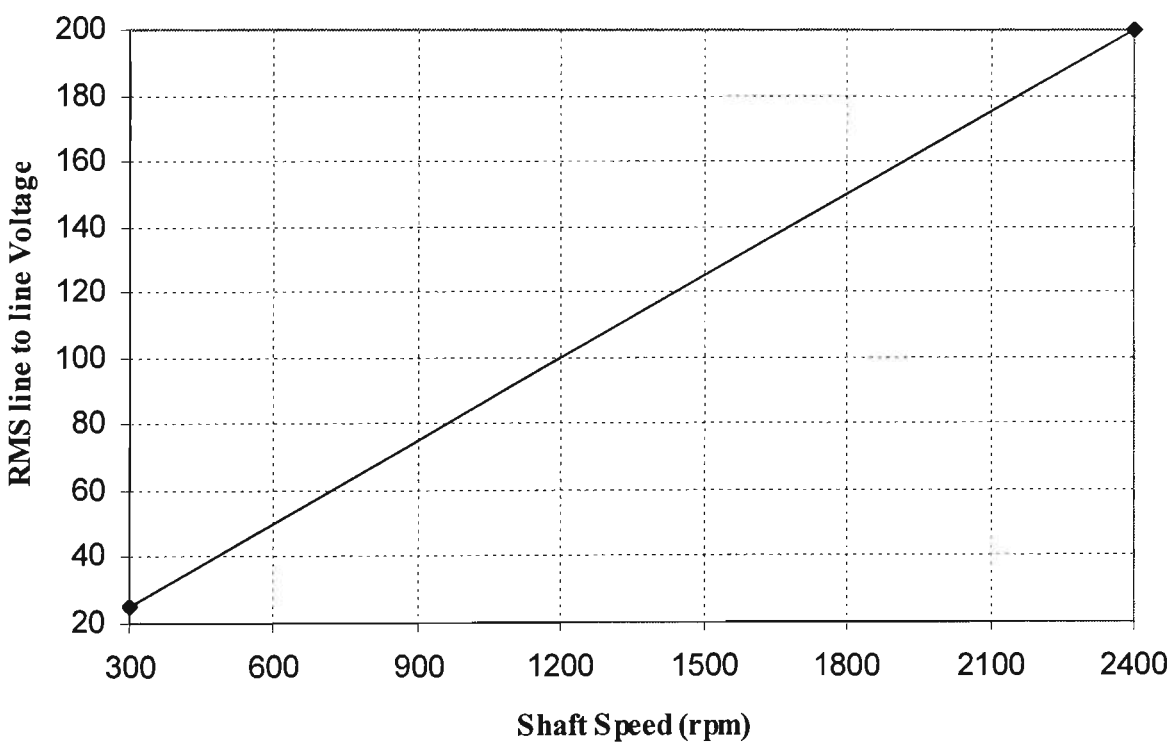


Figure 2.2 Line to Line Voltage vs. Shaft Speed of Generator

It should be pointed out again that the electrical output frequency is three times the shaft speed. Also it should be noted that these parameters represent the generator output when running at a constant shaft speed which produces a sinusoidal output waveform. This is not at all the case for the time varying speed (piston velocity) that occurs in practice.

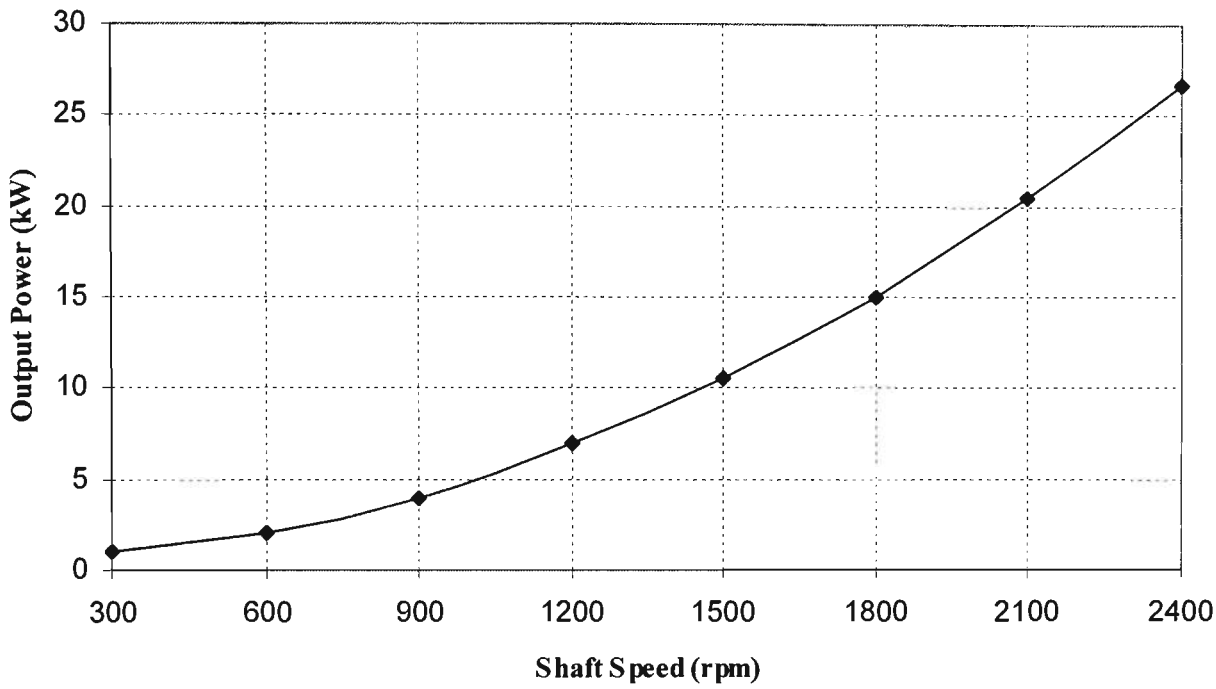


Figure 2.3 Output Power vs. Shaft Speed of Generator

Now consider the time varying case where the piston stroke motion and velocity are given by:

$$Sp = SM \sin\left(\frac{2\pi t}{Ts}\right) \qquad \text{meters}$$

$$Vp = \frac{d}{dt}(Sp) = 2\pi \frac{SM}{Ts} \cos\left(\frac{2\pi t}{Ts}\right) \qquad \text{meters / second} \qquad (2.1)$$

where T_S is the wave period and S_M is the peak piston displacement. (typically, ¼ of H_S, the peak to trough wave height).

Through the hydraulic motor action, the linear piston motion S_P produces an angular shaft rotation and angular velocity.

$$\theta s = \frac{S_p}{r_o} \qquad \text{radians}$$

$$\omega_s = \frac{d}{dt(\theta_s)} = \frac{V_p}{r_o} \quad \text{radians / sec} \tag{2.2}$$

The line to line generator voltage is

$$V_{LL}(t) = \frac{(2)^{0.5}\, k_E\, \omega_s(t) \sin(n_{PP}\theta_s(t))}{(2\pi)} \tag{2.3}$$

This is a very complex waveform. As can be seen, it is both amplitude and frequency modulated by the piston motion. Substituting, equations (2.1) and (2.2) in equation (2.3).

$$V_{LL}(t) = \frac{2^{0.5}\, k_E\, S_M}{(r_o\, T_S) \cos(2\pi t / T_S) \sin(n_{PP}\, S_M / r_o \sin(2\pi t / T_S)} \tag{2.4}$$

Since r_o is purposely designed to be small to achieve high shaft speeds, the FM modulation index is very high, resulting in a large number of sidebands with significant power levels over a broad frequency range. An example to highlight this is.

Assume H_S = 2.5m and T_S = 11.5s which correspond to the peak of the two dimensional probability distribution at the Portland site. Further assume that S_M = 0.25 H_S and that the other parameters k_E, n_{PP} and r_o are the constants of the generator and hydraulic motor previously specified. The time series waveform of the line voltage and its frequency spectrum are shown below in Figures 2.4 and 2.5.

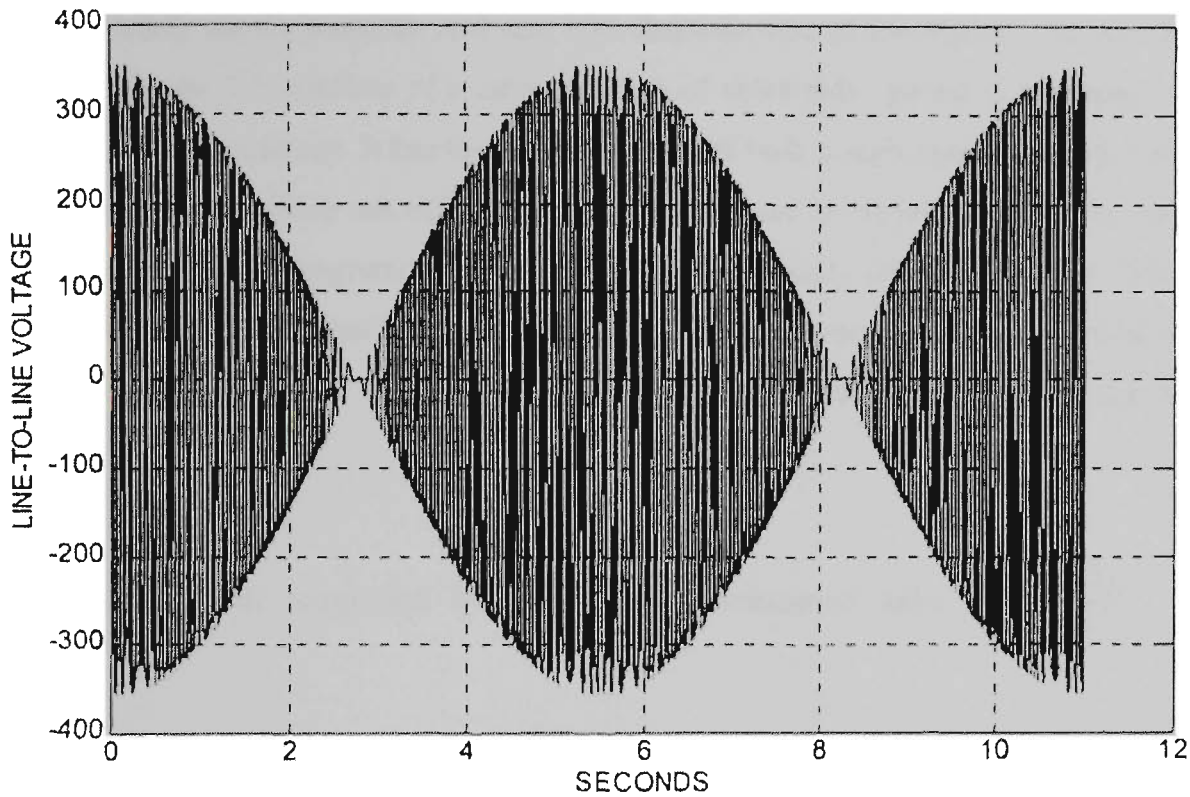


Figure 2.4 Line voltage waveform

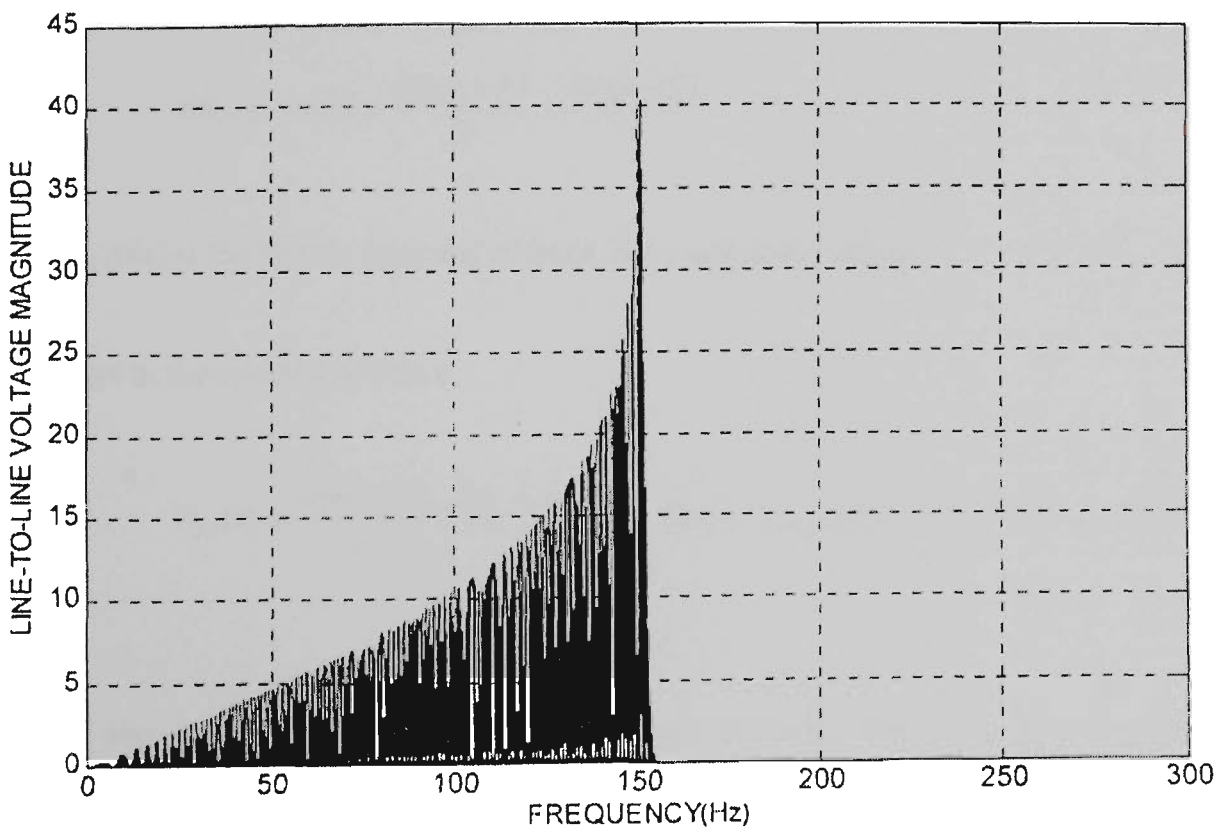


Figure 2.5 Line voltage spectrum

Figure 2.4 clearly shows both the AM and FM characteristic of the signal. The spectrum of line voltage Figure 2.5 consists of a large number of sidebands spaced at multiples of $1/T_S$ around the carried frequency. It has the characteristics of both a high pass filter and a low pass filter with a relatively sharp cut off. The high pass response is expected due to the inductive nature of the permanent magnetic generator. The high frequency falls off at about 150Hz. An important point to note is that the above waveform and spectrum correspond almost exactly with the experimental data obtained in numerous wave tank tests. This is far from a theoretical exercise.

Equation (2.4) can be simplified by defining the modulation index $m = \frac{n_{pp} S_M}{r_o}$ and employing.

Identities

$$\sin(m \sin(\omega_s t)) = 2\sum_0^\infty J_{2n+1}(m) \sin((2n+1)\,\omega_s t)$$

$$\sin(a)\cos(b) = \frac{\sin(a+b)}{2} + \frac{\sin(a-b)}{2}$$

where $J_{2n+1}(m)$ is the Bessel function of order 2n+1 and argument m.

This results in the spectral solution

$$V_{LL}(t) = \frac{sqrt(2)\, k_E\, S_M}{(r_o\, T_S)} \sum_0^\infty (J_{2n+1}(m) + J_{2n+3}(m)) \sin(2n\omega_s t) \qquad (2.5)$$

From the above, only even harmonics of the wave frequency show up in the spectrum as should be expected from the Figure 2.5 waveform. Secondly, the amplitude of the harmonics are dependent on the behaviour of the Bessel functions as both large orders and magnitudes,

and combined with the Bessel recurrence relation, explains the sharp fall off with frequency as follows. The recurrence relation is:

$$J_{n+1}(m) = \frac{2n}{mJ_n(m) - J_{n-1}(m)} \qquad (2.6)$$

It is seen that when $2n/m$ approaches 1, $Jn+1<Jn$ which makes $Jn+2 < Jn+1$, etc. This represents a rapid descent to 0 amplitude. By defineing fc the upper cut off frequency of the signal by setting $2n/m = 1$

$$f_c = n_{pp}\, S_M / (r_o\, T_S) \qquad (2.7)$$

Note: these analytic results are based on the Bessel formulation and have been cross-checked by extensive simulation.

This also makes sense physically since f_c is the highest instantaneous electrical frequency in the signal coupled with the fact that the power available is proportional to $(V_{LL})_{rms}^2$.

The upper cut-off frequency of transformer bandwidth is determined from equation (2.7) by the maximum expected wave height and minimum expected wave period. Similarly, the lower cut-off can be determined from the minimum wave height and maximum wave period. This will result in a transformer with a pass band of 20-600 Hz which will cover the worst case scenario. However, meeting the worst case is always much too expensive.

First, integrating the power spectrum of Figure 2.5 results in the fact that 95% of the signal power is contained in the internal from 0.5 f_c to f_c, i.e. halving the bandwidth results in only a 5% loss. The second factor is that certain combinations of wave height H_S and period T_S rarely occur and contribute little weighted average power. This is seen in the probability matrix as shown in Table 2.1.

Table 2.1 Table of wave probability matrix at OPT Portland site

Wave Probability Matrix at OPT Portland Site

Period (sec.)	Height (m)								
	1.25	1.75	2.25	2.75	3.25	3.75	4.25	4.75	5.25
4.5	0	11	4	2	0	0	0	0	0
5.5	0	15	33	55	11	2	0	0	0
6.5	0	11	46	63	46	17	0	0	0
7.5	0	2	37	59	74	74	17	4	2
8.5	0	0	24	11	28	35	22	7	2
9.5	2	26	33	35	26	26	13	17	2
10.5	7	181	375	262	155	113	41	31	9
11.5	15	399	805	560	366	129	35	22	2
12.5	13	288	473	403	188	96	22	4	0
13.5	22	301	438	327	229	94	31	7	2
14.5	15	303	329	303	170	150	26	2	0
15.5	7	201	214	118	131	85	17	9	2
16.5	2	137	70	55	68	68	11	0	0
17.5	2	20	35	9	17	9	0	0	0

The entries in the above table are the probabilities of occurrences x 10^4. For example, the probability of a wave having a height 2.25 meters and a period of 11.5 seconds are 0.0805 or, stating it another way, such a wave occurs 8.05% of the time. The data represents the average of more than 4500 samples taken over 13 month duration.

The proper way to assess the loss in efficiency is to take the above matrix weighted the available power (0.5 H_S^2 T_S Kw/m) and double integrates over all heights and periods. Then same calculation is performed with limited heights and periods. The ratio of these 2 results is the power loss due to the limited bandwidth transformer. This was done for several variants

on height and period. The most favourable trade off occurs for $1.75 < H_S < 4.25$ m and $6.5 < T_S < 16.5$ s which reduces the power out by only 6%. The transformer bandwidth is reduced from 200-600 Hz to 70-450 Hz.

2.5 Description of microprocessor based circuit breaker

The ABB Isomax S4 circuit-breakers for protection in AC power system can be fitted with PR212/P overcurrent constructed using electronic microprocessor-based technology. This allows protection functions to be obtained which guaranteeing a high level of reliability and tripping precision and which are unaffected by the external ambient. The power supply needed for correct operation is supplied directly by the release current transformers, in the presence of a phase current higher than or equal to 18% of their rated current, even with a single phase supplied with voltage. There is only one adjustment for all the phases and the neutral and the release is simultaneous for all the circuit-breaker poles, with trip characteristics which are unaffected by the external ambient. The functions and settings of the protections can be operated up to the frequency of 400Hz.

It is particularly suitable in applications with this project requirement, for earthing protection against the evolutionary faults, and for remote control and parameterisation, network supervision and centralised load management. The PR212/P with functions (LSIG) provides protection functions against overload (L), delayed short-circuit (S) and instantaneous short-circuit (I), and against earth fault (G).

For three-pole circuit-breakers fitted with PR212/P - LSIG, protection of the neutral can be set to 50% or 100% of the phase protection setting (by means of dip-switches on the front of the circuit breaker). On request, it is possible to obtain full protection of the neutral with setting equal to 100% of the protection.

The PR212/P microprocessor-based device are self-supplied and ensure correct operation of the protection functions, even with only a single phase supplied with voltage, in the presence of a current higher than or equal to 18% of the rated phase value. The protection device

consists of current transformers, the PR212/P protection unit and a demagnetising opening solenoid which acts directly on the circuit-breaker operating mechanism group.

The current transformers are housed inside the release box and supply the energy needed for correct operation of the protection and the signal required to determine the current. When the protection intervenes, the circuit-breaker opens by means of the opening solenoid, which changes over a contact for signalling tripped. Resetting the signal is of mechanical type and takes place with resetting of the circuit-breaker operating lever.

In the versions with PR212/P - LSIG device, it is possible to set the adjustment parameters of the protection functions directly from the front (dip-switch positioned on MAN). In case of any anomalies in remote parameterisation, the protection automatically uses the set of parameters set manually on the front of the circuit-breaker. This allows the adjustment parameters to be set even with the circuit-breaker open.

The view of physical configuration microprocessor based circuit breaker is shown in Appendix C1.

2.6 Research Methodology

The principle and knowledge of over current relays and differential relays are used to design and implement a secure power system algorithm. The microprocessor-based relay that operates on the samples of voltage and current are written to produce the estimated parameters of interest for protection. The concept of parameter is important since it is necessary to model the system or the waveforms in order to develop an algorithm. Most of the existing algorithms proposed for use in digital relaying is based on a model of the voltage and current waveforms. In this work, these waveforms will be generated using an algorithmic language MATLAB and power systems blockset for investigation and simulation of the power system protection.

Initially, analyse the permanent magnet generator characteristic is analysed and extensive experimental data gathered on this motor operating as a generator and models are developed which enable the prediction of output power as a function of shaft speed and load resistance. The three phase AC voltage produces rms value directly proportional to shaft speed. The proportionality constant k is specified in line to line rms volts/rps. For the Portland system, the generator k is about 5.0 volts/rps. A new simulation model is created by the experimental data for the MATLAB simulation.

In addition, the conventional current transformers are not ideal because of their nonlinear excitation characteristic and their ability to retain large flux level in the cores. This research study on the analysis varies with frequency of the steady state and transient behaviours. The characteristic function is verified from experiment in the power laboratory.

Subsequently, MATLAB and Power Systems Blockset are used for the simulation of the overall ocean power generating system as shown in Appendix A and the numerical solutions are computed by the tested parameters of generator, transformers, transmission lines, inverters, load, etc. Thus the overall solutions with comprehensive analytic asymmetrical

power system load flow and fault study are based on theory of symmetrical components. Simulated results are synthesized to investigate generating power systems.

Finally, the parameters of characteristic of over current and differential relays algorithms is worked out and developed in the program for microprocessor based protection system. The performance of protection relay is tested under the real time by the PC parallel port link to MATLAB and the functional testing is subjected to several normal and abnormal conditions.

Chapter 3 METHOD OF ANALYSIS

3.1 Introduction

The use of computer simulation tools is essential in power system studies. Software tools are widely used by utilities for transient event simulations, power flow studies, stability analysis, and operational planing. Most of the commercial software packages are available and is designed to work with large system models. The use of such tools is not well suited to the small power system. A new power system simulation tool that uses an interactive, object oriented interface is the SIMULINK [24] modelling environment.

SIMULINK is a window oriented dynamic modelling package built on top of MATLAB numerical workspace. The MATLAB environment has also been used to develop analysis tools for small scale power system studies [25]. However, the SIMULINK environment has the advantage that models are entered as block diagrams with an intuitive graphical interface. Model parameters are entered in menus and can be changed interactively during simulation. Simulation results can be viewed during the simulation and then exported to the MATLAB workspace for subsequent off-line analysis. While the simulation is not real time, the immediate feedback of the interactive simulation environment provides the user with far more intuition about simple system dynamics than batch-mode simulations.

The SIMULINK modelling environment consists of a library of basic building blocks [26], which can be combined to form a dynamic model. Groups of block can be combined into single customized block to form highly specialised modelling construction. The modelling environment described in this thesis consists of a library of customized blocks for power system components that are easily connected to form a power system model. Models are constructed in block diagram form, with a separate block, or object, used for each generator, transmission line, transformer and loads in the simulated system. The connections between

blocks in the model reflect a phasor representation of voltage and current, and the connections are structured so that Kirchhoff's voltage and current laws are satisfied. The simulation models can be developed to simulate balanced or unbalanced conditions based on the modelling method.

3.2 Simulation Method Analysis

In the present simulation method, the power system is represented as two parts: a state-space model for the linear circuit and a feedback model (using current injection) for the non-linear elements as shown in Figure 3.1.

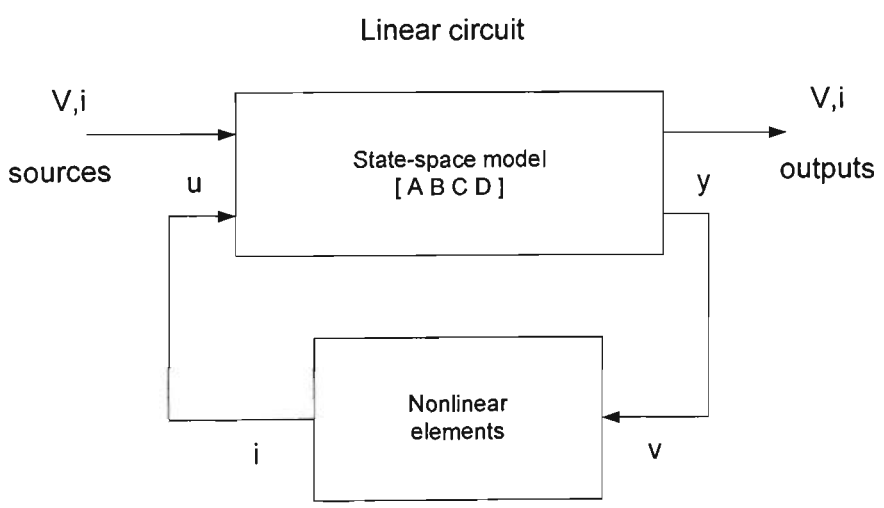


Figure 3.1 Modelling a power system using state-space model with feedback

State-space model of the linear circuit

The differential equation of a linear circuit consisting of resistors, inductors, capacitors and mutually coupled inductors can be written in the form of two state equations:

$$x' = Ax + Bu \tag{3.1}$$

$$y = Cx + Du \tag{3.2}$$

where x and x' are the state variables and their derivatives, u is the input vector, y is the output vector, and A, B, C, D are state matrices.

In the linear circuit, the state variables are capacitor voltages and inductor currents. Inputs are the voltage and current sources. Outputs are the measured voltages and currents.

Simulation of non-linear elements

Non-linear elements such as transformer saturation branches, non-linear inductances, switches and electric machines are modelled using non-linear v-i relations. Each non-linear model uses the voltage across the element as an input and returns a current which is re-injected into the linear circuit state model. A feedback loop is thus formed between the outputs and inputs of the state-space model of the linear circuit.

This method requires that non-linear elements have non-zero impedance. Also, they must contain at least one state in order to avoid an algebraic loop. SIMULINK integration algorithms can solve algebraic loop, but at the expense of slower simulation.

State initialisation-Calculation of steady-state

Once the state-space model of the linear system is obtained, all states must be initialised so that the simulation will start with steady state. The initial vector X_o to be used is calculated as follows.

The system states can be expressed as a function of the inputs:

$$X = (sI - A)^{-1} BU \tag{3.3}$$

The transfer function matrix (*n* output x *n* input) is derived from equations (3.2) and (3.3):

$$H = \frac{Y}{U} = C(sI - A)^{-1} B + D \tag{3.4}$$

where s is Laplace operator, X, U, Y are Laplace transforms of state, input, and output vectors, I is (n x n) identity matrix.

By setting $s = j(a)$ in equation (3.4) and using the phasors of input voltages and currents, the steady-state value of the state vector (complex vector) and its initial value X_o at $t = 0$ (real vector) are calculated. The output vector at frequency t_o is directly obtained from equation (3.4). If the system contains voltage and current sources at different frequencies, the above procedure is repeated for each frequency and the X_o vector is the sum of the vectors obtained at each individual frequency.

Frequency response calculation

Once the state-space model of the linear system is known, it can be used for studies in time and frequency domains. The user can access a variety of MATLAB functions and toolboxes for processing and plotting of results.

A simulation tool within SIMULINK environment

The Power System Blockset (PSB) is a graphic tool that allows building schematics and simulation of power systems in the SIMULINK environment. The blockset uses the SIMULINK environment to represent common components and devices found in electrical power networks. It consists of a block library that includes electrical models such as RLC branches and loads, transformers, lines, surge arresters, electric machines, power electronic devices, etc. Diagrams can be assembled simply by using click and drag procedures into SIMULINK windows. The Power System Blockset uses the same drawing and interactive dialogue boxes to enter parameters as in standard SIMULINK blocks.

Simulation results can be visualised with SIMULINK scopes connected to outputs of measurement blocks available in the PSB library. These measurement blocks acts as an interface between the electrical blocks and the SIMULINK blocks. The voltage and current measurement blocks can be used at selected points in the circuit to convert electrical signals into SIMULINK signals. Non-linear elements requiring control, such as power electronic devices, have a SIMULINK input that allows control from a SIMULINK system.

Initialization and simulation

An initialization process is executed each time the simulation is started. It computes the state-space representation of the circuit and verifies if the circuit is consistent with electrical rules and builds the SIMULINK model of the electrical network [27,28].

The PSB graphical interface also includes an interactive tool to set initial conditions of the capacitor voltages and inductor currents. This allows simulation with initial conditions, or to start the simulation with steady-state. The same interface can also display the steady-state values of all voltages and currents including state variables, electrical sources, nonlinear elements and measurement blocks.

A load flow computational engine allows to initialise three-phase circuits containing synchronous and asynchronous machines, so that the simulation directly starts in steady-state.

3.3 Identification of the Components of System

3.3.1 Mathematical Model of Permanent Magnet (PM) Synchronous Machine

The mathematical equation of the circuit model is shown in Figure 3.2. In general, the permeance along the q-axis and d-axis are not same. While the mmfs of the rotor windings are always directed along q-axis or d-axis, the direction of the resultant mmf of the stator windings relative to these two axes will vary with the power factor. A common approach to handling the magnetic effect of the stator's resultant mmf is to resolve it along the d-axis and q-axis, where it could be dealt with systematically.

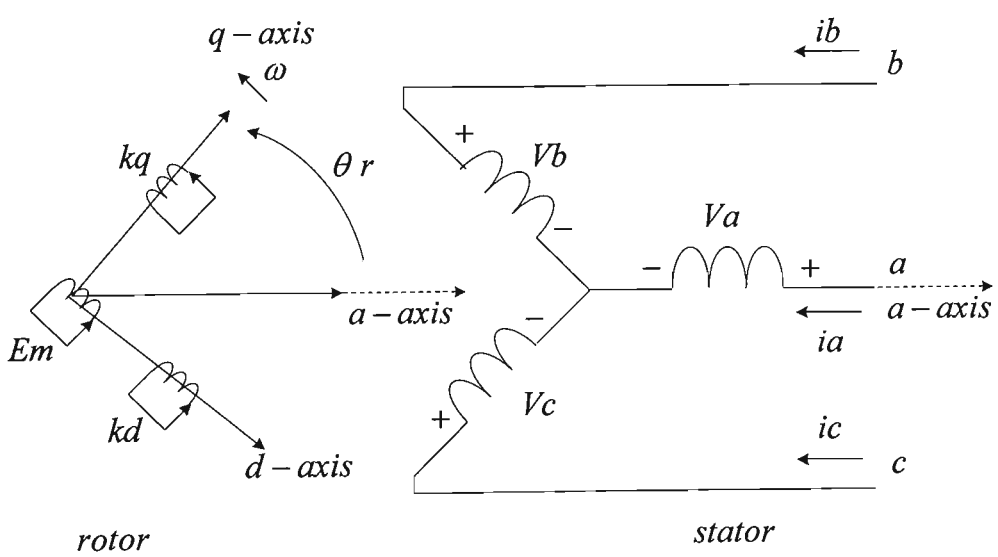


Figure 3.2 Circuit representation of an ideal PM machine

3.3.2 Voltage Equation in the Rotor's qd0 Reference Frame

A summary of the winding equations for Permanent Magnetic Synchronous Machine in rotor's q and d reference frame with all rotor quantities to the stator is as follows:

qd0 equation of a permanent magnetic machine

$$V_q = r_s i_q + \frac{d\lambda_q}{dt} + \lambda d \frac{d\theta_r}{dt}$$

$$V_d = r_s i_d + \frac{d\lambda_q}{dt} - \lambda q \frac{d\theta_r}{dt} \qquad (3.5)$$

$$V_0 = r_s i_o + \frac{d\lambda_0}{dt}$$

$$0 = r'_{kd} i'_{kd} + \frac{d\lambda'_{kd}}{dt}$$

$$0 = r'_{kd} i'_{kd} + \frac{d\lambda'_{kq}}{dt}$$

where the Flux linkages are:

$$\lambda_q = L_q i_q + L_{mq} i'_{kq}$$

$$\lambda_d = L_d i_d + L_{md} i'_{kd} + L_{md} i'_m \qquad (3.6)$$

$$\lambda_0 = L_{ls} i_0$$

$$\lambda'_{kq} = L_{mq} i_q + L'_{kqkq} i'_{kq}$$

$$\lambda'_{kd} = L_{md} i_d + L'_{kdkd} i'_{kd} + L_{md} i'_m$$

An equivalent circuit representation of a permanent magnet machine is based on the above set of voltage and flux relation. The winding equations and equivalent circuit diagram have been derived for an ideal machine. The equivalent circuit of Figure 3.3 for permanent magnetic can be substituted, the equivalent *qd0* circuit representation is shown in Figure 3.4.

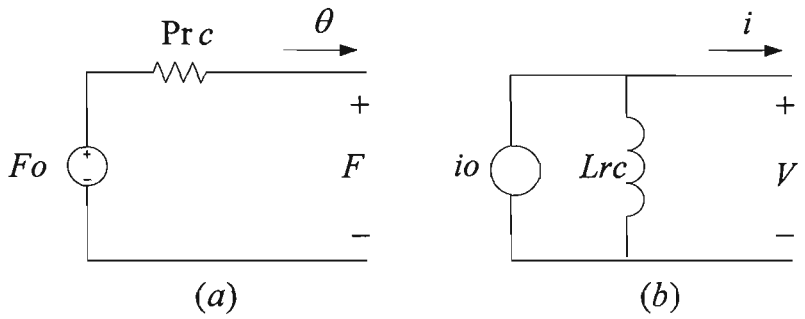


Figure 3.3 Equivalent magnetics circuit

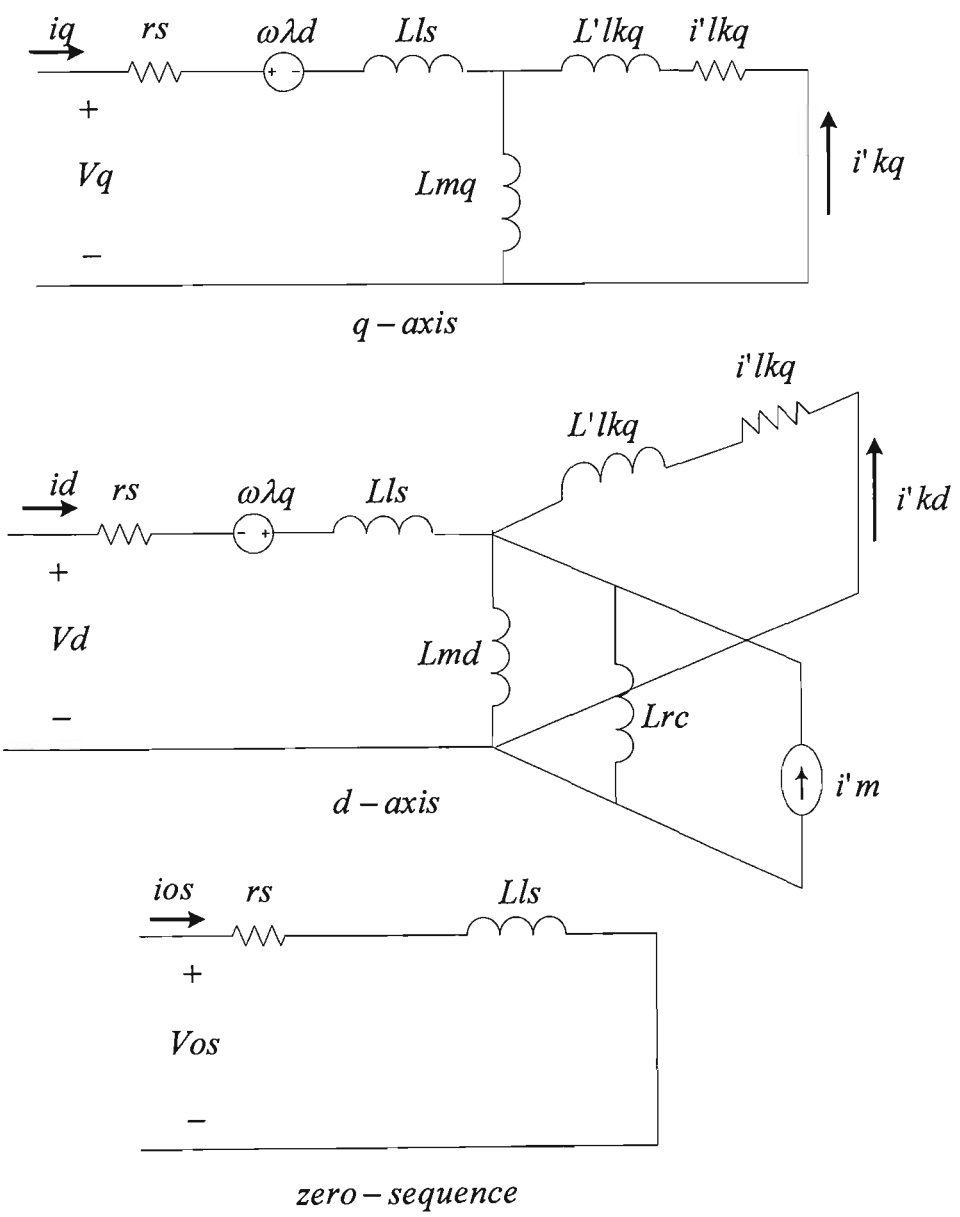


Figure 3.4 Equivalent *qd0* circuits of a permanent magnet synchronous machine

For modelling purposes, the permanent magnet inductance, *Lrc* is associated with its recoil slope, the case of a linear recoil characteristic of a magnet through its current operating point m(Hm, Bm) mapped directly onto the mmf-flux (F vs. Φ) or the current –flux linkages (I vs. λ) plane. It can be lumped with the common d-axis mutual inductance of stator and damper windings and the combined d-axis mutual inductance denoted still by *Lmd*. The current, *i'm* is the equivalent magnetizing current of the permanent magnets, referred to the stator side. The corresponding *qd0* equation for the above equivalent *qd0* circuits of the permanent magnet machine.

3.3.3 Electromagnetic torque

The expression of the electromagnetic torque developed by the machine can be obtained from the component of the input power that is transferred across the air gap. The total input power into the machine is given by:

$$P_{in} = V_a\, i_a + V_b\, i_b + V_c\, i_c + V_m\, i_m \qquad \text{(W)} \qquad (3.7)$$

When the stator phase quantities are transformed to the rotor *qd0* reference frame that rotates at a speed of $\omega r = d\theta / dt$, Equation (3.7) becomes:

$$P_{in} = \frac{3}{2}(V_q\, i_q + V_d\, i_d) + 3\, V_0\, i_0 + V_m\, i_m \qquad \text{(W)}$$

or

$$P_{in} = \frac{3}{2}(r_s(i^2_q + i^2_d) + i_q\frac{d\lambda_q}{dt} + i_d\frac{d\lambda_d}{dt} + \omega_r(\lambda_d i_q - \lambda_q i_d)) + 3\, i^2_0 r_0 + 3 i_0\frac{d\lambda_0}{dt} + i^2_m r_m + \frac{d\lambda_m}{dt} \qquad (3.8)$$

Eliminating terms in Equation (3.8) identify with the ohmic losses and the rate of change in magnetic energy. The above expression of the electromechanical power developed reduces to:

$$P_{em} = \frac{3}{2}\omega_r(\lambda_d\, i_q - \lambda_q\, i_d) \qquad \text{(W)} \qquad (3.9)$$

For a P-pole machine, $\omega_r = (P/2)\omega_{rm}$ with ω_{rm} being the rotor speed in mechanical radians per speed. Thus, equation (3.9) for a P-pole machine can also be written as:

$$P_{em} = \frac{3}{2}\frac{P}{2}\omega_{rm}(\lambda_d\, i_q - \lambda_q\, i_d) \qquad \text{(W)} \qquad (3.10)$$

Dividing the electromechanical power by the mechanical speed of the rotor can obtain the expression for the electromechanical torque developed by a P-pole machine:

$$T_{em} = \frac{P_{em}}{\omega_{rm}} = \frac{3}{2}\frac{P}{2}(\lambda_d\, i_q - \lambda_q\, i_d) \qquad \text{(N.m)} \qquad (3.11)$$

or

$$T_{em} = \frac{3}{2}\frac{P}{2}(L_d - L_q)i_d\, i_q + \frac{3}{2}\frac{P}{2}(L_{md}\, i'_{kd}\, i_q - L_{mq}\, i'_{kq}\, i_d) + \frac{3}{2}\frac{P}{2}(L_{md}\, i'_m\, i_q) \quad \text{(N.m)} \qquad (3.12)$$

3.3.4 Currents in Term of Flux Linkages

In equation (3.12) the developed electromagnetic torque is separated into two components a reluctance component, which is negative when $Ld < Lq$; an excitation components from the field of the permanent magnet.

The mutual flux linkages in the q-axis and d-axis may be expressed by

$$\lambda_{mq} = L_{mq}(i_q + i'_{kd}) \qquad \text{(Wb.turn)}$$

$$\lambda_{md} = L_{md}(i_d + i'_m + i'_{kd}) \qquad (3.13)$$

As before, the winding currents can be expressed as

$$i_q = \frac{1}{L_{ls}}(\lambda'_{kq} - \lambda_{mq}) \qquad i_d = \frac{1}{L_{ls}}(\lambda'_{kd} - \lambda_{md}) \qquad \text{(A)}$$

$$i'_{kq} = \frac{1}{L'_{lkq}}(\lambda'_{kq} - \lambda_{mq}) \qquad i'_{kd} = \frac{1}{L'_{lkd}}(\lambda'_{kd} - \lambda_{md}) \qquad (3.14)$$

With the above relations for the *d*-axis winding currents substituted into equation (3.13) and simplifying will obtain.

$$\lambda_{md} = L_{MD}\left(\frac{\lambda_d}{L_{ls}} + \frac{\lambda'_{kd}}{L'_{lkd}} + i'_m\right) \qquad \text{(Wb.turn)} \qquad (3.15)$$

where

$$\frac{1}{L_{md}} = \frac{1}{L_{ls}} + \frac{1}{L'_{lkd}} + \frac{1}{L_{md}} \qquad (3.16)$$

Similar expressions for λ_{mq} and L_{mq} can be written for the *q*-axis.

3.3.5 Steady state operation

When performing simulation, the steady state equations can be used to determine the proper steady state values to initialise the simulation so that it starts with the desired steady state condition. More importantly, the knowledge of the steady state behaviour is indispensable for checking the correctness of the simulation results.

A balanced steady state condition with the rotor rotating at the synchronous speed, ωe, and the field excitation held constant. Refer to *q*-axis on the rotor as the *qr*-axis, the *q*-axis of the synchronously rotating frame as the *qe*-axis, and the *q*-axis of the stationary reference frame which is along the axis of the *a*-phase winding as *qs*. When dealing with just one machine, it

is convenient to use the *a*-phase terminal voltage of its stator as the synchronous reference phasor. From which all phase angles are measured; in other words, the stator phase voltages can be expressed as:

$$V_a = V_m \cos(\omega_e t) \qquad \text{(V)}$$

$$V_b = V_m \cos(\omega_e t - \frac{2\pi}{3})$$

$$V_c = V_m \cos(\omega_e t - \frac{4\pi}{3}) \qquad (3.17)$$

The space vectors and phasors of the above phases voltages are referred to the *qe*-axis of a synchronously rotating frame, which start with an initial angle of θe (0) = 0 from *qs*-axis.

For balanced operation, the steady state stator currents flowing into the machine may be expressed by:

$$i_a = \mathrm{Im} \cos(\omega e t + \phi) \qquad \text{(A)}$$

$$i_b = \mathrm{Im} \cos(\omega_e t + \phi - \frac{2\pi}{3})$$

$$i_c = \mathrm{Im} \cos(\omega_e t + \phi - \frac{4\pi}{3})$$

It is evident from the above voltage and current expression that the terminal power factor angle is ϕ; the value of ϕ is positive for a leading terminal power factor condition and negative for lagging terminal power factor.

At this stage, one does not know the orientation of the rotor's *qr*-axis with respect to the synchronously rotating *qe*-axis, since the rotor is also rotating at synchronous speed in steady state, the angle between the *qr*-axis and *qe*-axis will have a steady value that is not varying with time. To locate the *qr*-axis, first transform the phase voltages and currents to the synchronously rotating frame. The *qd* components in the synchronously rotating reference frame are:

$$V^e{}_q - jV^e{}_d = V_m + j0 = V_m{}^{e^{j0}}$$

$$i^e q - ji^e d = \mathrm{Im}\cos\phi + j\,\mathrm{Im}\sin\phi = \mathrm{Im}^{e^{j\phi}} \qquad (3.18)$$

Note that, in steady state, the stator q and d voltage and current components in the synchronously rotating frame are constant. The zero sequence components of the balanced sets of phase voltages and currents are zero.

Steady state equation of the stator

Usually, only the f field winding is externally excited, that is $V'f \neq 0$, and the other rotor winding have no external input, that is $V'kd = V'kq = 0$. In steady state, the rotor is rotating at synchronous speed, that is $\omega r(t) = d\theta r(t)/dt = \omega e$. The relative speed of the rotor to the synchronously rotating resultant field in the air gap is zero; such as there will be no speed voltages in the rotor windings. Therefore, $i'f = (V'f/r'f)$, and the other rotor currents, $i'kd$ and $i'kq$ are zero. Since both stator and rotor currents are constant, the flux linkage, λd and λq, will also be constant and the $d\lambda d/dt$ and $d\lambda q/dt$ term in equation (3.1) will be zero. Thus, in steady state, the qd voltage equations of the stator windings in the rotor qd reference frame will reduce to:

$$Vq = rs\,iq + \omega e\,Ld\,id + Em \qquad (V)$$

$$Vd = rs\,id - \omega e\,Lq\,iq \qquad (3.19)$$

Locating the rotor's qr-axis

Now define the angle, $\delta(t)$, between the qr and de axis as

$$\delta(t) = \theta r(t) - \theta e(t) \qquad \text{(elect.rad.)} \qquad (3.20)$$

$$\delta(t) = \int_0^t \{\omega r(t) - \omega e\} dt + \theta r(0) - \theta e(0) \qquad (3.21)$$

where $\theta r(t)$ is the angle between the rotor's qr-axis and the axis of the stator a-phase winding, and $\theta e(t)$ is the angle between the qe-axis of the synchronously rotating reference frame and the same a-phase axis. As defined, δ is the angle between the qr-axis of the rotor and the qe-axis of the synchronously rotating reference frame, measured with respect to the qe-axis. In steady state, with the rotor rotating at the same synchronous speed, that is $\omega r(t) = \omega e$, the angle δ, will be constant. The steady state value of δ will have accumulated contributions from all three terms on the right side of the second equation in equation (3.20).

The value of δ can be determined as follow: when the qd component equations in equation (3.19) are put in the complex form, that is:

$$(Vq - jVd) = (rs + j\omega e\, Lq)(iq - j\, id) + \omega e(Ld - Lq)\, id + Em \quad (V) \qquad (3.22)$$

3.3.6 Steady state Torque Expression

The total complex power into all three phases of the stator windings is given by:

$$S = 3(\vec{V}q - j\vec{V}d)(\vec{I}q - j\vec{I}d)^* \qquad (VA) \qquad (3.23)$$

The electromagnetic power developed by the machine is obtained by subtracting from the input real power the losses in the stator, which in this model is just the copper losses in the stator windings. Thus, subtracting $3(I^2q + I^2d)rs$ from the real part of the input power the expression for electromagnetic power is:

$$Pem = \Re\left[3\,(\omega\, eLd\,\vec{I}\,d + \vec{E}\,m + j\,\omega\, e\,Lq\,\vec{I}\,q)(\vec{I}\,d + j\,\vec{I}\,q)\right] \quad \text{(W)}$$

$$Pem = 3\,(Em\,Iq + \omega\, e(Ld - Lq)Id\,Iq) \qquad (3.24)$$

The expression for the electromagnetic torque developed by the machine is obtained by dividing the expression for the electromagnetic power by the actual rotor speed, that is:

$$Tem = \frac{Pem}{\omega sm} = \left(\frac{2}{P\omega e}\right)Pem \qquad \text{(Nm)}$$

$$Tem = 3\left(\frac{2}{P\omega e}\right)\{Em\,Iq + \omega\, e\,(Ld - Lq)\,Id\,Iq\} \qquad (3.25)$$

The first torque component is the main torque component in a permanent magnet synchronous machine with constant magnet field excitation. The second component is referred to as the reluctance torque component. It is present only when there is rotor saliency.

3.3.7 Simulation of three Phase Permanent Magnet Synchronous Machine

The winding equation of the permanent magnet synchronous machine model derived in last section can be implemented in a simulation that uses voltages as input and currents as output. The main input to the machine simulation is the stator *abc* phase voltages and the applied mechanical torque to the rotor.

The *abc* phase voltages of the stator windings must be transformed to the *qd* reference frame attached to the rotor. Although the angle $\theta r(t)$ increases with time, $\cos \theta r(t)$ and $\sin \theta r(t)$ can be obtained from a variable frequency oscillator circuit which has a provision for setting the proper initial value of θr. The transformation from *abc* to *qd* rotor may be performed in two separate steps. In two steps, the intermediate output from the first step are the stator voltage in the stationary *qd* reference frame, that is:

$$V^s q = \frac{2}{3} Va - \frac{1}{3} Vb - \frac{1}{3} Vc$$

$$V^s d = \frac{1}{\sqrt{3}} (Vc - Vb)$$

$$V0 = \frac{1}{3} (Va + Vb + Vc) \qquad (3.26)$$

$$Vq = V^s q \cos \theta r(t) - V^s d \sin \theta r(t)$$

$$Vd = V^s d \sin \theta r(t) - V^s q \cos \theta r(t) \qquad (3.27)$$

where

$$\theta r(t) = \int_0^t \omega r(t)\, dt + \theta r(0) \qquad \text{(elect.rad.)} \qquad (3.28)$$

Expressing the *qd0* voltage equations as integral equations of the flux linkages of the windings, the above stator *qd0* voltage along with other inputs can be used in the integral equations to solve for the flux linkages of the windings. For the case of a machine with only

one field winding in the d-axis and a pair of damper winding in the *d*-axis and *q*-axis, the integral equations of the winding flux linkages are as follows:

$$\psi q = \omega b \int \left\{ Vq - \frac{\omega r}{\omega b} \psi d + \frac{rs}{xls} (\psi mq - \psi q) \right\} dt$$

$$\psi d = \omega b \int \left\{ Vd - \frac{\omega r}{\omega b} \psi q + \frac{rs}{xls} (\psi md - \psi d) \right\} dt \qquad (3.29)$$

$$\psi 0 = \omega b \int \left\{ (V0 + \frac{\omega r}{\omega b} \psi 0) \right\} dt$$

$$\psi' kq = \frac{\omega b \, r' kq}{x' lkq} \int (\psi mq - \psi' kq) \, dt = 0$$

$$\psi' kd = \frac{\omega b \, r' kd}{x' lkd} \int (\psi md - \psi' kd) \, dt = 0$$

The above equations are in generating convention, that is with the currents, *iq* and *id* into the positive polarity of the stator windings' terminal voltages. As before, to handle the cut set of inductors in the *q*-axis and *d*-axis circuits, express the mutual flux linkages in terms of the total flux linkages of the windings as:

$$\psi mq = xMQ \left(\frac{\psi q}{xls} \right)$$

$$\psi md = xMD \left(\frac{\psi d}{xls} + i'm \right) \qquad (3.30)$$

Having the values of the flux linkages of the windings and those of the mutual flux linkages along the *q*-axis and *d*-axis, can determine the winding currents using:

$$iq = \frac{\psi q - \psi mq}{xls} \qquad i' kq = \frac{\psi' kq - \psi mq}{x' lkq}$$

$$id = \frac{\psi d - \psi md}{xls} \qquad i'kd = \frac{\psi' kd - \psi md}{x'lkd} \tag{3.31}$$

The stator winding *qd* currents can be transformed back to *abc* winding currents using the following rotor to stationary *qd* and stationary *qd0* to *abc* transformations:

$$i^S q = iq \cos\theta r(t) + id \sin\theta r(t)$$

$$i^S d = -id \sin\theta r(t) + id \cos\theta r(t) \tag{3.32}$$

$$ia = i^S q + i0$$

$$ib = -\frac{1}{2} i^S q - \frac{1}{\sqrt{3}} i^S q + i0$$

$$ic = -\frac{1}{2} i^S q + \frac{1}{\sqrt{3}} i^S d + i0 \tag{3.33}$$

Torque expression is as given in equation (3.25) and the electromechanical torque developed by a machine with P-poles convention is:

$$Tem = \frac{P\,em}{\omega rm} = \frac{3}{2}\frac{P}{2}(\lambda d\, iq - \lambda q\, id) \qquad \text{(Nm).} \tag{3.34}$$

The value of T_{em} from the above expression is positive for motoring and negative for generating operation.

The net acceleration torque, $T_{em} + T_{mech} - T_{damp}$, is in direction of the rotor's rotation. T_{em}, the torque developed by the machine, is positive when the machine is motoring and negative when the machine is generating, T_{mech}, the externally applied mechanical torque in the direction of rotation, will be negative when the machine is motoring a load and will be

positive when the rotor is being driven by a prime mover as in generating; and T_{damp}, the frictional torque, acts in a direction opposite to the rotor's rotation. Equating the net acceleration torque to the inertia torque:

$$Tem + Tmech - Tdamp = J\frac{d\omega rm(t)}{dt} = \frac{2J}{p}\frac{d\omega r(t)}{dt} \quad \text{(Nm)} \qquad (3.35)$$

Equation (3.35) for the motion of the rotor assembly, expressed in per unit is:

$$Tem(pu) + Tmech(pu) - Tdamp(pu) = \left(\frac{1}{Tb}\right)\frac{2J}{p}\frac{d\omega r(t)}{dt} \quad \text{(pu)} \qquad (3.36)$$

When the base torque is $T_b = S_b/\omega bm$, the base mechanical angular frequency, ωbm is $2\omega_b/P$, where ω_b is the base electrical angular frequency and P is the number of poles.

In terms of the inertia constant, H that is defined as $H = \frac{1}{2}J\omega^2_{bm}/S_b$ expression is:

$$Tem(pu) + Tmech(pu) - Tdamp(pu) = 2H\frac{d(\omega r/\omega b)}{dt} \quad \text{(pu)} \qquad (3.37)$$

Figure 3.5 shows an overall diagram of the SIMULINK simulation. Figures 3.6 to 3.12 show the subsystems of variables in a simulation of a three phase permanent magnet synchronous machine as the rotor reference frame.

The input to the simulation is the stator *abc* voltages, and the externally applied mechanical torque, T_{mech}, on the rotor. The direction of the stator currents are into positive polarity of stator winding's terminal voltages. Since the thesis mainly uses the simulation for generating

operation, Figure 3.5 shows the stator *qd0* currents opposite in sign to those used for motoring. This step is taken merely for convenience.

In Figure 3.5 the transformation of the input stator *abc* voltages to the rotor *qd* reference frame are performed inside the *abc_to_qd0* block. Details of the *abc_to_qd0* block are shown in Figure 3.6. The transformation uses the *cos θr(t)* and *sin θr(t)* generated by the *oscillator* block.

Figure 3.7 the *qd_gen* block contains the simulation of the generator proper in its rotor reference frame. The simulation of the *q*-axis circuit equations with one damper winding on the rotor is performed inside the *q_cct* block, that of the *d*-axis circuit with one rotor winding and magnetic field inside *d_cct* block. The details of the *q_cct* and *d_cct* blocks are shown in Figure 3.8 and 3.9 respectively. A close examination of the manner in which the rotor winding equations are implemented in the simulation reveals that the addition of another rotor winding having a common magnetizing reactance with other windings of the same axis can be done.

The equation associated with rotor motion and rotor angle are implemented inside the rotor block. The details of the *rotor* block are shown in Figure 3.10.

When using a two stage transformation between *abc* and *qd0* variables, the *cos θr(t)* and *sinθr(t)* terms are generated by a variable frequency oscillator. Details of such an oscillator circuit are shown in Figure 3.11. The initial values of the *cos* and *sin* integrators in this block can be set to give any desired initial values of *θr(0)*. For example, using initial values of $\cos\theta r(0) = 1$ and $\sin\theta r(0) = 0$ corresponds to starting the rotor q-axis and d-axis with an initial value of $\theta r(0) = 0$ to the axis of the stator winding and *θr(t)* will be equal to *ωr(t)*.

The transformation of the *qd0* rotor reference currents back to *abc* stator currents are performed inside the *qd_to_abc* block. The details of the *qd_to_abc* block are shown in Figure 3.12.

In Figure 3.13 the transformation of three phase supply source block is controlled by the torque and is connected to *abc_to qd0* block.

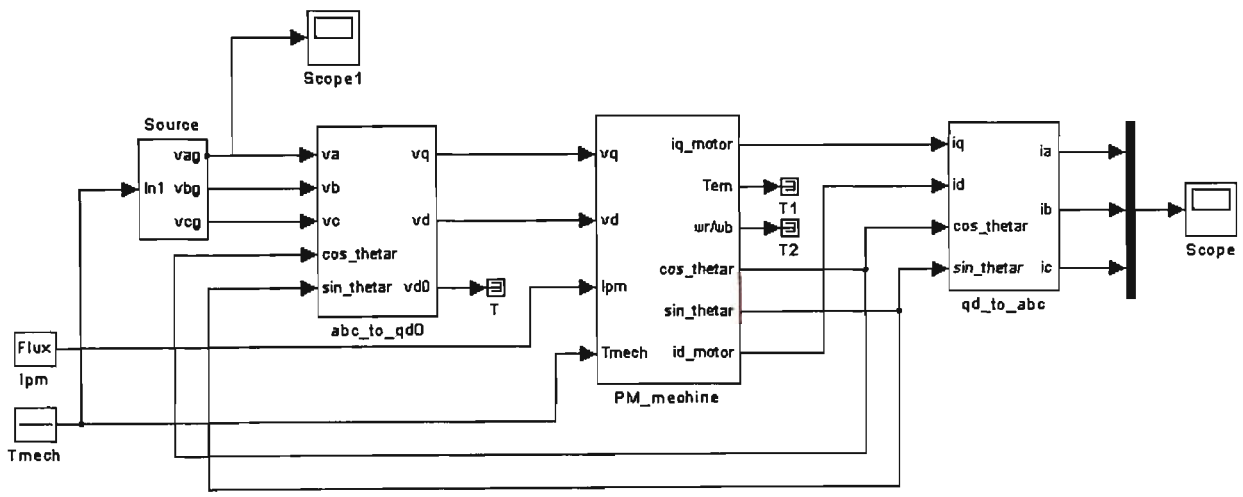


Figure 3.5 Simulation overall diagram of permanent magnet generator

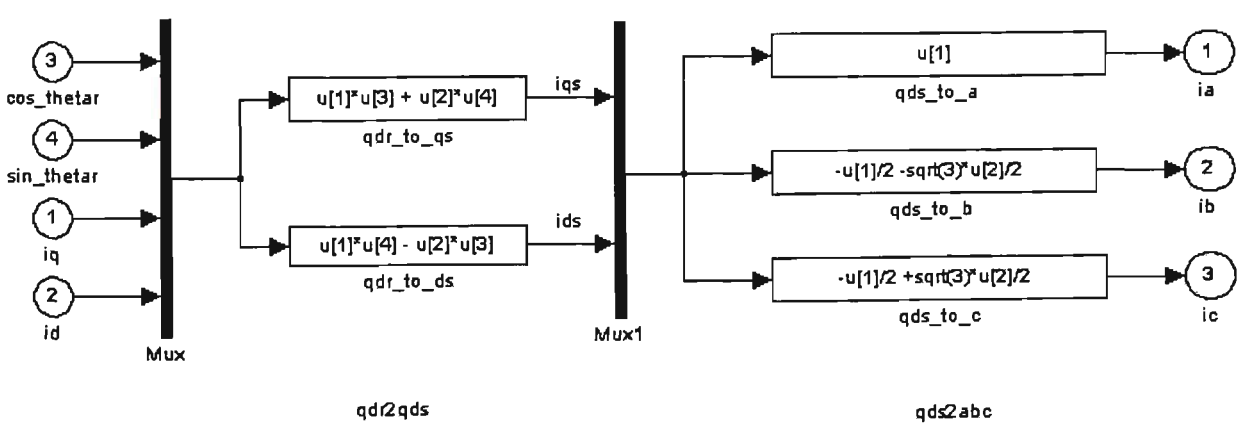


Figure 3.6 Inside the abc_to_qd0 block

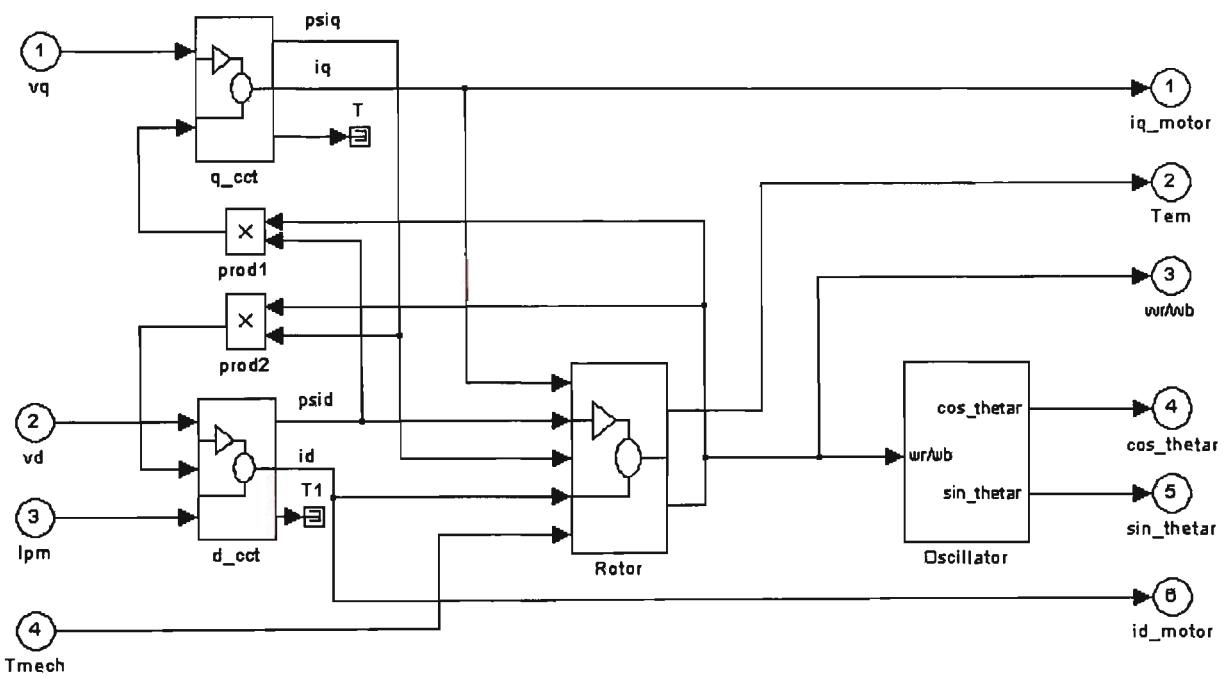


Figure 3.7 Inside the qd_gen block

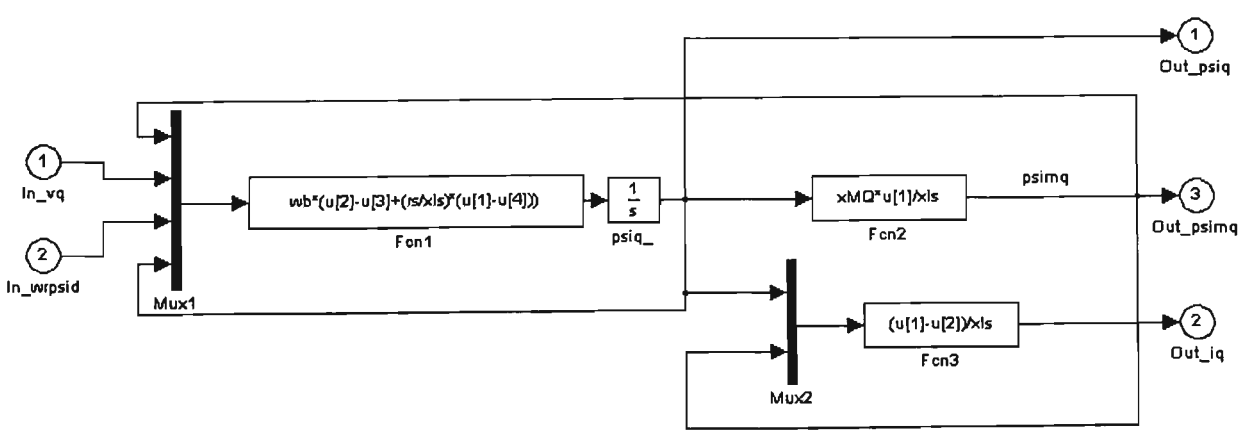


Figure 3.8 Inside the q_cct block

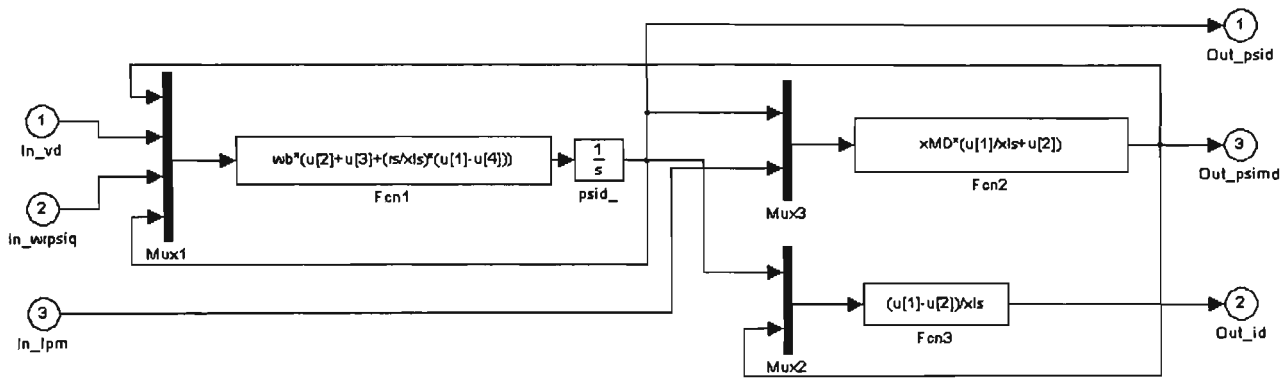


Figure 3.9 Inside the d_cct block

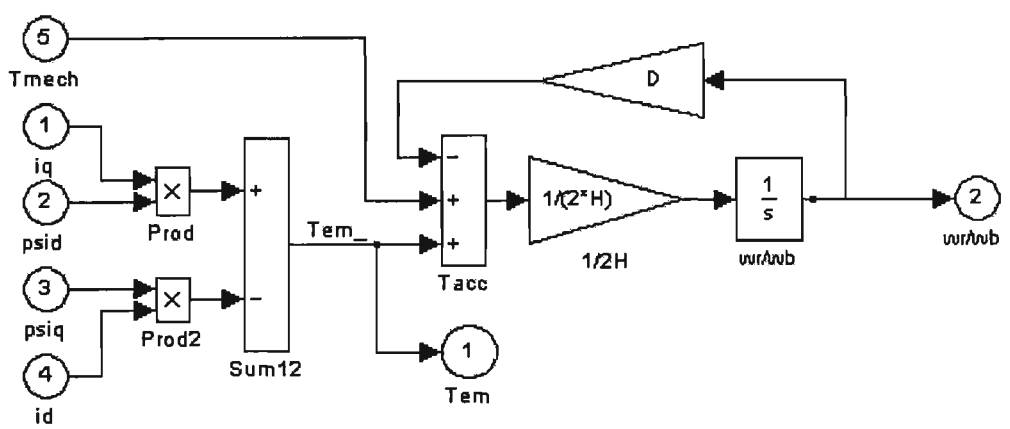


Figure 3.10 Inside the rotor block

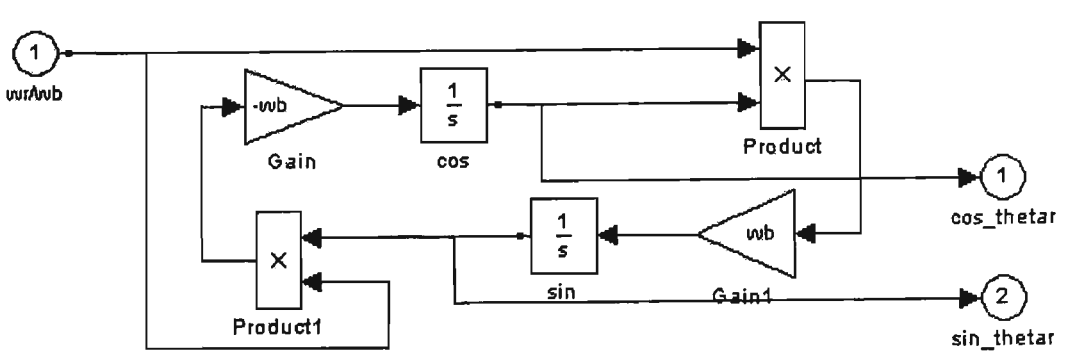


Figure 3.11 Inside the oscillator block

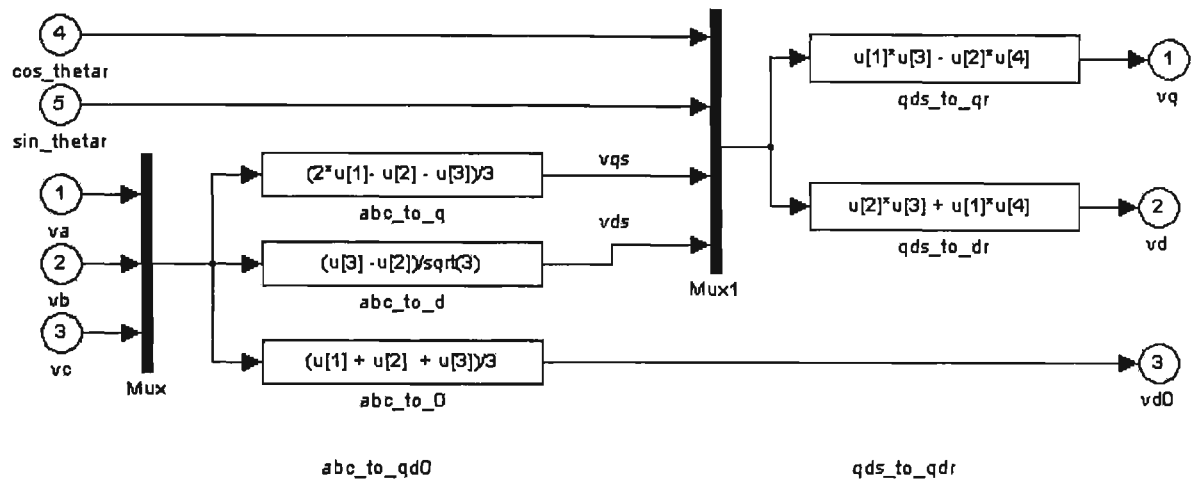


Figure 3.12 Inside the qd_to_abc block

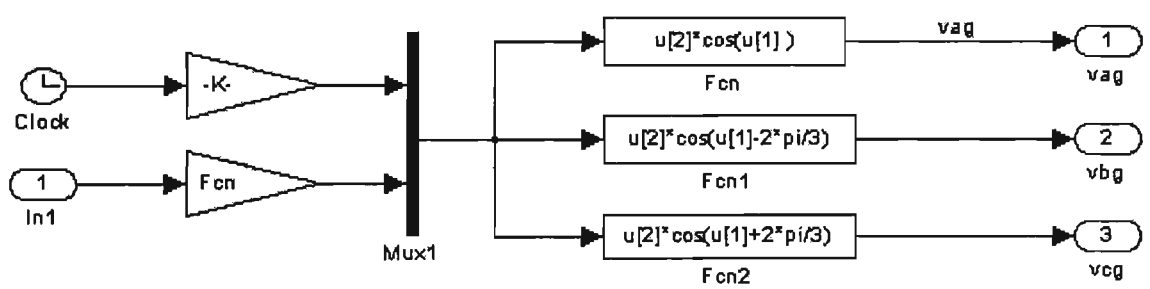


Figure 3.13 Inside the source block

Interfacing the Electrical Circuit with Simulink

The Voltage Measurement block acts as an interface between the PSB and the SIMULINK blocks. The link is done from the electrical system to the SIMULINK system. The Voltage Measurement block converts the measured voltages into SIMULINK signals. Note that the Current Measurement block from the Measurements library of POWERLIB can also be used to converted to any measured current into a SIMULINK signal [29,30].

The link from SIMULINK blocks to the electrical system is also possible. For example, one can use the Controlled Voltage Source block (Figure 3.14) to inject a voltage in an electrical circuit. The voltage is then controlled by a SIMULINK signal.

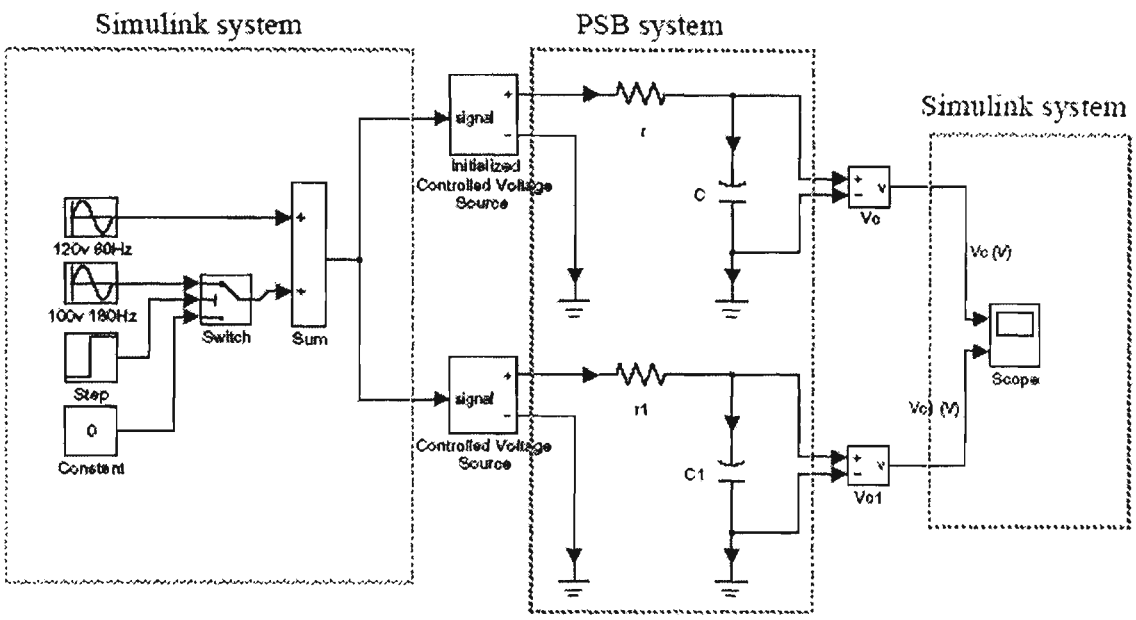


Figure 3.14 Example for SIMULINK Blocks link to Power System Blockset system

Permanent magnet generator parameters

The machine data from manufacturers are usually in the form of reactance, resistance, power rating, rated currents, voltages, torque and time constants. The data of permanent magnet generator shows as following:

Horse power	35.7 HP
Kilowatts	26.7 kW
Maximum operating speed	3000 rpm
Speed at rated torque	2400 rpm
Continuous rated torque at rated speed	105 Nm
Continuous stall torque	123 Nm
Peak torque	370Nm

Peak current	186 Amps
Maximum theoretical acceleration	28,461 rad / sec^2
Torque sensitivity	2.0 Nm / Amp
Back EMF line to line	158 Vrms / Krpm
DC resistance	0.193 Ohms
Inductance	3.16 MilliHenries
Rotor inertia	0.0130 Kg - m^2
Static friction	1.5 Nm

3.3.8 Transmission Line parameters

In this project, Powercor Australia Ltd provided the information of transmission line. The transmission line constructs underground cable. The cable is about 3km long and 4 core three phase and netural 16mm^2 each core is 7 strand. When the transmission line is shorter than 80km then it is classified short transmission line. The circuit is solved as a simple series AC circuit.

$$I_{sending\ end} = I\ receiving\ \ end$$

$$V_{sending\ end} = V_{\ receiving\ end} + I_{\ receivin\ gend} \times Z$$

where Z is the total series impedance of the line.

From the measurement given, phase to earth resistance is 27.5 Ohms and the phase to earth is 1.6 degree of phase angle.

The PSB provides a series of models. The three phase RLC series branch is selected for this case. The parameters are converted to resistance and inductance is applied in the power system.

3.3.9 The three phase transformer parameters

Two identical three phase step down transformers are applied in this power system project. From the information obtained, three phase star connection 4050V to 266V step down or up transformers, the capacity is 25kVA power. The rated frequency is 70Hz. The lack of information can not be directly applied in the MATLAB power system model. The open and short circuit transformer test is used to work out the parameters of transformer.

The practical test performed and the results are as follow:

Table 3.1 Transformer open circuit test:

	A phase - B phase	B phase – C phase
Voltage	246.2V	247.4V
Watt	386.7W	-607.2-3W
P.F.	0.46	-0.82
Ampere	3.275A	2.965A
VA	837.4VA	737.1VA
VAR	741.5VAR	420.5VAR

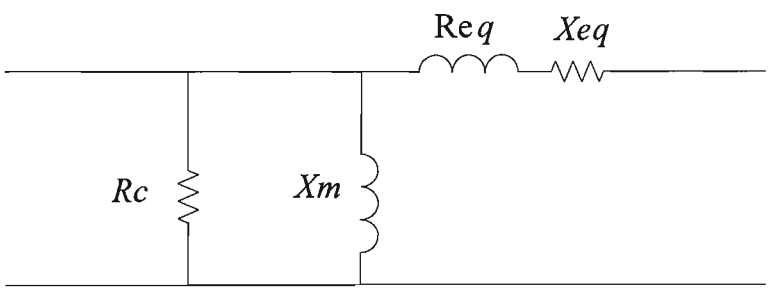
Table 3.2 Transformer short circuit test:

	A phase - B phase	B phase – C phase
Voltage	40.11V	43.07V
Watt	-113.6W	-133.9W
P.F.	-0.84	-0.96
Ampere	3.371A	3.469A
VA	134.7VA	139.2VA
VAR	72.43VAR	38.34VAR

Table 3.3 Transformer DC resistance test:

A – N = 1.652 Ohms	a – n = 0.0466Ohms
B – N = 1.686 Ohms	b – n = 0.0444Ohms
C – N = 1.765 Ohms	c – n = 0.0433 Ohms

The equivalent circuit (referred to H.V.) side is:



where Req = 1677.35 Ohms Leq = 301.74 Ohms

Xm = 63482 Ohms Rc = 63485 Ohms

The results are obtained from experiment and mathematic calculation. MATLAB transformer block model complied with industry standards [31]. The per unit conversion is applied to the resistance and inductance of the windings in per unit (p.u.). The values are based on the transformer rated power Pn, in VA, nominal frequency fn, in Hz, and nominal voltage Vn, in Vrms, of the corresponding winding. For each winding, the per unit resistance and inductance are defined as:

$$R(p.u.) = \frac{R(\Omega)}{R_{base}}$$

$$L(p.u.) = \frac{L(H)}{L_{base}}$$

Magnetization resistance and reactance

For the magnetization resistance *Rm* and inductance *Lm*, the p.u. values are based on the transformer rated power and on the nominal voltage of winding at the high voltage side. The resistance and inductance simulate the core active and reactive losses, both in p.u. The p.u. values are based on the nominal power *Pn*. Typical value of *Rm* = 500p.u. and *Lm* = 500p.u. is used in order to specify 0.2% of active and reactive core losses at nominal voltage.

3.4 Load Flow & Fault Analysis

3.4.1 Load flow simulation analysis

The load flow simulation is a general simulation application program to illustrate and determine the power transfer in electrical power system. An electrical power system can be represented by a single line diagram, which comprises an interconnected system of entities or elements including generators, transformers, transmission and distribution lines and loads. All electrical components such as cables and transformers have resistances, which will experience real power losses in electrical network distribution as well as real and reactive power flows for all equipment connecting the buses can be computed by means of load flow simulation

MATLAB has been chosen as the simulation tool for this research because of the simplicity of manipulation of matrix structures and inputs. It has in-built routines such as inverse functions, absolute function, graphing facilities to plot convergence of load flow and a scripting system which allows to develop and modify the software for own needs.

The SIMULINK model result is as shown in Figure 3.15. The input of data is created by the ocean wave energy. The data was obtained as explained in Chapter 2. The waveforms present the variance frequency, torque and load to gain different results. These results are further discussed in the Chapter 5.

3.4.2 Fault simulation analysis

Fault simulation is an important part of power system analysis. The problem consists of determining the generator voltages and line currents during various types of faults. Faults on power system are divided into three phase balanced and unbalanced faults. Different types of unbalanced faults are single line to ground, line to line, and double line to ground and the simulation results are represented in Chapter 5.

The information from the simulation results of fault analysis are used for proper setup of the microprocessor based protection and coordination. The three phase balanced fault information is used to select and setup the phase protection, while the line to ground fault is used for ground protection. Fault analysis is also used to obtain the rating of the protective switchgears.

The magnitude of the fault current depends on the internal impedance of the generator plus the impedance of the intervening circuit. The reactance of a generator under short circuit condition is not consistent. For the purpose of fault analysis, the generator behaviour is presented under the transient period, covering a relatively longer time.

MATLAB Power System Blockset provides a fault model. It can apply to simulate the fault studies. The SIMULINK model is the same as pervious model as shown in Figure 3.15. The faults are distributed at different location of the power system. The faulted voltages and currents waveforms are presented in Chapter 5.

3.5 Frequency Response Analysis

Frequency response of a system is the steady state response of the system to a sinusoidal input signal. The advantage of the frequency response is that the transfer function of a system can be determined experimentally by simulation frequency response. Furthermore, the design of a system in frequency domain provides the control over the system bandwidth and disturbance on the system response [32].

Frequency response analysis is a classical method for obtaining models of linear and non-linear time invariant systems. However, the quality of the results strongly depends on the simulation, experiment design and choice of parameters. Likewise, different problems may demand fault diagnostic. Frequency response analysis is most widely used methods to determine the dynamic of a stable linear system. In this research, frequency response analysis is used to analyse a simulation model of the ocean wave power system. This is also used to find out the impedance via frequency response in the power system.

MATLAB power system blocks provide an Impedance Measurement block. It can measure the impedance of a circuit as a function of frequency. The Impedance Measurement block measures the impedance between two nodes of a circuit as a function of frequency. It consists of a current source *Iz,* connected between inputs one and two of the Impedance Measurement block, and a voltage measurement *Vz*, connected across the terminals of the current source. The network impedance is calculated as the transfer function *H(s)* from the current input to the voltage output of the state-space model. The measurement takes into account the initial state of the Breaker and Ideal Switch blocks. It also allows impedance measurements with distributed parameter line blocks in power system circuit.

Impedance Measurement block connects on the end of the power system such as the end of step down transformer. The result shows single phase of zero-sequence impedance of a balanced three-phase circuit as shown in Figure 3.15.

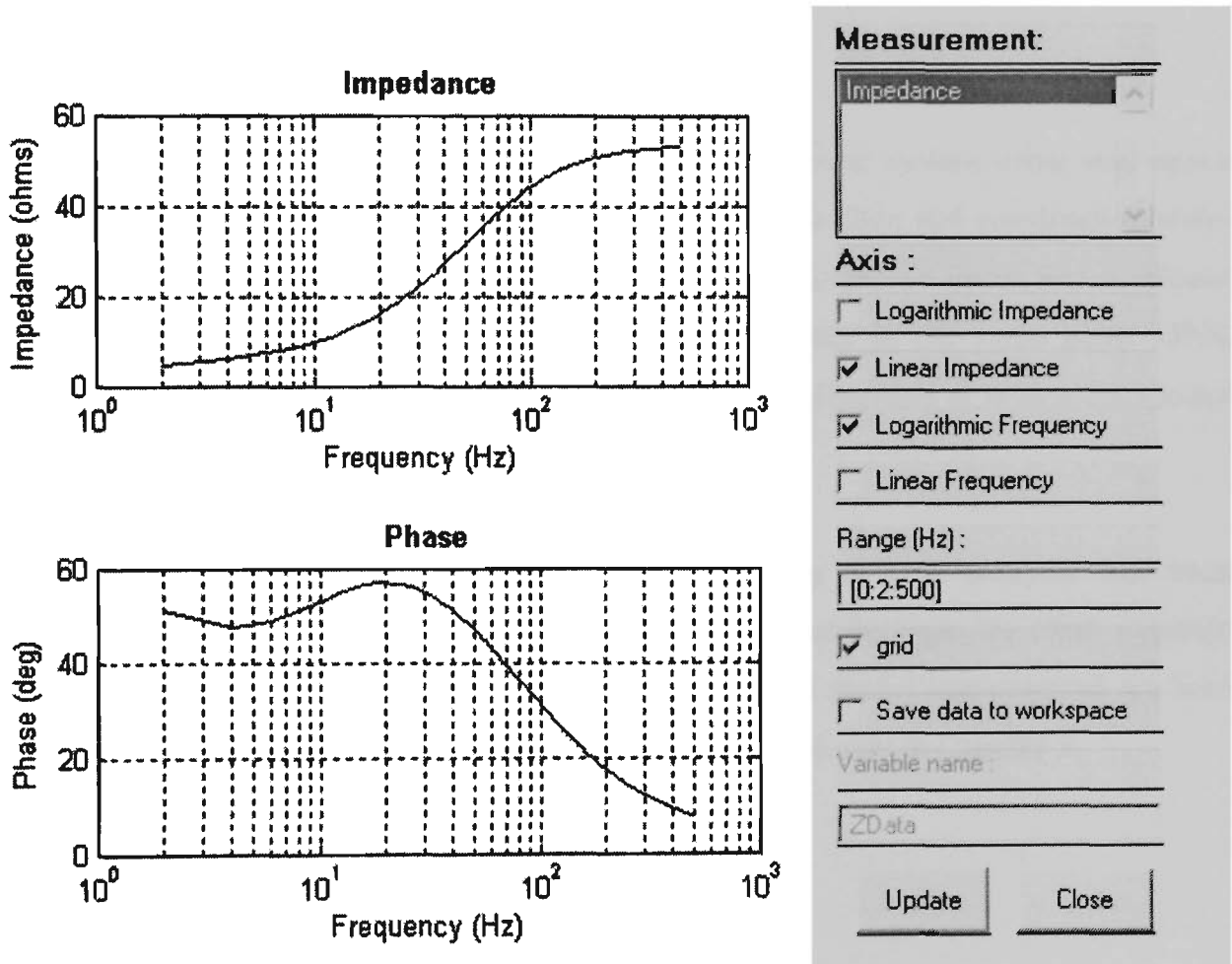


Figure 3.15 The impedance measurement result

The first plot is the impedance of a frequency response in Ohms. The second plot is the phase function $\theta(w)$. The logarithmic representation is useful in that it shows both the low and high frequency characteristics of the transfer function in the power system.

Due to limitation of MATLAB, during the impedance measurement it can simulate only non-linear blocks like the Breaker, the Ideal Switch, and the Distributed Parameter Line. All other non-linear blocks, such as machines and power electronic devices, are not considered, and they are disconnected during the measurement.

3.6 Conclusion

A unified method for modelling and simulation of electrical power system using state-space approach has been presented. By using a SIMULINK control toolbox and constructing state-space model of permanent magnet generator it is possible to combine linear and nonlinear circuits, electric machine, power electronic, and control blocks in the same SIMULINK diagram. Simulation with either variable time step integration algorithm or with a discretized system using fixed time step is possible.

MATLAB SIMULINK models for wave power generating system analysis has been presented in Figure 3.16. These models represent the interaction between the other dynamic elements within the power network. The overall power system model implemented are load flow, transient fault and frequency analysis. These analysis are shown in Chapter 5.

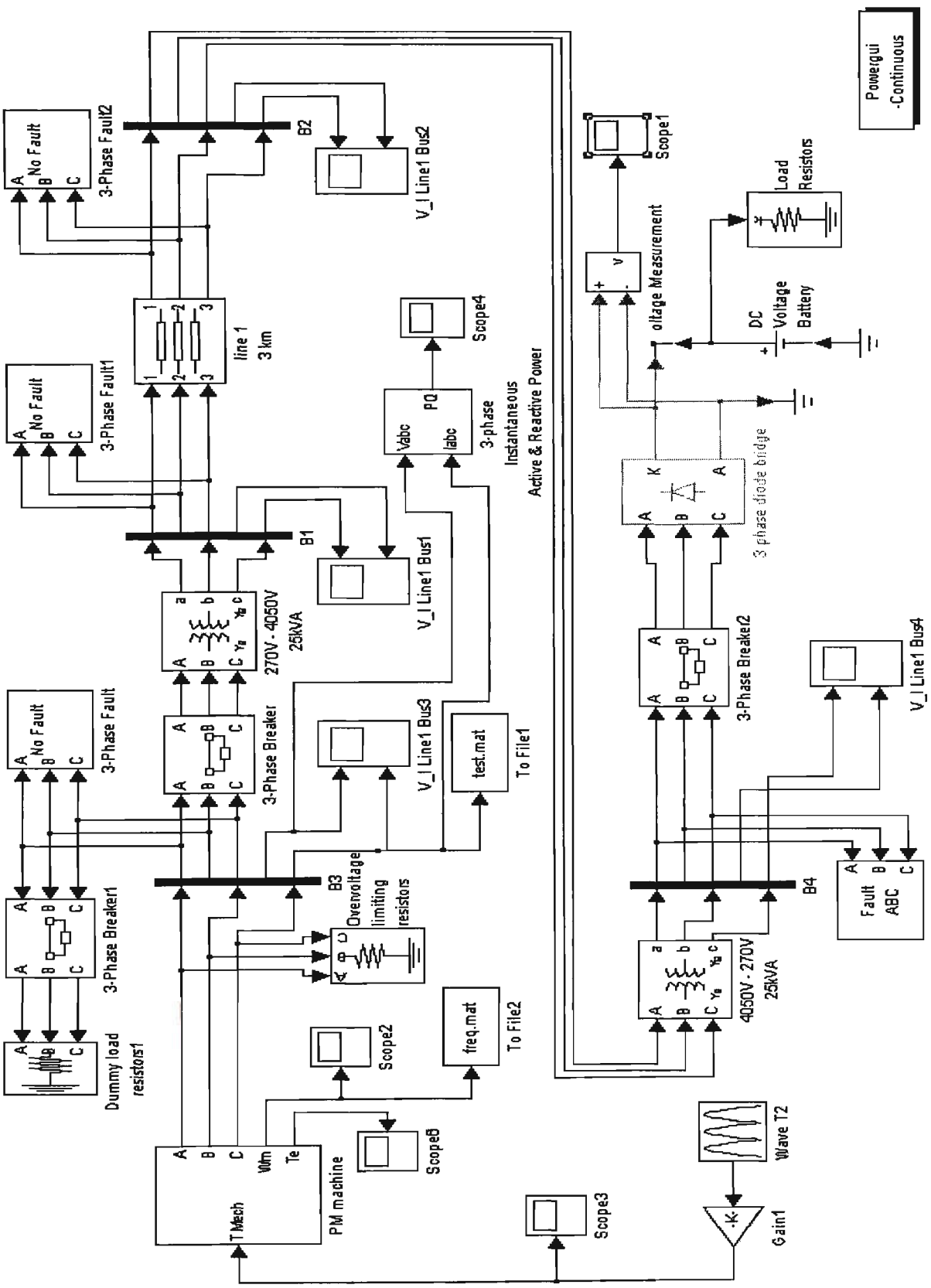


Figure 3.16 MATLAB simulink models for wave power generating system

Chapter 4 HARDWARE DESIGN & DEVELOPMENT

4.1 Introduction

The experimental system has been developed and commissioned in the power laboratory at Victoria University of Technology. This system has been set up to verify the performance of microprocessor based circuit breaker and determine the rated setting.

The experimental system consists of the following components:

1. A 50kVA IGBT DC to AC power converter.
2. ABB Isomax S4 100A microprocessor-based three phase circuit breaker.
3. Three of single phase 2kVA 400V/50V step down transformers.
4. A Pentium III 866MHz computer as running a real time windows target with MATLAB software package and WINDOWS98 system.
5. A three phase 125A isolation AC and DC current transducer.
6. Digital oscilloscope, Power meter, Stopwatch, etc.

The hardware, software design, development and testing of each component of the experimental system has been carried out in this study and are described in the following section.

4.2 50kVA IGBT DC to AC power converter

4.2.1 Introduction

The goal of this section is to present the design and construction of a SIMULINK based real time controlled three phase converter. The purpose of this converter is to provide an existing current from the DC–AC converter by the simulation result. The SIMULINK based controlled converter allows to adjust parameters and quickly iterate it to achieve required results. The SIMULINK toolbox in MATLAB is used to simulate switching schemes and the generated C-code for parallel output port is generated by Real Time Workshop. Using these technologies along with a readily available parallel port from the Real Time Windows Target, a customised code is directly generated and downloaded to the control port. The advantage of this three phase converter are shown as follows:

- Experimental Model: Assorted types of switching schemes and loads can be easily added and tested.
- SIMULINK based control: It employs graphical user interface (GUI) for building and simulating models stressing quick setup times.
- Measurement transducer: Provide access to diversity of measurements to the basic part of the converter.

With respect to this experiment, it is desirable to show how the converter is used for the purpose of demonstrating and analysing the converter output waveform. Section 4.3.2 presents the software development methodology to obtain the pulse width modulation (PWM) switching schemes from SIMULINK based in MATLAB.

4.2.2 Design of Power Converter

The objective of this section is to design a three phase converter as driving variable frequency to operate the microprocessor based circuit breaker. In order to realise this objective, it was necessary to build a converter that is able to run with different frequency of pulse width modulation (PWM) switching schemes and loads. Computer interface with parallel port is required to accomplish control schemes and data acquisition.

As shown in Figure 4.1, the overall system consists of a three phase input power supply, rectifier, filter, full bridge converter and load. The three phase input power supply is from the utility. The rectifier converts an AC input to DC output in an uncontrolled manner. A low pass filter makes the DC output to be as ripple free as possible. A three phase full bridge converter changes a DC input into a symmetrical AC output. Since DC input is not controllable, the AC output of the converter is controlled by pulse width modulation (PWM) and fed to the load such as step down transformer. From the converter setup block diagrams, relevant hardware issues are discussed in subsequent section.

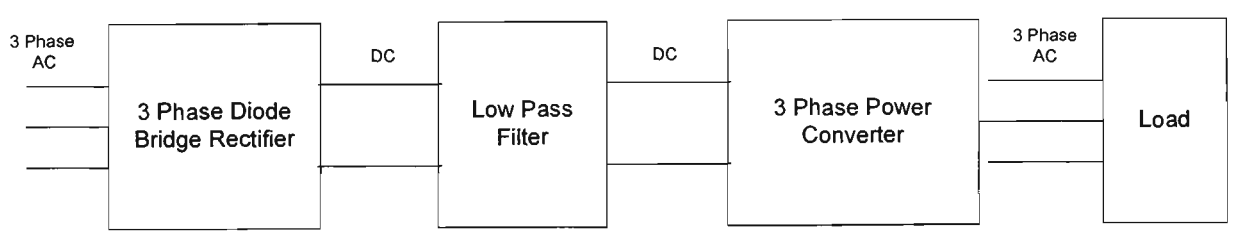


Figure 4.1 Converter block system

4.2.3 Rectifier Circuit

The function of a rectifier is to convert an AC signal into an unidirectional signal. The input is directly supplied from the utility. The rectifier draws highly distorted current during turn off time in each diode and generates harmonic to the load at the same time. In order to obtain the

DC input voltage, a three phase diode bridge and a filter are used to achieve lower ripple content in the waveforms and higher power. The power distribution board of the power laboratory of Victoria University provides the three phase full bridge rectifier and transformer. Each sub-distribution board supplies the DC output voltage of 280V and output current of 30A.

4.2.4 Filters

Since the output of the rectifier is DC, a low pass filter is desired. A DC filter is designed to smooth out the DC output and filter out harmonic on the input of the converter.

The Figure 4.2 shows that the higher order harmonic contents are attenuated by the inductor, while the capacitor helps keeping the DC voltage constant. In addition, the capacitor also acts as a snubber for switches in the converter. It is necessary to make the load impedance much greater than the impedance of the capacitor, to allow the n^{th} harmonic current to pass through the filter capacitor. The converter is designed to handle a range of loads, therefore only the relationship between the inductor and capacitor can be determined. For anticipated loads this relationship is:

$$L \times C = 10.051 x 10^{-6}$$

An inductor was wound and tested. The range used was from 5.5mH to 14.3mH over a varying load current. Due to availability, the capacitor chosen was 450V, 1700uF. Hence the L x C relationship while can be obtained from 9.35 x 10^{-6} to 24.31 x 10^{-6}.

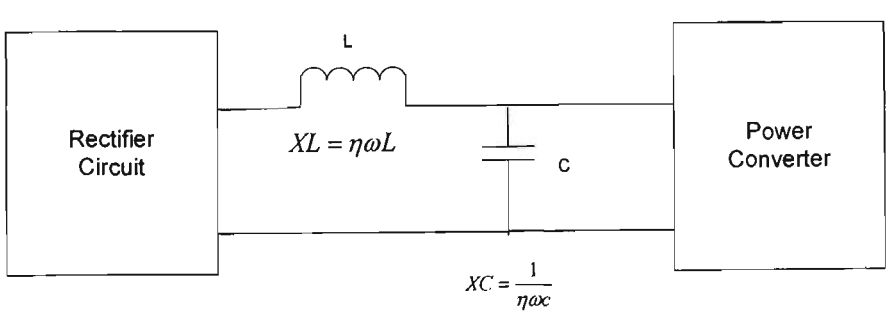


Figure 4.2 Low pass filter circuit

4.2.5 Description of Converter

The function of the converter is to change a DC input voltage to a symmetrical AC output voltage with controllable magnitude and frequency. The DC input of the converter is obtained from output of the low pass filter of the rectifier. Each switch of the converter is operated by a small control signal. Switches in the same leg are not turned on at the same time, to prevent short circuit in a leg. Blanking time must be added to make sure that there are no two switches in the same leg turning on at the same time. Switches in the converter are fully controllable power devices with fast switching speed, because this converter is expected to handle various kind of switching schemes, including ones that require high switching speeds. High negative blocking capability is also necessary for preventing the switch from damage. The voltage and current rating of power devices in converter depends on the types of load and switching schemes. For unidirectional switching, as in this converter, an anti-parallel connected diode is needed to provide a path for the current in the event that there is an inductive load. If an attempt is made to open switch, the energy stored in the inductor will be transformed in order to maintain the direction of the current. This may cause damages to the switch. Such a diode is called a free-wheeling diode.

4.2.6 Description of IGBT Drive Circuit

The function of a drive circuit is to turn the switch from an *off* state to *on* state and vice-versa. The power rating of the drive circuit may vary depending on the type of switches being used. A drive circuit also creates a blanking time for switches; therefore, a high speed drive circuit is desired for such a converter that is able to operate with various switching schemes including high speed switching. A drive circuit must electrically isolate the control signal from power switch, for safety reason and undesirable interferences. Overcurrent protection for the power switch is also essential [33], in which a communication between a drive circuit and control board is required [34-38].

Theoretically, the operation of the circuit is quite straightforward. A modulation signal such as parallel port output drives through an optocoupler to logical gate, passes through a set-reset latch structure which locks out when an over current is detected, and then is level converted by the TC4429 high speed power MOSFET driver to drive the IGBT gate through the *Rg*. However, the particular selection and arrangement of the components of the drive circuit is a good compact and warrants protecting output circuit.

Figure 4.3 shows the schematic of the Insulated Gate Bipolar Transistors (IGBT) gate driver circuit.

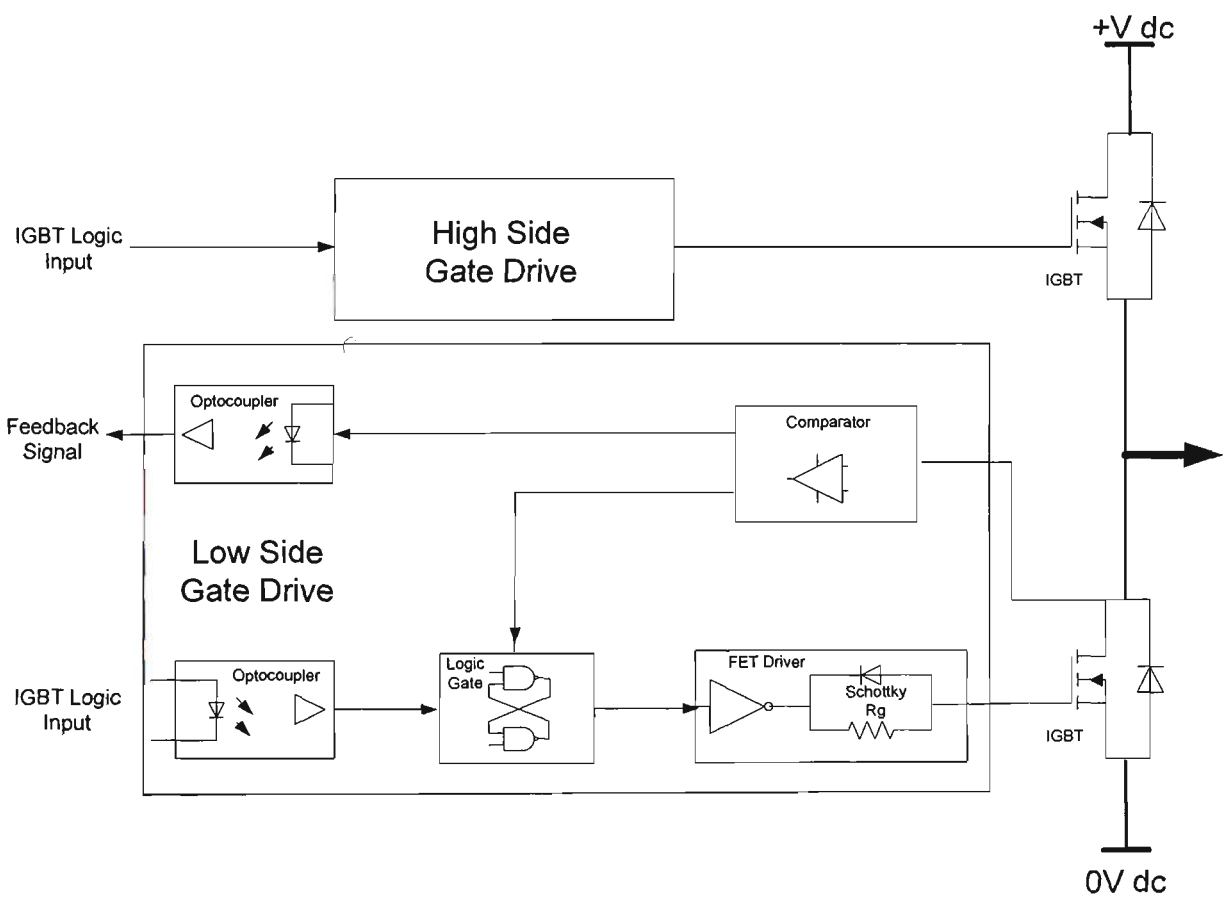


Figure 4.3 IGBT Drive Circuit with Instantaneous Over current Protection

Otpocoupler

The selection of the optocoupler is critical. For the high side in a typical inverter application, this component is subject to voltage changes in excess of 1000V/μsec, with common mode voltages of up to 600V DC. Furthermore, under these conditions it must reliably change state in less than 50-100nsec, without any significant capacitive coupling across the isolation barrier. However, the particular part chosen Very High CMR, Wide Vcc Logic Gate Optocoupler HCPL-2211 is specifically designed for these applications, which guarantees common mode transient immunity of 10kV/μsec at a common mode voltage of up to 1000V, and propagation delays of typically less than 90nsec.

Logic Gates

The output of the optocoupler feeds into a set and reset latch structure which is made up from simple NAND gates, and is arranged to lock out when an over current condition is detected, until the logic gate drive signal is turned off. The latch and resets automatically to allow the next modulation cycle to attempt to turn the IGBT *on* again. This approach has the benefit of not requiring any external over current reset action to be taken, but of course has the disadvantage of hiding the fact of an over current trip from modulation signal. Its main function is to act as an overriding last resort protection for IGBT, rather than to be used for integrated current limiting action. However, the logic could be easily extended to reflect the over current lock-out state back through another optocoupler to the modulation signal.

High Speed MOSFET Driver

From the set-reset latch circuit, the logic drive signal is translated by a TC4429 high speed MOSFET driver. This IC is specifically designed to drive high capacitive power MOSFET gates, and can source or sink up to 6A peak through 2.5Ω output impedance whilst only requiring a 450μA supply current. It can operate at supply voltages of upto 18V, and can

swing high or low within 25nsec or after a propagation delay time of less than 55nsec. The main attraction of this type of driver IC is very fast, high current switched output voltage that it can achieve without external circuity. The disadvantage is its limited supply capability of 18V, which makes it difficult to use when a negative *off* state gate voltage of -15V is desired since this would require a supply voltage of +30V to also achieve a +15V *on* state gate voltage.

IGBT Modules

The output of the gate driver IC directly drives the gate of the IGBT through the gate resistor *Rg*. The recommended value of this resistor from various manufacturers can vary from 10Ω to 220Ω, depending on IGBT size and rating, it has a significant effect on device switching speeds and switching losses. SKM75GB123D IGBT dual module package made by SEMIKRON manufacturer has been chosen. This IGBT module is designed to drive high capacitive power current upto 60A at high voltage of 1200V; it has low inductance drive and low tail current with low temperature dependence. It can operate ±20V signal swing at *Rg* gate resistor of typical 22Ω.

Schottky Diode

In addition, the parallel Schottky Diode *D5* is a significant component in the circuit, since it allows the driver IC to hold the IGBT gate firmly at near zero voltage in the off state, irrespective of the value of the gate resistor *Rg*. This greatly assists in preventing spurious turn *on* during high *dv/dt* switching transition, and it is used to minimise the diode forward bias voltage when the IGBT is turned off.

Comparator

In this drive circuit, overcurrent protection is implemented as collector emitter desaturation voltage detection, using a LM339 comparator. This technique is in contrast to the more conventional technique of measuring the voltage developed across a small resistor in series with the IGBT emitter, and has the benefit of not introducing additional stray inductance into the main power circuit. It does not require a negative control supply voltage and additional amplifying circuits. Desaturation protection is now well accepted as an effective way of achieving overcurrent control.

Significant Components

One difficulty with desaturation protection is its sensitivity to collector emitter switching noise and care should be taken to avoid nuisance trips. Figure 4.4 shows the schematic of the overload detection circuit. For this circuit, resistor *Rd* and capacitor *Cd* minimise any voltage spikes which might cause problems. However, difficulty occurs when IGBT is turned *on*, at which time the device may take over a microsecond to achieve minimum *on*-state voltage after the gate voltage has been applied. Desaturation protection must be disabled during this period to avoid a spurious overcurrent trip, and this is achieved with components *D1*, *Rg* and *Cg*.

When the IGBT is *off*, the output of the gate driver IC is low, and this limits the voltage *Vd* to approximately 0.5V such as determined by diode *D1* and the driver IC output low voltage. When the gate driver output goes high and the IGBT is *on*, *Vd* becomes free to rise, at a rate determined by the time constant *Rd* and *Cd*. However, provided that *Vd* does not rise above the desaturation trip voltage before the IGBT turns *on* and the collector emitter voltage falls to its *on*-state value, the desaturation protection will not react during turn *on*, and hence will not cause an invalid overcurrent trip. This same disable delay also prevents an over-current trip response to the higher current that can occur at turn on because of diode reverse recovery current from the other side switch of the phase leg. In practice, the values of *Rd* and *Cd* given in Figure 4.4 achieves a lockout time of few μsec, which is suitable for most devices.

Schottky diode *D2* is another important component in the circuit, although it does not have a direct role in the operation of the gate drive. Instead its role is to clamp negative voltage spikes which are generated by the discharge of the reverse bias capacitance of diode *D4*, as the main IGBT turn *on* and its drain source voltage collapses. Without this protection, it was found that the negative voltage spiles applied to the positive input of comparator LM339 caused a spurious output response such as an erroneous overcurrent trip, probably because the semiconductor substrate in the comparator is forward biased during the turn-*on* process.

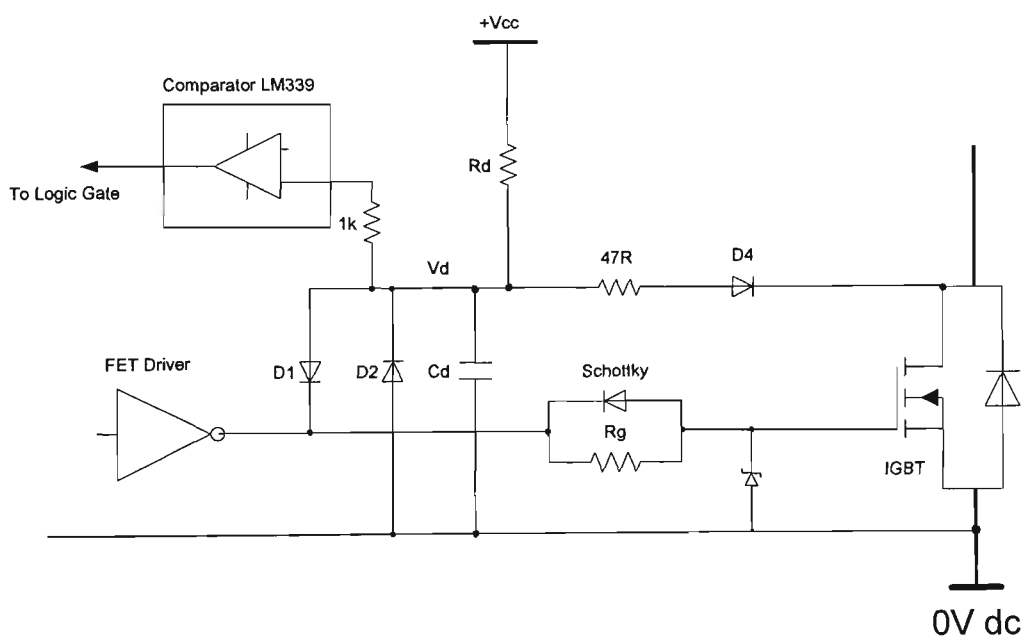


Figure 4.4 Schematic of the overload detection circuit

DC Power Supply

The isolated +15V and +5V supplies for each high and low side drive circuits must be considered. During the IGBT switching, this capacitance causes a significant transient charge and discharge current through the drive circuit ground conductors, which affects the upper switch turn *on* and turn *off* characteristics and can make the operation of the upper switch drive circuit more susceptible to noise and interference. The effective solution is to use

separate DC to DC converter NMF1215 to supply each drive circuit. Because of this converter IC has 1000V DC isolation with single isolated and controllable output and efficiency of 62%.

Feedback Signal

In the feedback circuit, when overcurrent protection is implemented as collector emitter desaturation voltage detection, another comparator monitors the voltage developed across resistor as well. The feedback signal is obtained from the response of overcurrent protection, and transmits through the optocoupler to interface the computer. It can provide double protection and control from external system. Alternatively, the precision temperature sensor LM135 provides the linear output to operate over +100°C temperature range. This device operates two different settings, the temperature rises up to about 50°C, and the exhaust fan is automatically turn-*on*. The 12V supply voltage will switch off if the temperature is more than 70°C.

Crossover Protection Delay

The IGBT switching transistors at the high and low sides of drive circuit at any phase of the inverter must not conduct simultaneously to prevent short circuit DC source. Thus, there must be a dead-time, *Td* which must elapse before high and low sides IGBT transistors can change state. The size of the dead-time is determined by the turn-*off* times of the switching devices used. Typically, this is in order of a few microseconds. The experimental circuit includes a module which accepts a TTL level signal and produces two switching signals for an inverter with a variable dead-time in microseconds at the transitions. The timing diagram of Figure 4.5 describes the operation of this circuit.

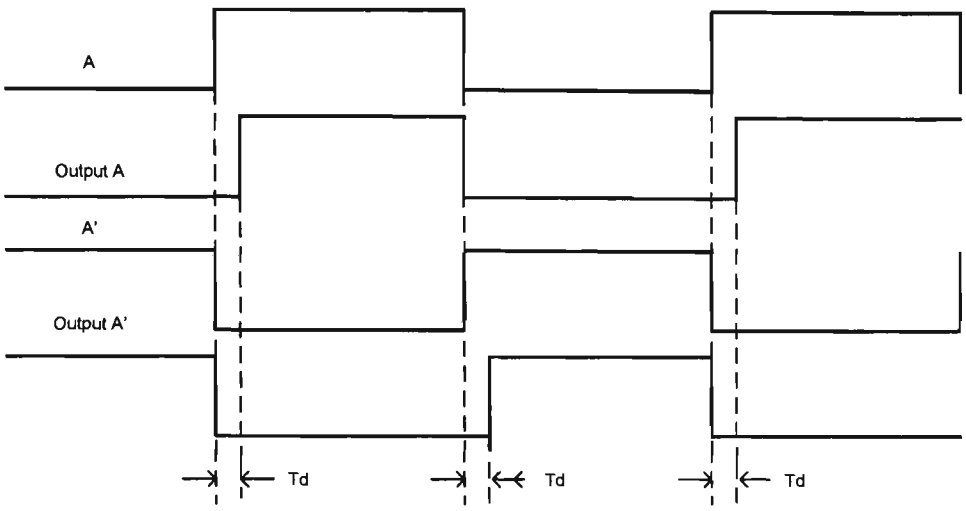


Figure 4.5 The waveforms of driving signal and high & low side phase output voltage

The hardware implementation of the crossover delay *Td* of about 4 μsecond is performed by the TTL Schmitt trigger logic circuit to ensure no overshoot through in any phase of the power converter. The circuit diagram is shown in Appendix B.

4.2.7 Conclusion

This comprehensive IBGT power drive circuit has been developed, which is capable of high speed switching transitions with fast acting overcurrent protection. This IGBT drive circuit has been built as a three phase converter, which has been used to test the response of microprocessor based circuit breaker under the difficult conditions such as variable frequency and amplitude.

The detail of circuit diagram of the power converter and the completed hardware configuration are shown in Appendix B.

The overview of 50kVA IGBT power converter and implementation in the testing is shown in Figure 4.6.

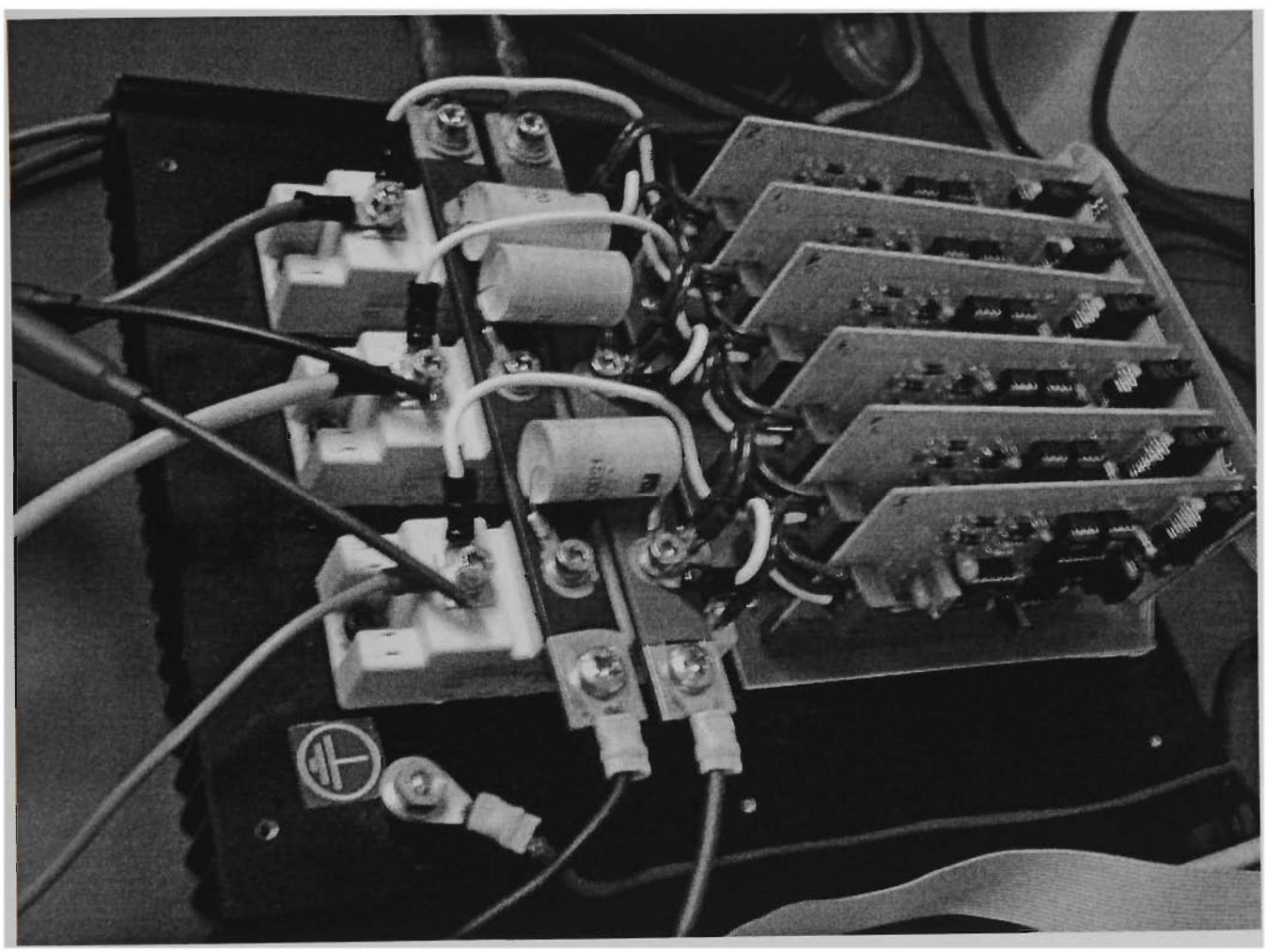


Figure 4.6 The view of 50kVA IGBT power converter

Isolation AC and DC current transducer

The current transducers are designed and developed is this project. The current transducers, LA125-P from Farnell was chosen to measure AC and DC current. The current transducers are isolation and Hall Effect type, which allows for accurate measurement of instantaneous values of AC and DC currents upto 125A. The output of current transducer is accurately and linearly related to the primary current flowing through the current core. Additionally, the operation amplifier provides the adjustable of output current signal for compensating the error of instrument. The circuit schematic diagram of three phase current transducer is shown in Appendix B.

The view of hardware current transducer configuration attached the microprocessor based circuit breaker is shown in Figure 4.7.

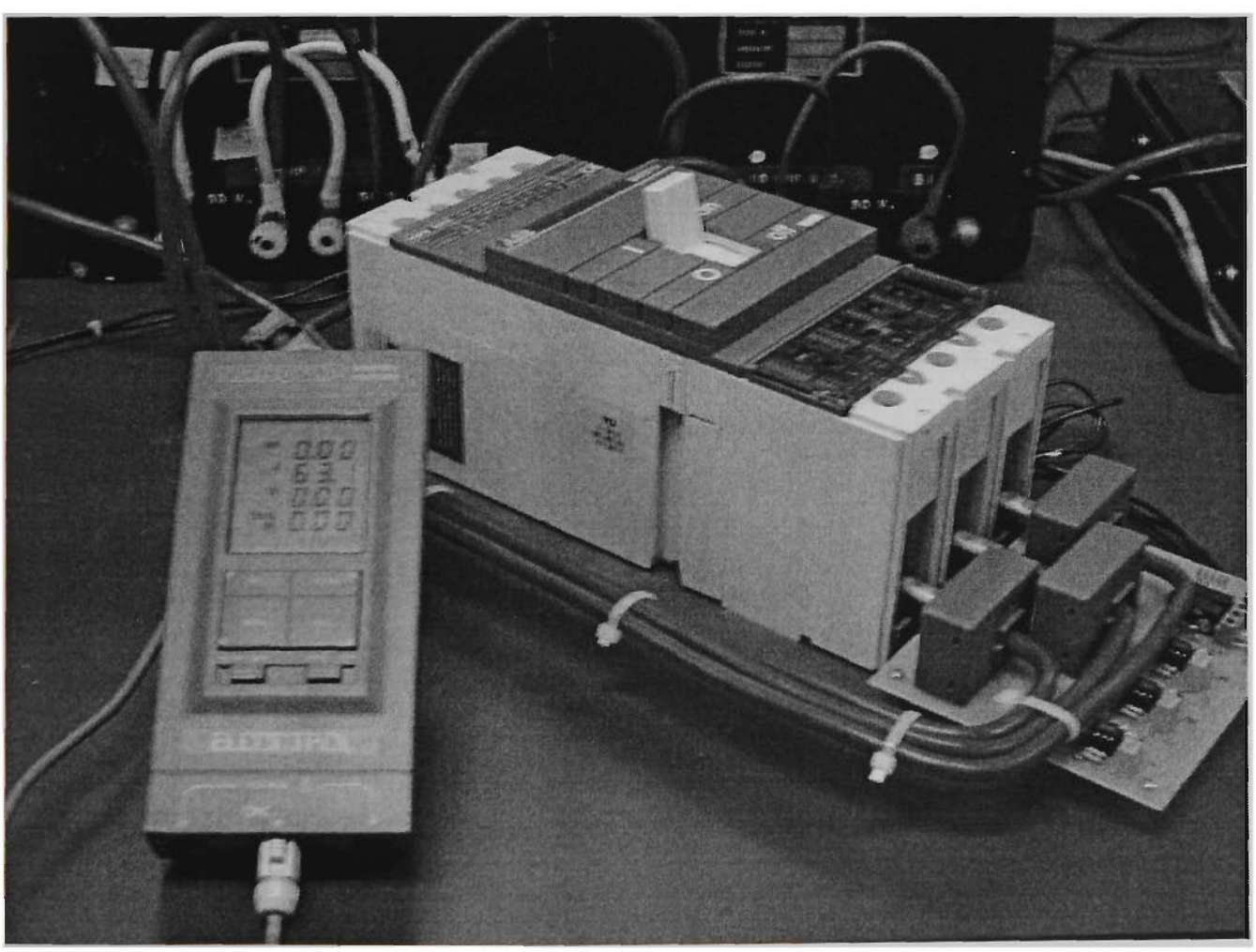


Figure 4.7 View of the current transducer and microprocessor based circuit breaker

4.3 Software development of generating PWM signal

4.3.1 Introduction

The software development is composed of four design tools including: MATLAB, SIMULINK, Real Time Workshop (RTW) and Real Time Windows Target (RTWT) that are structured in a hierarchical manner as shown in Figure 4.8. Each of these software components can be executed on standard computer PC hardware running on the Windows operating system. Figure 4.8 illustrates the hierarchical structure of the computer system environment along with its interface to external physical system. Brief description of each component is given as follows:

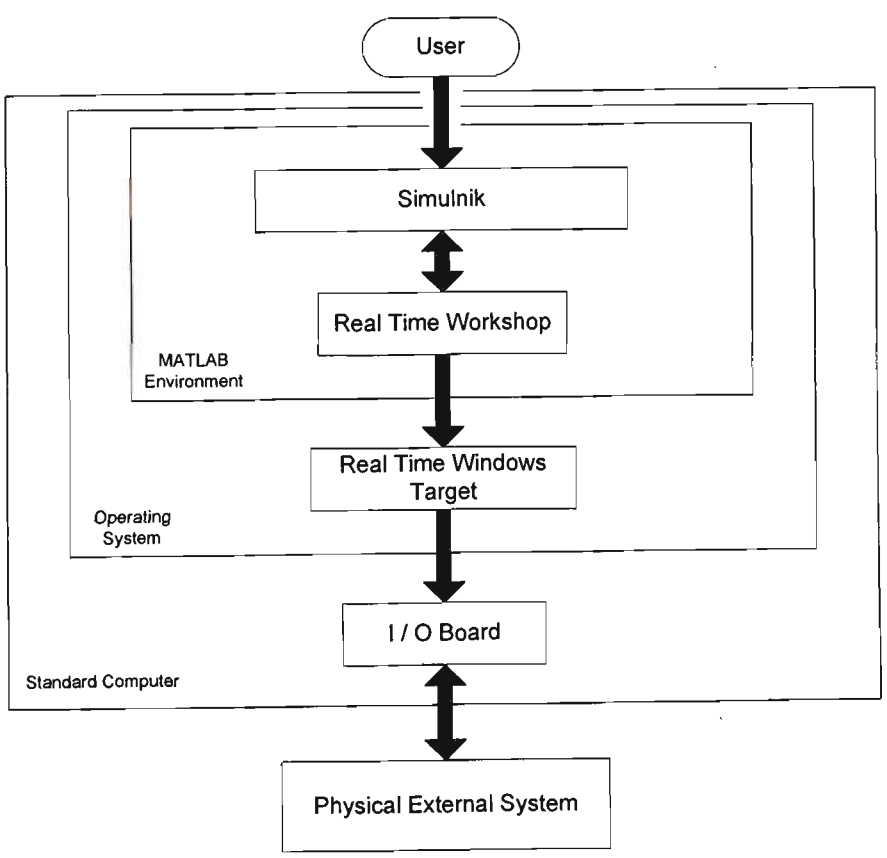


Figure 4.8 The hierarchical structure of the computer system environment

MATLAB is a software environment [39] that allows a user to easily integrate computation and visualization tasks. The main advantage of MATLAB is that problems and solutions are expressed in familiar mathematical notations. Due to the fact that numerous toolboxes and other software packages have been developed for MATLAB, it has become the tool of choice for computation, algorithm development, modelling, simulation, data analysis, visualization, engineering graphics and application development (including graphical user interface GUI development).

SIMULINK is a software package for modelling, simulating and analysing dynamic systems in the MATLAB environment. SIMULINK supports both linear and nonlinear systems that are modelled in continuous time and discrete time. The SIMULINK GUI is used to create block diagram models. The Power System Blockset library provides pre-configured blocks and connectors that can be incorporated into a model by simple drag and drop operations. Different type of sources in this library allows the user to apply different inputs. After the model is defined, the user can simulate the response of the system by selecting the appropriate time integration method. SIMULINK also allows for on-line parameter tuning in order to assess the change in system response. Scopes and other display blocks allow the user to view the simulation results while the simulation is still running.

Real Time Workshop (RTW) is an automatic C language code generator for SIMULINK, which runs within the MATLAB environment. RTW generates C-code directly from the SIMULINK models and automatically constructs a file that can be executed in real time environment. In conjunction with RTW, SIMULINK provides a powerful front-end for developing executable code without requiring a large amount of computer skills. The block diagram interface of Simulink coupled to the RTW code generator allows the user to concentrate on modelling and control issues as opposed to programming issues.

Real Time Windows Target (RTWT) is a window based software package that merges the power of SIMULINK Block diagrams and the C-code conversion ability of RTW into one package that is able to implement a control algorithm. It has the ability to run SIMULINK models under Windows 98 in real time on standard PC while interfacing to real hardware

using PC I/O boards [40,41]. It allows the user to collect the run time data in a MAT-file format for later analysis and visualization in MATLAB. During operation, other Windows applications continue to run and can use all CPU cycles not needed by the real time task.

4.3.2 Description of generating pulse width modulation (PWM) signal

A PWM signal is created by comparing a triangle wave with a sinusoidal wave (sinusoidal PWM type). Since SIMULINK does not provide a triangular signal generator, it has to be created. The modulating frequency, frequency modulation ratio (m_f) and the amplitude modulation ratio (m_a) can be changed by entered values. Moreover, an unlimited number of switching and control schemes can be developed in Simulink. Figure 4.9 shows an overall PWM generating signal diagram of the SIMULINK simulation.

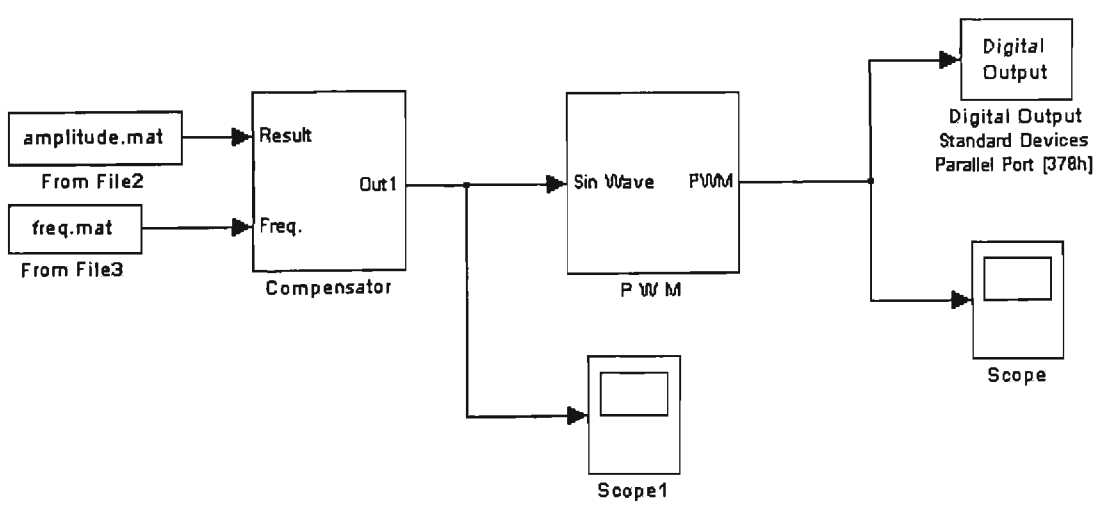


Figure 4.9 An overall three phase PWM generating block diagram

Figure 4.10 shows the sub-system of three phase sinusoidal signal as compared with the triangular signal and converts to logical output signal. The compensator block provides the external device compensation, such as the step down transformer saturated characteristic. Figure 4.11 shows the sub-system of compensator block.

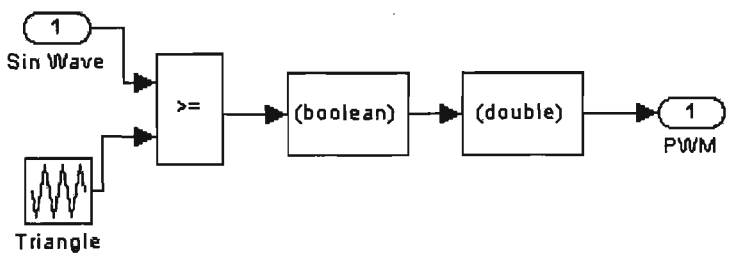


Figure 4.10 The sub-system of PWM block

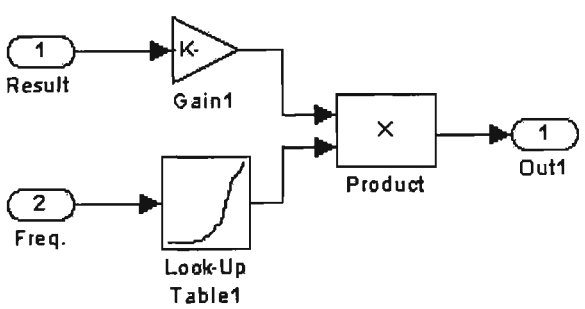


Figure 4.11 The sub-system of compensator block.

The two input MAT files are created from the generating power system simulation results. Two sets of MAT files are compared with the results obtained from the fault analytical simulation in different frequency and fault current. This combined signal presents with compensation response of a set of testing equipment.

Real Time Workshop (RTW) generates source code as defined by SIMULINK model. A parallel port device driver block provided by Real Time Windows Target (RTWT) must be included in SIMULINK model in order to run with a specific model. The following steps in RTW are needed:

- Create main program for the control board to execute the generated code.
- Create a system target file. This allows the Target Language Complier (TLC) to transform the model into generated code.
- Create a template makefile to build the real time executable code.
- Create device drivers to communicate between a real time program and an I/O device.

A main program and the generated model code are complied and linked together with the device drivers by a template makefile to build an executable code. This code is downloaded to the external device.

4.3.3 Performance of sampling Time

Although most of the signals have an unlimited frequency spectrum, the carrier frequency signal needs to be noticed. These are often found in range upto 2kHz in the normal power system with 50Hz. Consequently, sampling period should be less than 50e-6 seconds to ensure that the sampling process does not disturb the desired precision. This means there must be a compromise between the complexity of the modelled component and the maximum analysed frequency [42]. In the experimental testing, the sampling frequency 10 kHz applies in this real time simulation.

4.3.4 Conclusion

A new approach to implement computer based real time simulation in MATLAB environment is described. The testing of actual microprocessor based circuit breaker with variable frequency and amplitude has been processed. Digital system and external physical system is connected by the parallel I/O port. This testing is a valuable attempt in the future design of the power system digital simulators. It has also set up a foundation of advanced analysis on real devices in various kinds of dynamic condition.

The overview of the experimental power system and testing the microprocessor based circuit breaker is shown in Figure 4.12.

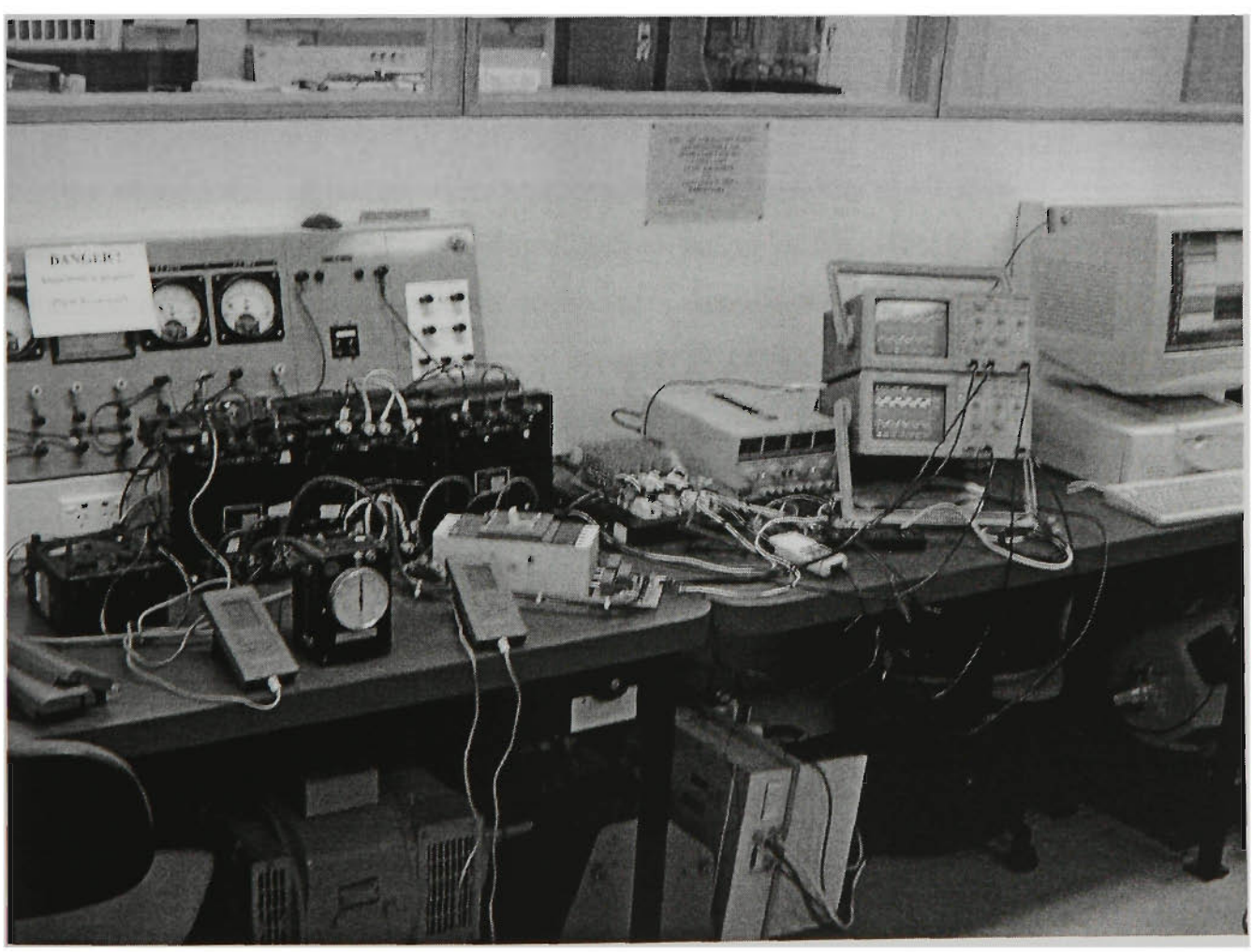


Figure 4.12 The overview of the experimental power system

Chapter 5 SIMULATION AND EXPERIMENT RESULTS

5.1 Introduction

When power systems are subjected to different abnormal operations such as short circuits or opening, it is very important to isolate the faulty sections of the system using the protection system. In order to be able to design microprocessor based protection systems, it is extremely vital to be able to carry out extensive research and development on wave power generating system under various fault condition. Accurate MATLAB simulation for analysing faulted transmission system and the precise nature of current and voltage waveforms, particularly during the first cycle is very important.

High-speed protection possesses significant travelling wave components in both faulted and healthy transmission. The microprocessor based protection has to be able to detect signals during a very short period of time following fault inception. Transient waveforms under fault condition depend on the nature of the earth return, conductor configuration, fault location and point on wave of fault initiation. An accurate representation of the protection system including the frequency variance provides realistic results, which can be used for developing and testing different algorithms for digital relaying and measurements purpose.

This chapter uses MATLAB program for simulating faulted power system to provide the appropriate transient waveforms for subsequent use for protection and measurement purpose. This is done in order to provide accurate information of the transient waveforms for further processing the protection system.

This chapter also contains the simulation results of the wave power generating system. The simulation results include the steady state and transient analysis. The simulation waveforms of transient voltages and currents under fault conditions, transients caused by switching of the generator, single line to ground fault, double line to ground fault, three line fault, the step up

and down transformers fault, etc. are plotted at various locations of interest on the power system. In all cases, the period of study under fault condition is dependant on the speed of the generator between 0.3 to 0.5 seconds. Each case of fault is illustrated by a figure drawn below the complete circuit diagram. This is followed by description of the fault and other interesting waveforms and comments. Most of waveforms contain changes in voltages or current in all three phases.

5.2 The simulation results of transient fault analysis

5.2.1 Fault located at the circuit breaker before the step up transformer of power system

This section describes short circuit in single phase, two phase and three phase to ground fault occurring at location X. The circuit breaker remains closed with the permanent magnet generator driving various speed and torque, as shown in Figure 5.1.

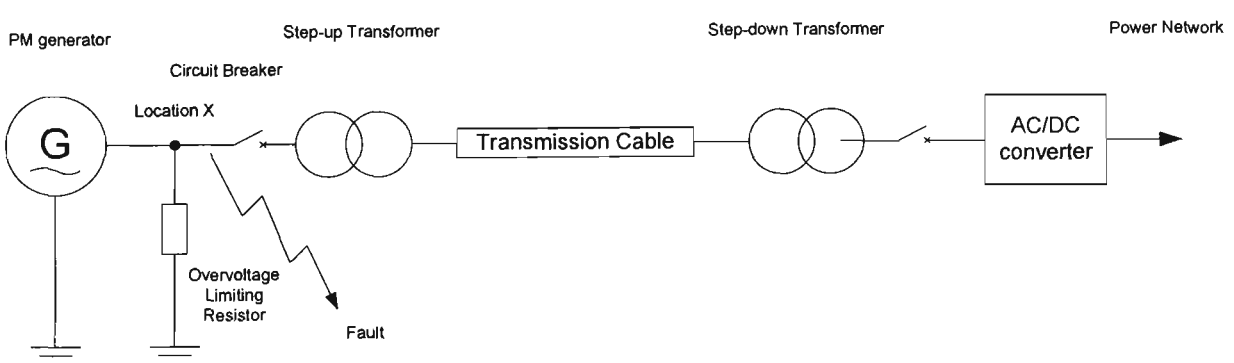


Figure 5.1 Illustration of the fault location X in the power system

The short circuit occurs after 0.4 second of normal steady state operated. The three phase generator output voltage and current waveforms are shown in Figures 5.2 to 5.18. The different faults and comments are described in Table 5.1.

Table 5.1 Table of results for simulation fault at location X.

Frequency	Phase A to earth fault	Phase A-B to earth fault	Three phase to earth fault
140Hz	Figures 5.2 and 5.3 Voltage: A phase – collapses to zero. B & C phases – maintain sinusoid but amplitudes increase due to the mutual coupling. Current: All phase still remains sinusoidal	Figures 5.4 and 5.5 Voltage: A-B phases – collapse to zero. C phase – maintains sinusoid and rises exponentially. Current: The envelope of fault currents occur in phase A and B. Thus the current through phase A and B are twice the normal current, which is due to earth fault.	Figures 5.6 and 5.7 Voltage: Three phases – collapse to zero. Current: The envelope of fault currents occur in three phases, so the fault current are several times more than the normal current.
70Hz	Figures 5.8 and 5.9 Voltage: A phase – collapses to zero. B & C phases - maintain sinusoid, the voltage rises similar to the situation of single phase faulted. Current: All phase still remains sinusoidal	Figures 5.10 and 5.11 Voltage: A-B phases – collapse to zero. C phase – maintains sinusoid oscillating waveform. Current: C phase maintains sinusoidal The envelope of fault currents in phase A and B are twice the normal current.	Figures 5.12 and 5.13 Voltage: Three phases – collapse to zero. Current: The envelope of fault currents occur in three phases, so the fault current are several times more than the normal current.
9Hz	Figures 5.14 and 5.15 Voltage: A phase – collapses to zero. B & C phases – collapse to zero rapidly, because the generator is running at starting speed and is unstable. Current: The three phase current also appears to have similar characteristics.	Figures 5.16 and 5.17 Voltage: A & B phase – collapses to zero. C phases – collapse to zero rapidly, because the generator is running at starting speed and is unstable. Current: The three phase current also appears to have similar characteristics.	Figures 5.18 and 5.19 Voltage: Three phase – collapses to zero. Current: The three phase current also appears to have similar characteristics.

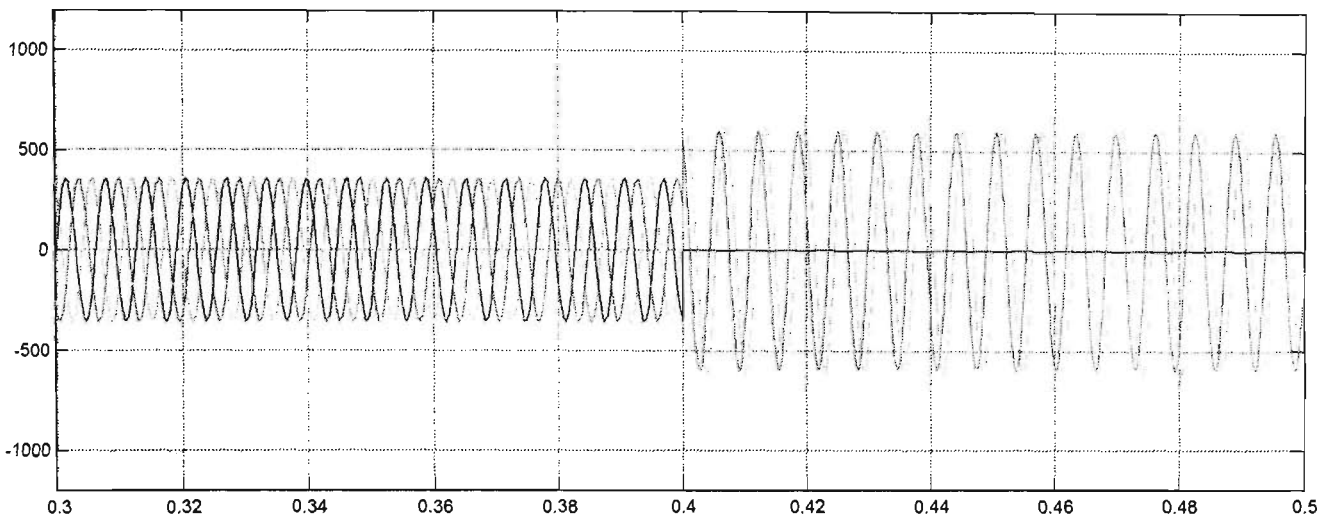


Figure 5.2 Phase A to ground fault on the location X (transient voltage) at 140Hz

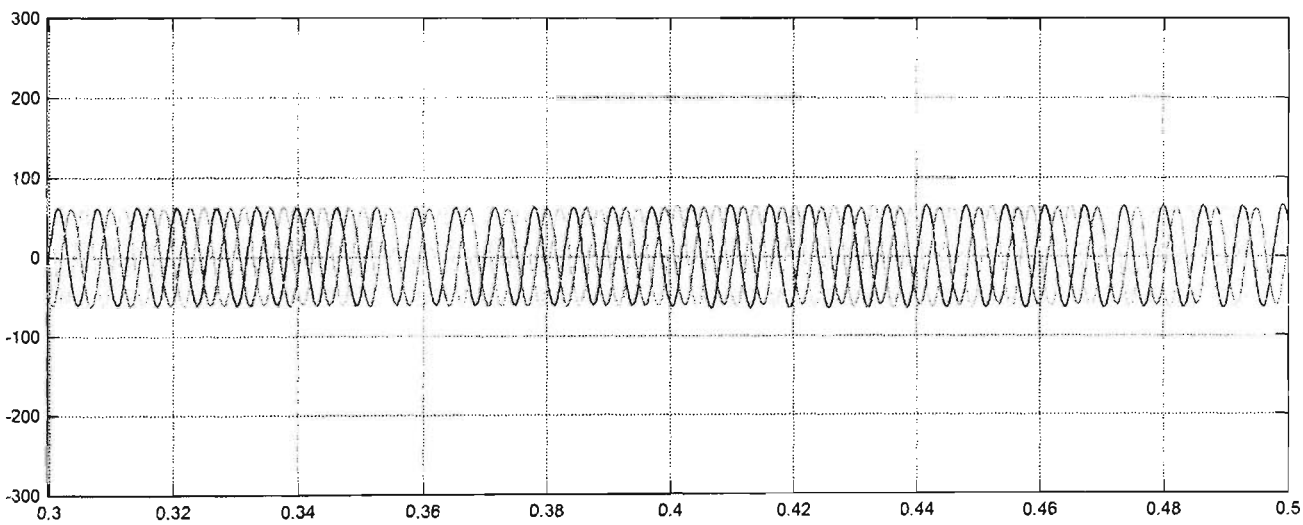


Figure 5.3 Phase A to ground fault on the location X (transient current) at 140Hz

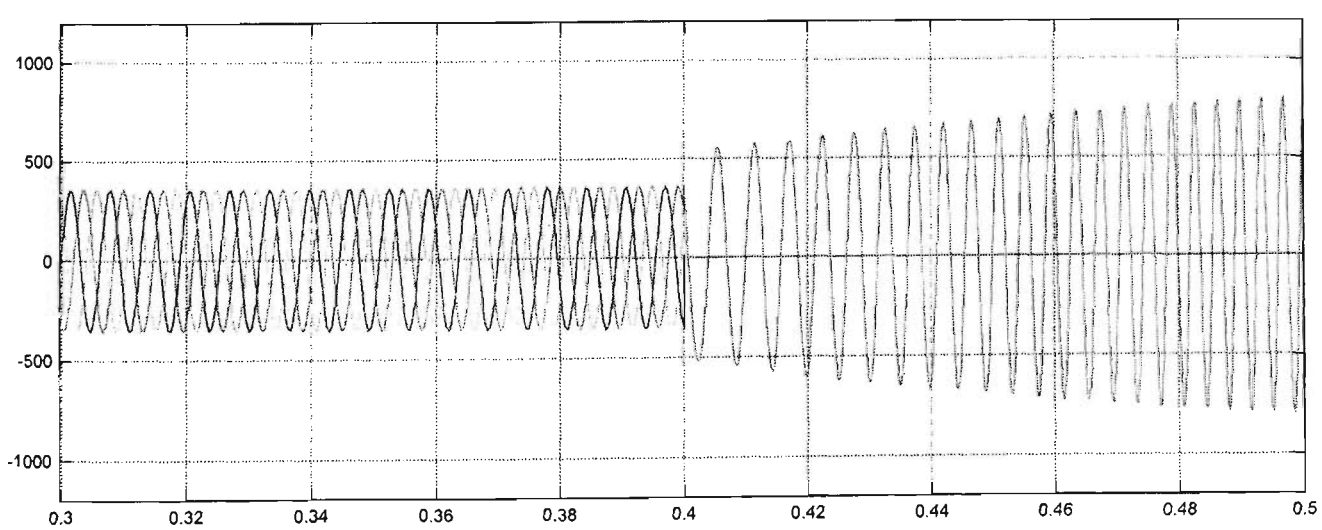


Figure 5.4 Phase A and B to ground fault on the location X (transient voltage) at 140Hz

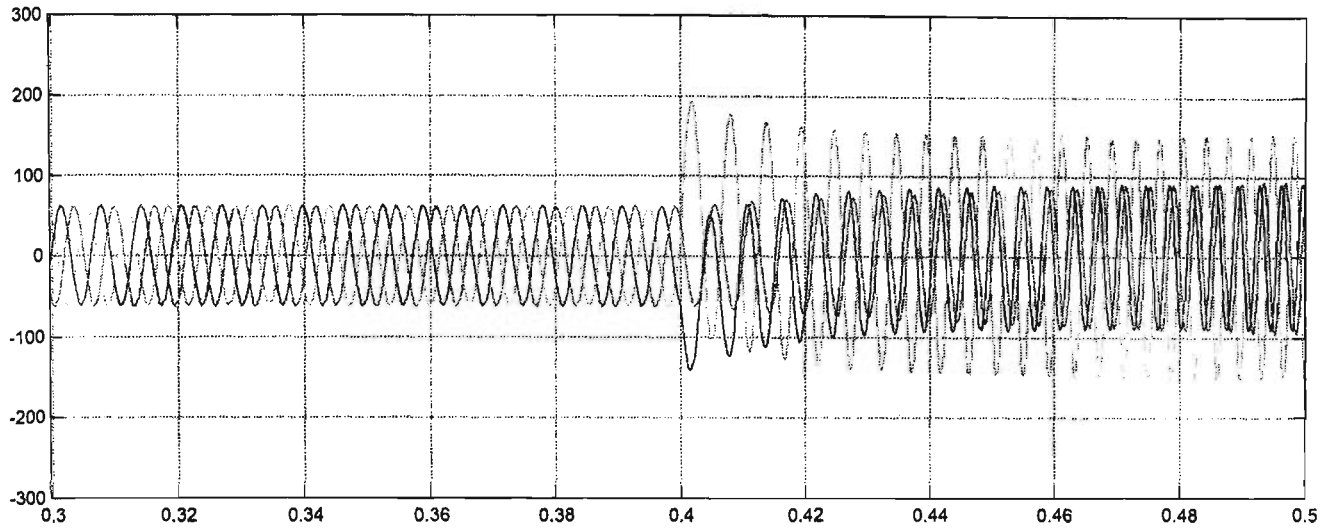


Figure 5.5 Phase A and B to ground fault on the location X (transient current) at 140Hz

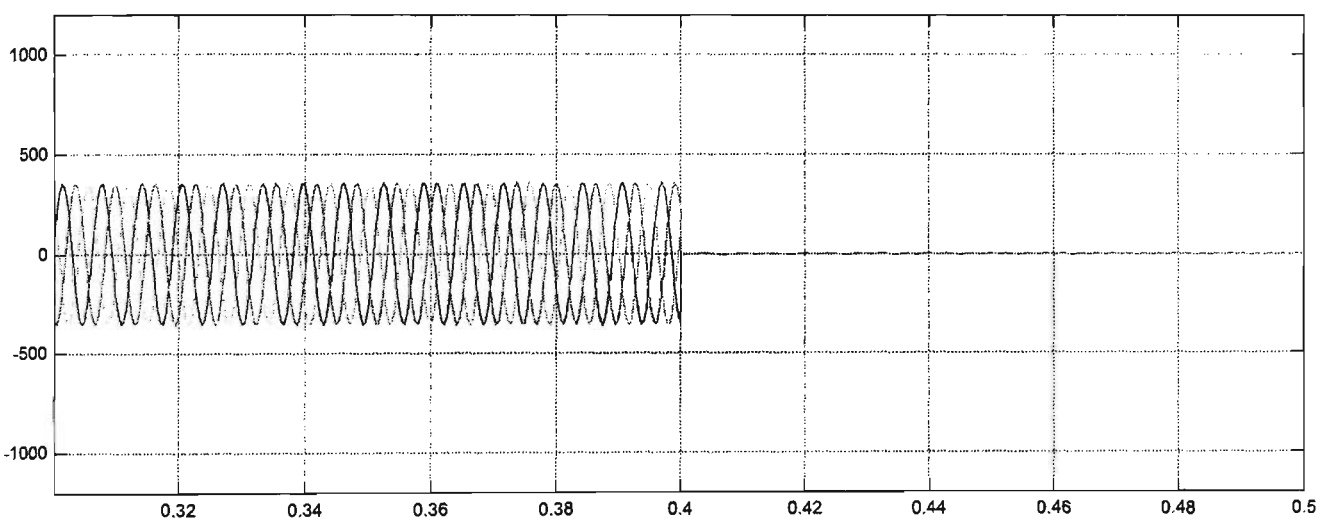


Figure 5.6 Three phase to ground fault on the location X (transient voltage) at 140Hz

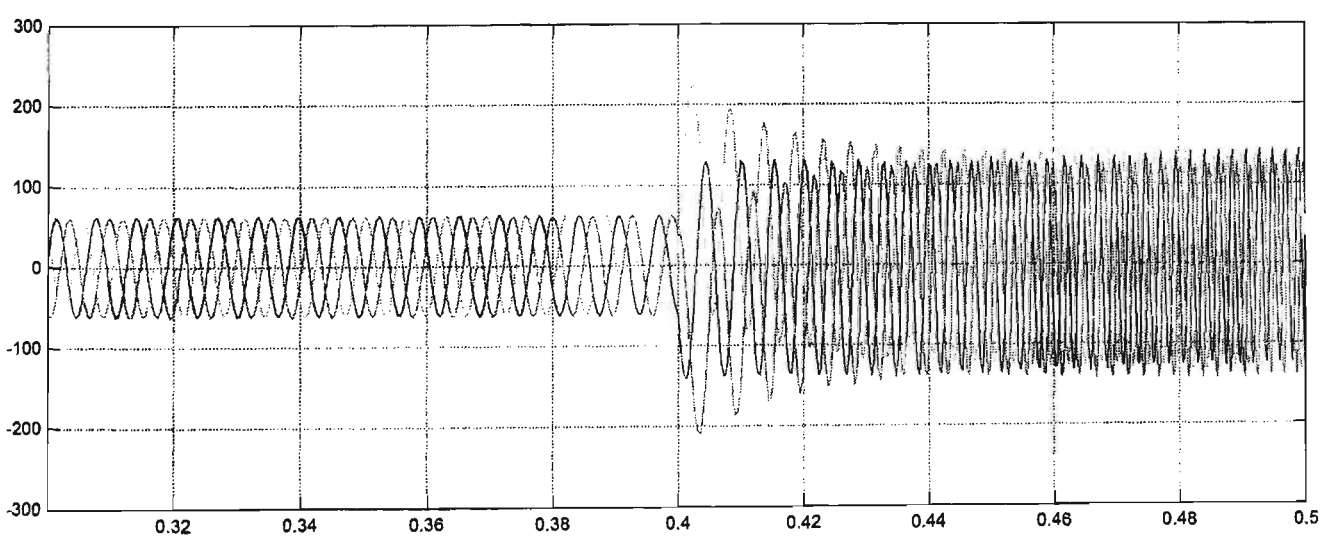


Figure 5.7 Three phase to ground fault on the location X (transient current) at 140Hz

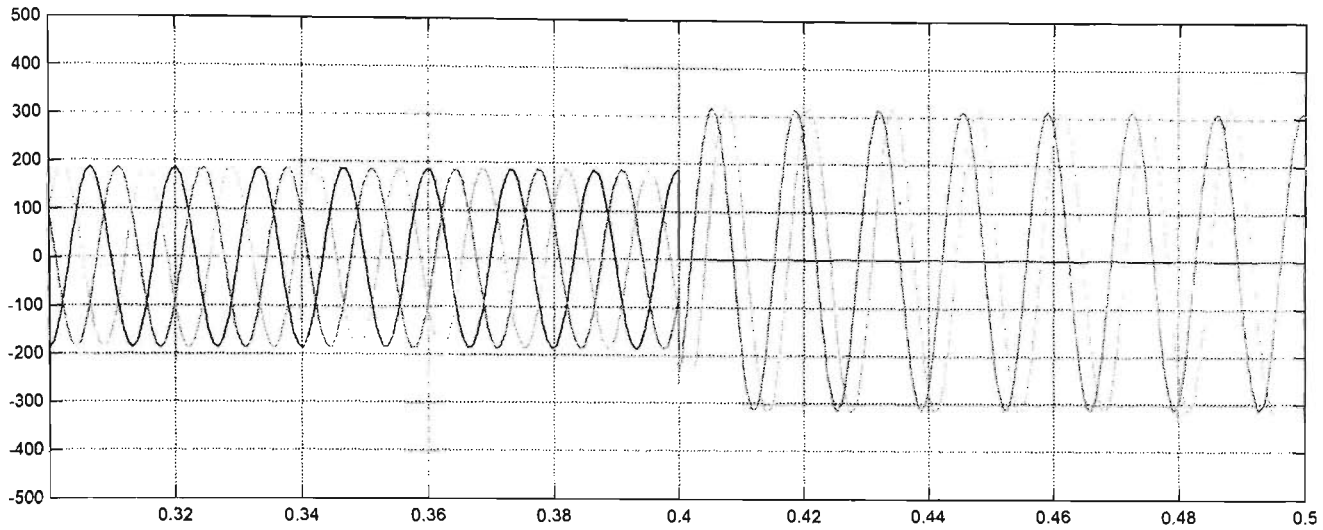


Figure 5.8 Phase A to ground fault on the location X (transient voltage) at 70Hz

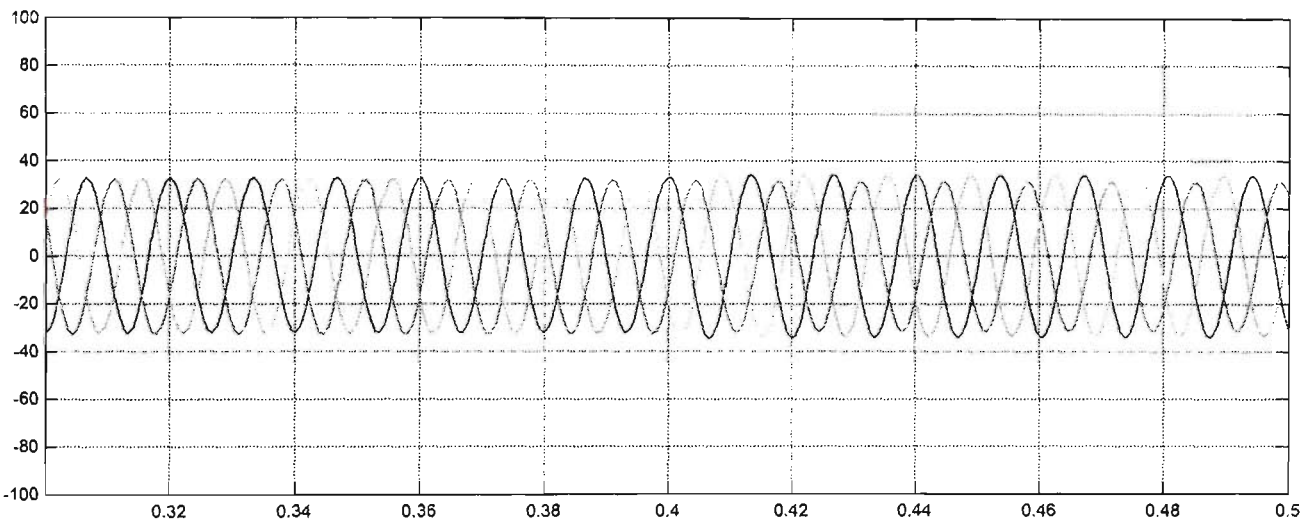


Figure 5.9 Phase A to ground fault on the location X (transient current) at 70Hz

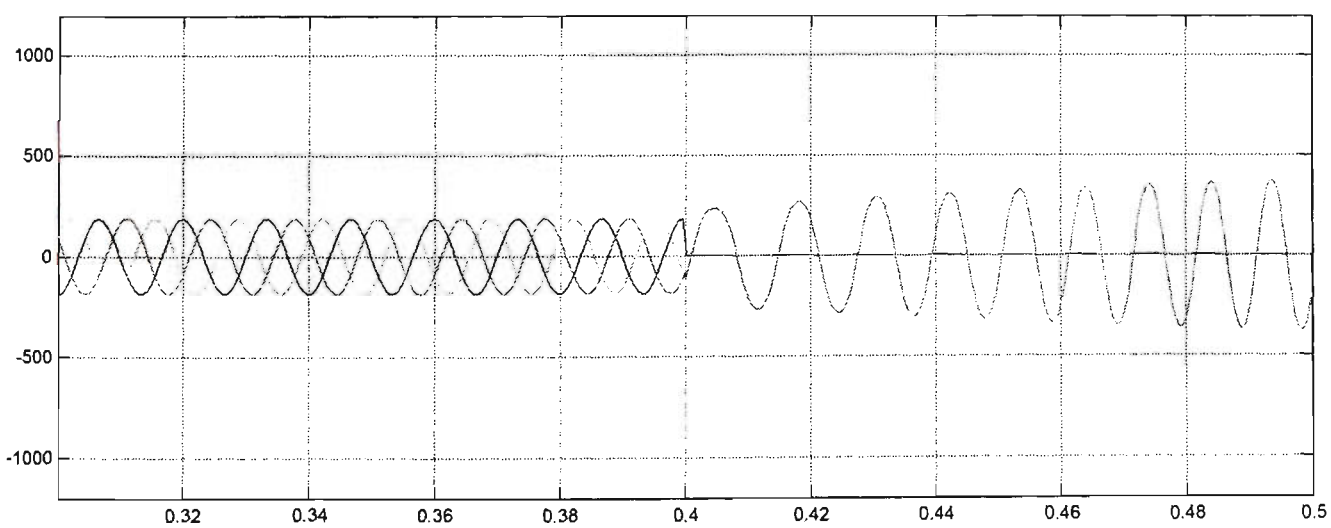


Figure 5.10 Phase A and B to ground fault on the location X (transient voltage) at 70Hz

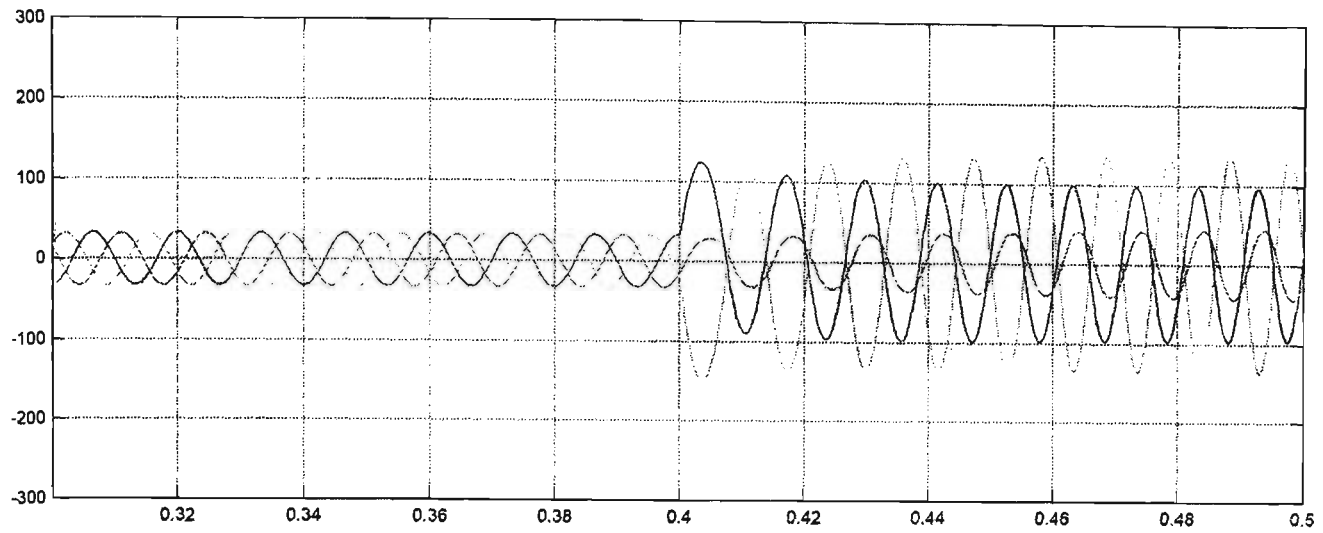


Figure 5.11 Phase A and B to ground fault on the location X (transient Current) at 70Hz

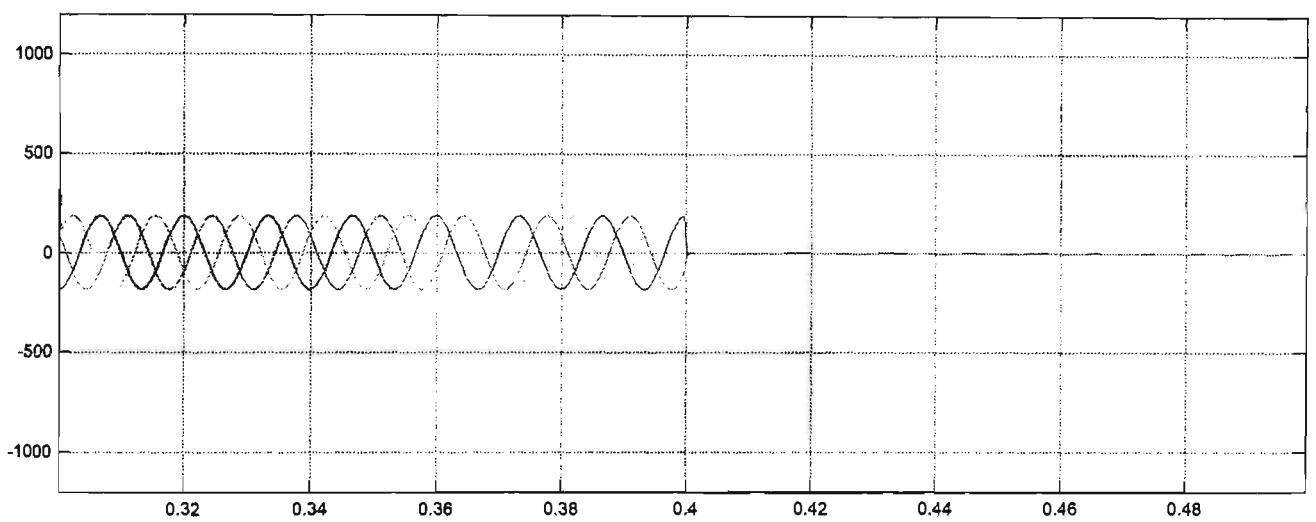


Figure 5.12 Three phase to ground fault on the location X (transient voltage) at 70Hz

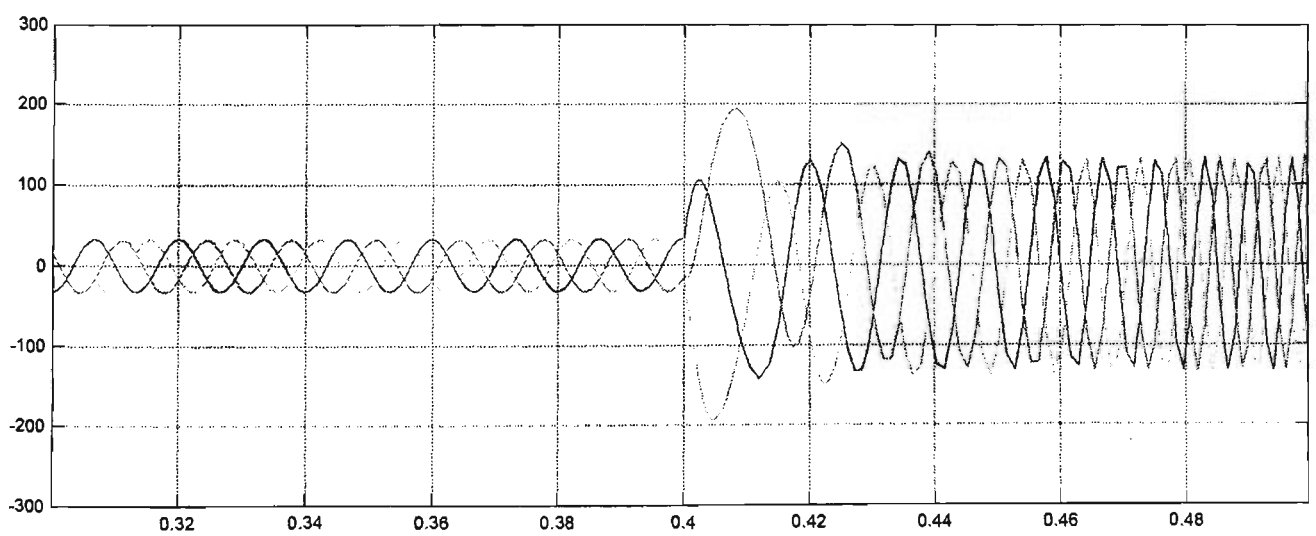


Figure 5.13 Three phase to ground fault on the location X (transient current) at 70Hz

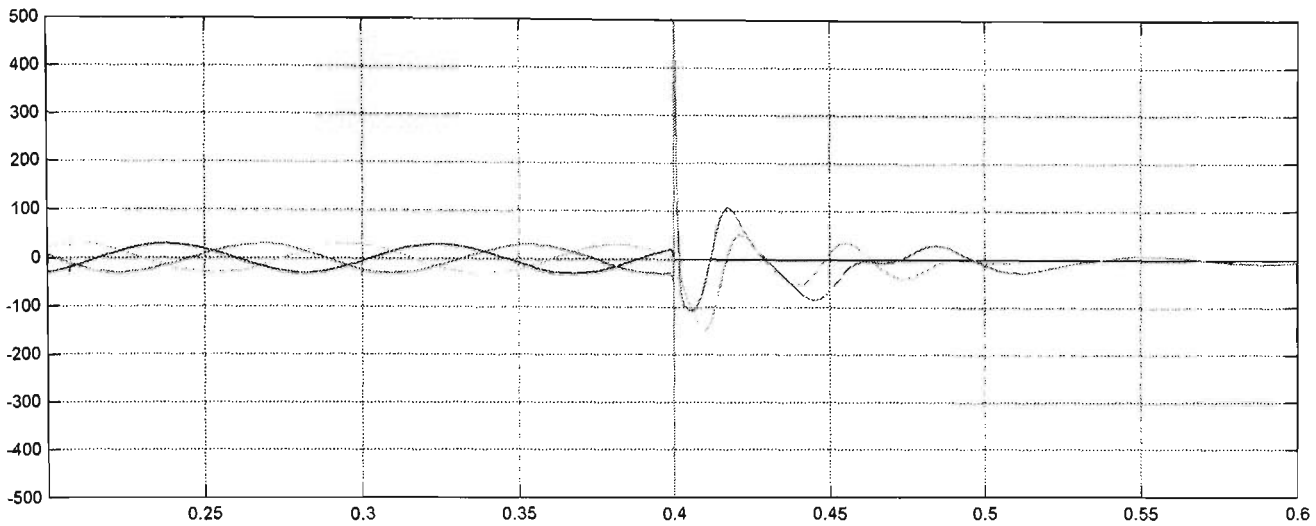


Figure 5.14 Phase A to ground fault on the location X (transient voltage) at 9Hz

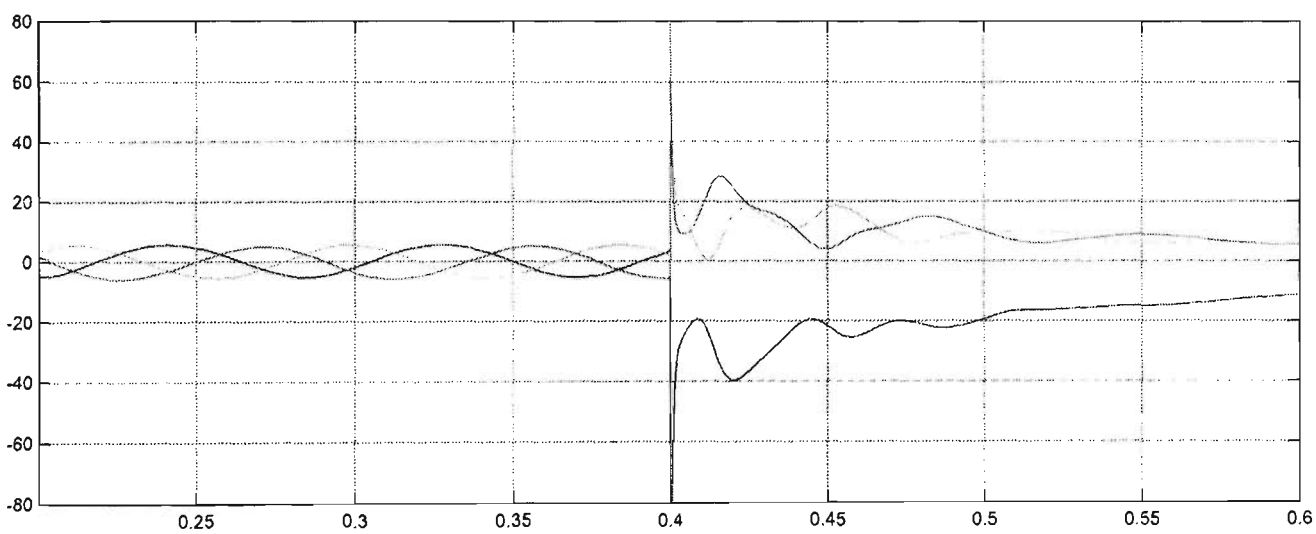


Figure 5.15 Phase A to ground fault on the location X (transient current) at 9Hz

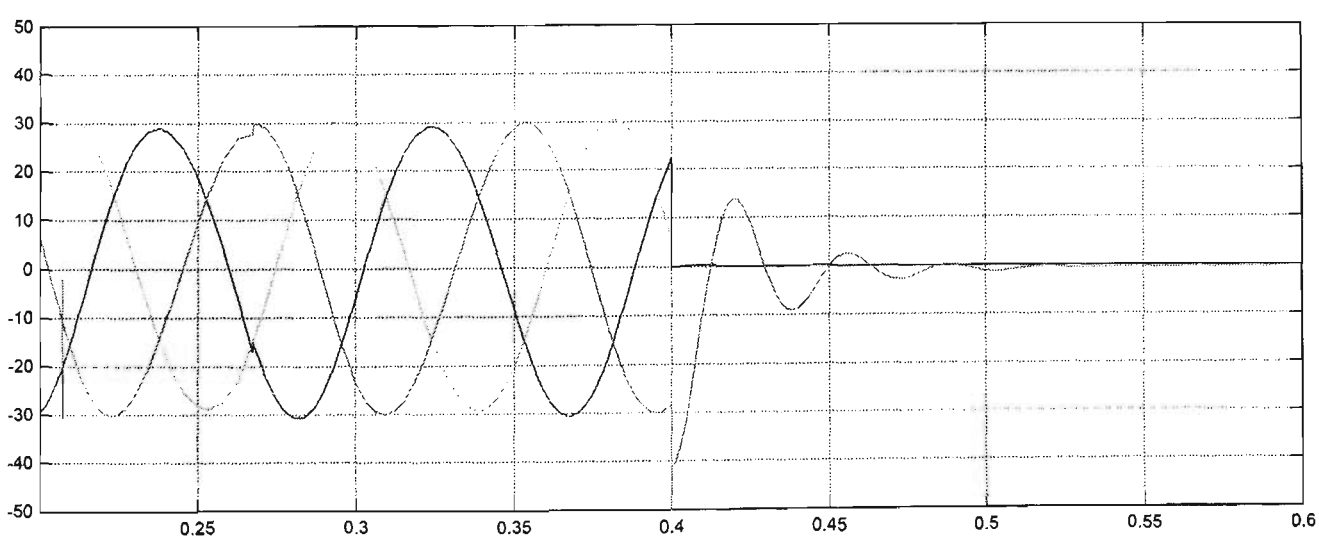


Figure 5.16 Phase A and B to ground fault on the location X (transient voltage) at 9Hz

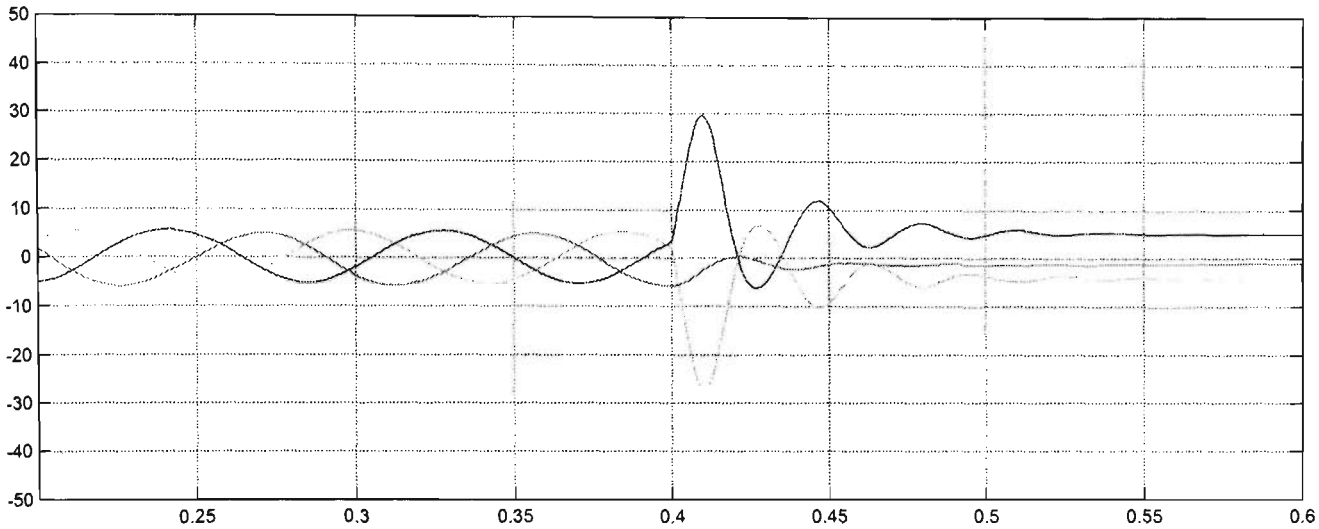


Figure 5.17 Phase A and B to ground fault on the location X (transient current) at 9Hz

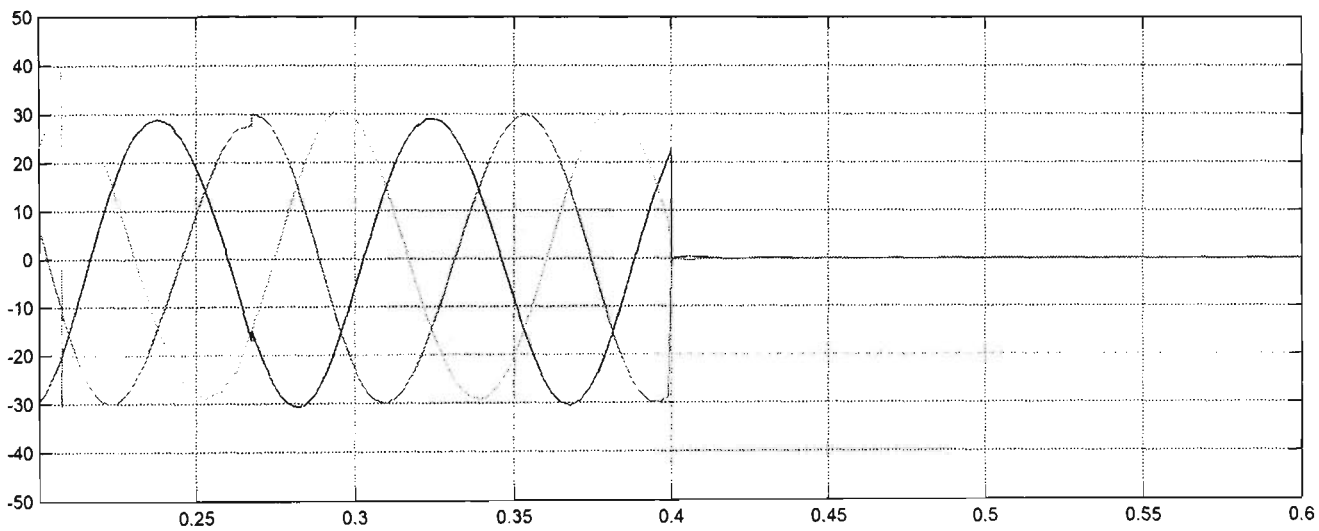


Figure 5.18 Three phase to ground fault on the location X (transient voltage) at 9Hz

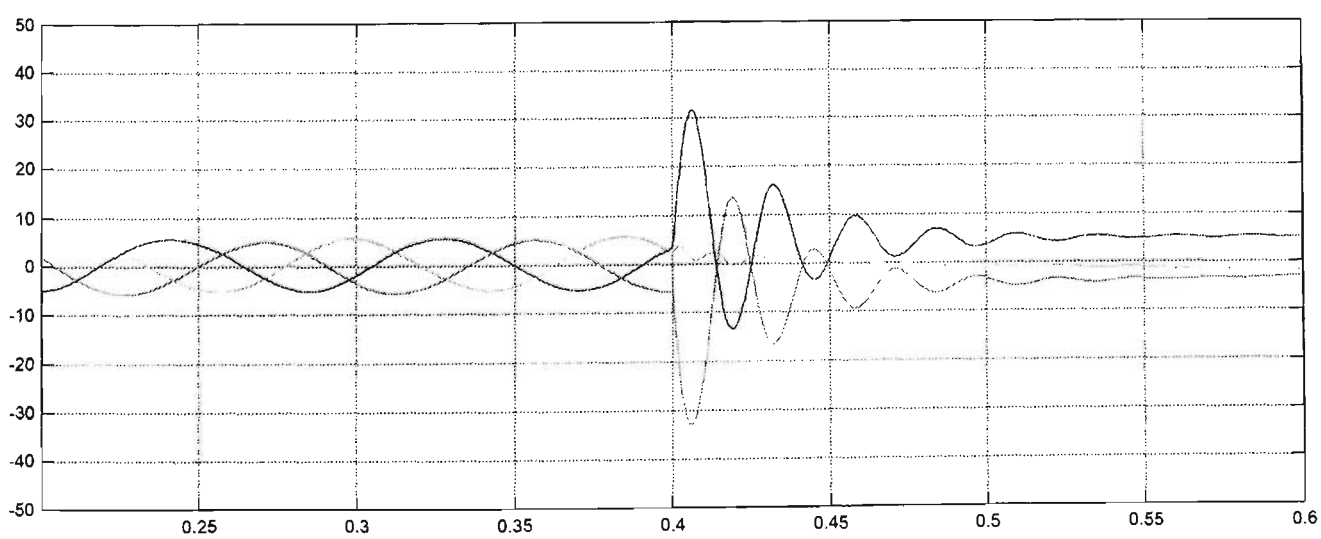


Figure 5.19 Three phase to ground fault on the location X (transient current) at 9Hz

5.2.2 Fault located at the end of the step down transformer of power system

In this section, single phase, two phases and three phases to ground fault occurrs at location Y with short circuit at the fault path. The generator produces power through the transmission system to the offshore station. Location Y is beyond the step down transformer and before the next circuit breaker. The circuit breaker remains closed with the permanent magnetic generator driving various speed and torque, as shows in Figure 5.20.

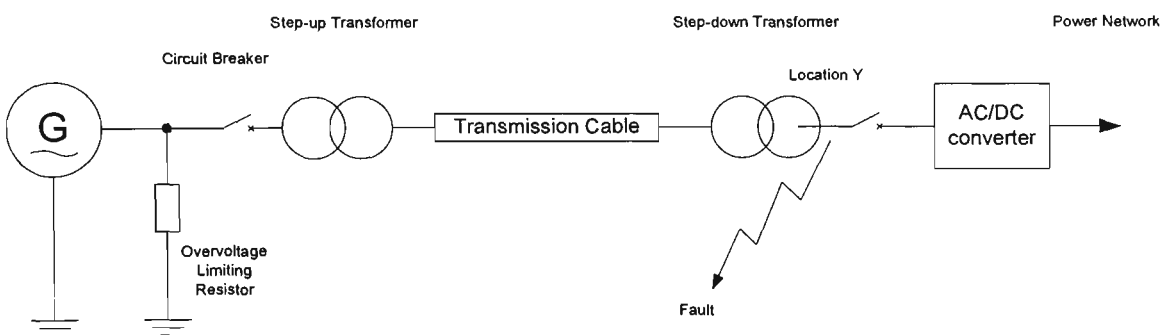


Figure 5.20 Illustration of the fault location Y in the power system

A short circuit in different phase to ground fault occurs at location Y of the power system after 0.4 second of normal steady state operated. The three phase generator output voltage and current waveforms are shown in Figures 5.21 to 5.38. The different faults and comments are described in Table 5.2.

Table 5.2 Table of results for simulation fault at location Y.

Frequency	Phase A to earth fault	Phase A-B to earth fault	Three phase to earth fault
140Hz	Figures 5.21 and 5.22 Voltage: A phase – collapses nearly to zero and small voltage exists after the fault occurrence. B & C phases – maintain sinusoidal but amplitudes increase due to the mutual coupling.	Figures 5.23 and 5.24 Voltage: A-B phases – collapses and the faulted phases again maintain voltage due to the fault impedance increases. C phase – maintains sinusoid and amplitudes increase.	Figures 5.25 and 5.26 Voltage: Three phase voltage decreases due to ground fault. Post fault voltage on the three phase high rather than collapsing to zero.

	Current: All phase still remains sinusoidal in nature.	Current: The envelope of fault currents occur in phases A and B. Thus the current through phases A and B are more than normal current, which is due to the earth fault. Phase A still remains sinusoidal	Current: The envelope of fault currents occur in three phases, so the fault current are two times more than the normal.
70Hz	Figures 5.27 and 5.28 Voltage: A phase – collapses to nearly zero but small voltage remains after the fault occurrence. B & C phases - maintains sinusoidal nature however the voltages are high due to the mutual coupling. Current: All phase still remains sinusoidal in nature.	Figures 5.29 and 5.30 Voltage: A-B phases – collapses but voltage remains at high level of voltage. The frequency of system begins to decrease and phases C is affected too. Current: The envelope of fault currents in phases A and B are twice the normal current. Again the frequency of system is changed.	Figures 5.31 and 5.32 Voltage: Three phases – collapses but maintains half of normal voltage after the fault occurrence. As to the fault impedance is high and the frequency of system commences to reduce. Current: The envelope of fault currents occur in three phases, so the fault current are double of normal current, also the frequency of system changes.
9Hz	Figures 5.33 and 5.34 Voltage: A phase – collapses but remains half of normal voltage. B & C phases - maintains sinusoid however the voltages are high due to the mutual coupling. Current: B & C phases current still remains sinusoidal. A-phase slightly increases in amplitude.	Figures 5.35 and 5.36 Voltage: Three phase is unstable and the waveforms collapse due to the generator running at starting speed. Current: The three phase current also appears to have similar characteristics.	Figures 5.37 and 5.38 Voltage: Three phase – collapses to near zero and the frequency of system changes very low value. Current: The three phase currents also appears to have similar characteristics.

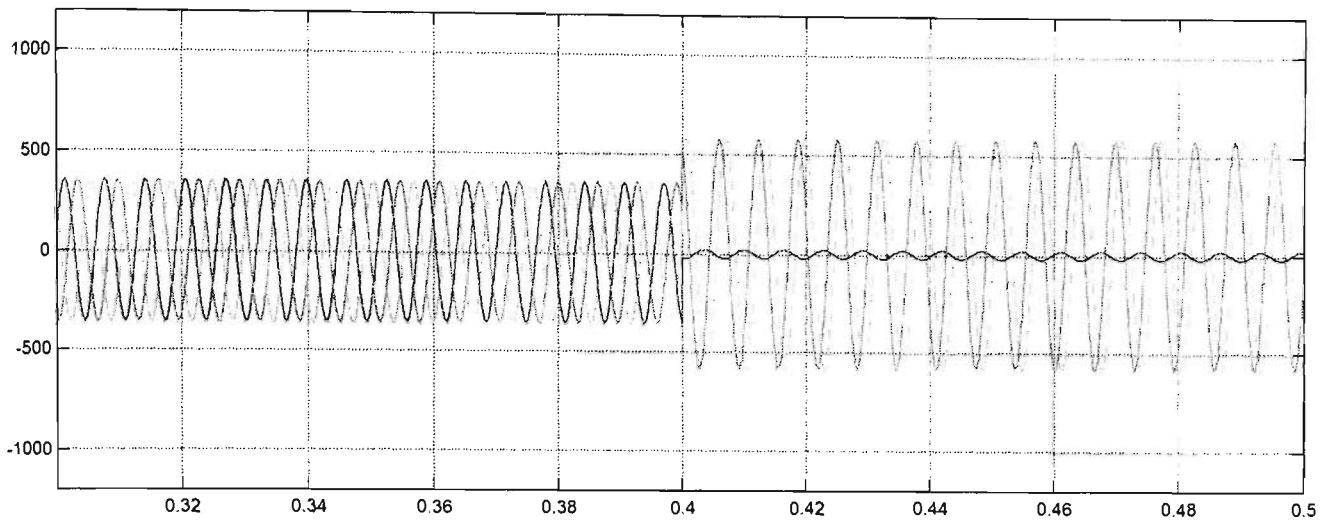


Figure 5.21 Phase A to ground fault on the location Y (transient voltage) at 140Hz

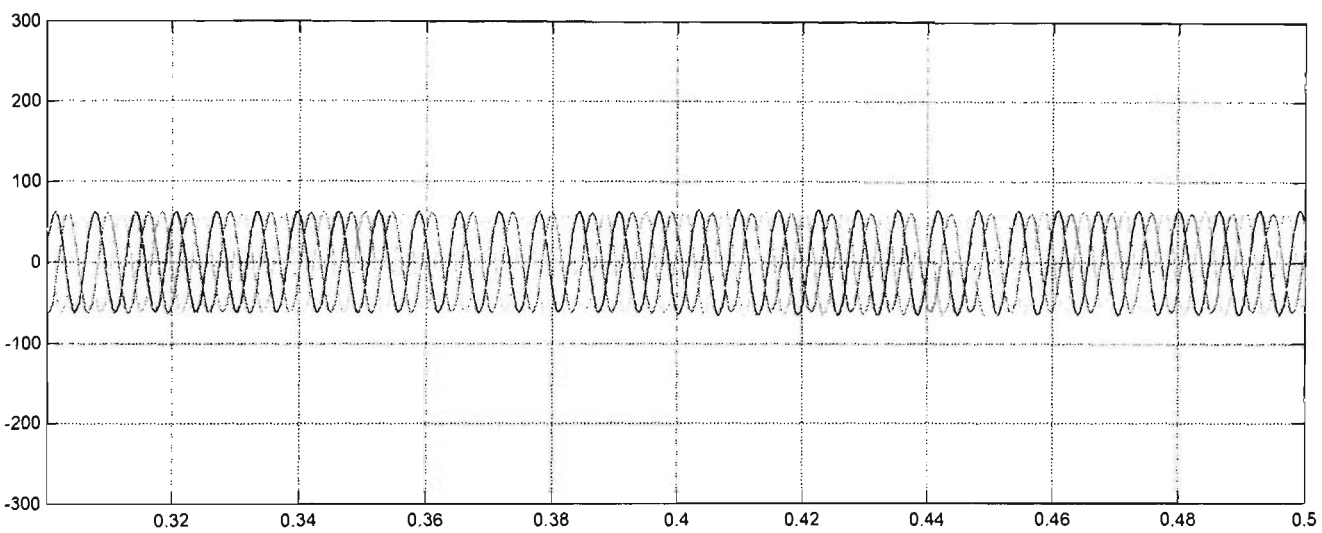


Figure 5.22 Phase A to ground fault on the location Y (transient current) at 140Hz

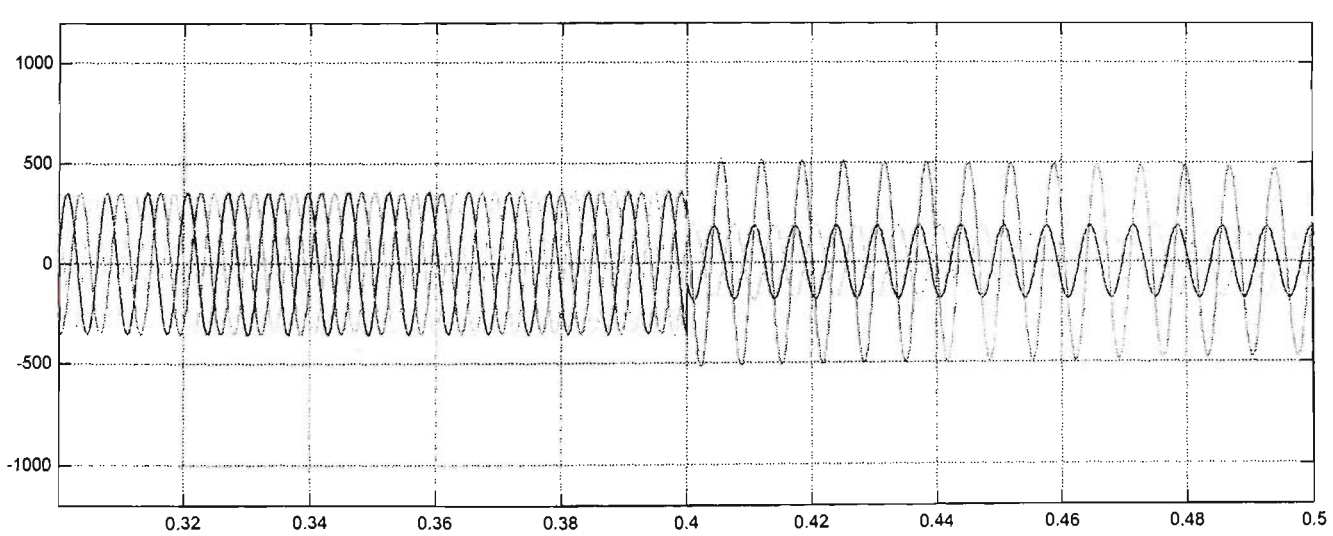


Figure 5.23 Phase A and B to ground fault on the point D (transient voltage) at 140Hz

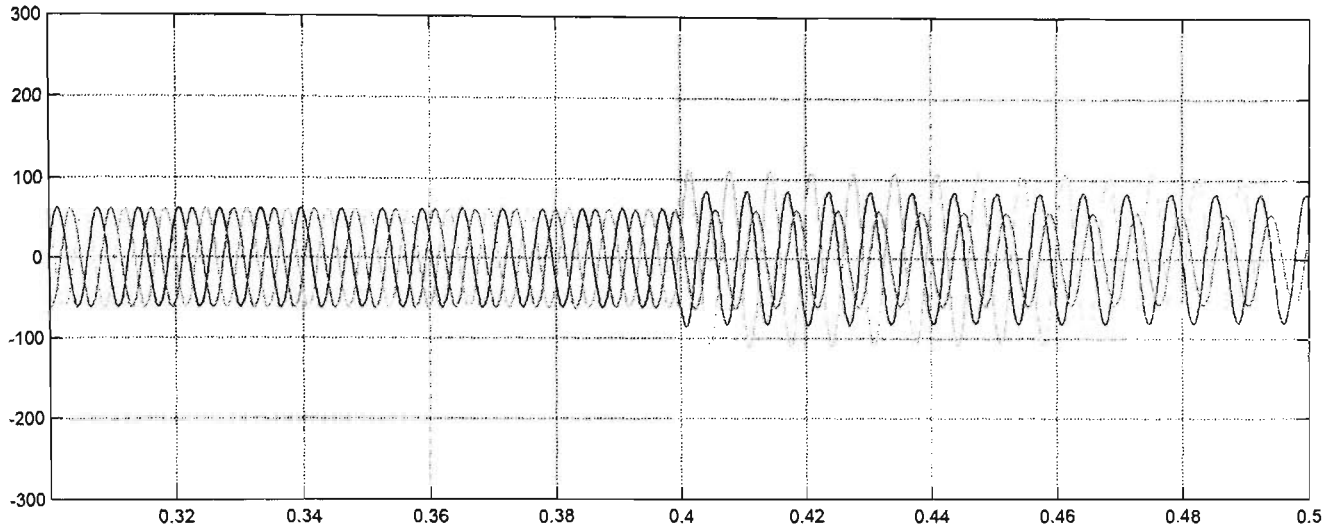


Figure 5.24 Phase A and B to ground fault on the location Y (transient current) at 140Hz

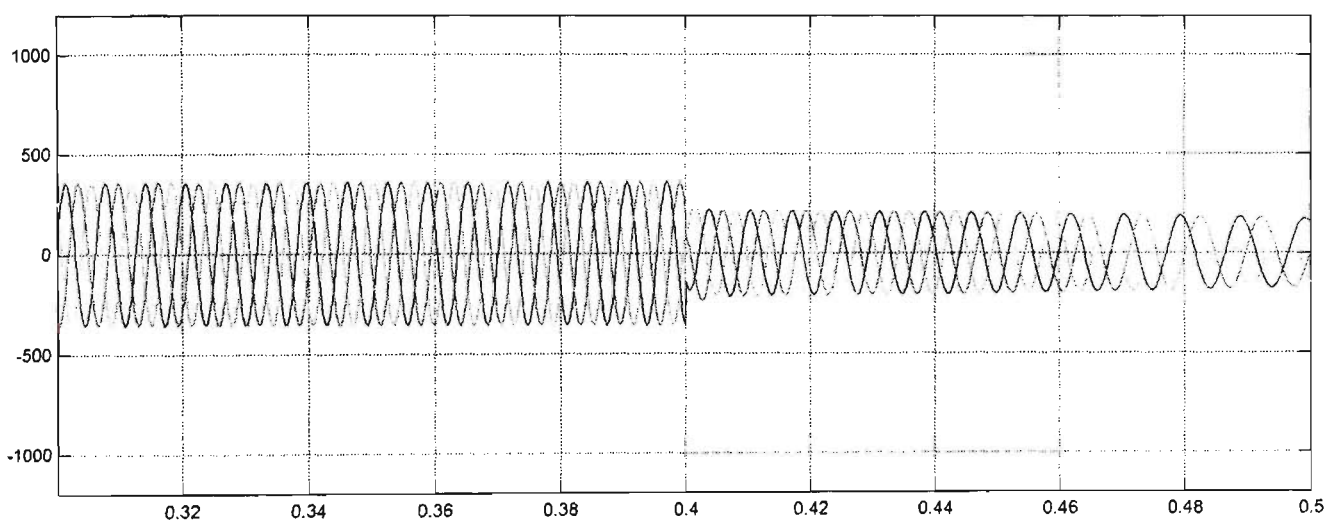


Figure 5.25 Three phase to ground fault on the location Y (transient voltage) at 140Hz

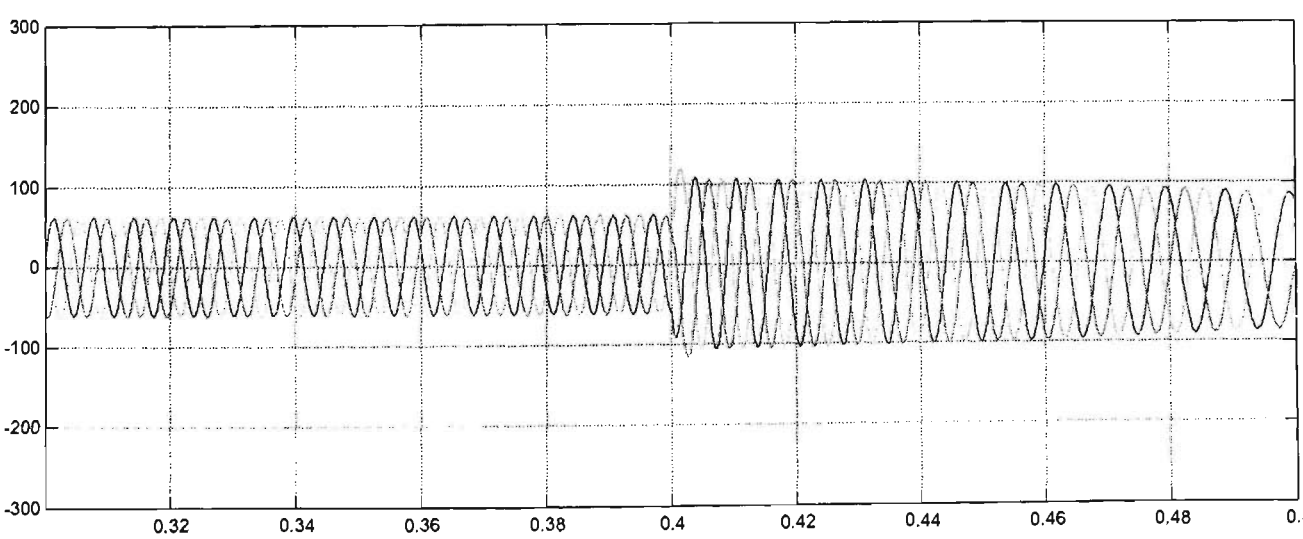


Figure 5.26 Three phase to ground fault on the location Y (transient current) at 140Hz

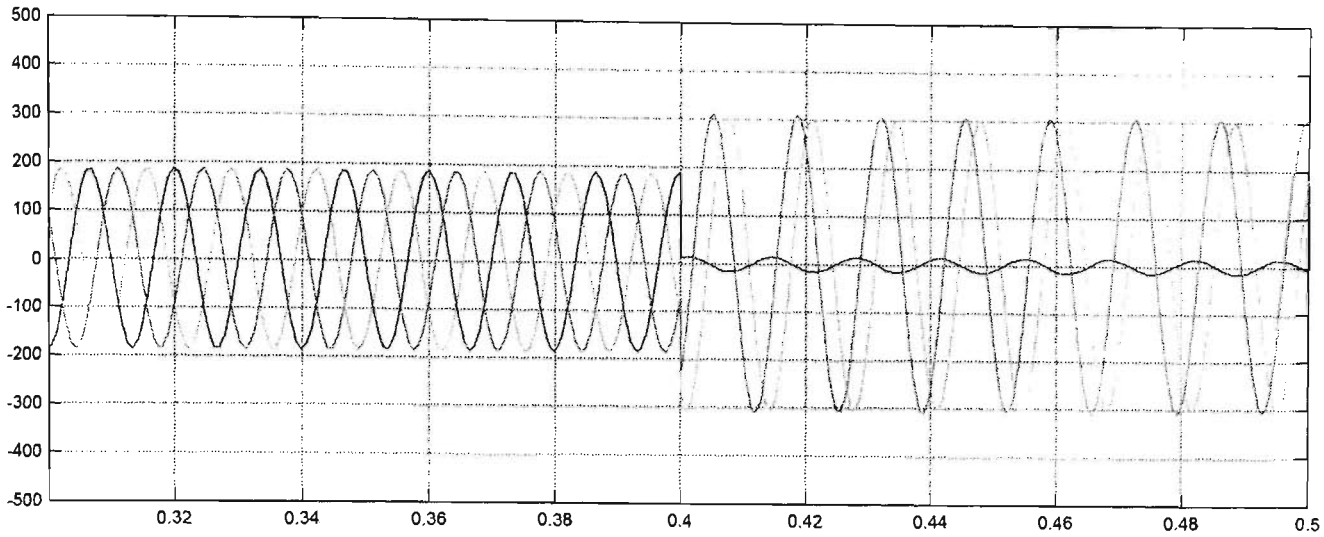


Figure 5.27 Phase A to ground fault on the location Y (transient voltage) at 70Hz

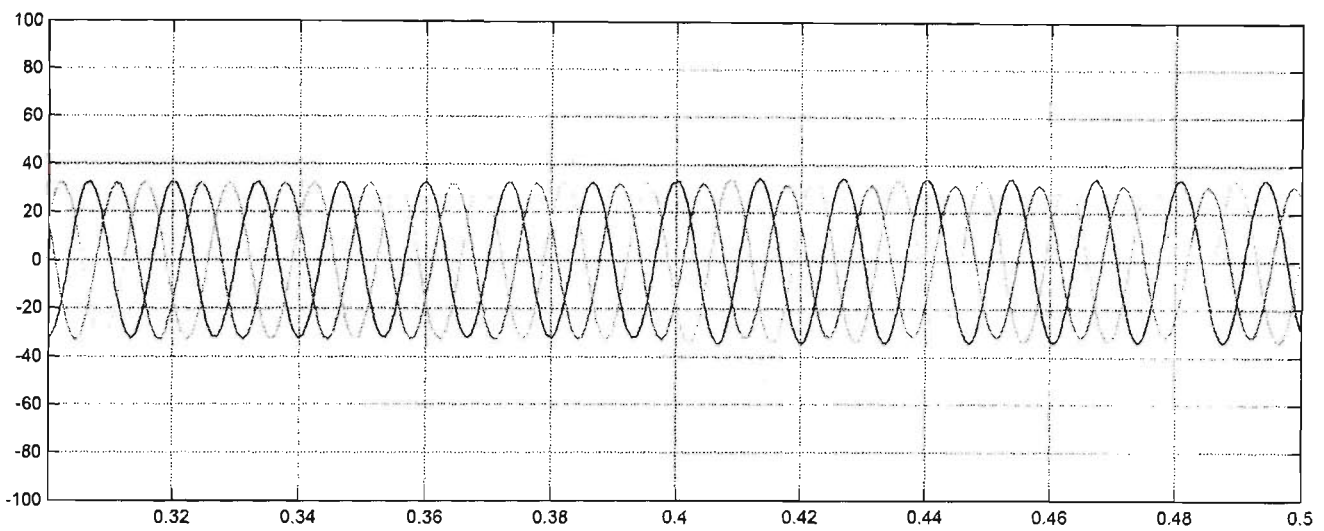


Figure 5.28 Phase A to ground fault on the location Y (transient current) at 70Hz

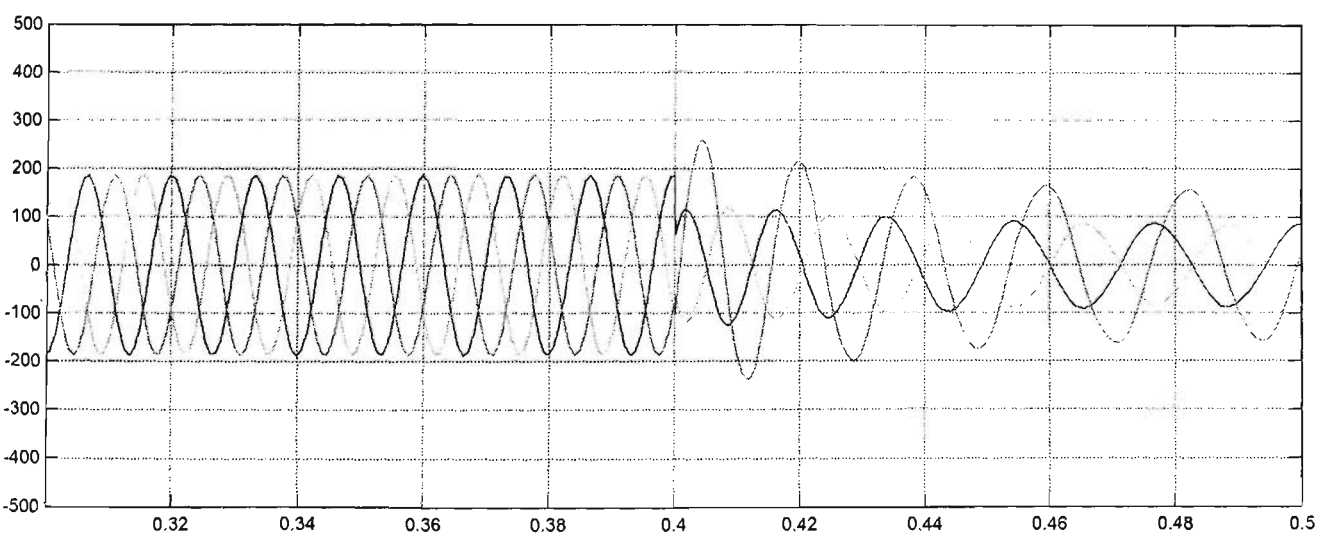


Figure 5.29 Phase A and B to ground fault on the location Y (transient voltage) at 70Hz

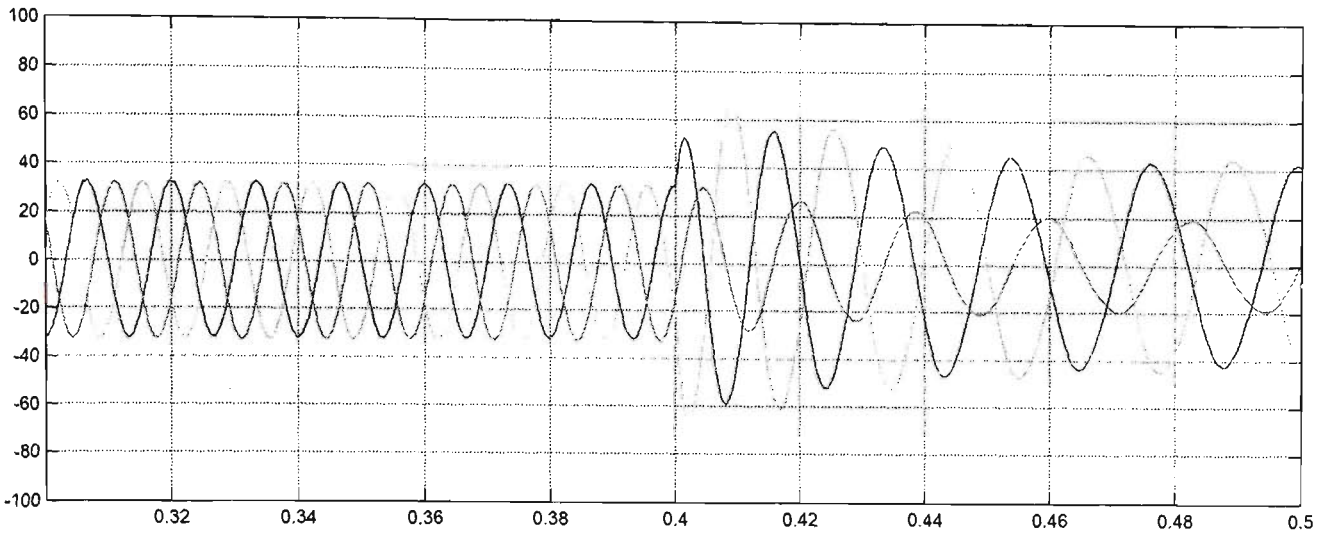


Figure 5.30 Phase A and B to ground fault on the location Y (transient current) at 70Hz

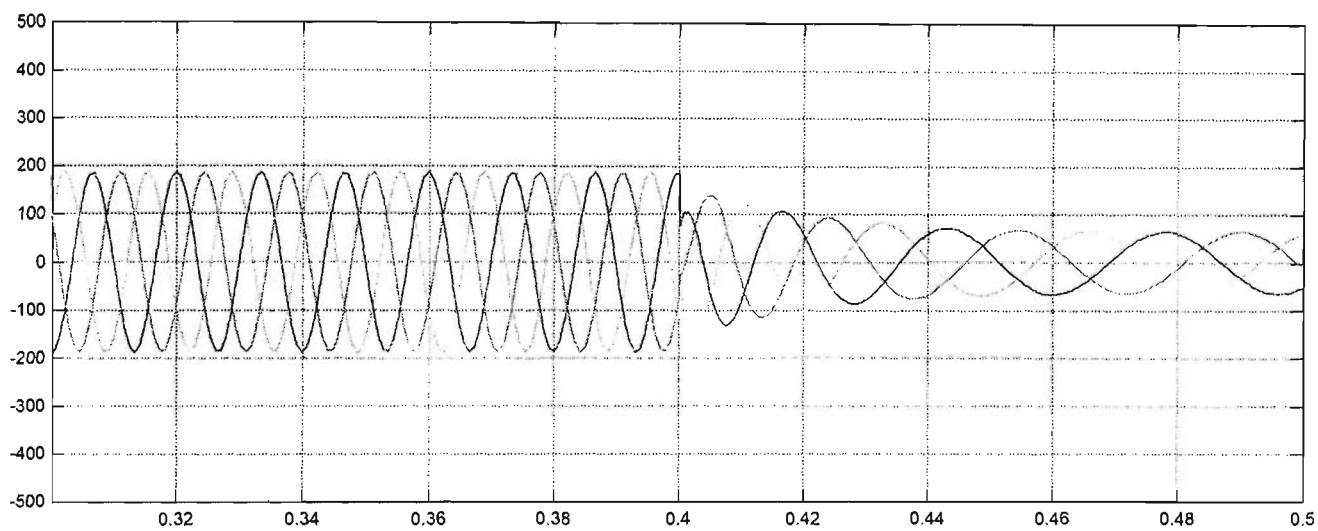


Figure 5.31 Three phase to ground fault on the location Y (transient voltage) at 70Hz

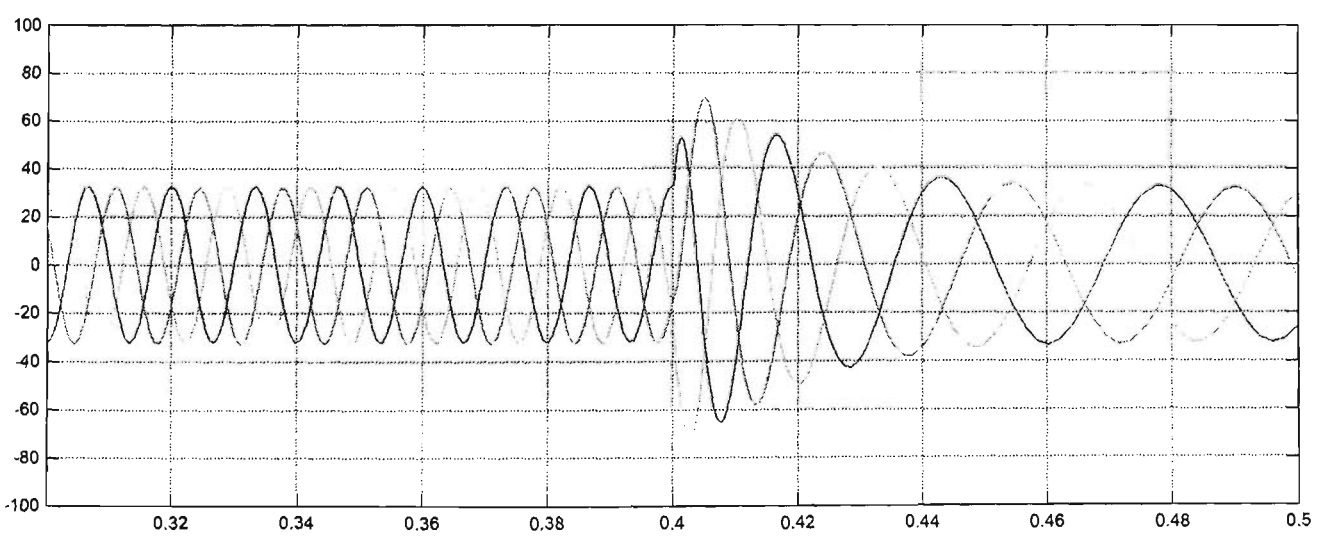


Figure 5.32 Three phase to ground fault on the location Y (transient current) at 70Hz

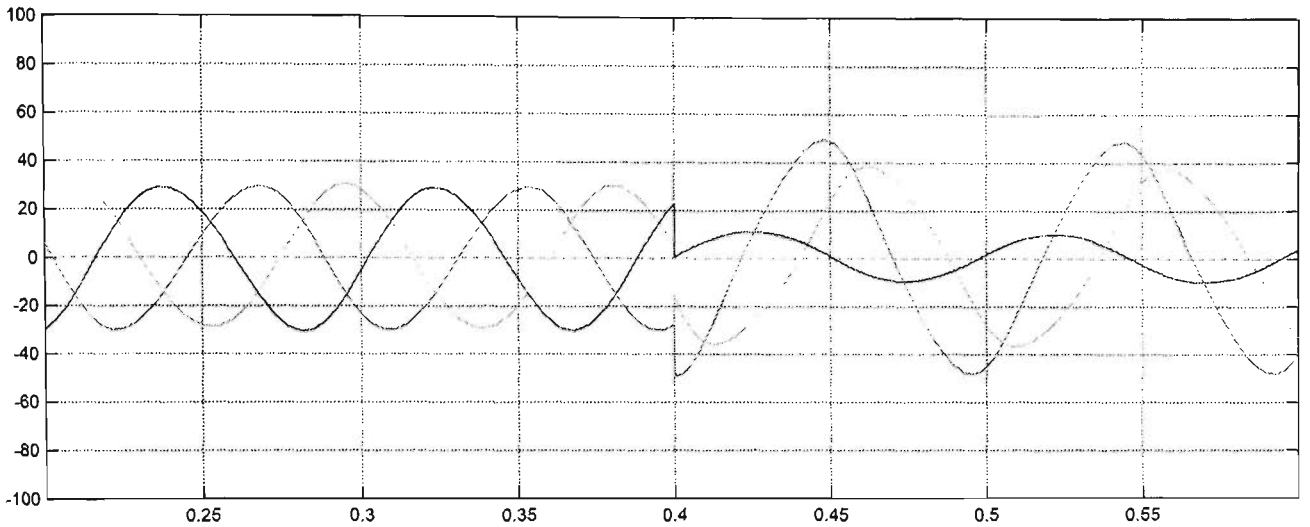


Figure 5.33 Phase A to ground fault on the location Y (transient voltage) at 9Hz

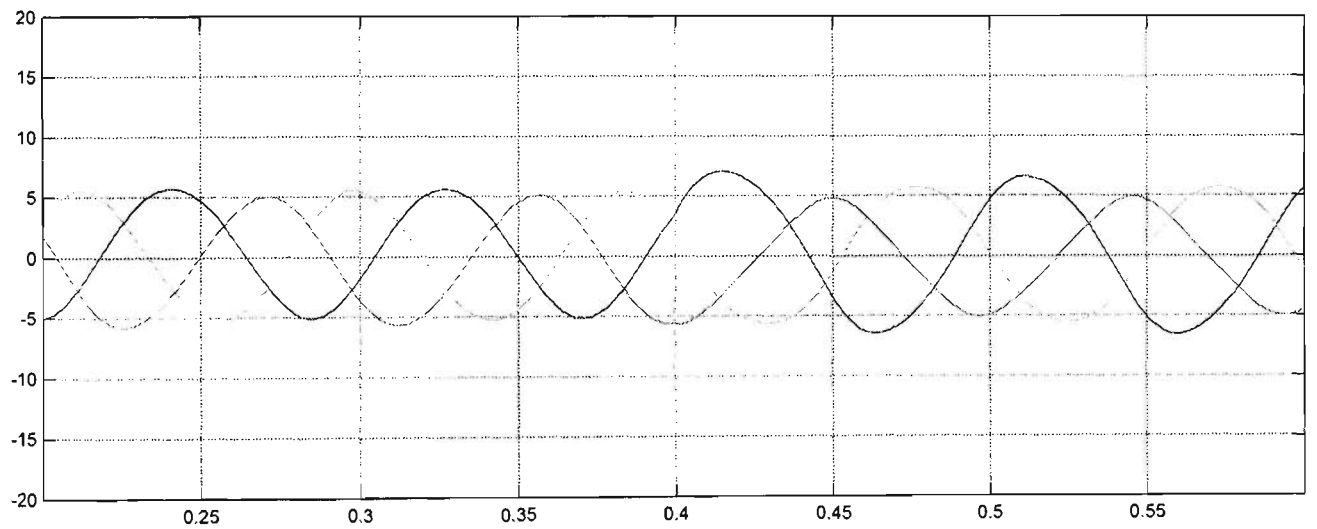


Figure 5.34 Phase A to ground fault on the location Y (transient current) at 9Hz

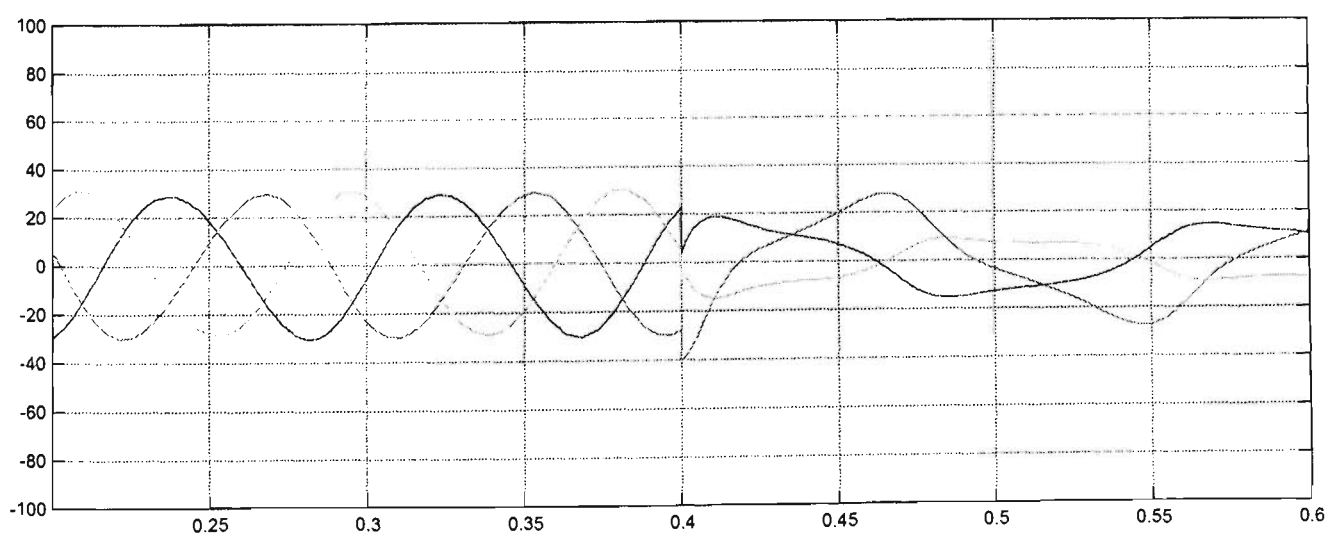


Figure 5.35 Phase A and B to ground fault on the location Y (transient voltage) at 9Hz

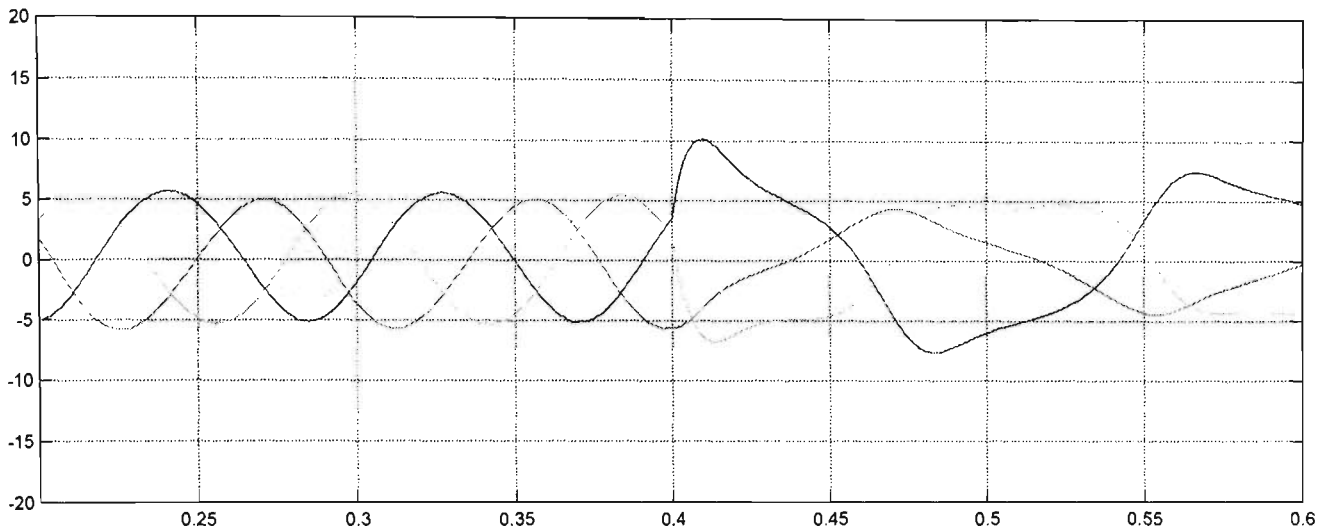


Figure 5.36 Phase A and B to ground fault on the location Y (transient current) at 9Hz

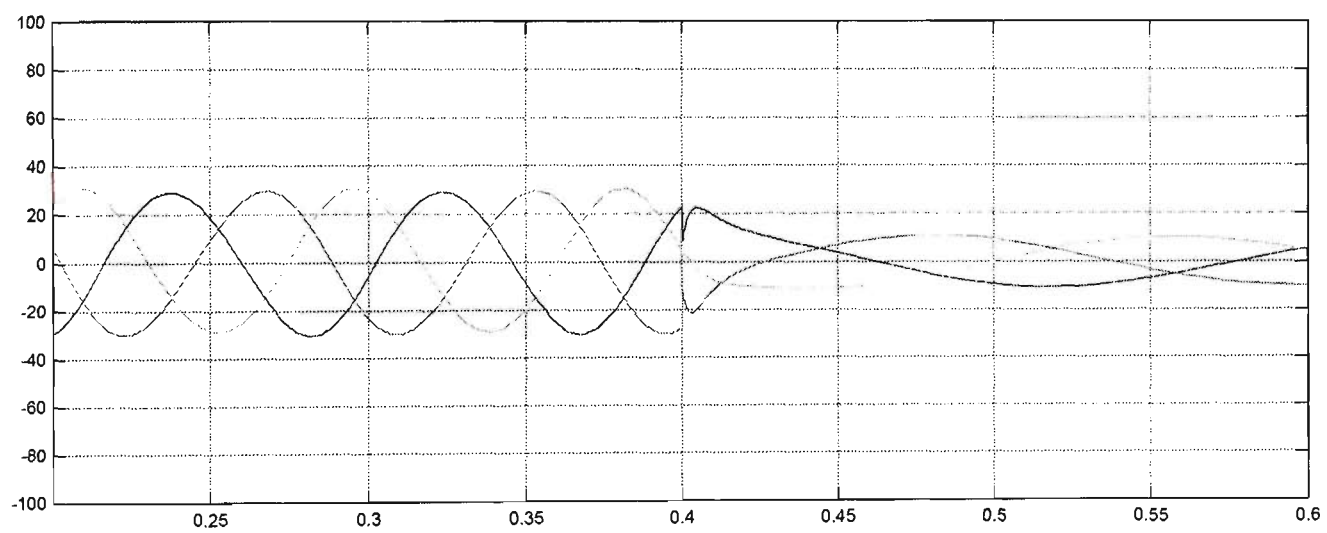


Figure 5.37 Three phase to ground fault on the location Y (transient voltage) at 9Hz

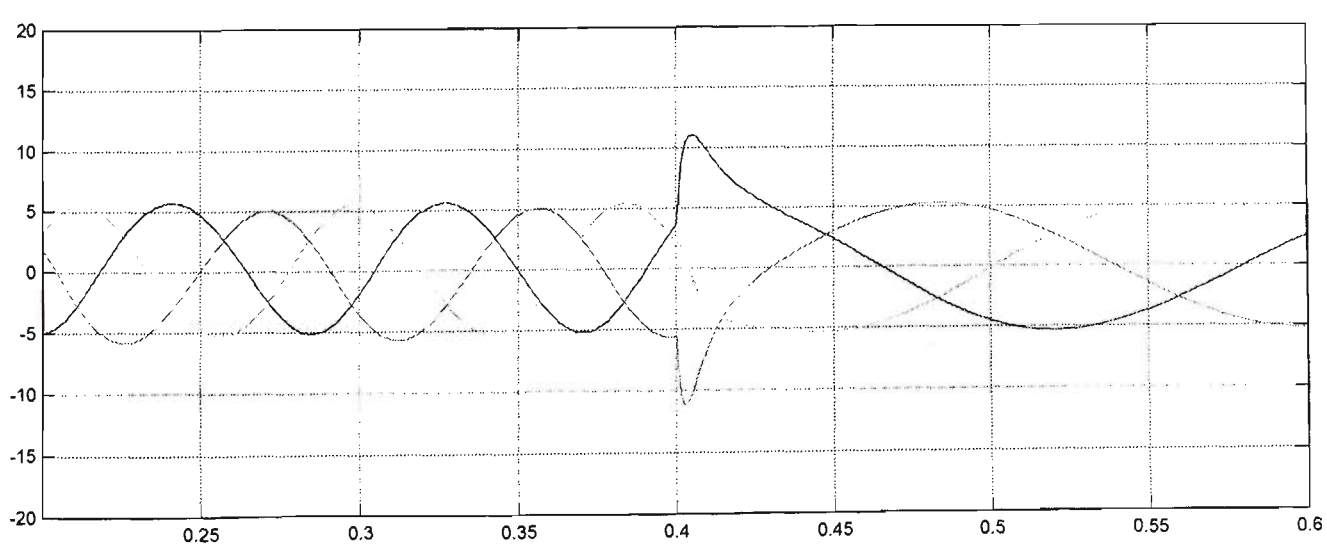


Figure 5.38 Three phase to ground fault on the location Y (transient current) at 9Hz

5.3 The simulation results of load flow analysis

This section describes the load flow analysis on a single line diagram for the wave power system as shown in Figure 5.39. The simulation contains a permanent magnet generator of 26.7kW power rating connected to three phase circuit breaker, step up and down transformers of 25kVA ratings, three phase transmission cable and one of 30KW inverter. Overall, this system consists of single feed to power grid system. A load flow simulation has been implemented in MATLAB program.

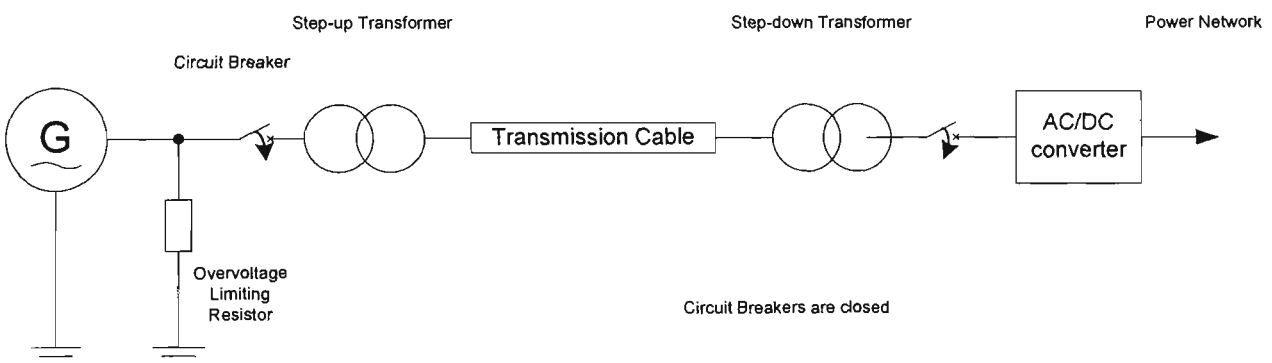


Figure 5.39 Illustration of a single line diagram of wave power system

Figures 5.40 to 5.51 show the line voltages and currents in all phase at the generator output measurement. The permanent magnet generator is simulated at variance frequency and torque. Also assumed are different load conditions at the DC-AC converter, which means that the battery in rapid charging mode to fully charged. Observation of the output of voltages and currents and comments are described in Table 5.3.

Table 5.3 Table of results for steady state simulation

Torque	No load	10kW load	Full load (20kW)
105N-m	Figures 5.40 and 5.41 Voltage: 420V peak line voltage 171V rms phase voltage Current: 62A peak current Frequency: 120Hz	Figures 5.42 and 5.43 Voltage: 300V peak line voltage 122V rms phase voltage Current: 62A peak current Frequency: 75Hz	Figures 5.44 and 5.45 Voltage: 250V peak line voltage 102V rms phase voltage Current: 62A peak current Frequency: 70Hz
60N-m	Figures 5.46 and 5.47 Voltage: 220V peak line voltage 90V rms phase voltage Current: 32A peak current Frequency: 50Hz	Figures 5.48 and 5.49 Voltage: 150V peak line voltage 62V rms phase voltage Current: 32A peak current Frequency: 36Hz	Figures 5.50 and 5.51 Voltage: 135V peak line voltage 55V rms phase voltage Current: 32A peak current Frequency: 30Hz

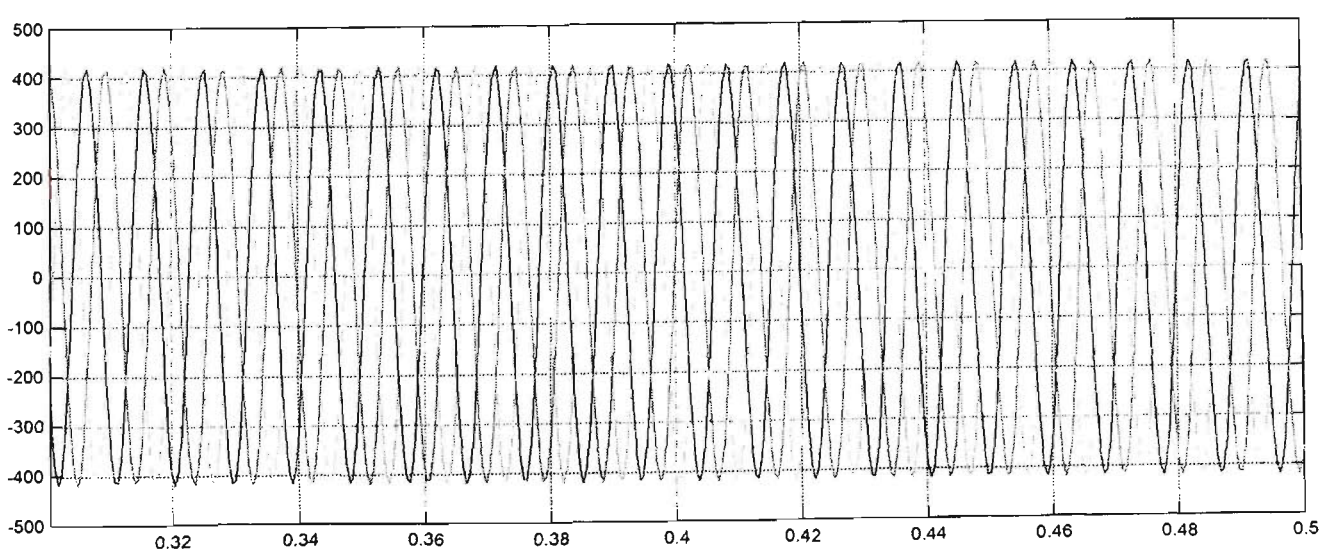


Figure 5.40 The voltage of generator drives at 105Nm on no load condition

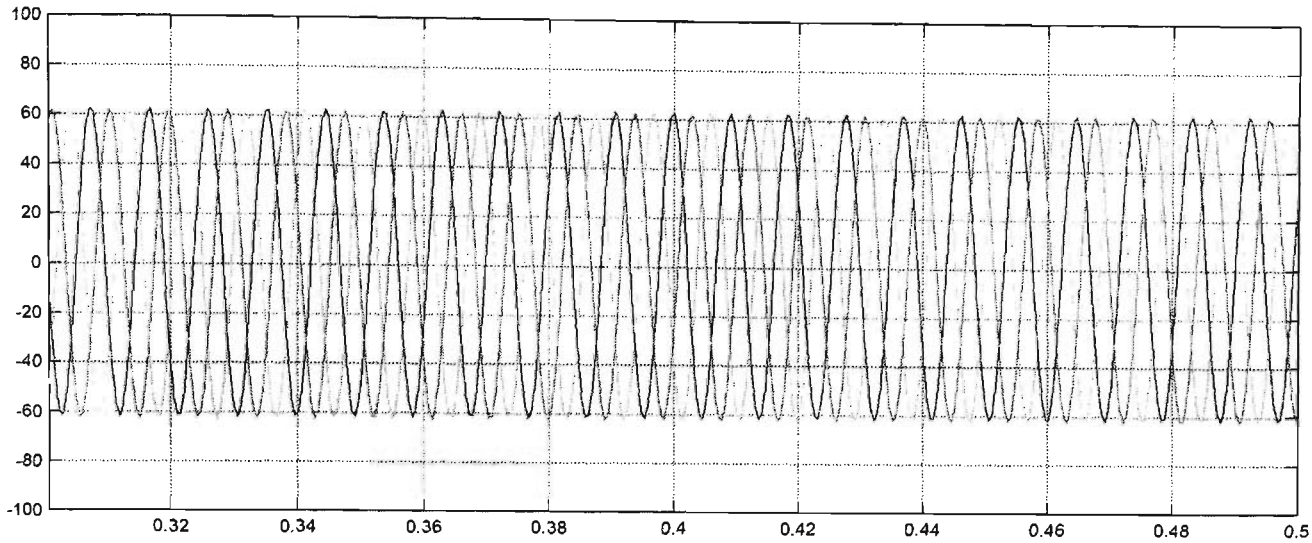


Figure 5.41 The current of generator drives at 105Nm on no load condition

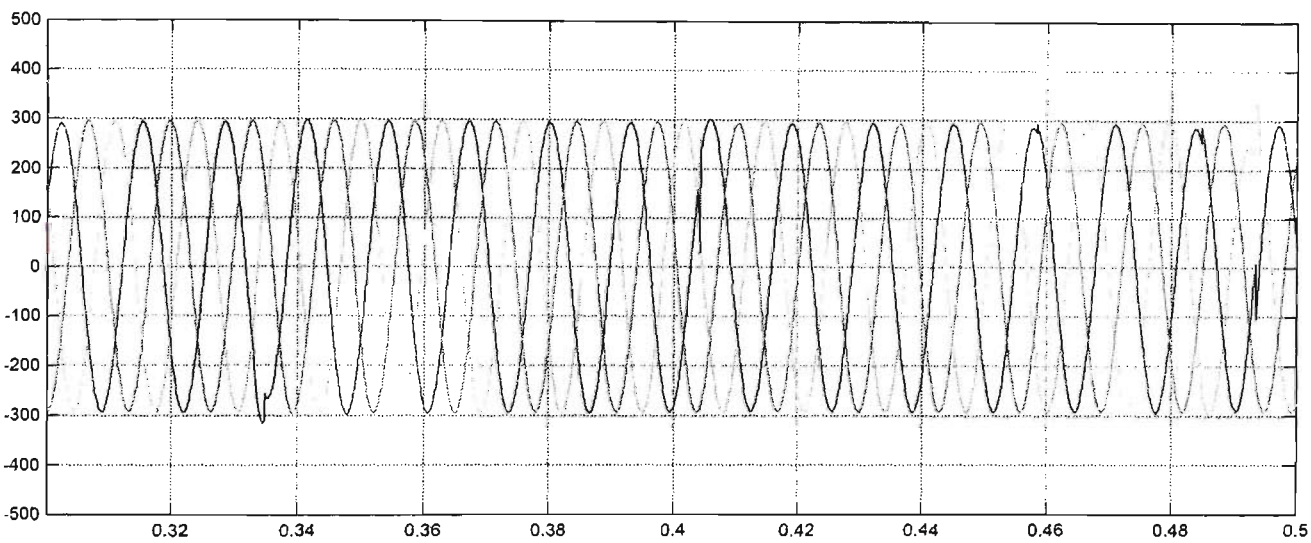


Figure 5.42 The voltage of generator drives at 105Nm on 10KW load condition

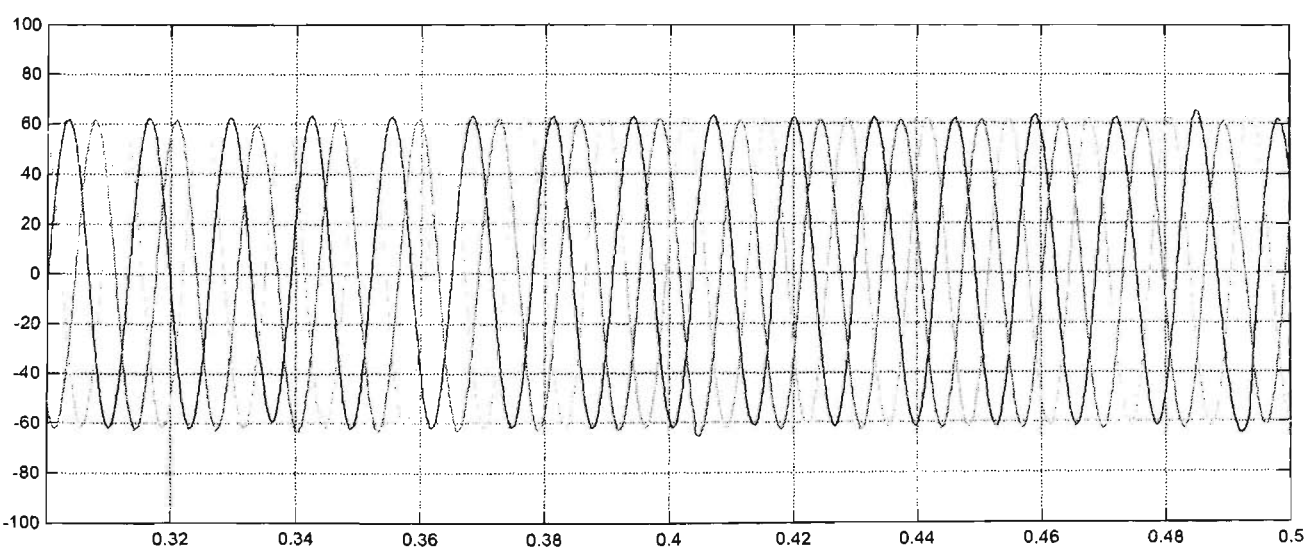


Figure 5.43 The current of generator drives at 105Nm on 10KW load condition

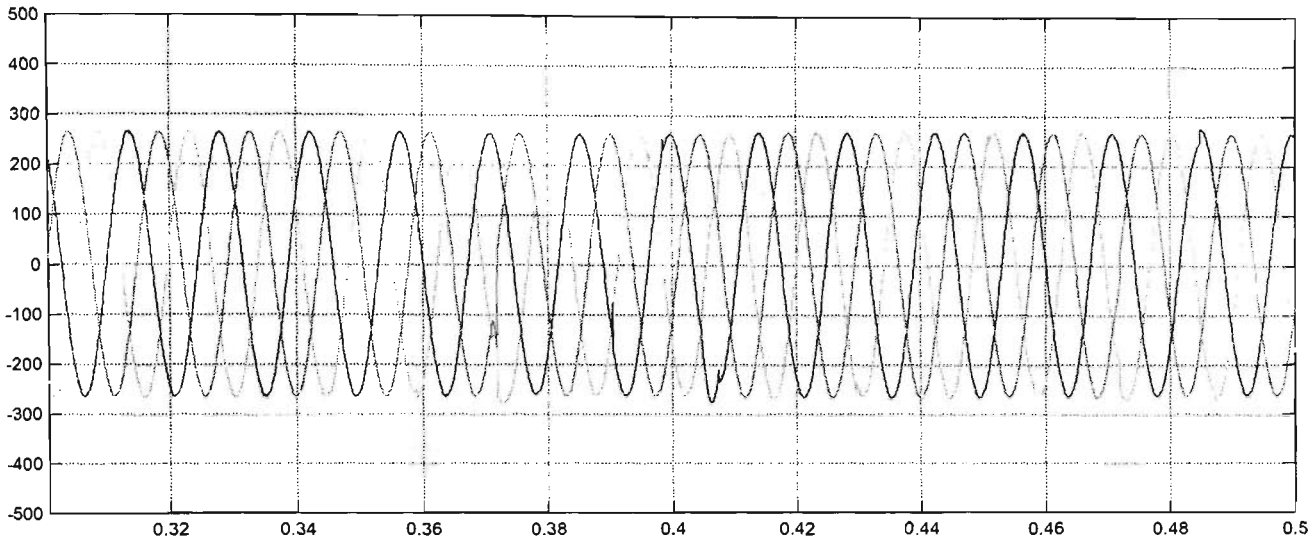


Figure 5.44 The voltage of generator drives at 105Nm on full load condition

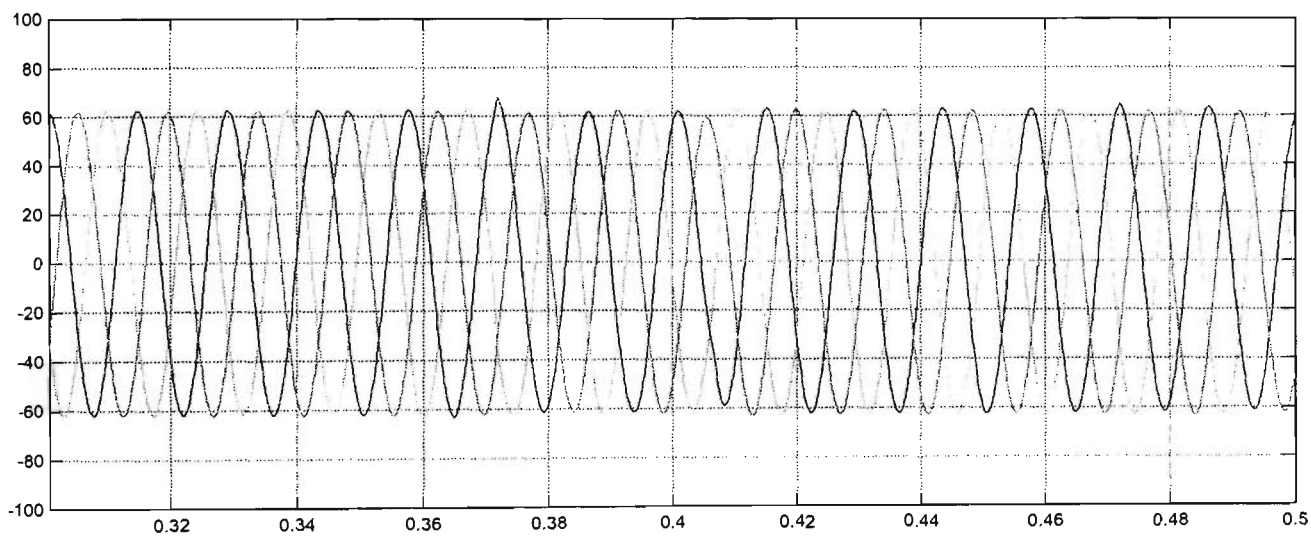


Figure 5.45 The current of generator drives at 105Nm on full load condition

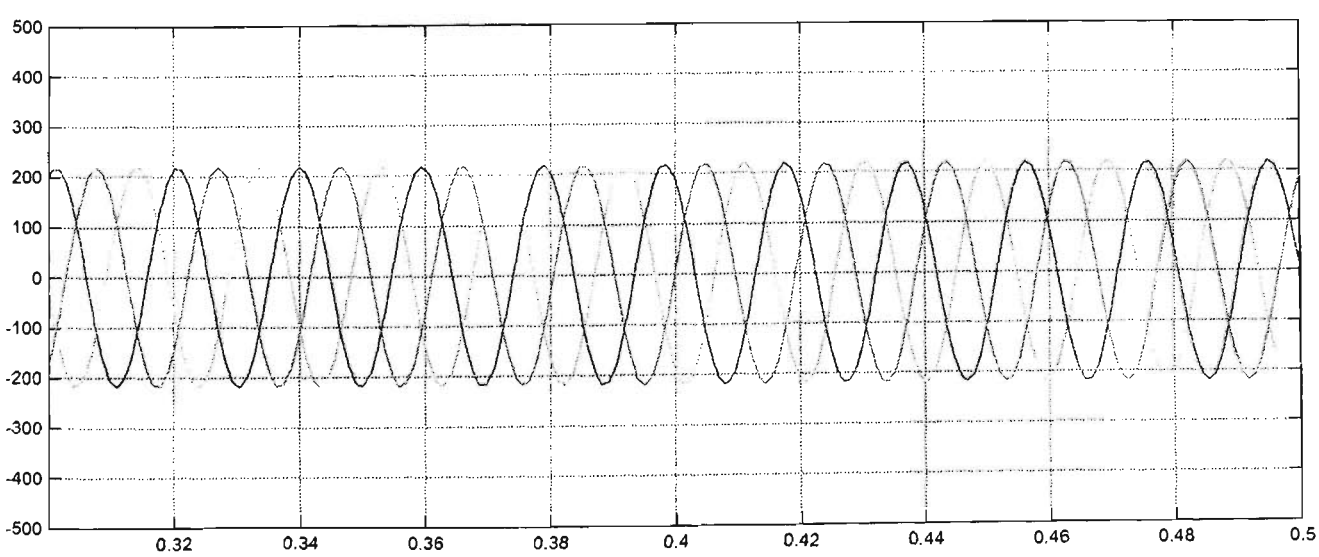


Figure 5.46 The voltage of generator drives at 60Nm on no load condition

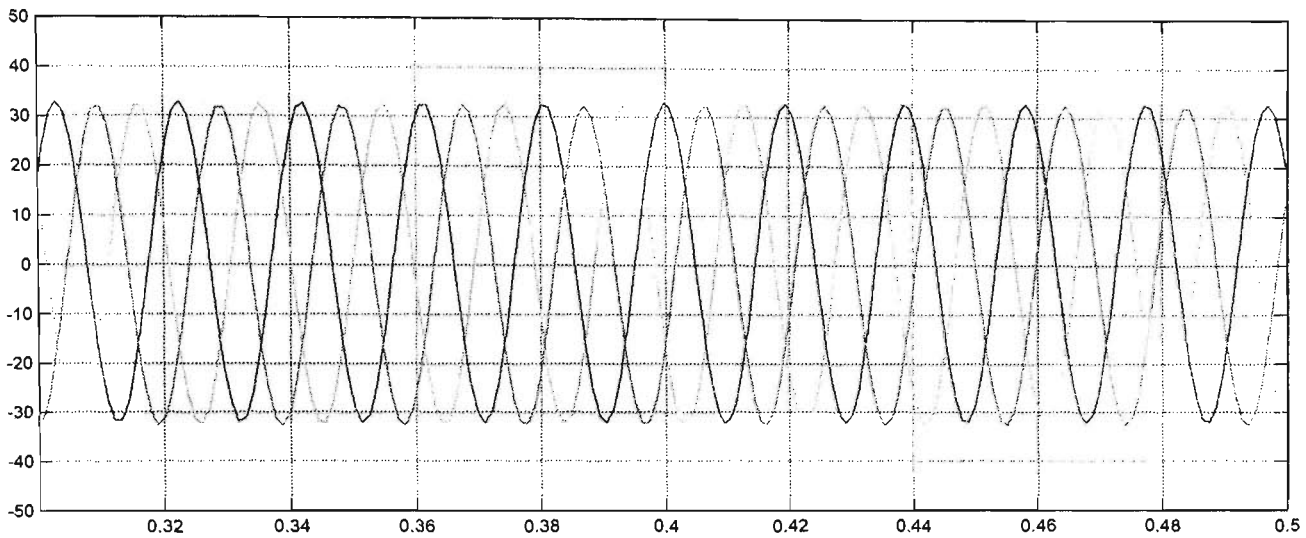


Figure 5.47 The current of generator drives at 10Nm on no load condition

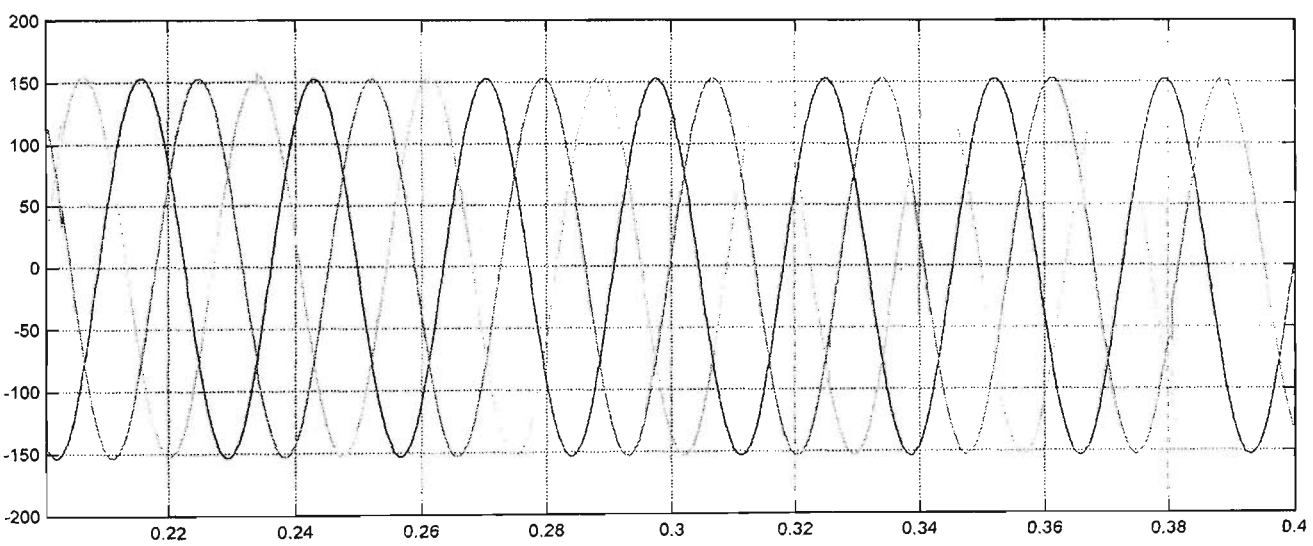


Figure 5.48 The voltage of generator drives at 60Nm on 10KW load condition

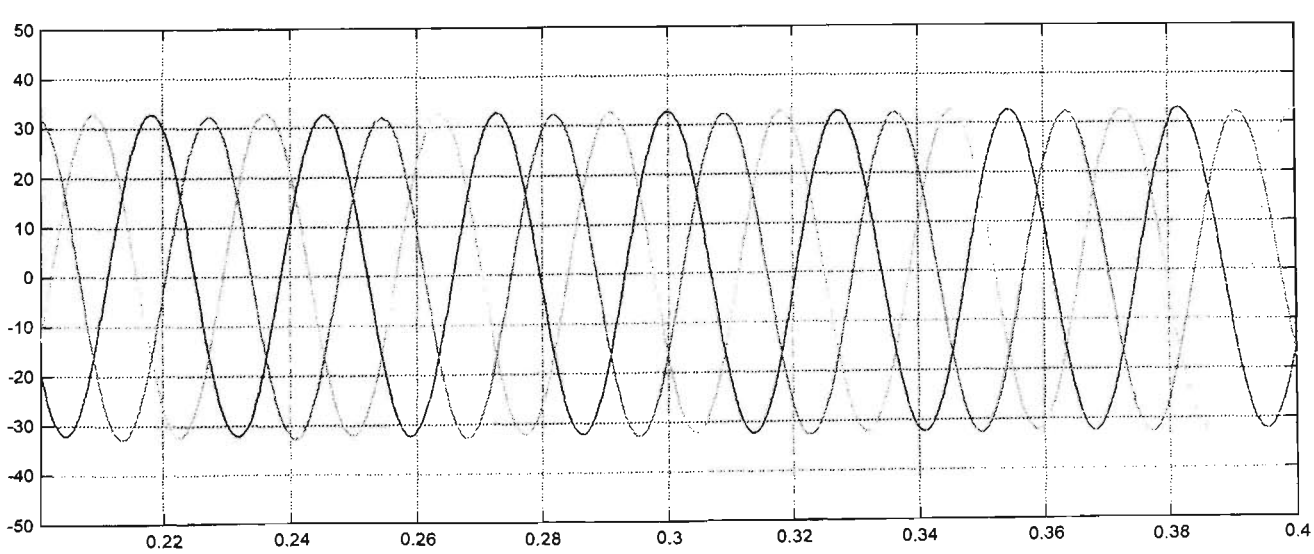


Figure 5.49 The current of generator drives at 60Nm on 10KW load condition

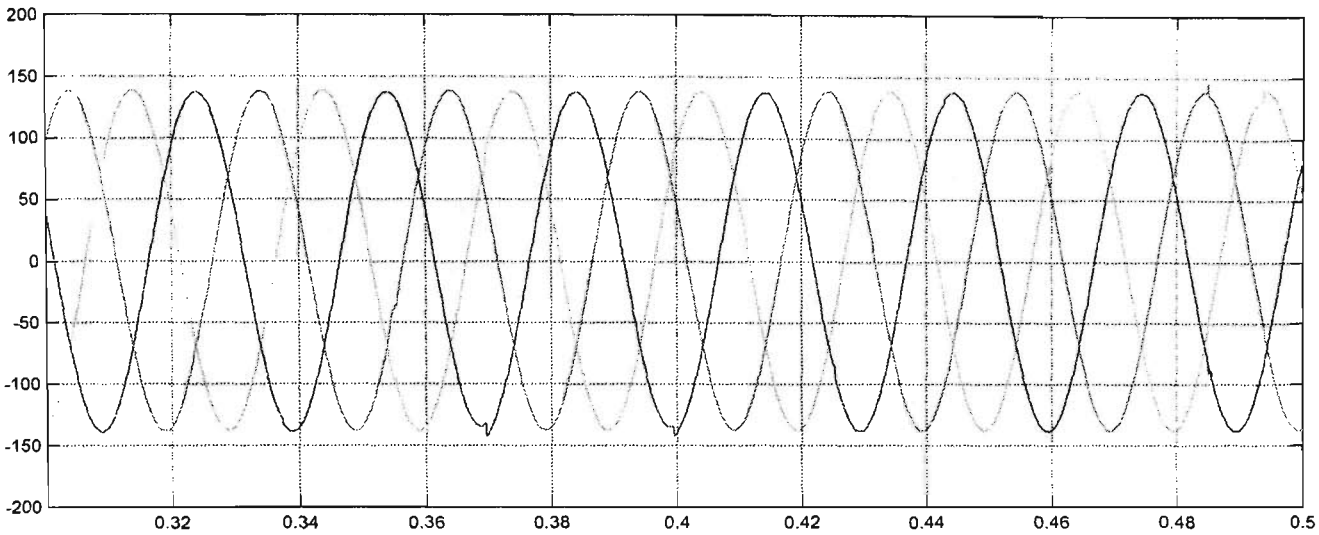


Figure 5.50 The voltage of generator drives at 60Nm on full load condition

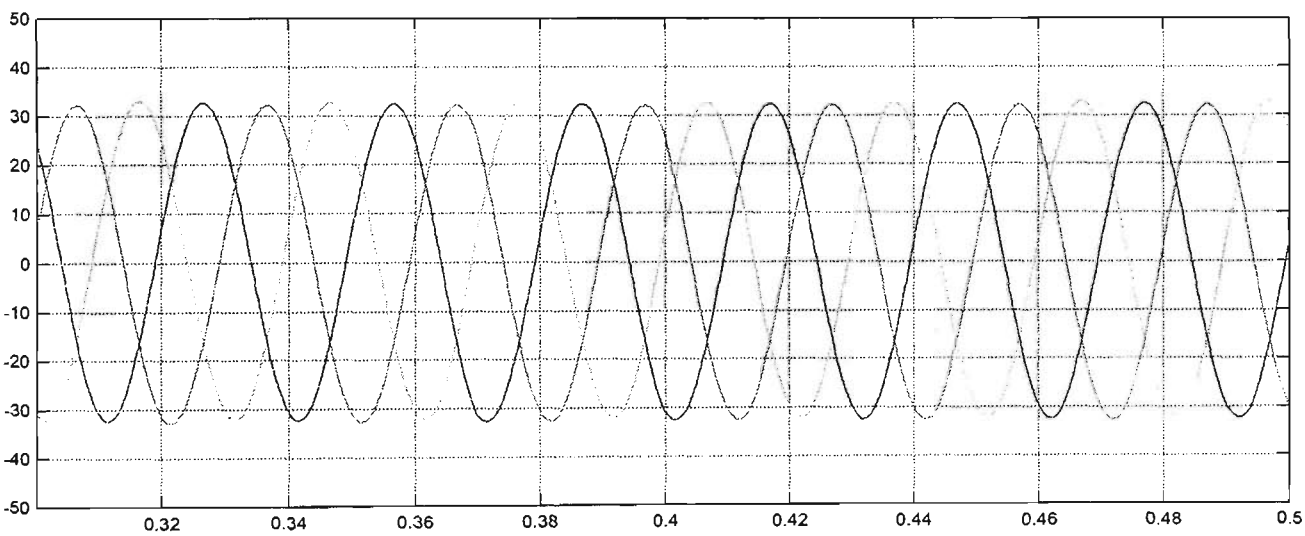


Figure 5.51 The current of generator drives at 60Nm on full load condition

5.4 Analysis Considering Load Condition

Figures 5.52 and 5.53 show the phase voltages and currents in all phase at the generator output measurement. The simulation period is set at 30-second. The permanent magnet generator simulated at the ocean wave energy produces waveforms from zero to maximum torque (105Nm). Also, assumes full load condition at the DC-AC converter is assumed that the battery and is rapid charging mode. The output currents and voltages are proportional to the frequency and torque of the generator. Furthermore, the power is dependant on the ocean wave amplitude.

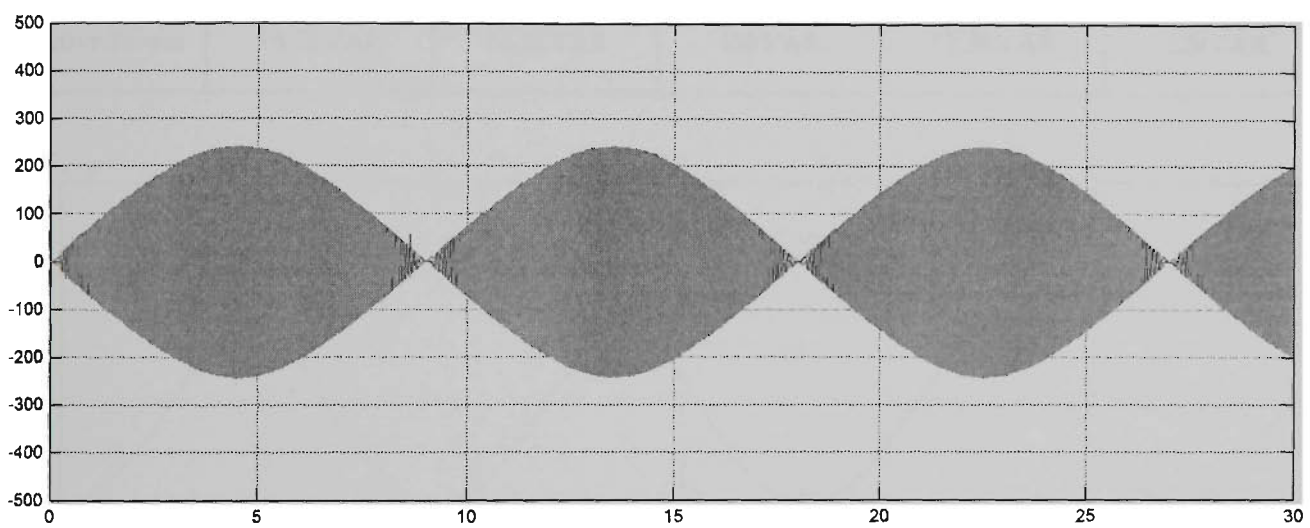


Figure 5.52 The voltage of generator drives wave power signal at full load condition

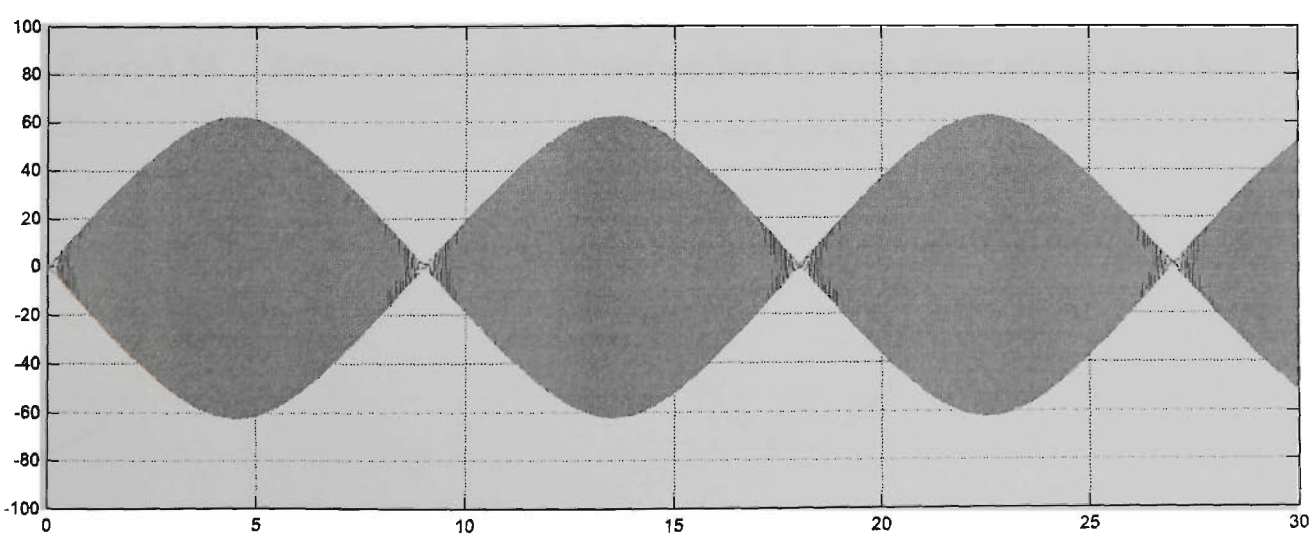


Figure 5.53 The current of generator drives wave power signal at full load condition

Table 5.4 shows that the generator drives almost at the rated power at no load condition. The output of current is absorbed by the overvoltage limiting resistors. At full load condition, the output of current is kept at maximum. Therefore, the frequency and voltage are decreased, as shown in Figure 5.44. The power of generator also drops to 20.5kW. Figures 5.54 to 5.58 show the simulation results of active and reactive power at different loads.

Table 5.4 The results of active and reactive power on different loads

	No load	5kW load	10kW load	15kW load	20kW load
Figure	5.54	5.55	5.56	5.57	5.58
Active Power	26.7kW	24kW	22.5kW	21kW	20.5kW
Reactive Power	16.5kVAR	14.5kVAR	13kVAR	12.5kVAR	12kVAR

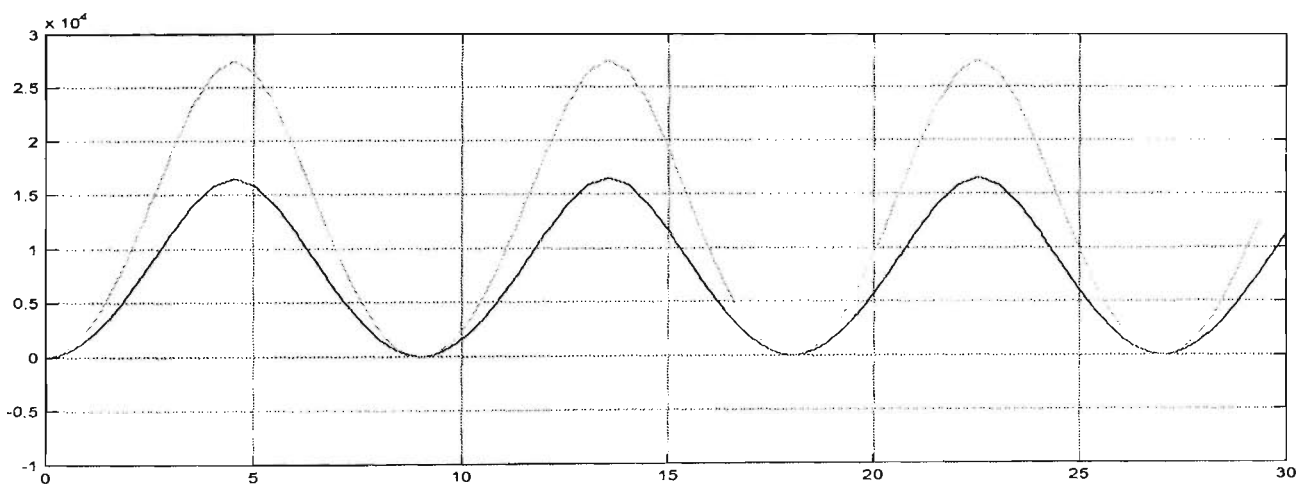


Figure 5.54 Active and Reactive Power produce by wave power energy on no load

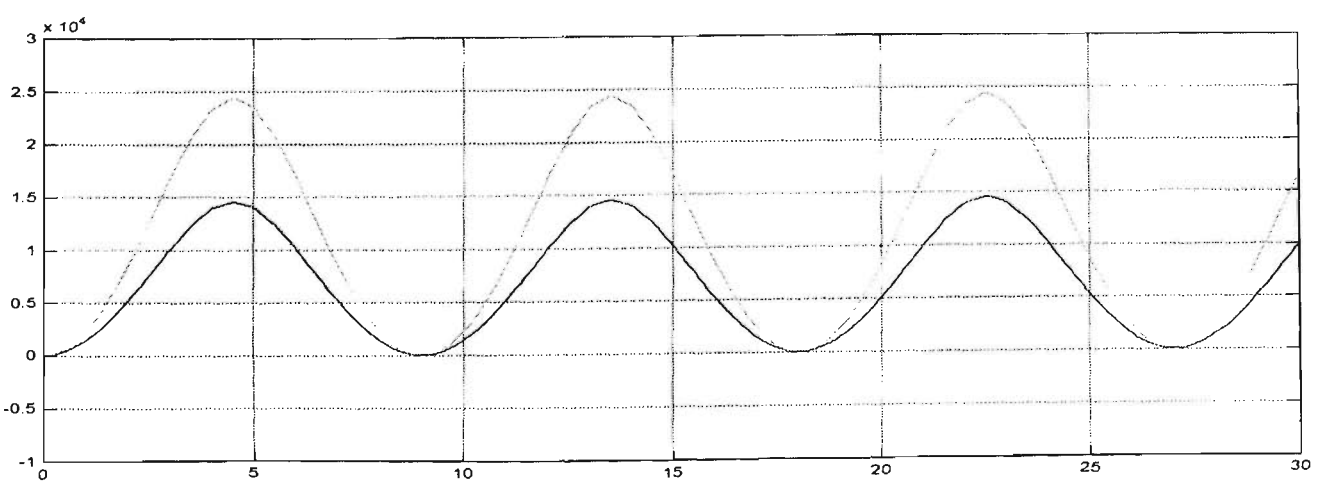


Figure 5.55 Active and Reactive Power produce by wave power energy on 5KW load

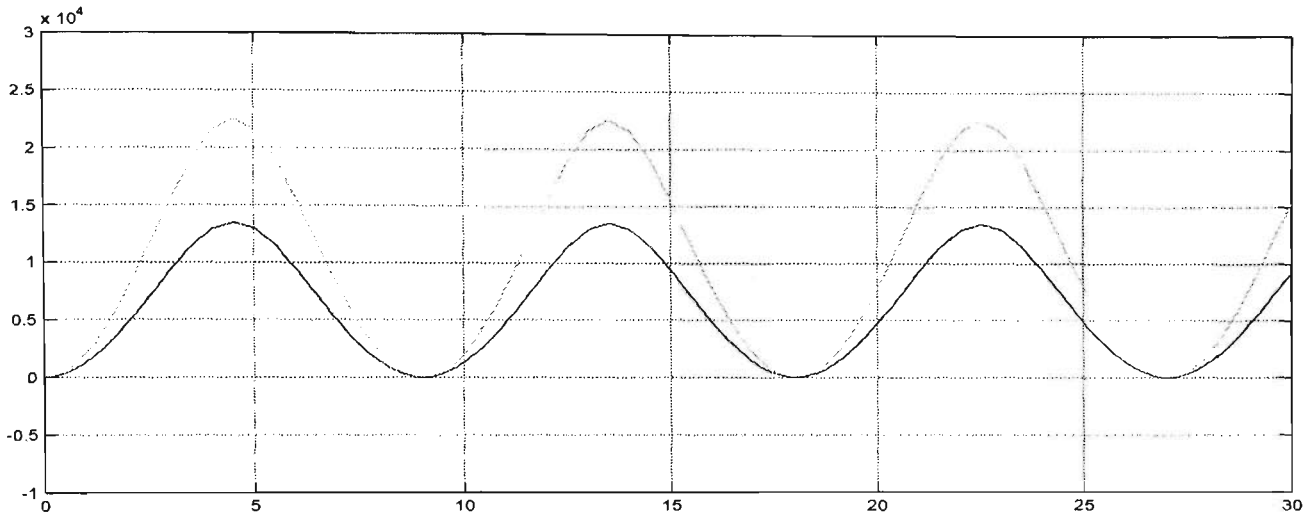


Figure 5.56 Active and Reactive Power produce by wave power energy on 10KW load

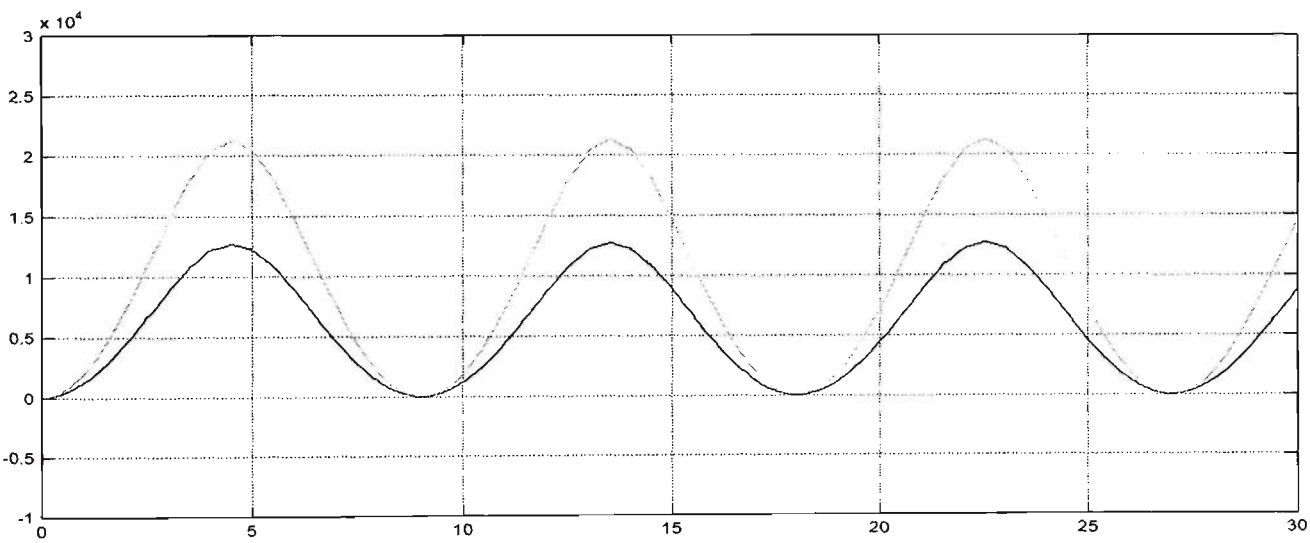


Figure 5.57 Active and Reactive Power produce by wave power energy on 15KW load

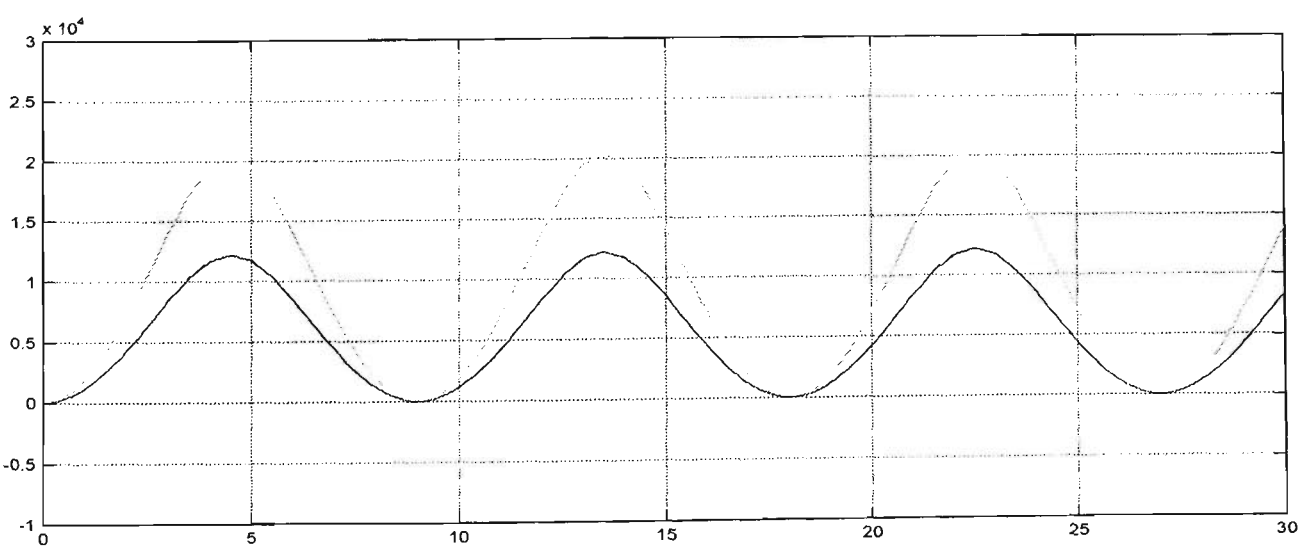


Figure 5.58 Active and Reactive Power produce by wave power energy on full load

5.5 Test system and experimental results

To verify the simulation results, it was necessary to implement a practical experiment in order to illustrate the application of the proposed theory in power system studies.

This section describes the two experimental results of steady state load flow and transient fault analysis. The performance of the power system is experimentally verified by simulation results and practically implemented the by real currents. All experimental results are analysed and recorded using a Tektronix TDS310 digital storage oscilloscope.

The simulation results were created in Chapter 3 and the hardware experimental was described in Chapter 4. The microprocessor based circuit breaker, analogue current transducers data acquisition, three of single phase step down transformers and power converter to link with computer are connected together to form the completed wave power energy system. This is tested under a range of load flow and fault cases (Figure5.59).

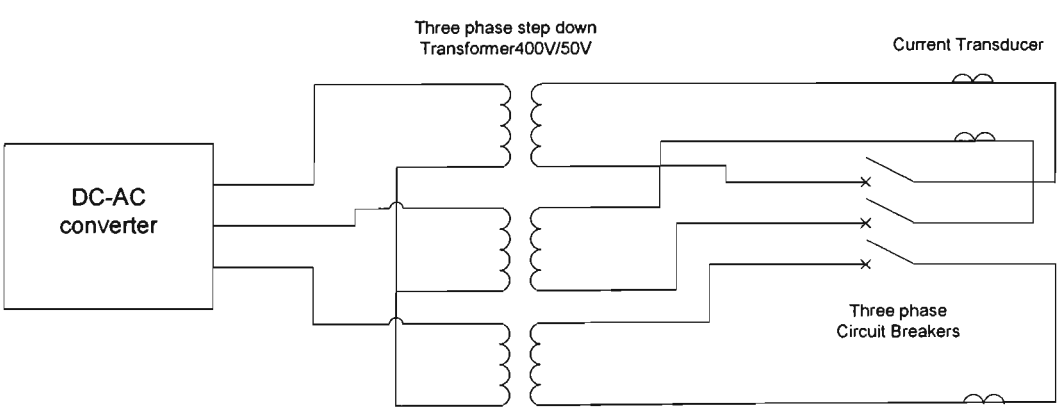


Figure 5.59 The experimental testing circuit diagram

5.5.1 Load flow experiment results

The channel one of the digital storage oscilloscope displays the A and B phase of generator producing the pulse width modulation signal. The measurement of period is 10ms per division and the amplitude of voltage is 5V per division. It converts to scale of three phase output currents to be 30A per division. The load flow analysis from simulation results generates power to supply 10kW load. The experimental results are shown in the Table 5.5 and the experimental waveforms are shown in the Figures 5.60 to 5.62.

Table 5.5 The results of load flow experiment results

Generator Torque	105 N-m	60 N-m	10 N-m
Peak Current (A)	62A	30A	6A
Frequency (Hz)	75Hz	36Hz	10Hz

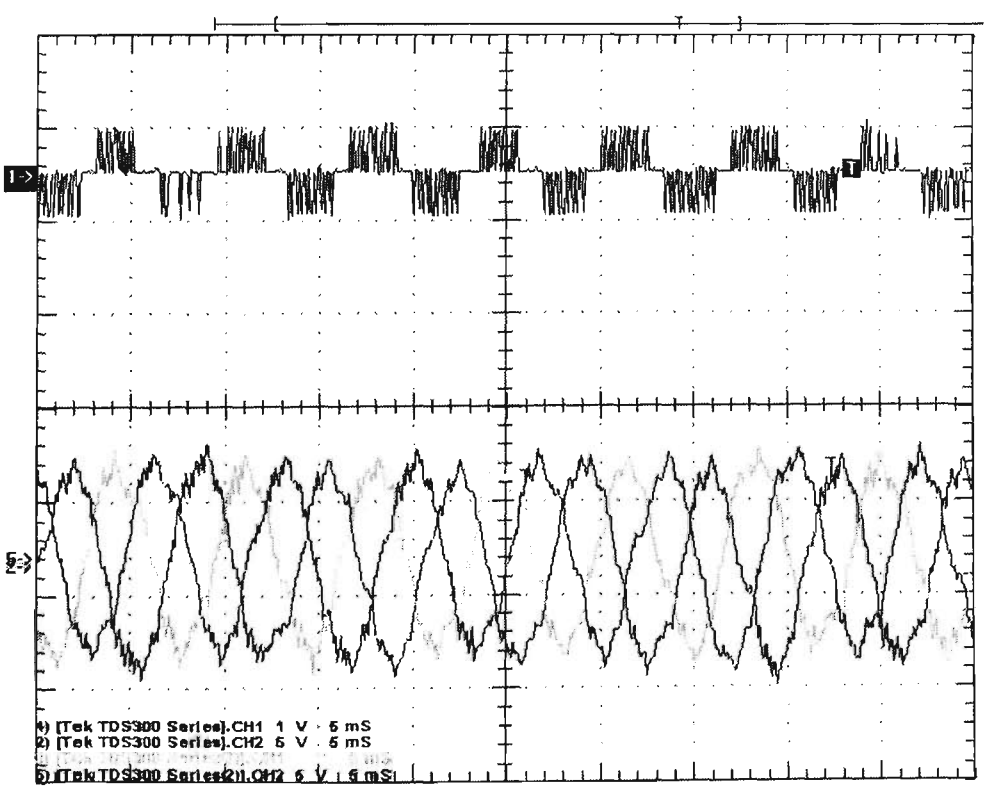


Figure 5.60 The PWM generates sin wave output signal at 105Nm

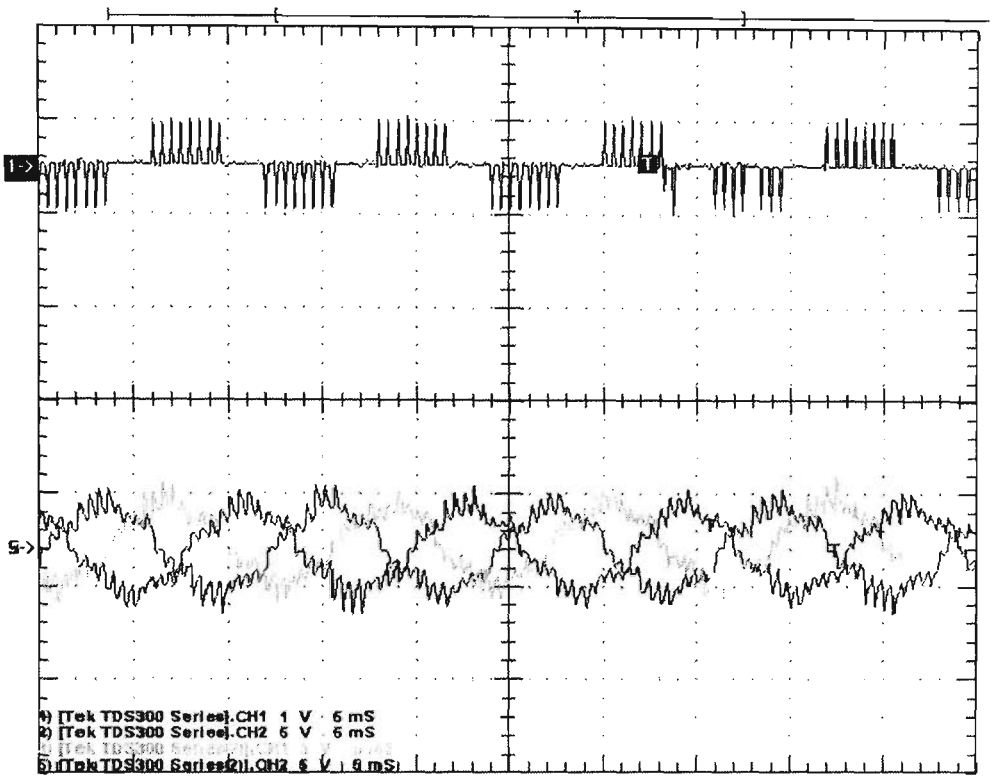


Figure 5.61 The PWM generates sin wave output signal at 60Nm

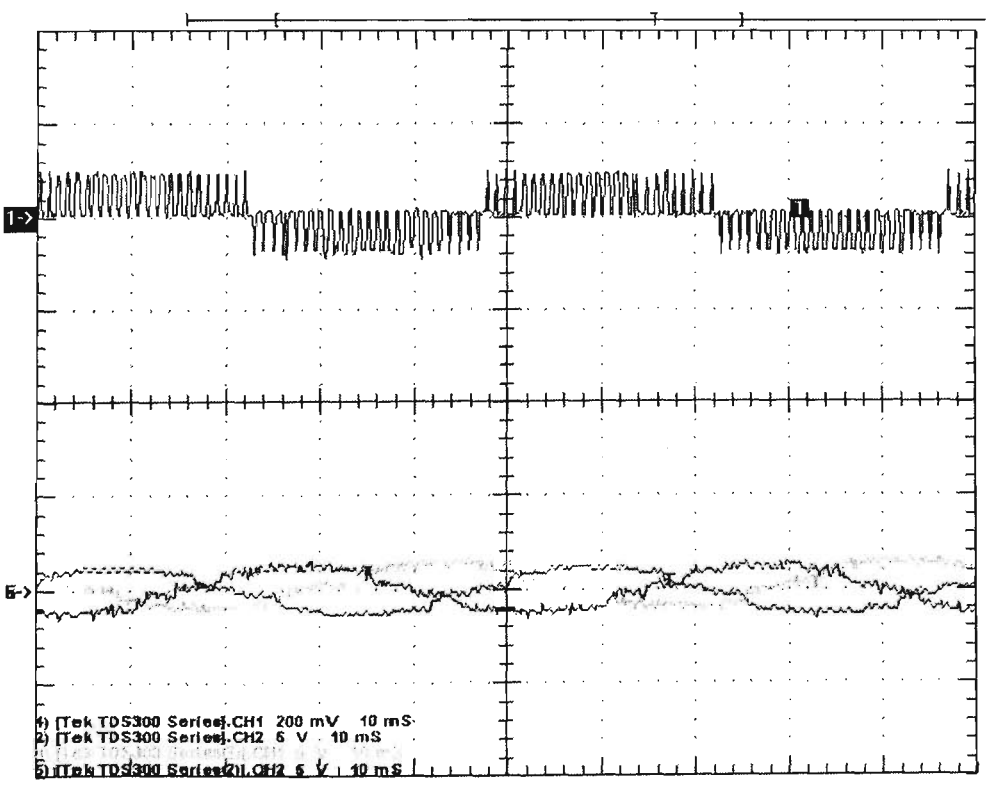


Figure 5.62 The PWM generates sin wave output signal at 10Nm

5.5.2 Transient current experiment results

The experimental results of generator for single phase, two phase and three phase to ground fault at the location X and Y. Occurring at 3 second are shown in Figures 5.63 to 5.64. The channel one of the digital storage oscilloscope shows the experiment result of generator producing the pulse width modulation signal to drive the step down transformers. The measurement of period is 10ms per division and the amplitude of voltage is 5V per division. It converts to scale of three phase current to be 30A per division. The full load condition is applied to experiment the transient fault analysis.

The tripping time determined by the microprocessor based protection setting, has been mentioned on section 5.6.2 and Table 5.8. The experiment results and comments are shown in Table 5.6 and the experimental waveforms are shown in Figures 5.63 to 5.68.

Table 5.6 Table of experiment results for transient fault analysis

Torque Location	Three Phase to earth fault	Phase B - C to earth fault	Phase A to earth fault
105Nm Location X	Figure 5.63 The circuit breaker tripping time is about 0.03 second after the fault. The envelope of fault currents occur in three phases, and the amplitudes of fault currents are about two times of normal current.	Figure 5.64 The tripping time also is about 0.03 second after the fault occurrence. The envelope of fault currents occur in phases B and C. Thus the current through phases A and B are more than normal current, which is due to earth fault. Phase A still remains sinusoidal form.	Figure 5.65 The circuit breaker is not to be tripped at any time after the fault occurrence. The amplitudes of fault current are same as the normal.
105Nm Location Y	Figure 5.66 The circuit breaker does not trip after the fault occurrence. The amplitude of fault currents only increases half time normal current during fault occurrence It means that the fault condition is no longer applicable in this case.	Figure 5.67 The circuit breaker is tripped at 0.07 second, and the amplitude of fault current of phases B and C is slightly increased after the fault occurrence.	Figure 5.68 The amplitude of fault current does not change after the fault occurrence. The circuit breaker also does not trip in this condition.

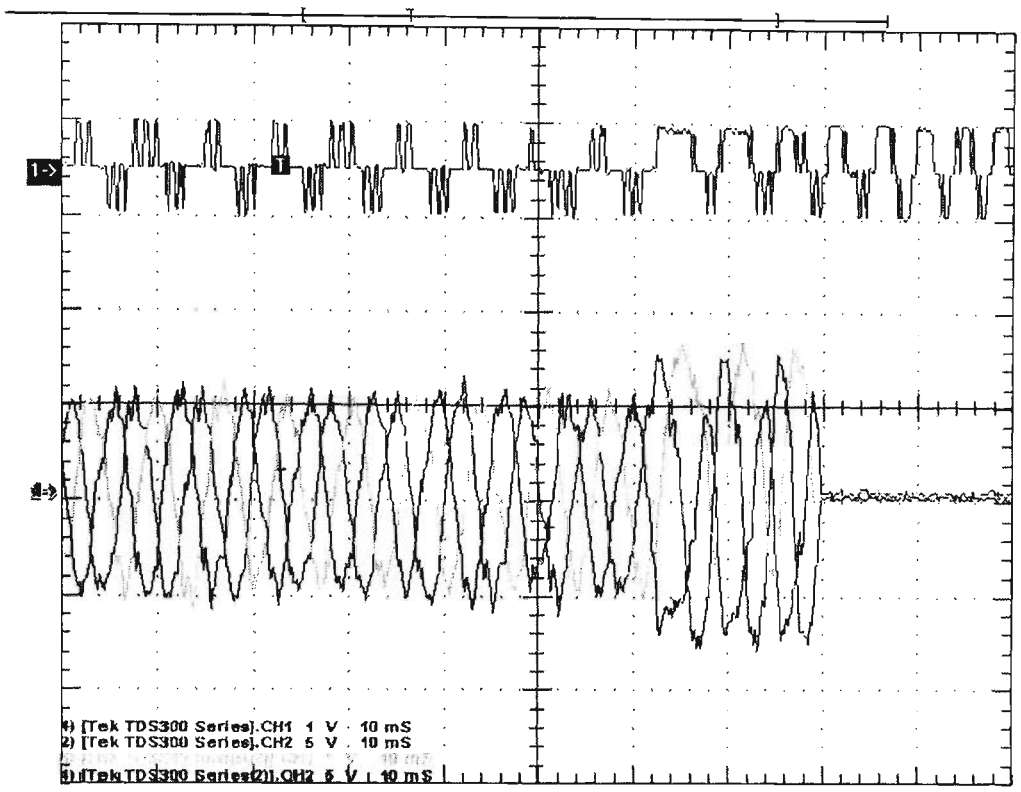


Figure 5.63 PWM signal and three phase fault to ground on location X (transient current) at 105Nm torque

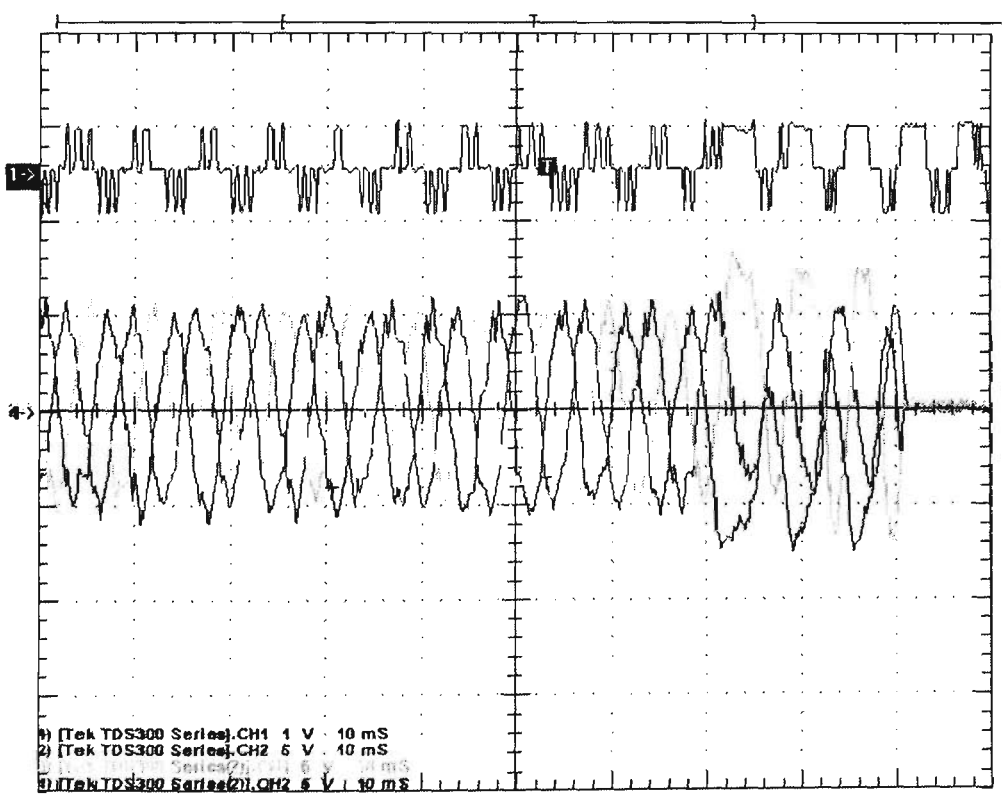


Figure 5.64 PWM signal and B-C phase fault to ground on location X (transient current) at 105Nm torque

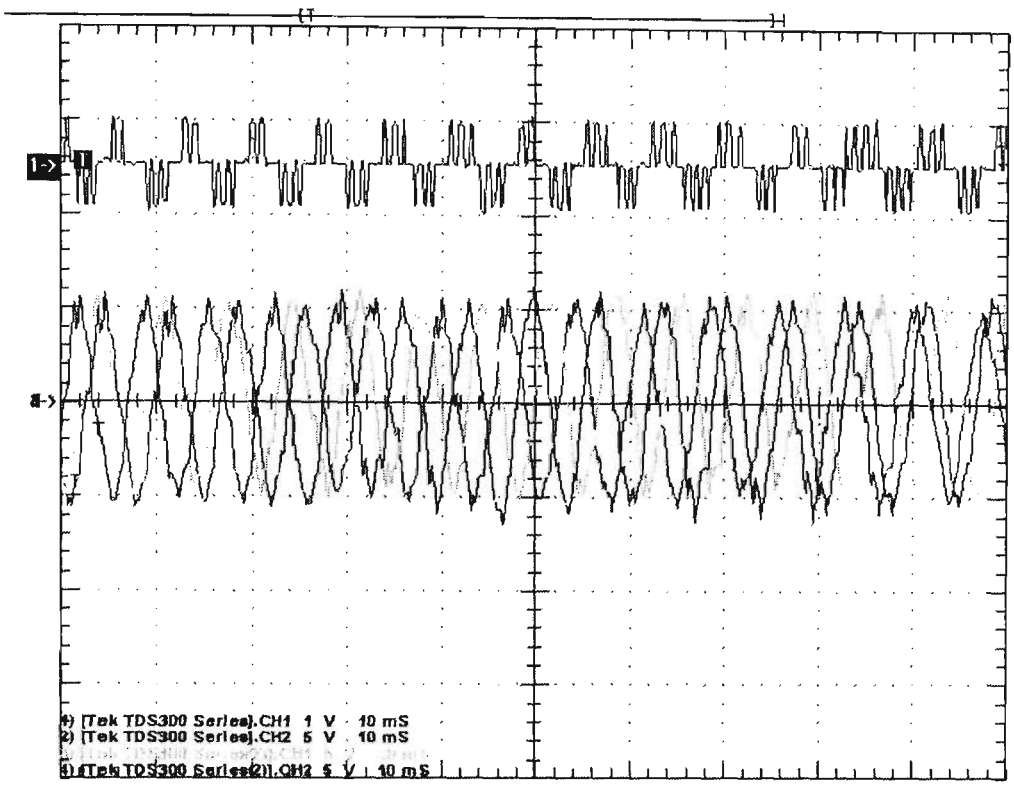


Figure 5.65 PWM signal and A phase fault to ground on location X (transient current) at 105Nm torque

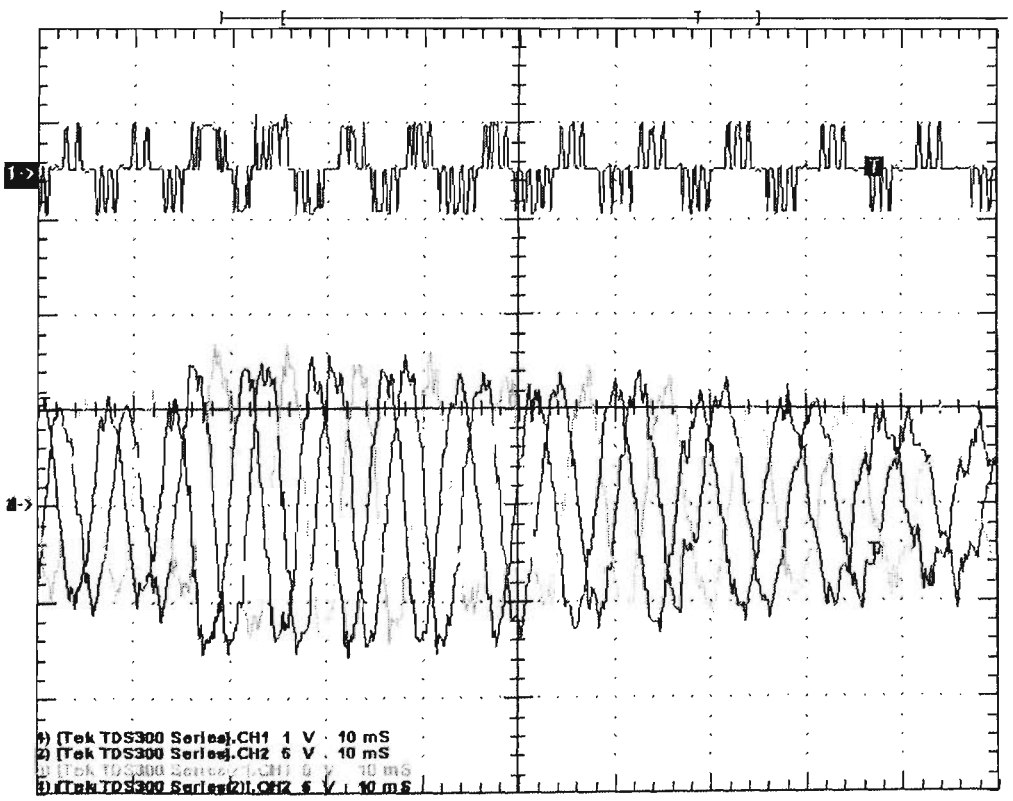


Figure 5.66 PWM signal and three phase fault to ground on location Y (transient current) at 105Nm torque

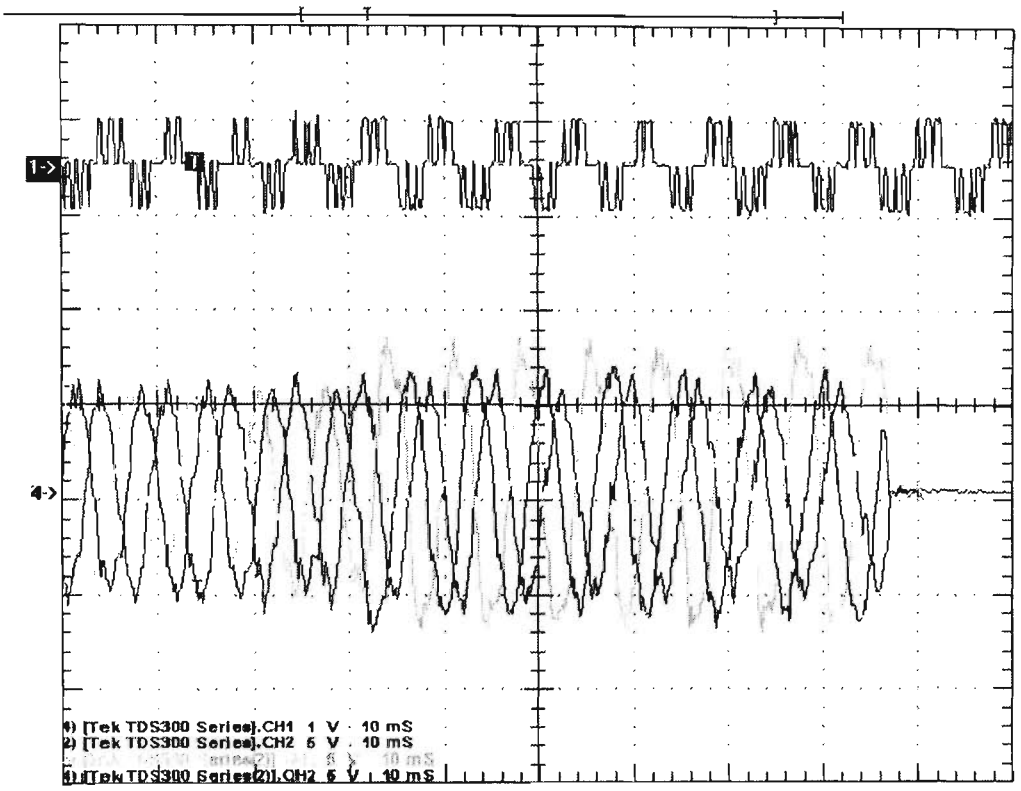


Figure 5.67 PWM signal and B-C phase fault to ground on location Y (transient current) at 105Nm torque

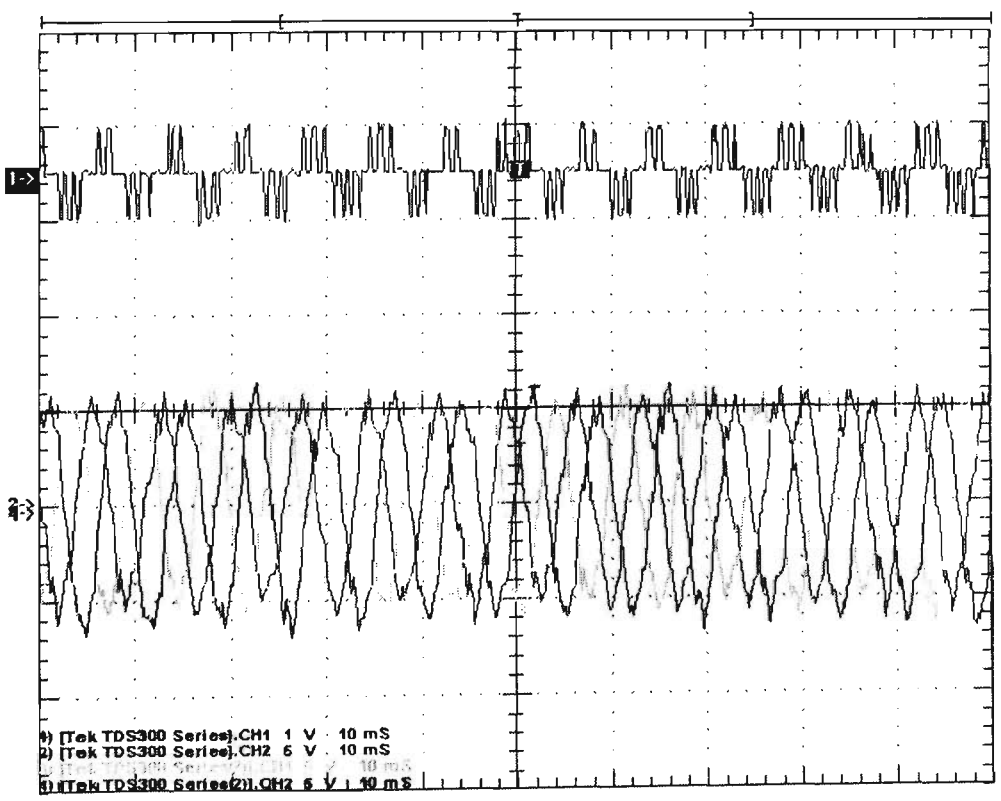


Figure 5.68 PWM signal and A phase fault to ground on location Y (transient voltage) at 105Nm torque

The generator is driven at 60Nm torque to test the transient fault analysis. The experiment results and comments are shown in Table 5.7 and the experiment current waveforms are also shown in Figures 5.69 to 5.71.

Table 5.7 The experiment results for transient fault analysis at 60Nm torque

Torque Location	Three Phase to earth fault	Phase B - C to earth fault	Phase A to earth fault
60Nm Location X	Figure 5.69 The circuit breaker trips at 0.05 second after the fault occurrence. The normal peak current is about 32A, the instantaneous fault current rises to two and half times of normal current.	Figure 5.70 The tripping time also is about 0.05 second after fault occurrence. The envelope of fault currents occur in phases B and C. The phase A still remains sinusoid but the form amplitude commences to increase.	Figure 5.71 The circuit breaker does not trip at any time after fault occurrence. The amplitudes of fault current are same as the normal. .

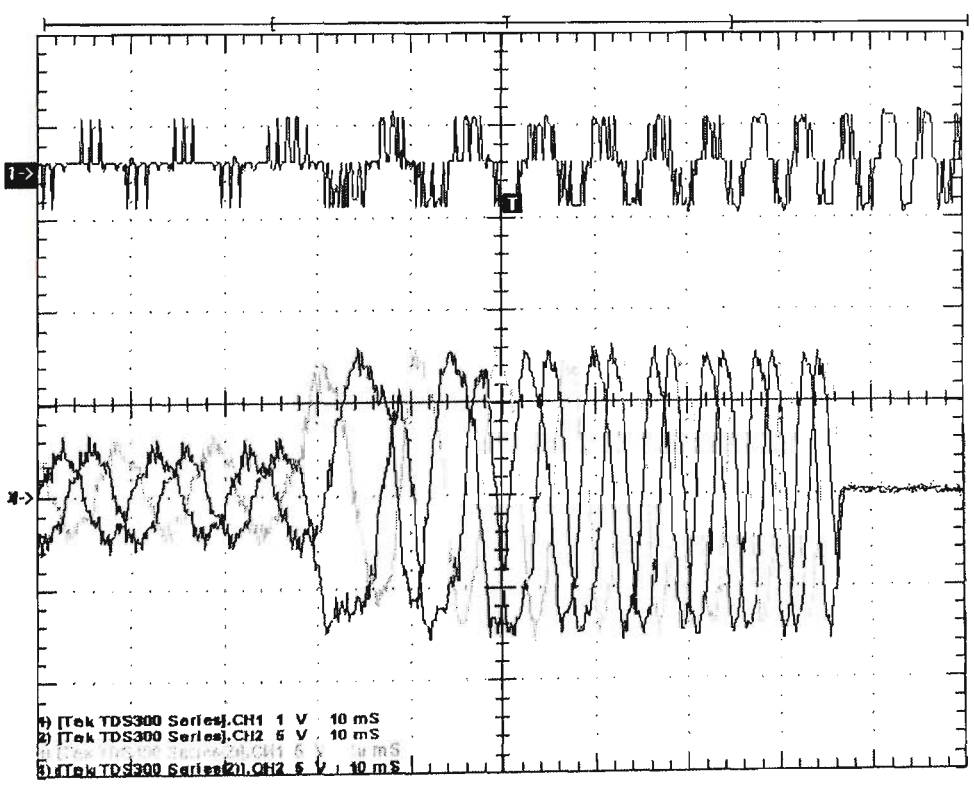


Figure 5.69 PWM signal and three phase fault to ground on location X (transient current)at 60Nm torque

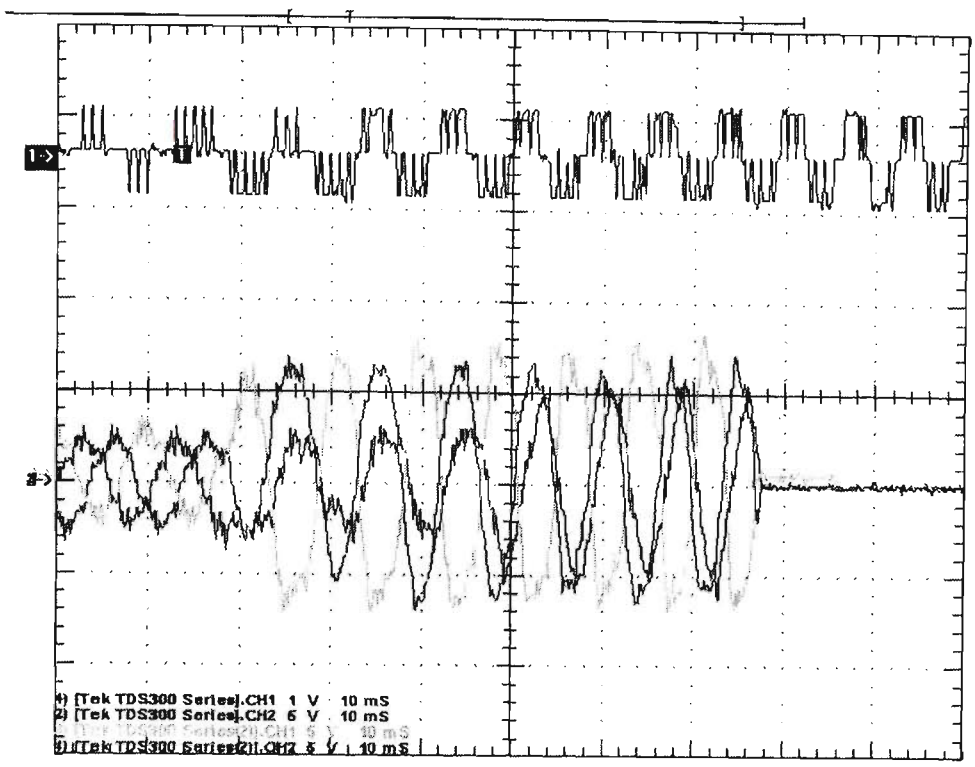


Figure 5.70 PWM signal and B-C phase fault to ground on location X (transient current) at 60Nm torque

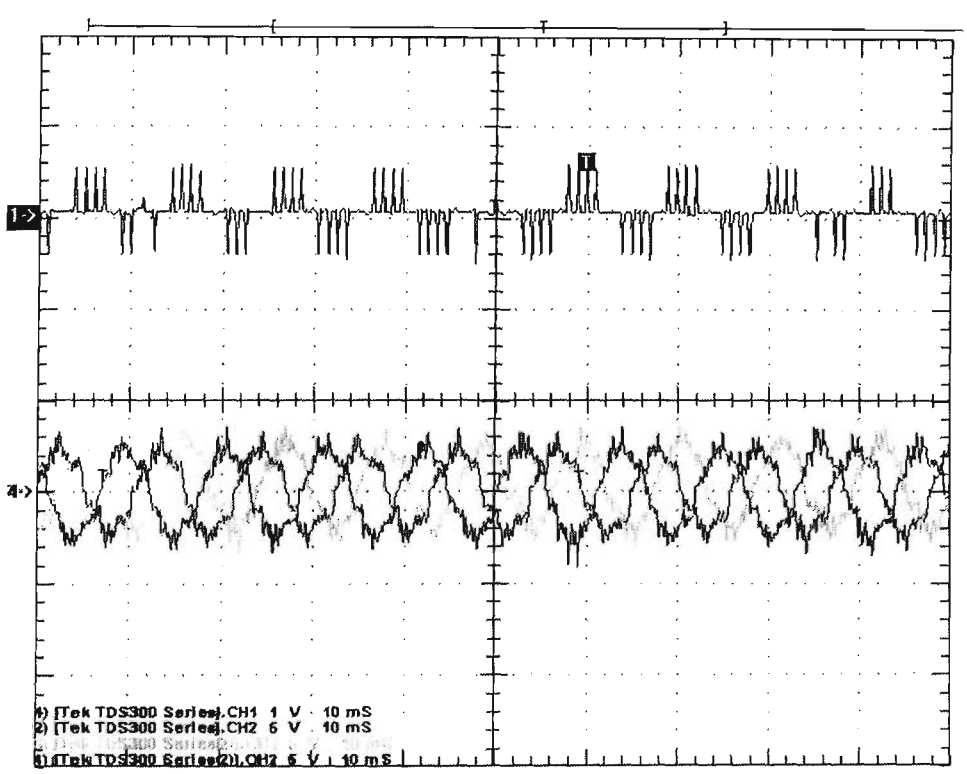


Figure 5.71 PWM signal and A phase fault to ground on location X (transient current) at 60Nm torque

5.6 Microprocessor based Protection Settings

5.6.1 Introduction

In this research, the microprocessor based circuit breaker selected is the ABB Isomax S4 160 *In* = 100A circuit breaker. It can be used as outgoing circuit breaker with overcurrent protection PR212/P microprocessor based release. From the operation and protection of three phase permanent magnet generator, which considers the rated current of generator (a function of its power and operating voltage) and the short circuit current at the point of application, which can be calculated in relation to transient reactance *X'' d* (%) the following formula is used.

$$I_{scc} = \frac{P \times 100}{U_e \times X''d} \qquad (5.1)$$

This normally has very small values compared to those of the power supply, and special attention must be paid to the overcurrent relays, which must be set to 2-3 times the rated current when it is used as the outgoing circuit breaker. Wherever the generator operates, it connects to the normal power supply network. The short circuit current that needs to be taken into consideration, when choosing the circuit breaker is the one due to the transformer since it has a higher value than that resulting from the generating faults [43-45]. The breaking capacity is determined in relation to the short circuit current for generator.

PR212/P device builds in microprocessor based electronic technology, the features ensure high reliability, precise tripping and immunity to the influence of ambient conditions. The power supply required for correct operation is supplied directly by the device's current transformer with one phase current larger than 15% of their rated currents. Even only one phase circuit powered on, one adjustment is required for all the phase and neutral. Tripping of

the device is simultaneous for all the poles of the circuit breaker with operating characteristic that are unaffected by ambient conditions.

5.6.2 Test response of microprocessor based circuit breaker

In this research, the simulation results were carried out with the parameters of power system such as the maximum load flow currents and transient currents under the fault operating conditions. In addition, the best fit settings are worked out for operating power system. From the last section, the simulation results show the normal maximum operating peak current is 62A, which is equal to 44A rms. The frequency varies with the input torque running the permanent magnet generator and depends on the output load. The range of frequency commences from 0Hz to maximum of about 120Hz.

PR212/P device provides protection functions against overloads (L), delayed short circuit (S), instantaneous short circuit (I) and earth fault (G). The functions of characteristics are shown in Appendix C.

Due to high frequency, the performance of the device is reclassified to take into account typical phenomena such as:

- Increased skin effect and inductive reactance in direct proportion to the frequency, which for the same copper cross-section area lead to decreased conductivity and increased inertia phenomena which causes overheating of the conductors or the copper circuit breaker parts through which current normally flows.

- The hysteresis cycle is lengthened and there are falls in the magnetic saturation and electromotive forces induced by the magnetic field which are generated for a given current.

In general these phenomena have repercussions on the behaviour of the copper parts both of the thermo magnetic device and the interrupting parts of the circuit breakers. It is necessary to reclassify their performance where the operating characteristics of the device are concerned. The PR212/P microprocessor based overcurrent device benefits from stable behaviour in the face of frequency variations, and these phenomena must be taken into account for the circuit breaker's copper parts and for the temperature increase due to frequency.

Figure 5.72 shows the characteristic of the frequencies via the tripping time. Current of 100A rms is injected to test the function. From the figure shows, two curves are stabile from 40Hz to 150 Hz. The tripping times are similar.

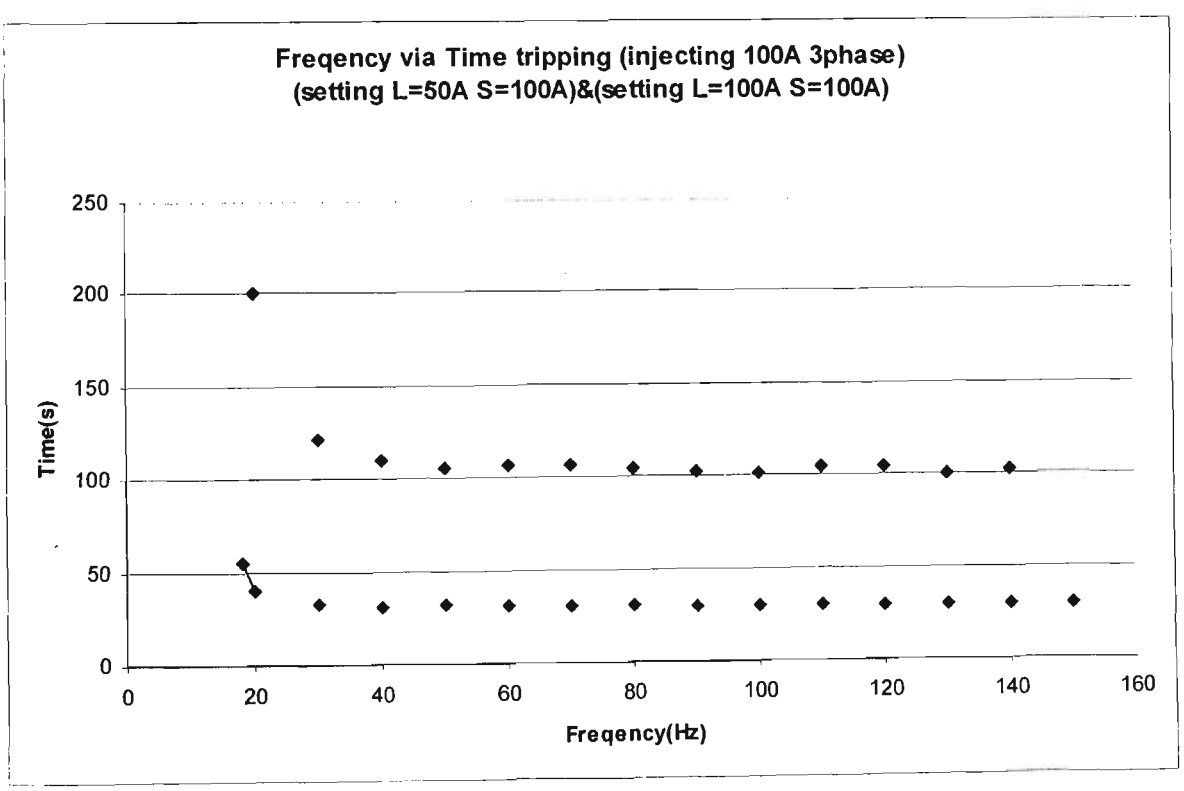


Figure 5.72 Test response of microprocessor based circuit breaker

From the experimental system, the best fit settings of microprocessor based protection are shown in Table 5.6.

Table 5.8 Table of settings for microprocessor based circuit breaker

Overloads L	Delayed short circuit S	Instantaneous short circuit I	Earth fault G
I1 = 0.65 x In	I2 = 1 x In	I3 = 1.5 x In	I4 = 0.2 x In
T1 = 3 S	T2 = 0.05 S	InN = 50% In	T4 = 0.1 S

It is possible to set the adjustment parameters of the protection functions directly from the front (dipswitch positioned on MAN). In case of any anomalies in remote parameterisation, the protection automatically uses the set of parameters set manually on the front of the circuit-breaker. This allows the adjustment parameters to be set even with the circuit-breaker open

Chapter 6 CONCLUSIONS AND RECOMMENDATIONS

6.1 Introduction

In this thesis, research has been conducted on the simulation and implementation of the settings of the microprocessor based circuit breaker for ocean power energy system. The project has utilised MATLAB simulink and Power System Blockset software package to simulate the output of generator and transport the wave power energy to connect the power grid under different operating condition. The load flow and transient fault analysis are conducted. The simulation output also produce the actual current to apply into the microprocessor based circuit breaker and test its function. This research project has been completed successfully.

This section offers the main conclusion derived from the observed results. Further, it also points out the direction of future work out to enhance this research.

6.2 Transient fault studies

Simulation transient fault studies were carried out in Chapter 3 to find out transient fault of the wave power system under various operating condition, such as different changing load. This analysis utilises the same MATLAB simulink program addition the Power System Blockset. The faults can be applied in different locations as shown in Figures 5.1 and 5.20. The simulation results are described in Chapter 5. When three phase fault occur at location X the magnitude of the transient current may suddenly become high. Voltages collapse to zero when connected to ground. If the fault occurs at location Y the magnitude of transient voltage

and current may be slightly different. The zero resistance faults are associated with the transmission line impedance.

In case of phase A to ground fault, the voltage of phase A collapses to zero. The other phases increase. The transient currents are still kept at constant level.

6.3 Load flow studies

Simulation load flow studies were carried out in Chapter 3 to find out the operation of different dynamic loading produced by the 30kW inverter such as fully charged or rapid charging condition and supplies different level of the wave power energy to generate the active and reactive power. From the simulation results as shown in Chapter 5 (Figures 5.44 to 5.45) it can be seen that the maximum speed and torque of permanent magnet generator can not produce the maximum of active and reactive power due to the design of power system which has a overvoltage limiting resistors in parallel to the generator. In no load condition, the generator drives it to almost at the rated power. The outputs of currents absorbed by the overvoltage limiting resistors then can limit output voltages as shown in Figure 5.40. If full load condition occurs, the total of impedance becomes extremely low and the output of current of generator is kept at the maximum value. Therefore, the frequency and voltage stall at a low level at about 70Hz and 290V peak voltage as shown in Figure 5.44. The active and reactive powers are 20kW and 13kVAR respectively as shown in Figure 5.58.

6.4 The function of microprocessor based circuit breaker

This experimental system is implemented and the function of microprocessor based circuit breaker and the best setting is determined. In this case, simulation results show the maximum current and voltage could be about 230V and 62A when running at the peak torque. This setting cannot fulfil all criteria under the different operating conditions. Since the output of generator is dependant on the ocean wave energy, and this energy is unpredictable. If the generator runs low speed then the power, current and voltage also become low. The

microprocessor based circuit breaker only can meet the basic requirement for most situations. The alterative protection will apply in this power system such as the overvoltage protection, which can monitor generator voltage output, the setting of overvoltage protection is set at 270V. It can protect the single phase fault issue and also compensate the system needs.

6.5 The 50kVA IGBT power converter

The power system experiment needs a power converter, which can produce the high currents and variance frequency in order to implement the microprocessor based circuit breaker function. The concept of design of converter is based on information from 10kVA converter, which was manufactured at Monash University. A IGBT power converter has been build for this research.

6.6 Recommendations for Future research

This research project implemented shows promising potential for a protecting power system. The ocean wave power system could include further enhancements as shown below:

1. Improvement of the protection system specifically at lower frequency needs further investigation.

2. The overvoltage limiting resistors absorbs high energy in order to limit the level of output voltage. Further development like the dynamic loading for control the output power of generator would be useful.

3. Development of the electrical control design specifically of the output load monitoring will enhance the work.

Reference

[1] G.D. Rockefeller, "Fault Protection with a Digital Computer", IEEE Trans. On Power Apparatus and Systems, Vol.88, No.4, pp.438-461, April 1969.

[2] J. Carr and R.V. Jackson, "Ferquency Domain Analysis Applied to Digital Transmission Line Protection", IEEE Trans. On Power Apparatus and Systems, Vol.94, No.4, pp.1157-1166, July/August 1975.

[3] M.S. Sachdev and M.A. Baribeau, "A Digital Computer Relay for Impedance Protection of Transmission Lines", Trans. of the Engineering and Operating Division, Canadian Electrical Association, Vol.18, Part3, No. 79-158, pp.1-5, 1980.

[4] M.S. Sachdev and M.A. Baribeau, "A New Algorthm for Digital Impedance Relay", IEEE Trans. On Power Apparatus and Systems, Vol.98, No.6, pp.2232-2240, December 1980.

[5] M.S. Sachdev and D.W. Wind, "An-Line Digital Computer Approach for Generator Differential Protection", Trans. of the Engineering and Operating Division, Canadian Electrical Association, No.73-149, Vol. 12, No. 3, pp.1-6, 1973.

[6] M.S. Sachdev and D.W. Wind, "Generator Differential Protection Using a Hybrid Computer", IEEE Trans. On Power Apparatus and Systems, Vol. 92, No.6, pp.2063-2072, November/December 1973.

[7] O.P. Malik, P.K. Dash and G.S. Hope, "Digital Protection of a Power Transformer", IEEE Publication 76CH 1075-1 PWR, Paper No.A76 191-7, IEEE PES Winter Meeting, New York, pp.1-7, January 1976.

[8] E.O. Schweitzer, R.R. Larson and A.J. Flechsig, Jr., "An Efficient Inrush Current Detection Algorithm for Digital Computer Relay Protectionof Transformer", IEEE Publication No. 77CH 1193-2 PWR, Paper No. A77 510-1, IEEE PES Summer Power Meeting, Mexico, pp.1-5, July 1977.

[9] R.R. Larson, A.J. Flechsig and E.O. Schweitzer, "The Design and Test of a Digital Relay for Transformer Protection", Ibid, pp.795-804, 1980.

[10] A.O. Phadke and J.S. Throp, "A Microprocessor Based Three-phase Transformer Differential Relay", Ibid, pp.426-432, 1982.

[11] A.O. Phadke and J.S. Thorp, "A New Computer-based Flux Restrained Current Differential Relay for Power Transformer Protection", IEEE Trans. On Power Apparatus and Systems, Vol.102, No.11, pp.3624-3629, November 1983.

[12] M.A. Rahman, B. Jeyasurya and A. Gangopadhyay, "Digital Differnetial Protection of Power Transformers Based on Walsh Functions", Transactions of the Engineering and Operating Division, Vol.24, Part 3, Paper No. 85-SP-149, pp.1-17. 1985.

[13] C.A. Kramer and W.A. Elmore, "Flexible Inverse Overcurrent Relaying Using a Microprocessor", IEEE Trans. on Power Delivery, Vol.5, No.2, pp.915-921, April 1990.

[14] T.S. Sidhu, H.C. Wood and M. Nagpal, "Design, Implementation and Testing of a Microprocessor Based High-Speed Relay for Detecting Transformer Winding Faults", Transactions on Power Delivery, Vol.7, No.1, pp. 108-117, January 1992.

[15] A.Kalam, A. Spicer, R. Coulter, A. Klebanowski, C. Biasizzo and H. McDonald, "Power System Protection", ESAA publication, 2000.

[16] E.E. Conner E.C. Wentz. And D.W. Allen, "Methods for Estimating Transient Performance of Practical Transformer for Relaying", IEEE Tran., PES-94, (1), pp.116-122, 1975.

[17] IEEE Power System Relaying Committee: "Transient Response of Current Transformer", IEEE Trans., PAS-96, (6), pp.1809-1814, 1977.

[18] L.J. Powell, "Current Transformer Burden and Saturation", IEEE Trans. IA-15, (3), pp.294-302, 1980.

[19] A.Wright, and C. Christopoulos, "Electrical Power System Protection", (Chapman & Hall), 1994.

[20] Y.C. Kang, S.H. Kang, J.K. Park, A.T. Johns and R.K. Aggarwal, "Development and Hardware Implementation of a Compensating Algorithm for the Secondary Current of Current Transformers", IEE Proc-Electr. Power Appl., Vol.143, No.1, January 1996.

[21] Y.C. Kang, S.H. Kang, J.K. Park, A.T. Johns and R.K. Aggarwal, "An Algorithm for Compensating Secondary Currents of Current Transformers", IEEE Trans. on Power Delivery, Vol.12, No.1, January 1997.

[22] J.R. Marti, L.R. Linares and H.W. Dommel, "Current Transformers and Coupling-Capactior Voltage Transformers in Real Time Simulations", IEEE Trans. on Power Delivery, Vol.12, No.1, January 1997.

[23] N. Locci and C. Muscas, "A Digital Compensation Method for Improving Current Transformer Accuracy", IEEE Trans on Power Delivery, Vol.15, No.4, October 2000.

[24] SIMLINK Dynamic System Simulation Software. Natick, MA The Mathworks Inc., 1994.

[25] G. Rogers and J. Chow, "Hands on teaching of power system dynamics", IEEE Computing Application Power, pp.12-16, Jan 1995.

[26] A. Bosin, "Electrical Power System Modelling and Simulation using Simulink", IEE 1998

[27] L. Hoang, S. Gilbert, "MATLAB/Simulink and PSpice as Modellong Tools for Power System and Power Electronics", IEEE 2000, pp.766-767.

[28] I. Ismail, S. Nadarajah, "Simulation of Power System and Machine of an Industrial Plant using the MATLAB/Simulink Power System Blockset", IEEE 2002, pp.293-296.

[29] L. Hoang, "Modelling and Simulation of Electrical Drives using MATLAB/Simulink and Power System Blockset", IEEE 2001, pp.1603-1611.

[30] L. Hoang, G. Sybille, P. Brunelle, "Theory and Application of Power system Blockset a MATLAB/Simulink Based Simulation Tools for Power system", IEEE 2000, pp.774-779.

[31] B. Mork, "Parameter for Modeling Transmission Lines and Transformer in Transient Simulations" IEEE 2001, pp.716-717.

[32] J. Larsson, "An Expert System for Frequency Response Analysis", IEEE 1994, pp.52-57.

[33] G. Castino, A. Dubhashi, S. Clemente and B. Pelly, "Protecting IGBT's against Short Circuit", International Rectifiers Application Note AN-984, 1991.

[34] A.J. Hefner, "An Investigation of the Drive Circuit Requirements for the Power Insulated Gate Bipolar Transistor", IEEE PESC-90, pp.126-137.

[35] S. Clemente. A. Dubhashi and B. Pelly, "IGBT Characteristics and Applications", International Rectifiers Application Note AN-983, 1991.

[36] R. Chokhawala, J. Catt, B. Pelly, "Gate Drive Considerations for IGBT Modules", IEEE IAS-92, pp.1186-1195.

[37] A. Petterteig, J. Lode, T.M. Undeland, "IGBT Turn-off Losses for Hard Switching and with Capacitive Snubbers", IEEE IAS-91, pp.1501-1507.

[38] N. LaWhite and M. Ilic, "Vector space decomposition of reactive power for periodic nonsinusoidal signals," IEEE Trans. Circuits System. I, Vol.44, April 1997.

[39] The Mathwork, Inc., http:// www.mathwork.com.

[40] J.J.R. Oilveria, H.M. Barros, M. Roitman, S.E. Santo, "Interfacing Digital Real Time Simulator to External Devices", ICDS'99, Vasteras, Sweden, May 25-28, 1999.

[41] T. Hiyama, A. Ueno, "Development of Real Time Power System Simulator in MATLAB/Simulink Environment", IEEE 2000, pp.2096-2100.

[42] D.G. Holmes, A. M^{c} Iver and A. Kotsopoulos, "The Effect of Gate Drive on IGBT Switching Characteristic", Proceeding of AUPEC'94, Vol.1, pp.167-122.

[43] M. Bayrak, "A New Power-Based Digital Algorithm for Generator Protection" IEEE 2000, pp.49-52.

[44] K. Al-Hadded, L.A.Dessaint, "Investigation on Power Quality Issues in Industrial Distribution System using the Power System Blockset Simulation Software", IEEE 1998, pp.114-121.

Reference

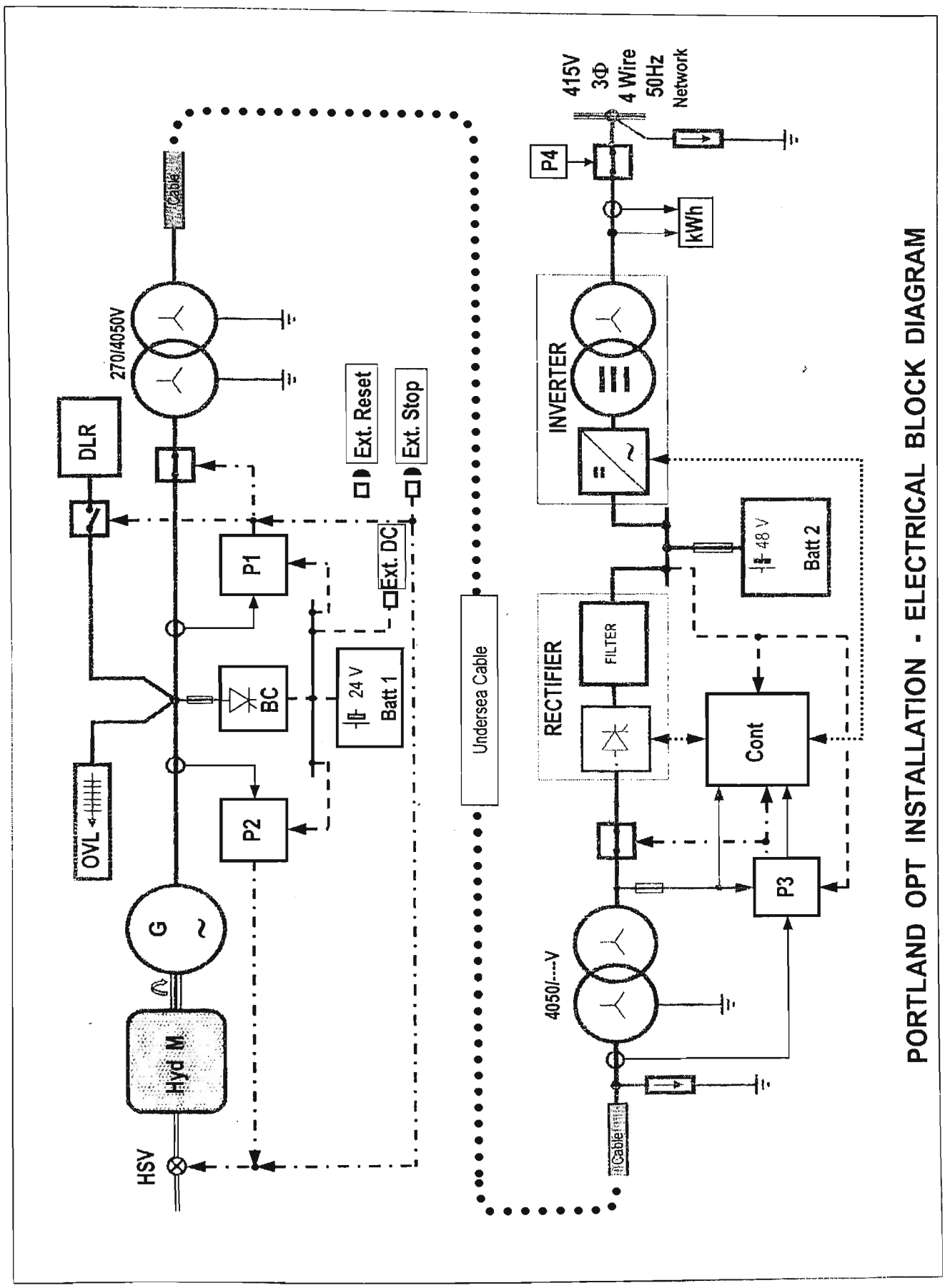
[45] M.A.Rahman, B. So, M.R. Zaman and M.A.Hoque, "Testing of Algorithms for a Stand-Alone Digital Relay for Power Transformers", IEEE Trans. Power Delivery, Vol.13, No.2, April 1998.

APPENDIX A

Electrical Block Diagram

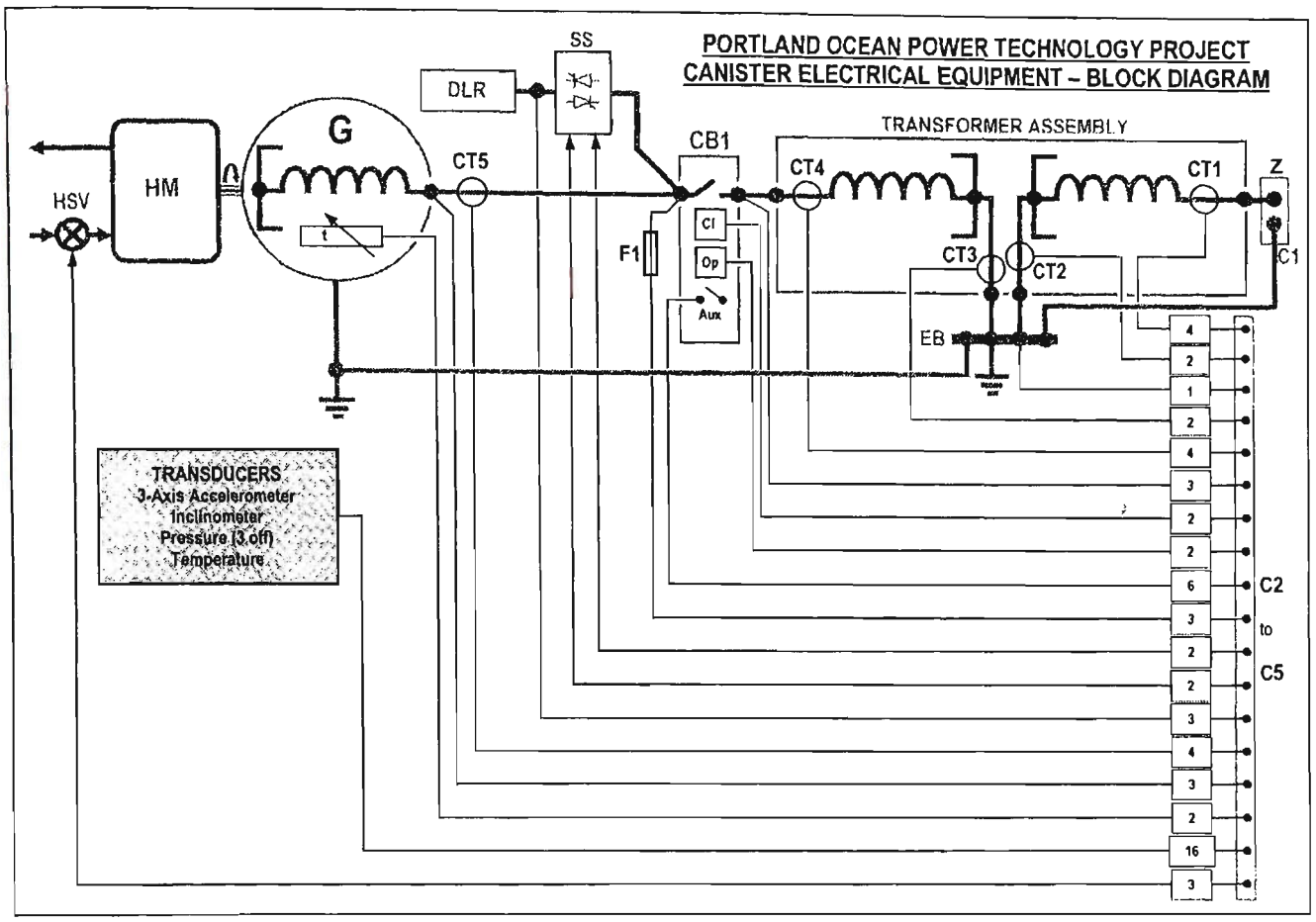
Appendix A shows the Portland Ocean Power Technology Project relative electrical diagram for reference.

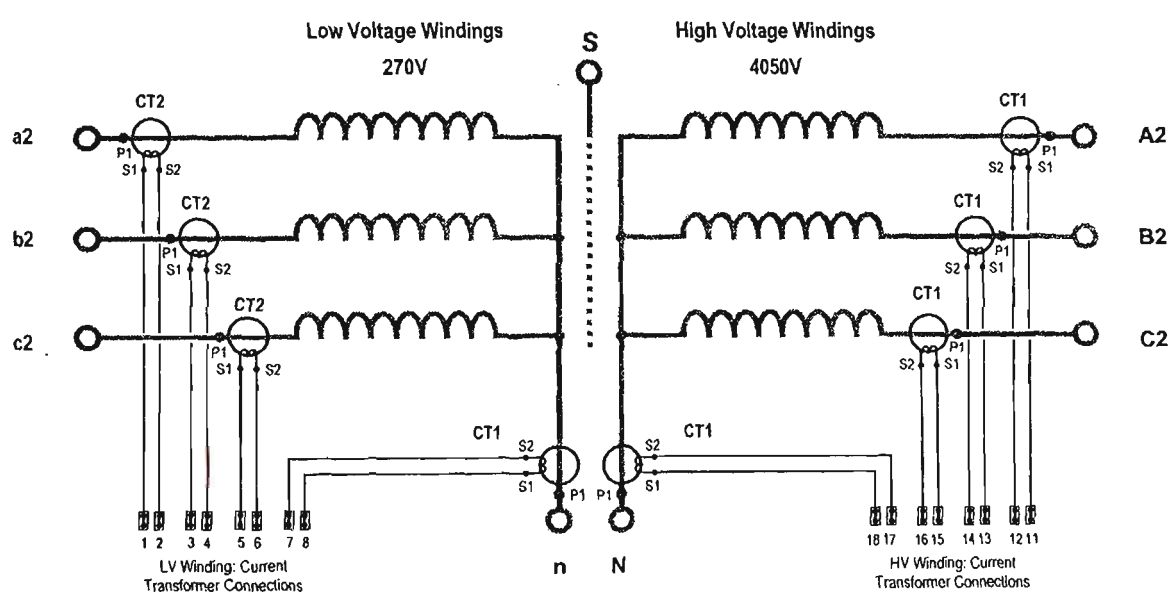
1. Portland OPT Electrical Block Diagram shows in Appendix A1
2. Canister Electrical Equipment Block Diagram
3. On-buoy Power Transformer Schematic Diagram
4. RTU Application on shore Arrangement
5. RTU Application at Buoy Arrangement



Portland OPT Electrical Block Diagram shows in Appendix A1

Appendix

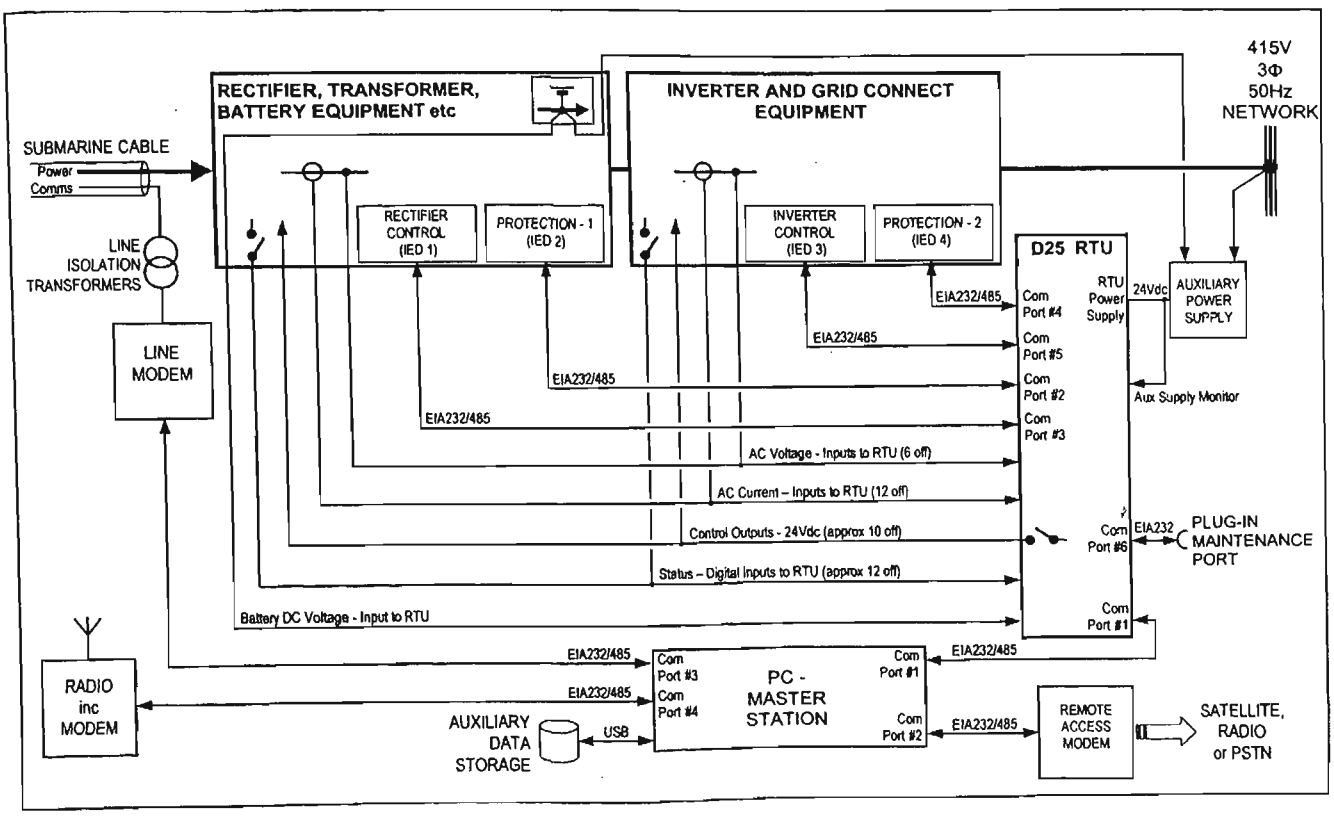




Current Transformer Specifications: CT1 - 5/1 Ratio, Protection Class, 0.025PL50R0.5 at 50Hz (to Australian Standard AS1675-1986)
CT2 - 75/1 Ratio, Protection Class, 0.025PL50R0.5 at 50Hz (to Australian Standard AS1675-1986)

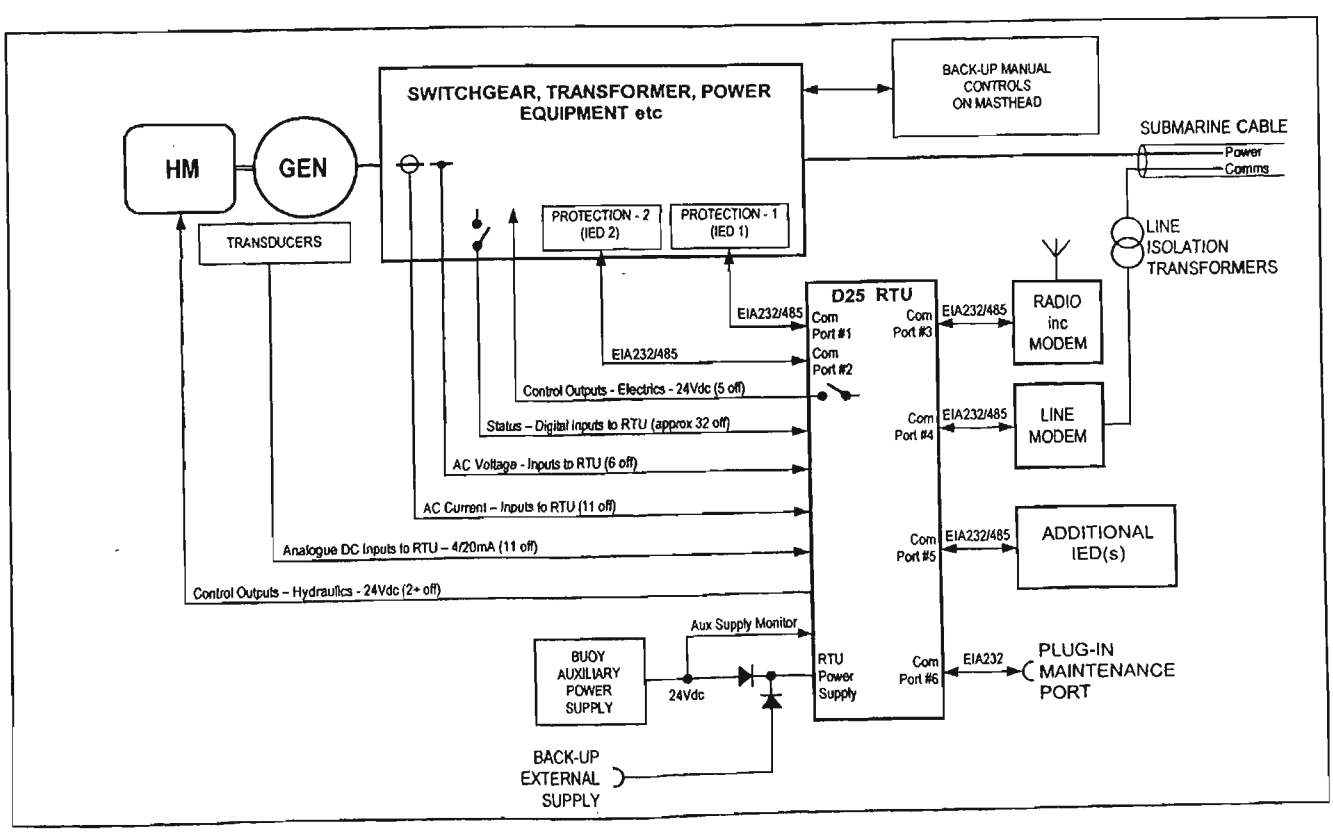
PORTLAND OCEAN POWER TECHNOLOGY PROJECT: ON-BUOY POWER TRANSFORMER SCHEMATIC DIAGRAM

Appendix



PORTLAND OCEAN POWER TECHNOLOGY PROJECT

RTU APPLICATION ON SHORE – OPTION S1: BASIC ARRANGEMENT



PORTLAND OCEAN POWER TECHNOLOGY PROJECT

RTU APPLICATION AT BUOY – OPTION B1: BASIC ARRANGEMENT

APPENDIX B

Circuit Schematics

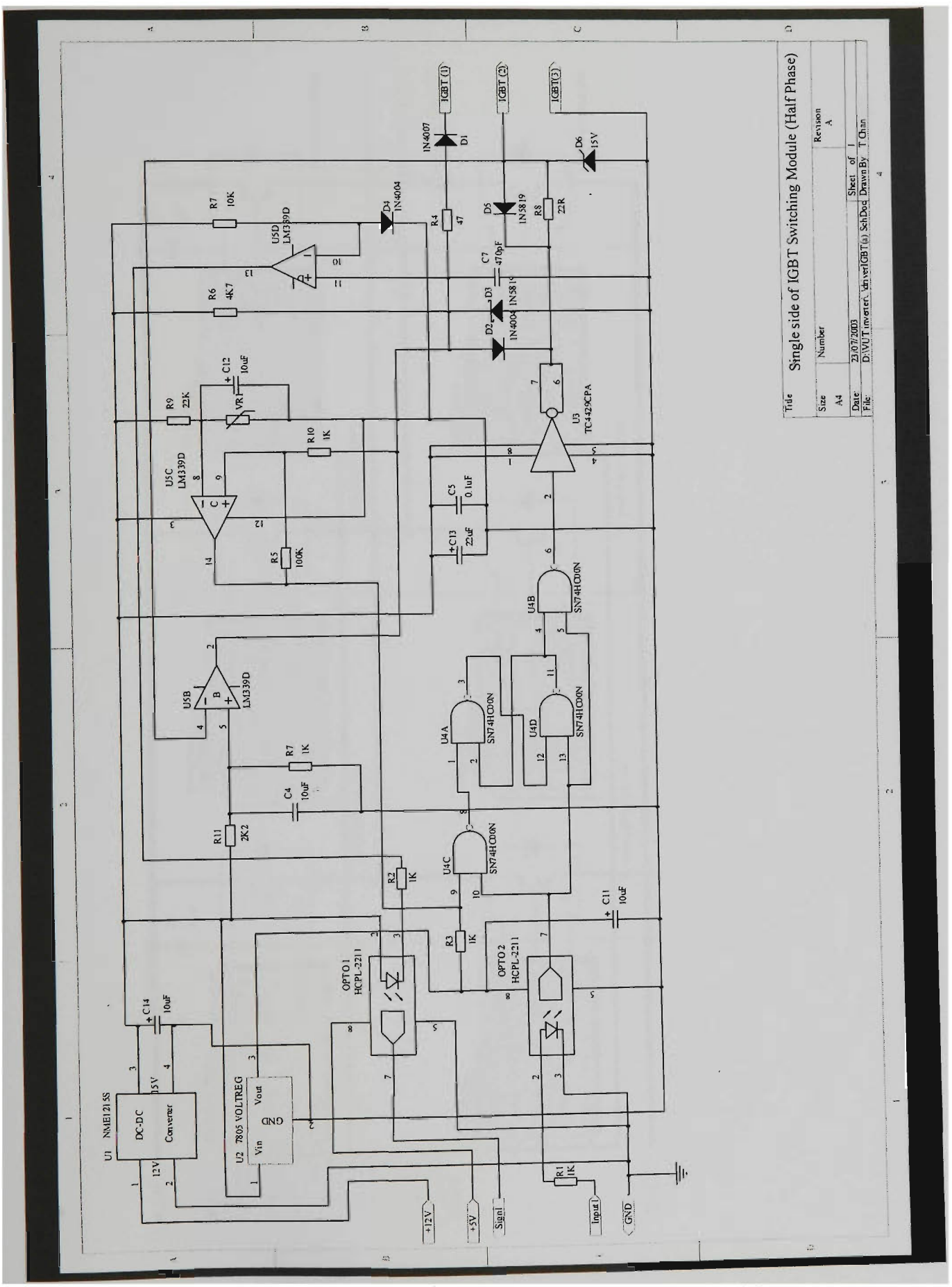
50KVA IGBT DC to AC power converter

1. The detail of IGBT circuit diagram of the power converter shows in Appendix B1,
2. The hardware configuration shows in Appendix B2.
3. The hardware implementation of the crossover delay the power driver circuit diagram shows in Appendix B3.

Isolation AC and DC current transducer

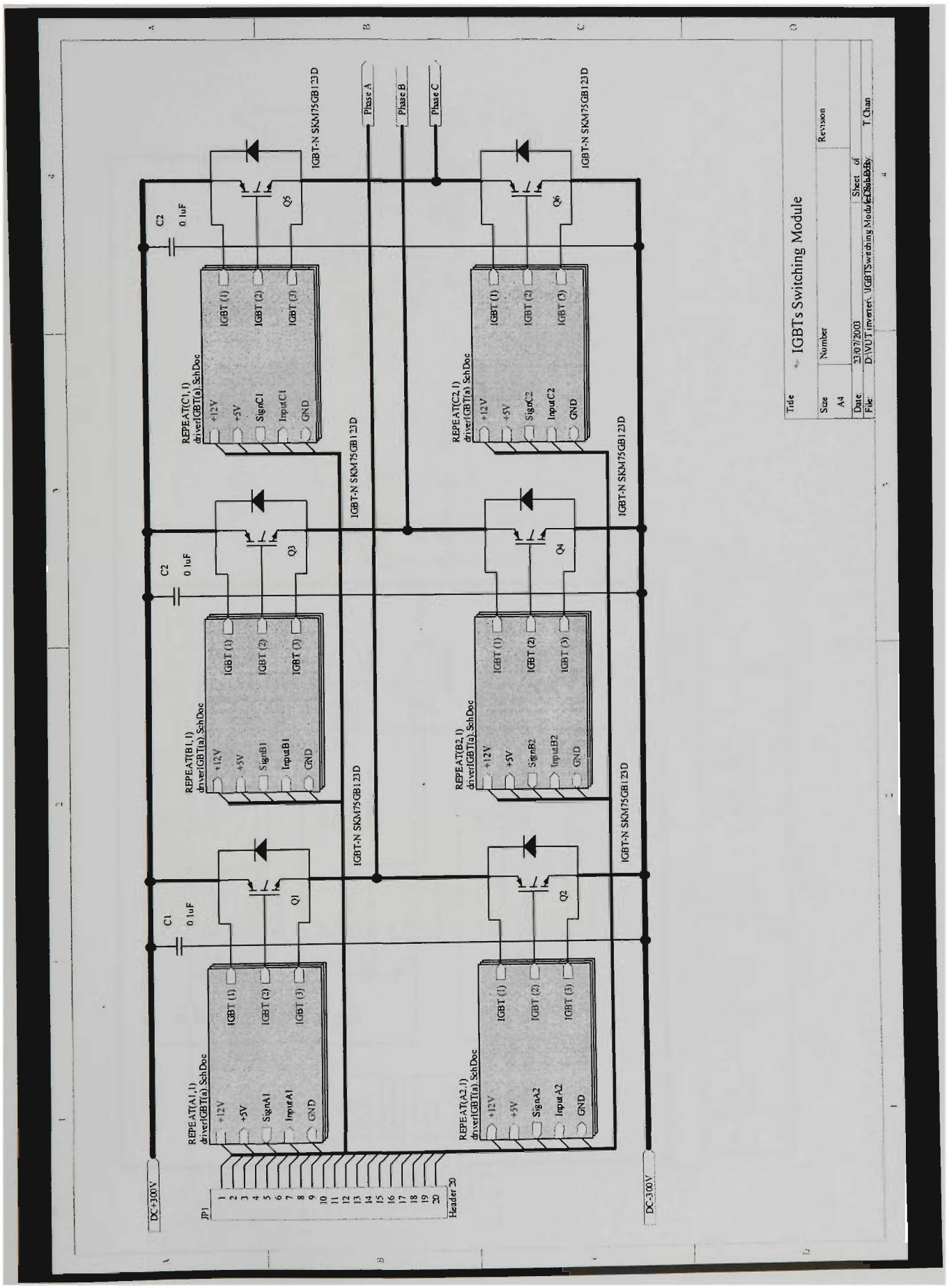
4. The circuit schematic diagram of three phase current transducer shows in Appendix B4.

Appendix



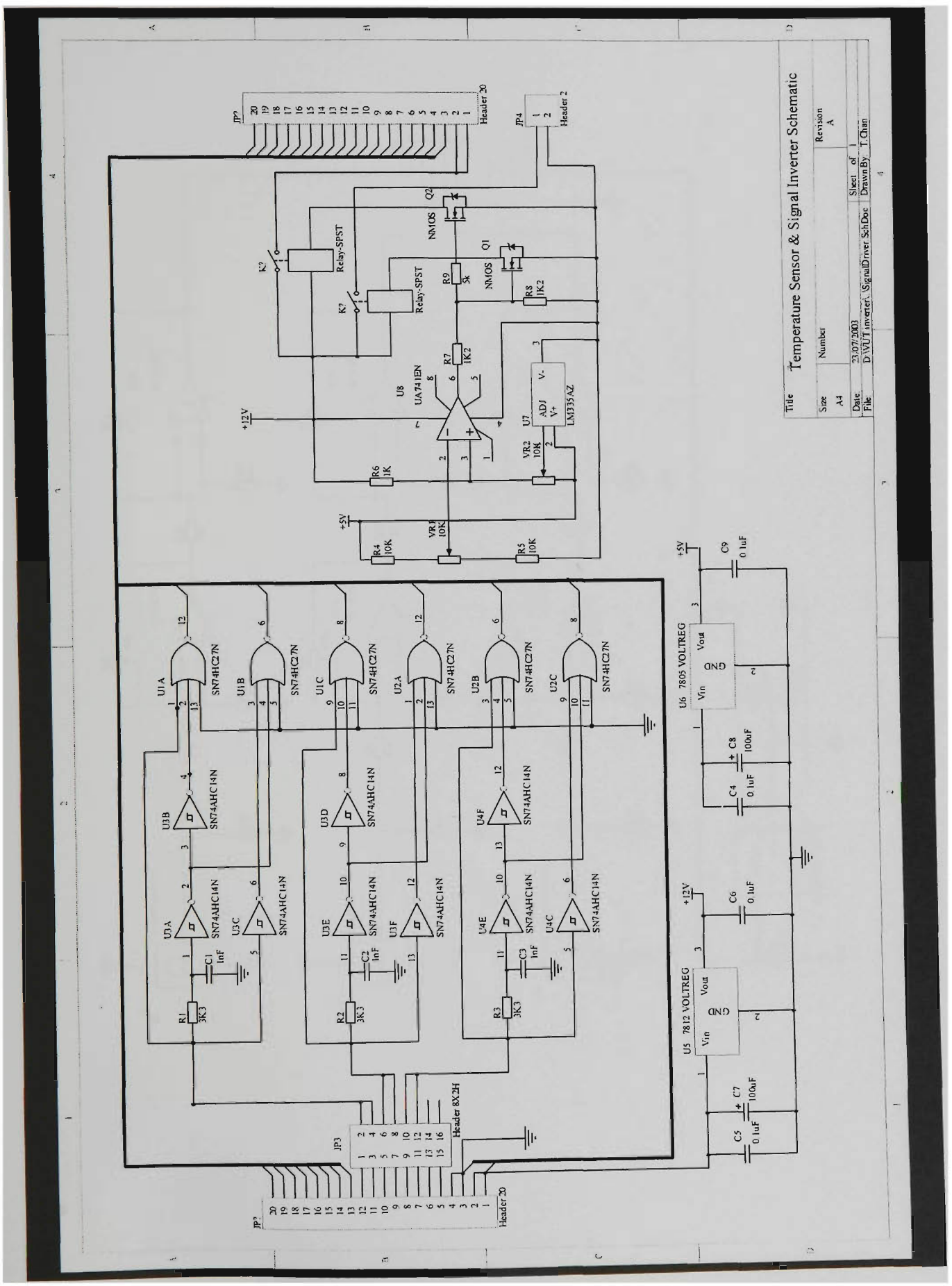
The detail of circuit diagram of the power converter shows in Appendix B1.

Appendix



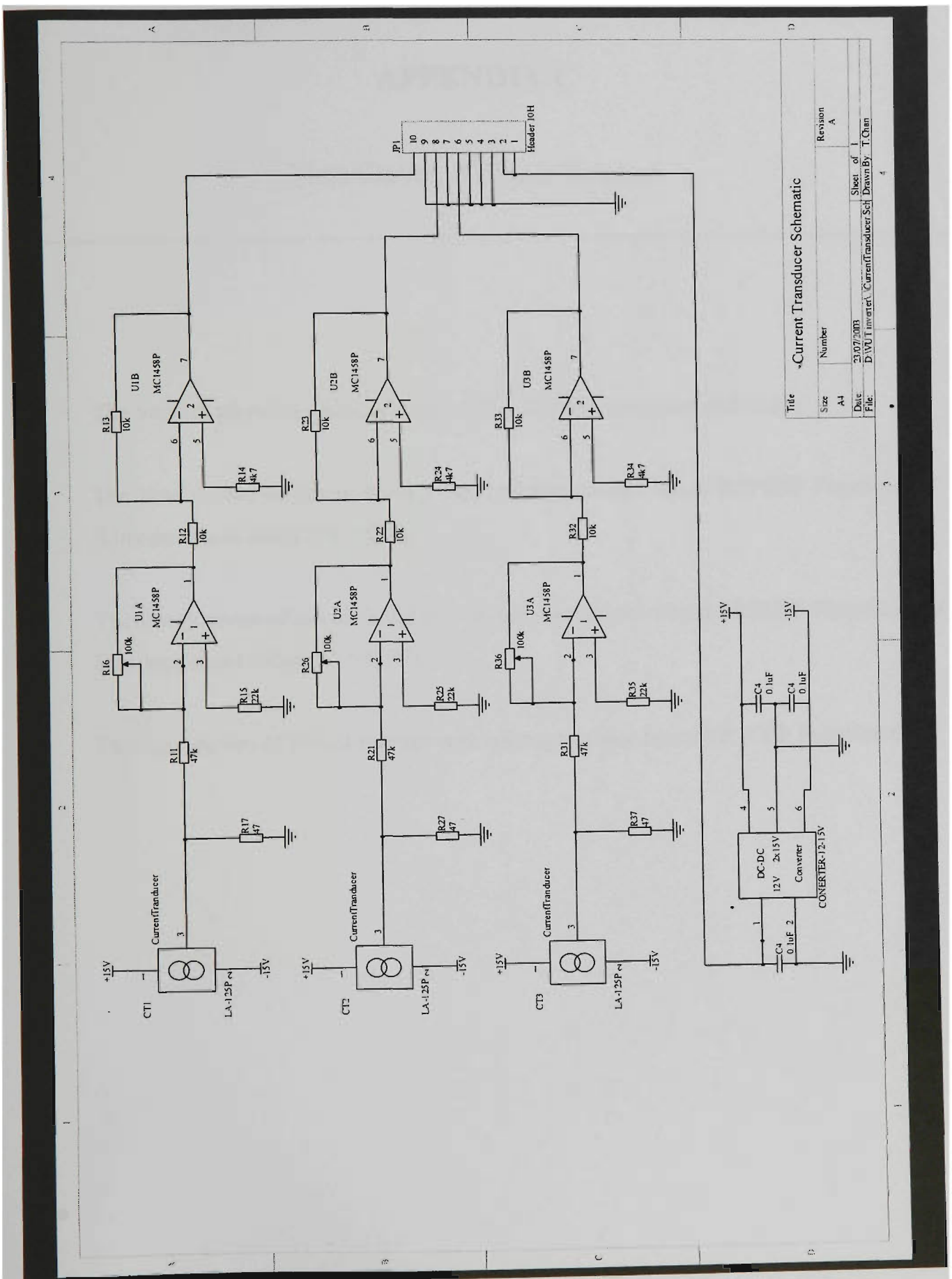
The hardware configuration shows in Appendix B2.

Appendix



The hardware configuration shows in Appendix B3.

Appendix



The circuit schematic diagram of three phase current transducer shows in Appendix B4.

APPENDIX C

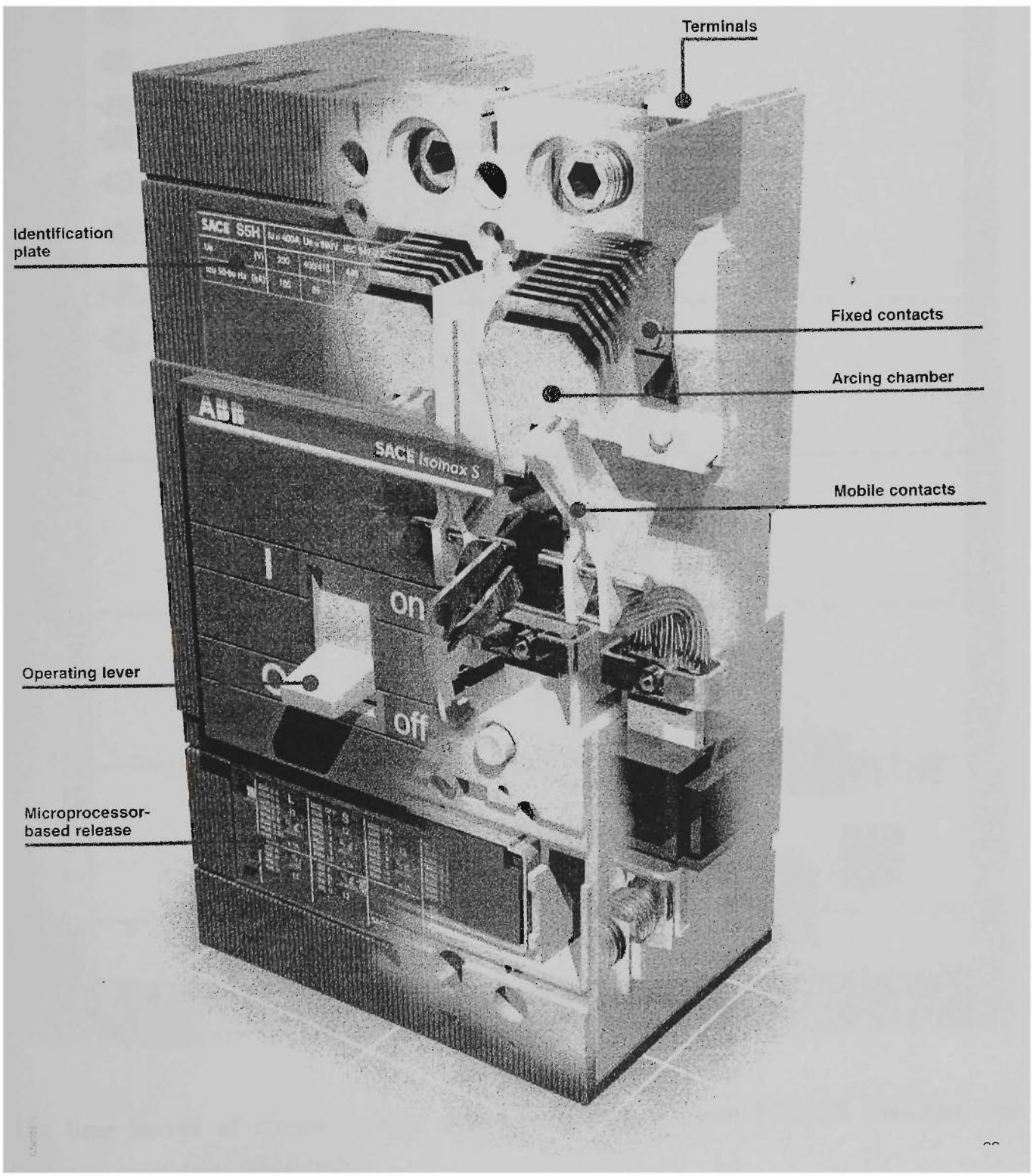
Time Curves of Circuit Breakers

1. The view of physical configuration microprocessor based circuit breaker

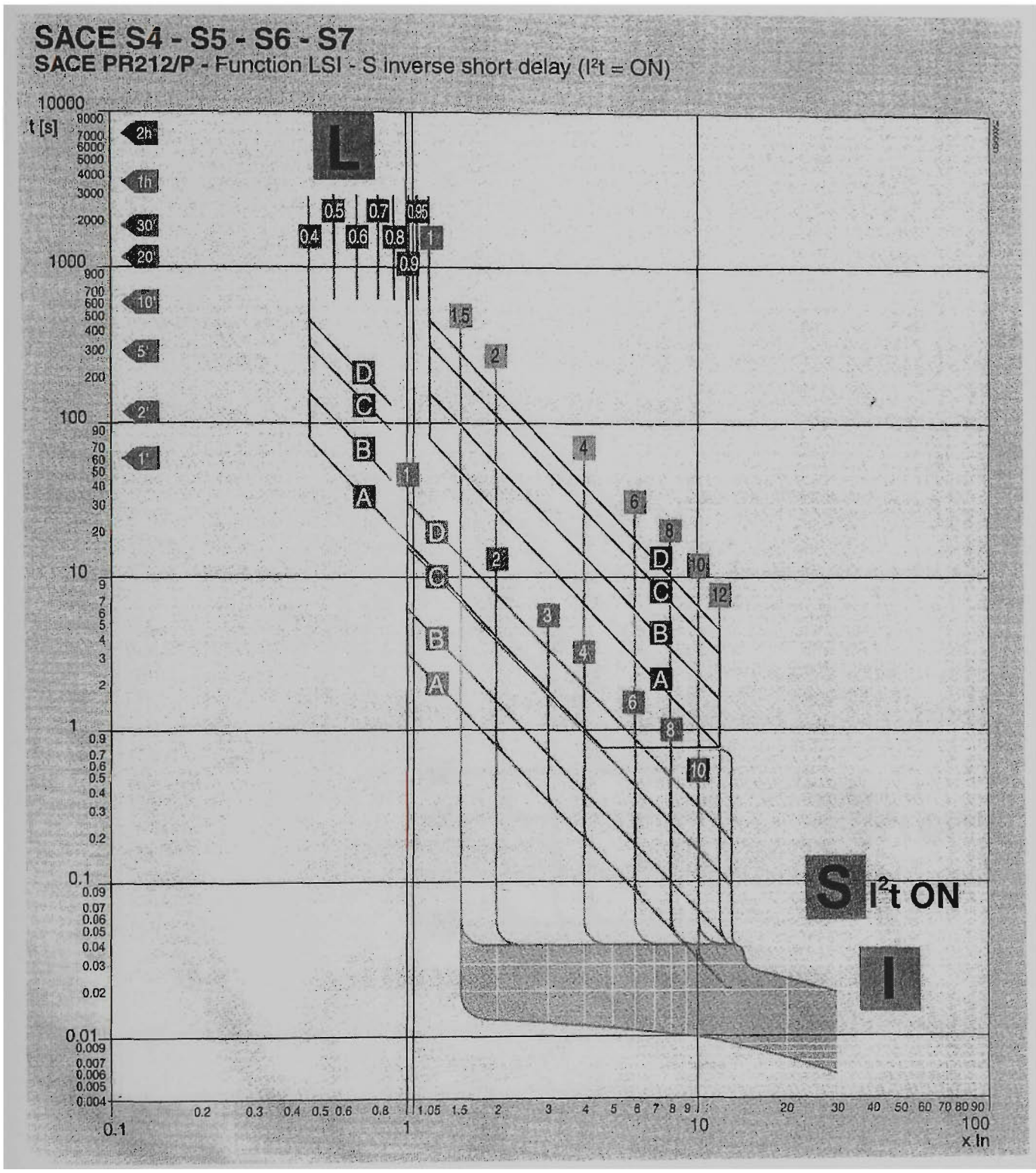
2. The time curves of circuit breaker with microprocessor based PR212/P Function LSI S inverse short delay (I^2t = ON).

3. The time curves of circuit breaker with microprocessor based PR212/P Function LSI S inverse short delay (I^2t = OFF).

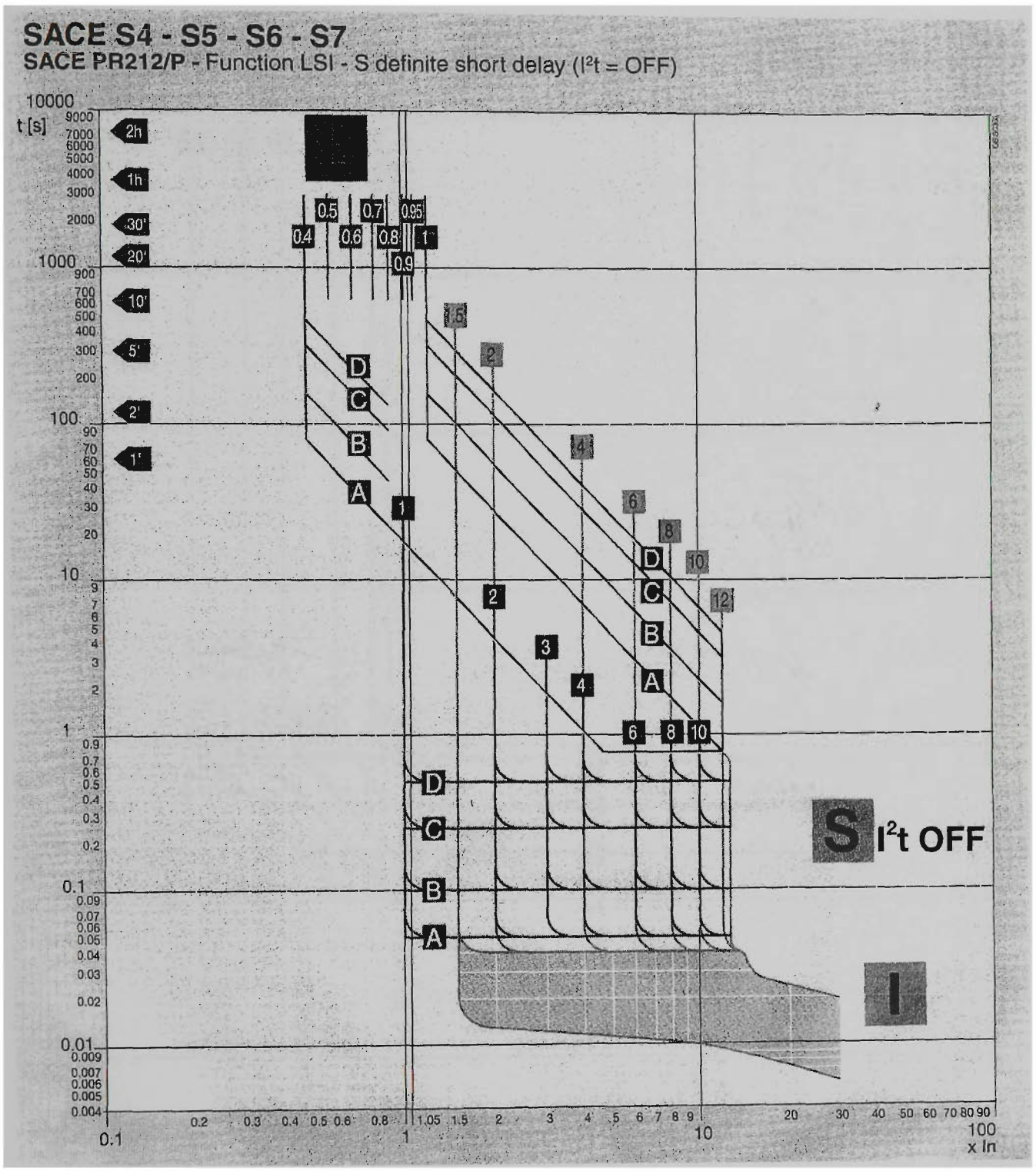
4. The time curves of circuit breaker with microprocessor based PR212/P Function G.



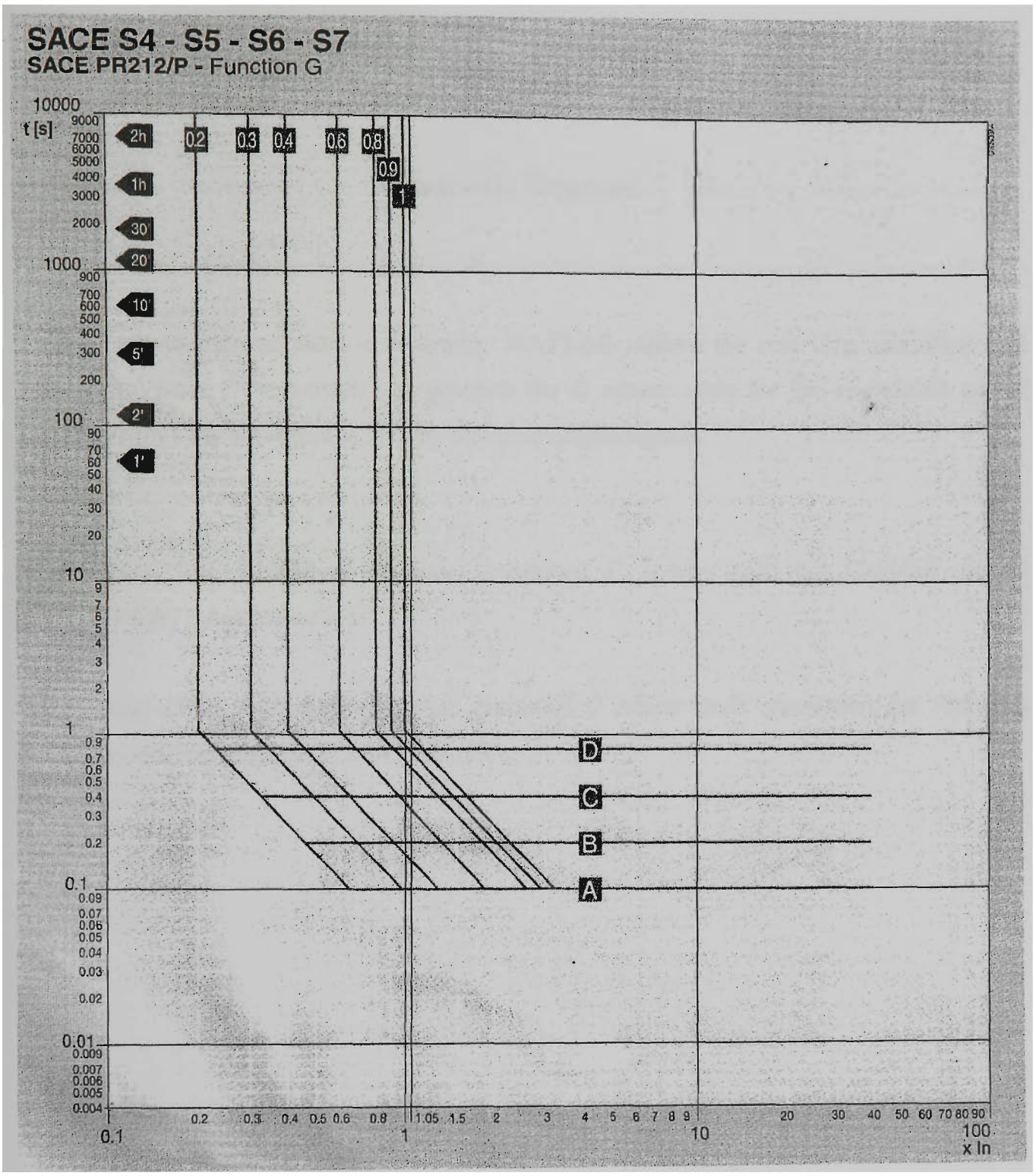
Appendix C1 The view of physical configuration microprocessor based circuit breaker



The time curves of circuit breaker with microprocessor based PR212/P Function LSI S inverse short delay (I^2t = ON).



The time curves of circuit breaker with microprocessor based PR212/P Function LSI S inverse short delay (I^2t = OFF).



The time curves of circuit breaker with microprocessor based PR212/P Function G.

APPENDIX D

Software Programs

There are two main simulation programs. MATLAB utilises the real time workshop and Microsoft Visual C++ complier to generate the C source code for the completed power system, as runs the acceleration and the real time output signals.

1. Real-Time Workshop Compiler generated C source code for Simulink model "FaultTransientTest.mdl".

2. Real-Time Workshop Complier generated C source code generation for Simulink model "SteadyStateSystem_acc.mdl".

Appendix

```
/*
 * Real-Time Workshop code generation for Simulink model "FaultTransientTest.mdl".
 *
 * Model Version                  : 1.22
 * Real-Time Workshop file version    : 5.0 $Date: 2002/05/30 19:21:33 $
 * Real-Time Workshop file generated on : Mon Aug 16 15:31:02 2004
 * TLC version                    : 5.0 (Jun 18 2002)
 * C source code generated on         : Mon Aug 16 15:31:03 2004
 */

#include <math.h>
#include <string.h>
#include "FaultTransientTest.h"
#include "FaultTransientTest_private.h"
#include "ext_work.h"

#include "FaultTransientTest_dt.h"
#include "simstruc.h"

/* user code (top of source file) */

/* list of Real-Time Windows Target boards */

int RTWinBoardCount = 1;
RTWINBOARD RTWinBoards[1] = {
  { "Standard_Devices/Parallel_Port", 888U, 0, NULL },
};

/* Block signals (auto storage) */
BlockIO rtB;

/* Block states (auto storage) */
D_Work rtDWork;

/* Parent Simstruct */
static SimStruct model_S;
SimStruct *const rtS = &model_S;

/* Start for root system: '<Root>' */
void MdlStart(void)
{

  /* FromFile Block: <Root>/From File2 */
  {
    static const real_T tuData[18016] = { 0.0, 1.0009547433332314E-010,
      1.0009547433332474E-010, 1.0009547433332776E-010, 1.0109743103335805E-010,
      1.0109743103336107E-010, 1.0209938773339136E-010, 1.0310134443342165E-010,
      1.0410330113345194E-010, 1.0710917123354280E-010, 1.1011504133363366E-010,
      1.1312091143372452E-010, 1.2514439183408799E-010, 1.3716787223445144E-010,
      1.4919135263481488E-010, 1.9728527423626868E-010, 2.4537919583772248E-010,
      2.9347311743917627E-010, 4.8584880384499146E-010, 6.7822449025080665E-010,
      8.7060017665662183E-010, 1.6401029222798826E-009, 2.4096056679031433E-009,
      3.1791084135264041E-009, 6.2571193960194475E-009, 9.3351303785124905E-009,
      1.2413141361005533E-008, 2.4725185290977705E-008, 3.7037229220949881E-008,
      4.9349273150922053E-008, 9.8597448870810754E-008, 1.4784562459069944E-007,
      1.9709380031058813E-007, 3.9408650319014293E-007, 5.9107920606969768E-007,
      7.8807190894925243E-007, 1.5760427204674715E-006, 2.3640135319856908E-006,
      3.1519843435039098E-006, 3.9399551550221292E-006, 7.8798092126132251E-006,
      1.1819663270204320E-005, 1.5759517327795415E-005, 1.9699371385386510E-005,
      1.9795912357930225E-005, 1.9829671962289782E-005, 1.9863431566649339E-005,
      1.9897191171008896E-005, 1.9930950775368454E-005, 1.9964710379728011E-005,
      2.0051706114693302E-005, 2.0138701849658594E-005, 2.0225697584623885E-005,
      2.0312693319589177E-005, 2.1082535572287480E-005, 2.1852377824985782E-005,
      2.2622220077684085E-005, 2.4348286276901085E-005, 2.6074352476118085E-005,
      2.7800418675335084E-005, 2.9526484874552084E-005, 3.2671789446246851E-005,
      3.5817094017941618E-005, 3.8962398589636385E-005, 4.2107703161331152E-005,
      4.5253007733025919E-005, 5.7034211466895084E-005, 6.8815415200764257E-005,
      8.0596618934633429E-005, 9.2377822668502602E-005, 1.1685325267215955E-004,
      1.4132868267581651E-004, 1.6580411267947347E-004, 1.9027954268313043E-004,
```

Appendix

2.1475497268678739E-004, 2.9166613240268766E-004, 3.6857729211858792E-004,
4.4548845183448819E-004, 5.2239961155038845E-004, 5.9931077126628872E-004,
7.2035015790584934E-004, 8.4138954454540996E-004, 9.6242893118497058E-004,
1.0834683178245312E-003, 1.2045077044640919E-003, 1.4081535868037953E-003,
1.6117994691434987E-003, 1.8154453514832021E-003, 2.0190912338229053E-003,
2.2227371161626084E-003, 2.4263829985023116E-003, 2.8346737355751561E-003,
3.2429644726480007E-003, 3.6512552097208452E-003, 4.0595459467936898E-003,
4.4678366838665348E-003, 4.8761274209393798E-003, 5.6585552546986771E-003,
5.8932836048264667E-003, 6.1280119549542563E-003, 6.3627403050820460E-003,
6.5974686552098356E-003, 6.8321970053376252E-003, 7.0669253554654149E-003,
7.9973350022524495E-003, 8.7647802130739663E-003, 9.5322254238954830E-003,
1.0299670634717000E-002, 1.1067115845538517E-002, 1.1834561056360033E-002,
1.2602006267181550E-002, 1.3369451478003067E-002, 1.4187761378262380E-002,
1.5006071278521693E-002, 1.5824381178781006E-002, 1.6642691079040317E-002,
1.7461000979299628E-002, 1.8279310879558940E-002, 1.9097620779818251E-002,
1.9915930680077562E-002, 2.0734240580336873E-002, 2.1552550480596185E-002,
2.1798043450673977E-002, 2.2043536420751769E-002, 2.2289029390829562E-002,
2.2534522360907354E-002, 2.2780015330985146E-002, 2.3151510064465908E-002,
2.3523004797946671E-002, 2.3894499531427433E-002, 2.4265994264908195E-002,
2.4735448632333484E-002, 2.5204902999758774E-002, 2.5674357367184063E-002,
2.6143811734609352E-002, 2.6613266102034641E-002, 2.7164934664278009E-002,
2.7716603226521377E-002, 2.8268271788764746E-002, 2.8819940351008114E-002,
2.9371608913251482E-002, 2.9982545081706787E-002, 3.0593481250162091E-002,
3.1204417418617396E-002, 3.1815353587072701E-002, 3.2426289755528005E-002,
3.3037225923983310E-002, 3.3700380152801100E-002, 3.4363534381618890E-002,
3.5026688610436681E-002, 3.5689842839254471E-002, 3.6352997068072261E-002,
3.7016151296890051E-002, 3.7692027360458832E-002, 3.8367903424027613E-002,
3.9043779487596394E-002, 3.9719655551165176E-002, 4.0395531614733957E-002,
4.0934263805640560E-002, 4.1472995996547163E-002, 4.2011728187453766E-002,
4.2550460378360369E-002, 4.3089192569266972E-002, 4.3627924760173575E-002,
4.4166656951080178E-002, 4.4727742015038881E-002, 4.5177132925615537E-002,
4.5626523836192194E-002, 4.6075914746768851E-002, 4.6525305657345507E-002,
4.6974696567922164E-002, 4.7424087478498821E-002, 4.7904985893540357E-002,
4.8385884308581893E-002, 4.8866782723623430E-002, 4.9347681138664966E-002,
4.9828579553706502E-002, 5.0309477968748038E-002, 5.0813021320289566E-002,
5.1316564671831093E-002, 5.1820108023372620E-002, 5.2323651374914147E-002,
5.2827194726455674E-002, 5.3330738077997202E-002, 5.3834281429538729E-002,
5.4428701402669116E-002, 5.5023121375799504E-002, 5.5617541348929891E-002,
5.6211961322060279E-002, 5.6806381295190667E-002, 5.7400801268321054E-002,
5.7995221241451442E-002, 5.8589641214581829E-002, 5.9081702887453458E-002,
5.9573764560325086E-002, 6.0065826233196715E-002, 6.0557887906068343E-002,
6.1049949578939972E-002, 6.1542011251811600E-002, 6.2034072924683228E-002,
6.2542031542816207E-002, 6.3049990160949185E-002, 6.3557948779082163E-002,
6.4065907397215141E-002, 6.4573866015348119E-002, 6.5081824633481097E-002,
6.5589783251614076E-002, 6.6097741869747054E-002, 6.6605700487880032E-002,
6.6666666666665167E-002, 6.6666666666667013E-002, 6.6666666666668525E-002,
6.6666870013149582E-002, 6.6667073359630638E-002, 6.6667276706111694E-002,
6.6667886745554863E-002, 6.6668496784998033E-002, 6.6669106824441202E-002,
6.6671546982213878E-002, 6.6673987139986554E-002, 6.6676427297759230E-002,
6.6686187928849935E-002, 6.6695948559940640E-002, 6.6705709191031345E-002,
6.6744751715394166E-002, 6.6783794239756986E-002, 6.6822836764119806E-002,
7.1514437680920742E+002, 7.1618901427759965E+002, 7.1722528503837975E+002,
7.1803868609700771E+002, 7.1866226055387608E+002, 7.1924313538978993E+002,
7.1988529610618696E+002, 7.2077252963554679E+002, 7.2189499230430818E+002,
7.2280029508755842E+002, 7.2353582475402709E+002, 7.2412727338647369E+002,
7.2469596156952559E+002, 7.2543227866415418E+002, 7.2644366088721790E+002,
7.2748319543795469E+002, 7.2831938575360573E+002, 7.2895953272828058E+002,
7.2952913333835943E+002, 7.3014056519197948E+002, 7.3099072392983271E+002,
7.3209615484062203E+002, 7.3300821289435839E+002, 7.3375613247766307E+002,
7.3435111992747647E+002, 7.3490362310399644E+002, 7.3561210662031942E+002,
7.3660148379220846E+002, 7.3763641514825906E+002, 7.3847890369331503E+002,
7.3912403453437730E+002, 7.3968656685576548E+002, 7.4021731407719994E+002,
7.4086117753344354E+002, 7.4168174508255038E+002, 7.4276508975597744E+002,
7.4370751483568029E+002, 7.4438939782159594E+002, 7.4493434724076303E+002,
7.4545481774112307E+002, 7.4615437468095871E+002, 7.4709842314545358E+002,
7.4814271699292055E+002, 7.4898125492655845E+002, 7.4957791226442987E+002,
7.5006634115236625E+002, 7.5062917247900987E+002, 7.5137808466833746E+002,
7.5242487508926502E+002, 7.5341387415893939E+002, 7.5416130683481788E+002,
7.5471694663688686E+002, 7.5518656355244377E+002, 7.5581791161939293E+002,
7.5671132965099628E+002, 7.5776461322886462E+002, 7.5853571409683821E+002,
7.5912441981458767E+002, 7.5961505659510021E+002, 7.6016453893463836E+002,
7.6093439880008737E+002, 7.6194494033272429E+002, 7.6290054887542692E+002,

```
7.6362342621080472E+002, 7.6416321484976027E+002, 7.6467008854079722E+002,
7.6525022086923059E+002, 7.6607499333399949E+002, 7.6712685730725218E+002,
7.6798179735424355E+002, 7.6867378132361591E+002, 7.6921657971239699E+002,
7.6971346103429994E+002, 7.7035430803729651E+002, 7.7127111357433046E+002,
7.7234766761800790E+002, 7.7312228392752911E+002, 7.7371970435974356E+002,
7.7420335713261977E+002, 7.7470972469076742E+002, 7.7541055495876049E+002,
7.7636956254099437E+002, 7.7732594501515825E+002, 7.7807735594699750E+002,
7.7864300319740607E+002, 7.7914197439133181E+002, 7.7966974473455809E+002,
7.8042417613438374E+002, 7.8144457753600864E+002, 7.8238058647425783E+002,
7.8306496764335111E+002, 7.8356826069989961E+002, 7.8404475901518958E+002,
7.8460675085671460E+002, 7.8541661197333610E+002, 7.8644240215143668E+002,
7.8727184433381854E+002, 7.8793918171610323E+002, 7.8845767882107145E+002,
7.8892579320588266E+002, 7.8953165944640557E+002, 7.9039835382935632E+002,
7.9142005438958870E+002, 7.9219355583313484E+002, 7.9279696274730384E+002,
7.9327431199437228E+002, 7.9374146710487253E+002, 7.9437569461496412E+002,
7.9527981834929460E+002, 7.9631900873335303E+002, 7.9705593778869763E+002,
7.9761455000593298E+002, 7.9806318567146968E+002, 7.9854146979236828E+002,
7.9921889935221964E+002, 8.0015087887816287E+002, 8.0107893877521030E+002,
8.0180669675631771E+002, 8.0235046141573434E+002, 8.0282100257310697E+002,
8.0331052071419492E+002, 8.0401957326903016E+002, 8.0496339830981742E+002,
8.0586698873823 74E+002, 8.0655912432934508E+002, 8.0706962896120831E+002,
8.0752429547089821E+002, 8.0802225576847536E+002, 8.0874969585727581E+002,
8.0973842602369325E+002, 8.1064187245184826E+002, 8.1129786297647377E+002,
8.1177406626991274E+002, 8.1221493851132482E+002, 8.1273256597715522E+002,
8.1349294492373872E+002, 8.1448030339942090E+002, 8.1535820287901959E+002,
8.1598423488373874E+002, 8.1643446349211001E+002, 8.1686853852962747E+002,
8.1730067684347455E+002, 8.1790474817339066E+002, 8.1874887055550744E+002,
8.1971393843837734E+002, 8.2050984590433575E+002, 8.2106183059917930E+002,
8.2150371178262253E+002, 8.2194817424698454E+002, 8.2256795541518909E+002,
8.2342143638516347E+002, 8.2437717010935557E+002, 8.2514407112183369E+002,
8.2566684012431392E+002, 8.2606171934870622E+002, 8.2651853761856376E+002,
8.2716065037488943E+002, 8.2809921974830559E+002, 8.2896169089668490E+002,
8.2959973554376927E+002, 8.3005995651171634E+002, 8.3048375047625746E+002,
8.3098441171530749E+002, 8.3172506969047492E+002, 8.3269056535535151E+002,
8.3355126243805341E+002, 8.3416547672224749E+002, 8.3460522834365361E+002,
8.3502253991525231E+002, 8.3543518688275924E+002, 8.3601589826403642E+002,
8.3683559134137306E+002, 8.3778303059807911E+002, 8.3857230483700198E+002,
8.3912041288395687E+002, 8.3954708248725501E+002, 8.3996662470624551E+002,
8.4055700836317396E+002, 8.4138213795506624E+002, 8.4232233071570965E+002,
8.4309013426397996E+002, 8.4361528784853544E+002, 8.4403308784489445E+002,
8.4445833377331951E+002, 8.4505663428867490E+002, 8.4588570296689898E+002,
8.4681885371249336E+002, 8.4756971774418605E+002, 8.4807779187658241E+002,
8.4848895104332337E+002, 8.4891790643388640E+002, 8.4952138802271622E+002,
8.5035243437494853E+002, 8.5120637133814535E+002, 8.5185373835619419E+002,
8.5232196400598264E+002, 8.5273016089361613E+002, 8.5318336901502914E+002,
8.5386035466281555E+002, 8.5479408593050414E+002, 8.5565585224377458E+002,
8.5628459703051476E+002, 8.5673660944904702E+002, 8.5713710959541675E+002,
8.5759168016330659E+002, 8.5827419188386625E+002 };

 rtDWork.From_File3_PWORK.PrevTimePtr = (void *) &tuData[0];
}

/* S-Function Block: <Root>/Digital Output */

{
 {
  double val[3];
  double *valp = val;

  *valp++ = *&rtP.Digital_Output_InitialValue;
  *valp++ = *&rtP.Digital_Output_InitialValue;
  *valp++ = *&rtP.Digital_Output_InitialValue;

  RTBIO_DriverIO(0, DIGITALOUTPUT, IOWRITE, 3,
   &rtP.Digital_Output_Channels[0], val, &rtP.Digital_Output_BitMode);
 }
 }
}

/* Outputs for root system: '<Root>' */
void MdlOutputs(int_T tid)
{
```

Appendix

```
/* local block i/o variables */
real_T rtb_From_File2[3];
real_T rtb_From_File3;
real_T rtb_Look_Up_Table1_a;
real_T rtb_Clock;
real_T rtb_Look_Up_Table1_b;

/* tid is required for a uniform function interface. This system
 * is single rate, and in this case, tid is not accessed. */
UNUSED_PARAMETER(tid);

{
  static const real_T *pStart = NULL;
  static boolean_T initBasePtr = TRUE;
  const real_T *pT = rtDWork.From_File2_PWORK.PrevTimePtr;
  real_T time = ssGetT(rtS);
  const real_T *pU = NULL;

  if (initBasePtr == TRUE) {
    pStart = (real_T *) rtDWork.From_File2_PWORK.PrevTimePtr;
    initBasePtr = FALSE;
  }

  pU = pStart + 4502;

  if (time <= pStart[0]) {
    pT = pStart;
  } else if (time >= pU[0]) {
    pT = pU;
  } else {
    if (time < pT[0]) {
      while (time < pT[0]) {
        pT--;
      }
    } else {
      while (time >= pT[1]) {
        pT++;
      }
    }
  }
  rtDWork.From_File2_PWORK.PrevTimePtr = (void *) pT;

  pU = pT + 4504;

  if (pT[0] == pT[1]) {
    rtb_From_File2[0] = pU[ (time < pT[0]) ? 0 : 1 ];
    pU += 4504;
    rtb_From_File2[1] = pU[ (time < pT[0]) ? 0 : 1 ];
    pU += 4504;
    rtb_From_File2[2] = pU[ (time < pT[0]) ? 0 : 1 ];
  } else {
    real_T f = (pT[1]-time)/(pT[1]-pT[0]);
    if (pU[0] == pU[1]) {
      rtb_From_File2[0] = pU[0];
    } else {
      rtb_From_File2[0] = f*pU[0]+(1.0-f)*pU[1];
    }
    pU += 4504;
    if (pU[0] == pU[1]) {
      rtb_From_File2[1] = pU[0];
    } else {
      rtb_From_File2[1] = f*pU[0]+(1.0-f)*pU[1];
    }
    pU += 4504;
    if (pU[0] == pU[1]) {
      rtb_From_File2[2] = pU[0];
    } else {
      rtb_From_File2[2] = f*pU[0]+(1.0-f)*pU[1];
    }
  }
}
```

Appendix

```
{
 static const real_T *pStart = NULL;
 static boolean_T initBasePtr = TRUE;
 const real_T *pT = rtDWork.From_File3_PWORK.PrevTimePtr;
 real_T time = ssGetT(rtS);
 const real_T *pU = NULL;

 if (initBasePtr == TRUE) {
  pStart = (real_T *) rtDWork.From_File3_PWORK.PrevTimePtr;
  initBasePtr = FALSE;
 }

 pU = pStart + 2250;

 if (time <= pStart[0]) {
  pT = pStart;
 } else if (time >= pU[0]) {
  pT = pU;
 } else {
  if (time < pT[0]) {
   while (time < pT[0]) {
    pT--;
   }
  } else {
   while (time >= pT[1]) {
    pT++;
   }
  }
 }
 rtDWork.From_File3_PWORK.PrevTimePtr = (void *) pT;

 pU = pT + 2252;

 if (pT[0] == pT[1]) {
  rtb_From_File3 = pU[ (time < pT[0]) ? 0 : 1 ];
 } else {
  real_T f = (pT[1]-time)/(pT[1]-pT[0]);
  if (pU[0] == pU[1]) {
   rtb_From_File3 = pU[0];
  } else {
   rtb_From_File3 = f*pU[0]+(1.0-f)*pU[1];
  }
 }
}

/* Lookup: '<S1>/Look-Up Table1' */

rtb_Look_Up_Table1_a = rt_Lookup(rtP.Look_Up_Table1_a_XData, 15,
 rtb_From_File3, rtP.Look_Up_Table1_a_YData);

/* Product: '<S1>/Product' incorporates:
 *  Gain: '<S1>/Gain1'
 *
 * Regarding '<S1>/Gain1':
 *  Gain value: rtP.Gain1_Gain
 */
rtB.Product[0] = (rtb_From_File2[0] * rtP.Gain1_Gain) * rtb_Look_Up_Table1_a;
rtB.Product[1] = (rtb_From_File2[1] * rtP.Gain1_Gain) * rtb_Look_Up_Table1_a;
rtB.Product[2] = (rtb_From_File2[2] * rtP.Gain1_Gain) * rtb_Look_Up_Table1_a;

/* Clock: '<S3>/Clock' */
rtb_Clock = ssGetT(rtS);

/* Lookup: '<S3>/Look-Up Table1' incorporates:
 *  Fcn: '<S3>/Fcn1'
 *  Sum: '<S3>/Sum'
 *  S-Function (sfun_tstart): '<S3>/startTime'
 *
 * Regarding '<S3>/Fcn1':
 *  Expression: rem(u[1],period)
 */
```

Appendix

```
rtb_Look_Up_Table1_b = rt_Lookup(rtP.Look_Up_Table1_b_XData, 4, (fmod(
  (rtb_Clock - (0.0)) , 0.0005) ), rtP.Look_Up_Table1_b_YData);

/* DataTypeConversion: '<S2>/Data Type  Conversion2' incorporates:
 *  RelationalOperator: '<S2>/Relational Operator1'
 */
rtB.Data_Type_Conversion2[0] = (real_T)(rtB.Product[0] >=
  rtb_Look_Up_Table1_b);
rtB.Data_Type_Conversion2[1] = (real_T)(rtB.Product[1] >=
  rtb_Look_Up_Table1_b);
rtB.Data_Type_Conversion2[2] = (real_T)(rtB.Product[2] >=
  rtb_Look_Up_Table1_b);

/* S-Function Block: <Root>/Digital Output */
{
  {

    RTBIO_DriverIO(0, DIGITALOUTPUT, IOWRITE, 3,
     &rtP.Digital_Output_Channels[0], &rtB.Data_Type_Conversion2[0],
     &rtP.Digital_Output_BitMode);
  }
 }
}

/* Update for root system: '<Root>' */
void MdlUpdate(int_T tid)
{

 /* tid is required for a uniform function interface. This system
  * is single rate, and in this case, tid is not accessed. */
 UNUSED_PARAMETER(tid);
}

/* Terminate for root system: '<Root>' */
void MdlTerminate(void)
{
 if(rtS != NULL) {

   /* S-Function Block: <Root>/Digital Output */

   {
    {
     double val[3];
     double *valp = val;

     *valp++ = *&rtP.Digital_Output_FinalValue;
     *valp++ = *&rtP.Digital_Output_FinalValue;
     *valp++ = *&rtP.Digital_Output_FinalValue;

     RTBIO_DriverIO(0, DIGITALOUTPUT, IOWRITE, 3,
      &rtP.Digital_Output_Channels[0], val, &rtP.Digital_Output_BitMode);
    }
   }
  }
 }

 /* Function to initialize sizes */
 void MdlInitializeSizes(void)
 {
  ssSetNumContStates(rtS, 0);          /* Number of continuous states */
  ssSetNumY(rtS, 0);                  /* Number of model outputs */
  ssSetNumU(rtS, 0);                  /* Number of model inputs */
  ssSetDirectFeedThrough(rtS, 0);      /* The model is not direct feedthrough */
  ssSetNumSampleTimes(rtS, 2);         /* Number of sample times */
  ssSetNumBlocks(rtS, 15);            /* Number of blocks */
  ssSetNumBlockIO(rtS, 2);            /* Number of block outputs */
  ssSetNumBlockParams(rtS, 45);        /* Sum of parameter "widths" */
 }

 /* Function to initialize sample times */
 void MdlInitializeSampleTimes(void)
 {
```

```
 /* task periods */
 ssSetSampleTime(rtS, 0, 0.0);
 ssSetSampleTime(rtS, 1, 0.0001);

 /* task offsets */
 ssSetOffsetTime(rtS, 0, 0.0);
 ssSetOffsetTime(rtS, 1, 0.0);
}

/* Function to register the model */
SimStruct *FaultTransientTest(void)
{
 static struct _ssMdlInfo mdlInfo;
 (void)memset((char *)rtS, 0, sizeof(SimStruct));
 (void)memset((char *)&mdlInfo, 0, sizeof(struct _ssMdlInfo));
 ssSetMdlInfoPtr(rtS, &mdlInfo);

 /* timing info */
 {
  static time_T mdlPeriod[NSAMPLE_TIMES];
  static time_T mdlOffset[NSAMPLE_TIMES];
  static time_T mdlTaskTimes[NSAMPLE_TIMES];
  static int_T mdlTsMap[NSAMPLE_TIMES];
  static int_T mdlSampleHits[NSAMPLE_TIMES];

  {
   int_T i;

   for(i = 0; i < NSAMPLE_TIMES; i++) {
    mdlPeriod[i] = 0.0;
    mdlOffset[i] = 0.0;
    mdlTaskTimes[i] = 0.0;
   }
  }
  (void)memset((char_T *)&mdlTsMap[0], 0, 2 * sizeof(int_T));
  (void)memset((char_T *)&mdlSampleHits[0], 0, 2 * sizeof(int_T));

  ssSetSampleTimePtr(rtS, &mdlPeriod[0]);
  ssSetOffsetTimePtr(rtS, &mdlOffset[0]);
  ssSetSampleTimeTaskIDPtr(rtS, &mdlTsMap[0]);
  ssSetTPtr(rtS, &mdlTaskTimes[0]);
  ssSetSampleHitPtr(rtS, &mdlSampleHits[0]);
 }
 ssSetSolverMode(rtS, SOLVER_MODE_SINGLETASKING);

 /*
  * initialize model vectors and cache them in SimStruct
  */

 /* block I/O */
 {
  void *b = (void *) &rtB;
  ssSetBlockIO(rtS, b);

  {
   int_T i;

   b =&rtB.Product[0];
   for (i = 0; i < 6; i++) {
    ((real_T*)b)[i] = 0.0;
   }
  }
 }

 /* parameters */
 ssSetDefaultParam(rtS, (real_T *) &rtP);

 /* data type work */
 {
  void *dwork = (void *) &rtDWork;
  ssSetRootDWork(rtS, dwork);
  (void)memset((char_T *) dwork, 0, sizeof(D_Work));
```

```
}

/* data type transition information (for external mode) */
{
 static DataTypeTransInfo dtInfo;

 (void)memset((char_T *) &dtInfo, 0, sizeof(dtInfo));
 ssSetModelMappingInfo(rtS, &dtInfo);

 dtInfo.numDataTypes = 13;
 dtInfo.dataTypeSizes = &rtDataTypeSizes[0];
 dtInfo.dataTypeNames = &rtDataTypeNames[0];

 /* Block I/O transition table */
 dtInfo.B = &rtBTransTable;

 /* Parameters transition table */
 dtInfo.P = &rtPTransTable;
}

/* Model specific registration */
ssSetRootSS(rtS, rtS);

ssSetVersion(rtS, SIMSTRUCT_VERSION_LEVEL2);
ssSetModelName(rtS, "FaultTransientTest");
ssSetPath(rtS, "FaultTransientTest");

ssSetTStart(rtS, 0.0);
ssSetTFinal(rtS, 2.0500000000000003E+000);
ssSetStepSize(rtS, 0.0001);
ssSetFixedStepSize(rtS, 0.0001);

ssSetChecksumVal(rtS, 0, 1316278515U);
ssSetChecksumVal(rtS, 1, 1666847045U);
ssSetChecksumVal(rtS, 2, 81993352U);
ssSetChecksumVal(rtS, 3, 203932709U);

{
 static const EnableStates rtAlwaysEnabled = SUBSYS_ENABLED;

 static RTWExtModeInfo rt_ExtModeInfo;
 static const void *sysModes[1];

 ssSetRTWExtModeInfo(rtS, &rt_ExtModeInfo);
 rteiSetSubSystemModeVectorAddresses(&rt_ExtModeInfo, sysModes);

 sysModes[0] = &rtAlwaysEnabled;

 rteiSetModelMappingInfoPtr(&rt_ExtModeInfo, &ssGetModelMappingInfo(rtS));

 rteiSetChecksumsPtr(&rt_ExtModeInfo, ssGetChecksums(rtS));

 rteiSetTPtr(&rt_ExtModeInfo, ssGetTPtr(rtS));
}

return rtS;
}
```

Appendix

```
/*
 * Real-Time Workshop code generation for Simulink model "SteadyStateSystem_acc.mdl".
 *
 * Model Version                  : 1.54
 * Real-Time Workshop file version     : 5.0 $Date: 2002/05/30 19:21:33 $
 * Real-Time Workshop file generated on : Mon Aug 16 15:01:39 2004
 * TLC version                    : 5.0 (Jun 18 2002)
 * C source code generated on          : Mon Aug 16 15:01:40 2004
 */

#include <math.h>
#include <string.h>
#include "SteadyStateSystem_acc.h"
#include "SteadyStateSystem_acc_private.h"
#include <stdio.h>
#include "simstruc.h"

#define CodeFormat                S-Function
#define AccDefine1                Accelerator_S-Function

real_T rtInf;
real_T rtMinusInf;

/* Output and update for enable system:
 *   '<S6>/Phasor Measurements'
 *   '<S7>/Phasor Measurements'
 *   '<S8>/Phasor Measurements'
 */
void SteadyStateSystem_Phasor(SimStruct *S, int_T tid, real_T fu0, const real_T
 fu1[3], const real_T fu2[3], rtDW_SteadyStateSystem_Phasor *localDW,
 rtP_SteadyStateSystem_Phasor *localP)
{
  /* local block i/o variables */
  real_T rtb_Complex_to_Magnitude_Angle[6];
  real_T rtb_temp116[6];

  /* detect enable/disable transitions */
  if (ssIsSampleHit(S, 1, tid)) {
    EnableStates prevEnableState = (EnableStates) localDW->Phasor_Mea_a_MODE[0];
    EnableStates enableState;

    if (ssIsSampleHit(S, 1, tid)) {    /* Sample time: [0.0, 1.0] */
      localDW->Phasor_Mea_a_MODE[1] = (fu0 > 0.0) ? SUBSYS_ENABLED :
        SUBSYS_DISABLED;
    }

    enableState = (EnableStates) localDW->Phasor_Mea_a_MODE[1];
    if (enableState == SUBSYS_ENABLED) {
      if (prevEnableState == SUBSYS_DISABLED) {
        /* SUBSYS_BECOMING_ENABLED */

        if ( ssGetT(S) != ssGetTStart(S) ) {
          ssSetSolverNeedsReset(S);
        }
        /* (system enable function is empty) */
        localDW->Phasor_Mea_a_MODE[0] = (int_T) SUBSYS_ENABLED;
      }
    } else {
      if (prevEnableState == SUBSYS_ENABLED) {
        /* SUBSYS_BECOMING_DISABLED */
        ssSetSolverNeedsReset(S);

        /* (system disable function is empty) */
        localDW->Phasor_Mea_a_MODE[0] = (int_T) SUBSYS_DISABLED;
      }
    }
  }

  /* run blocks if enabled */
  if (localDW->Phasor_Mea_a_MODE[0] == SUBSYS_ENABLED) {
    if (ssIsContinuousTask(S, tid)) {
```

```
    /* ComplexToMagnitudeAngle: '<S36>/Complex to Magnitude-Angle' */
    rtb_Complex_to_Magnitude_Angle[0] = fabs(fu1[0]);
    if (fu1[0] >= 0.0) {
      rtb_temp116[0] = 0.0;
    } else {
      rtb_temp116[0] = RT_PI;
    }
    rtb_Complex_to_Magnitude_Angle[1] = fabs(fu1[1]);
    if (fu1[1] >= 0.0) {
      rtb_temp116[1] = 0.0;
    } else {
      rtb_temp116[1] = RT_PI;
    }
    rtb_Complex_to_Magnitude_Angle[2] = fabs(fu1[2]);
    if (fu1[2] >= 0.0) {
      rtb_temp116[2] = 0.0;
    } else {
      rtb_temp116[2] = RT_PI;
    }
    rtb_Complex_to_Magnitude_Angle[3] = fabs(fu2[0]);
    if (fu2[0] >= 0.0) {
      rtb_temp116[3] = 0.0;
    } else {
      rtb_temp116[3] = RT_PI;
    }
    rtb_Complex_to_Magnitude_Angle[4] = fabs(fu2[1]);
    if (fu2[1] >= 0.0) {
      rtb_temp116[4] = 0.0;
    } else {
      rtb_temp116[4] = RT_PI;
    }
    rtb_Complex_to_Magnitude_Angle[5] = fabs(fu2[2]);
    if (fu2[2] >= 0.0) {
      rtb_temp116[5] = 0.0;
    } else {
      rtb_temp116[5] = RT_PI;
    }

    /* Gain: '<S36>/180//pi'
     *
     * Regarding '<S36>/180//pi':
     *  Gain value: localP->pi_Gain
     */
    {
      int_T i1;

      const real_T *u0 = &rtb_temp116[0];
      real_T *y0 = &rtb_temp116[0];

      for (i1=0; i1 < 6; i1++) {
        y0[i1] = u0[i1] * localP->pi_Gain;
      }
    }
  } }
}

/* Outputs for root system: '<Root>' */
static void mdlOutputs(SimStruct *S, int_T tid)
{
  /* simstruct variables */
  SteadyStateSystem_BlockIO *SteadyStateSystem_B = (SteadyStateSystem_BlockIO *)
    _ssGetBlockIO(S);
  SteadyStateSystem_ContinuousStates *SteadyStateSystem_X =
    (SteadyStateSystem_ContinuousStates*) ssGetContStates(S);
  SteadyStateSystem_D_Work *SteadyStateSystem_DWork = (SteadyStateSystem_D_Work
    *) ssGetRootDWork(S);
  SteadyStateSystem_Parameters *SteadyStateSystem_P =
    (SteadyStateSystem_Parameters *) ssGetDefaultParam(S);

  /* local block i/o variables */
  real_T rtb_Int;
  real_T rtb_Clock_a;
```

```
real_T rtb_iq;
real_T rtb_id_a;
real_T rtb_Switch3_a;
real_T rtb_Switch3_e;
real_T rtb_Fcn2;
real_T rtb_Fcn3;
real_T rtb_Elementary_Math;
real_T rtb_Elementary_Math1;
real_T rtb_Complex_to_Magnitude_Angle;
real_T rtb_temp143;
real_T rtb_temp144;
real_T rtb_temp145;
boolean_T rtb_Relational_Operator1[6];

if (ssIsSampleHit(S, 1, tid)) {      /* Sample time: [0.0, 1.0] */

  /* Constant: '<S69>/Constant' */
  SteadyStateSystem_B->Constant_a = SteadyStateSystem_P->Constant_a_Value;

  /* Constant: '<S70>/Constant' */
  SteadyStateSystem_B->Constant_b = SteadyStateSystem_P->Constant_b_Value;

  /* Constant: '<S71>/Constant' */
  SteadyStateSystem_B->Constant_c = SteadyStateSystem_P->Constant_c_Value;
}

if (ssIsContinuousTask(S, tid)) {    /* Sample time: [0.0, 0.0] */

  /* Integrator: '<S59>/Int' */
  rtb_Int = SteadyStateSystem_X->Int_CSTATE;

  /* Gain: '<S13>/Gain'
   *
   * Regarding '<S13>/Gain':
   *  Gain value: SteadyStateSystem_P->Gain_a_Gain
   */
  SteadyStateSystem_B->Gain_a = rtb_Int * SteadyStateSystem_P->Gain_a_Gain;

  /* Scope: '<Root>/Scope2' */
  /* Call into Simulink for Scope */
  ssCallAccelRunBlock(S, 10, 5, SS_CALL_MDL_OUTPUTS);

  /* Clock: '<S14>/Clock' */
  rtb_Clock_a = ssGetT(S);

  /* S-Function (sfun_tstart): '<S14>/startTime' */

  /* S-Function Block (sfun_tstart): <S14>/startTime */
  SteadyStateSystem_B->startTime = ssGetTStart(S);

  /* Gain: '<Root>/Gain1' incorporates:
   *  Lookup: '<S14>/Look-Up Table1'
   *  Fcn: '<S14>/Fcn1'
   *  Sum: '<S14>/Sum'
   *
   * Regarding '<Root>/Gain1':
   *  Gain value: SteadyStateSystem_P->Gain1_a_Gain
   *
   * Regarding '<S14>/Fcn1':
   *  Expression: rem(u[1],period)
   */
  SteadyStateSystem_B->Gain1_a =
    rt_Lookup(SteadyStateSystem_P->Look_Up_Table1_XData, 19, (fmod(
    (rtb_Clock_a - SteadyStateSystem_B->startTime) , 9.0) ),
    SteadyStateSystem_P->Look_Up_Table1_YData) *
    SteadyStateSystem_P->Gain1_a_Gain;

  /* Scope: '<Root>/Scope3' */
  /* Call into Simulink for Scope */
  ssCallAccelRunBlock(S, 10, 12, SS_CALL_MDL_OUTPUTS);
}
```

```
if (ssIsSampleHit(S, 1, tid)) {     /* Sample time: [0.0, 1.0] */

 /* Constant: '<S64>/Constant' */
 {
  int_T i1;

  real_T *y0 = &SteadyStateSystem_B->Constant_d[0];
  const real_T *p_Constant_d_Value =
   &SteadyStateSystem_P->Constant_d_Value[0];

  for (i1=0; i1 < 6; i1++) {
   y0[i1] = p_Constant_d_Value[i1];
  }
 }
}

if (ssIsContinuousTask(S, tid)) {    /* Sample time: [0.0, 0.0] */

 /* Integrator: '<S59>/Int1' */
 rtb_temp143 = SteadyStateSystem_X->Int1_CSTATE;

 /* ElementaryMath: '<S57>/Elementary Math' */
 rtb_Elementary_Math = sin(rtb_temp143);

 /* ElementaryMath: '<S57>/Elementary Math1' */
 rtb_Elementary_Math1 = cos(rtb_temp143);

 /* Integrator: '<S62>/iq ' */
 rtb_iq = SteadyStateSystem_X->iq_CSTATE;

 /* Integrator: '<S61>/id ' */
 rtb_id_a = SteadyStateSystem_X->id_a_CSTATE;

 /* Fcn: '<S60>/Fcn'
  *
  * Regarding '<S60>/Fcn':
  *   Expression: u[3]*u[2] + u[4]*u[1]
  */
 SteadyStateSystem_B->Fcn = rtb_iq * rtb_Elementary_Math1 +
  rtb_id_a * rtb_Elementary_Math;

 /* Fcn: '<S60>/Fcn1'
  *
  * Regarding '<S60>/Fcn1':
  *   Expression: (u[2]*(-u[3]-sqrt3*u[4]) + u[1]*(sqrt3*u[3]-u[4]))*0.5
  */
 SteadyStateSystem_B->Fcn1_b = ( rtb_Elementary_Math1 * ( ( -rtb_iq ) -
  1.7320508075688772E+000 * rtb_id_a ) + rtb_Elementary_Math * (
  1.7320508075688772E+000 *
  rtb_iq - rtb_id_a ) ) * 0.5;

 /* Integrator: '<S65>/Integrator' */
 if (ssIsMajorTimeStep(S)) {
  {
   int_T i1;

   real_T *xc = &SteadyStateSystem_X->Integrator_CSTATE[0];

   for (i1=0; i1 < 6; i1++) {
    if (xc[i1] >= SteadyStateSystem_P->Integrator_UpperSat ) {
     xc[i1] = SteadyStateSystem_P->Integrator_UpperSat;
    } else if (xc[i1] <= SteadyStateSystem_P->Integrator_LowerSat ) {
     xc[i1] = SteadyStateSystem_P->Integrator_LowerSat;
    }
   }
  }
 }
 {
  int_T i1;

  real_T *y0 = &SteadyStateSystem_B->Integrator[0];
  real_T *xc = &SteadyStateSystem_X->Integrator_CSTATE[0];
```

```
  for (i1=0; i1 < 6; i1++) {
    y0[i1] = xc[i1];
  }
 }
}

if (ssIsSampleHit(S, 1, tid)) {      /* Sample time: [0.0, 1.0] */

 /* Constant: '<S65>/0 2' */
 SteadyStateSystem_B->id_b = SteadyStateSystem_P->id_b_Value;

 /* RelationalOperator: '<S65>/Relational Operator1' */
 if (ssIsMajorTimeStep(S)) {
  {
   int_T i1;

   const real_T *u0 = &SteadyStateSystem_B->Integrator[0];
   int_T *mode = &SteadyStateSystem_DWork->Relational_Operator1_MODE[0];

   for (i1=0; i1 < 6; i1++) {
    mode[i1] = (int_T)(u0[i1] > SteadyStateSystem_B->id_b);
   }
  }
 }
 {
  int_T i1;

  boolean_T *y0 = &rtb_Relational_Operator1[0];
  int_T *mode = &SteadyStateSystem_DWork->Relational_Operator1_MODE[0];

  for (i1=0; i1 < 6; i1++) {
   y0[i1] = (real_T)(mode[i1]);
  }
 }

 /* DataTypeConversion: '<S65>/Data Type  Conversion' */
 {
  int_T i1;

  const boolean_T *u0 = &rtb_Relational_Operator1[0];
  real_T *y0 = &SteadyStateSystem_B->Data_Type_Conversion[0];

  for (i1=0; i1 < 6; i1++) {
   y0[i1] = (real_T)u0[i1];
  }
 }
}

if (ssIsContinuousTask(S, tid)) {    /* Sample time: [0.0, 0.0] */

 /* Level2 S-Function Block: <S52>/State-Space (sfun_psbcontc) */
 /* Call into Simulink for MEX-version of S-function */
 ssCallAccelRunBlock(S, 10, 25, SS_CALL_MDL_OUTPUTS);

 /* Scope: '<Root>/Scope5' */
 /* Call into Simulink for Scope */
 ssCallAccelRunBlock(S, 10, 26, SS_CALL_MDL_OUTPUTS);

 /* Fcn: '<S59>/Te'
  *
  * Regarding '<S59>/Te':
  *  Expression: 1.5*p*(lam*u(1)+(Ld-Lq)*u(1)*u(2))
  */
 SteadyStateSystem_B->Te = 1.5 * 3.0 * ( 0.426 * rtb_iq + ( 0.00316 -
  0.00316) * rtb_iq * rtb_id_a );

 /* Scope: '<Root>/Scope6' */
 /* Call into Simulink for Scope */
 ssCallAccelRunBlock(S, 10, 28, SS_CALL_MDL_OUTPUTS);

 /* Gain: '<S7>/Ki'
```

```
 *
 * Regarding '<S7>/Ki':
 *   Gain value: SteadyStateSystem_P->Ki_a_Gain
 */
SteadyStateSystem_B->temp107[0] = SteadyStateSystem_B->State_Space[21] *
  SteadyStateSystem_P->Ki_a_Gain;
SteadyStateSystem_B->temp107[1] = SteadyStateSystem_B->State_Space[22] *
  SteadyStateSystem_P->Ki_a_Gain;
SteadyStateSystem_B->temp107[2] = SteadyStateSystem_B->State_Space[23] *
  SteadyStateSystem_P->Ki_a_Gain;

/* ToFile: '<Root>/To File1' */
/* Call into Simulink for To File */
ssCallAccelRunBlock(S, 10, 30, SS_CALL_MDL_OUTPUTS);

/* ToFile: '<Root>/To File2' */
/* Call into Simulink for To File */
ssCallAccelRunBlock(S, 10, 31, SS_CALL_MDL_OUTPUTS);

/* Gain: '<S7>/Kv'
 *
 * Regarding '<S7>/Kv':
 *   Gain value: SteadyStateSystem_P->Kv_a_Gain
 */
SteadyStateSystem_B->temp108[0] = SteadyStateSystem_B->State_Space[11] *
  SteadyStateSystem_P->Kv_a_Gain;
SteadyStateSystem_B->temp108[1] = SteadyStateSystem_B->State_Space[12] *
  SteadyStateSystem_P->Kv_a_Gain;
SteadyStateSystem_B->temp108[2] = SteadyStateSystem_B->State_Space[13] *
  SteadyStateSystem_P->Kv_a_Gain;

/* Scope: '<Root>/V_1 Line1 Bus1' */
/* Call into Simulink for Scope */
ssCallAccelRunBlock(S, 10, 33, SS_CALL_MDL_OUTPUTS);

/* Gain: '<S6>/Kv'
 *
 * Regarding '<S6>/Kv':
 *   Gain value: SteadyStateSystem_P->Kv_b_Gain
 */
SteadyStateSystem_B->temp108[0] = SteadyStateSystem_B->State_Space[8] *
  SteadyStateSystem_P->Kv_b_Gain;
SteadyStateSystem_B->temp108[1] = SteadyStateSystem_B->State_Space[9] *
  SteadyStateSystem_P->Kv_b_Gain;
SteadyStateSystem_B->temp108[2] = SteadyStateSystem_B->State_Space[10] *
  SteadyStateSystem_P->Kv_b_Gain;

/* Gain: '<S6>/Ki'
 *
 * Regarding '<S6>/Ki':
 *   Gain value: SteadyStateSystem_P->Ki_b_Gain
 */
SteadyStateSystem_B->temp107[0] = SteadyStateSystem_B->State_Space[18] *
  SteadyStateSystem_P->Ki_b_Gain;
SteadyStateSystem_B->temp107[1] = SteadyStateSystem_B->State_Space[19] *
  SteadyStateSystem_P->Ki_b_Gain;
SteadyStateSystem_B->temp107[2] = SteadyStateSystem_B->State_Space[20] *
  SteadyStateSystem_P->Ki_b_Gain;

/* Scope: '<Root>/V_1 Line1 Bus2' */
/* Call into Simulink for Scope */
ssCallAccelRunBlock(S, 10, 36, SS_CALL_MDL_OUTPUTS);

/* Gain: '<S8>/Kv'
 *
 * Regarding '<S8>/Kv':
 *   Gain value: SteadyStateSystem_P->Kv_c_Gain
 */
SteadyStateSystem_B->temp108[0] = SteadyStateSystem_B->State_Space[14] *
  SteadyStateSystem_P->Kv_c_Gain;
SteadyStateSystem_B->temp108[1] = SteadyStateSystem_B->State_Space[15] *
  SteadyStateSystem_P->Kv_c_Gain;
```

```
SteadyStateSystem_B->temp108[2] = SteadyStateSystem_B->State_Space[16] *
  SteadyStateSystem_P->Kv_c_Gain;

/* Gain: '<S8>/Ki'
 *
 * Regarding '<S8>/Ki':
 *  Gain value: SteadyStateSystem_P->Ki_c_Gain
 */
SteadyStateSystem_B->temp107[0] = SteadyStateSystem_B->State_Space[24] *
  SteadyStateSystem_P->Ki_c_Gain;
SteadyStateSystem_B->temp107[1] = SteadyStateSystem_B->State_Space[25] *
  SteadyStateSystem_P->Ki_c_Gain;
SteadyStateSystem_B->temp107[2] = SteadyStateSystem_B->State_Space[26] *
  SteadyStateSystem_P->Ki_c_Gain;

/* Scope: '<Root>/V_I Line1 Bus3' */
/* Call into Simulink for Scope */
ssCallAccelRunBlock(S, 10, 39, SS_CALL_MDL_OUTPUTS);
}

if (ssIsSampleHit(S, 1, tid)) {      /* Sample time: [0.0, 1.0] */

 /* Constant: '<S1>/Constant' */
 SteadyStateSystem_B->Constant_e = SteadyStateSystem_P->Constant_e_Value;

 /* Constant: '<S1>/Constant1' */
 SteadyStateSystem_B->Constant1_a = SteadyStateSystem_P->Constant1_a_Value;

 /* Constant: '<S1>/Constant2' */
 SteadyStateSystem_B->Constant2_a = SteadyStateSystem_P->Constant2_a_Value;

 /* Constant: '<S1>/Constant3' */
 SteadyStateSystem_B->Constant3_a = SteadyStateSystem_P->Constant3_a_Value;

 /* Constant: '<S1>/N' */
 SteadyStateSystem_B->N_a = SteadyStateSystem_P->N_a_Value;

 /* Constant: '<S16>/Constant' */
 SteadyStateSystem_B->Constant_f = SteadyStateSystem_P->Constant_f_Value;

 /* Constant: '<S17>/Constant' */
 SteadyStateSystem_B->Constant_g = SteadyStateSystem_P->Constant_g_Value;

 /* Constant: '<S18>/Constant' */
 SteadyStateSystem_B->Constant_h = SteadyStateSystem_P->Constant_h_Value;

 /* Constant: '<S19>/Constant1' */
 SteadyStateSystem_B->Constant1_b = SteadyStateSystem_P->Constant1_b_Value;

 /* Constant: '<S19>/C4' */
 SteadyStateSystem_B->C4_a = SteadyStateSystem_P->C4_a_Value;

 /* Constant: '<S3>/com' */
 SteadyStateSystem_B->com_a = SteadyStateSystem_P->com_a_Value;

 /* Constant: '<S3>/C4' */
 SteadyStateSystem_B->C4_b = SteadyStateSystem_P->C4_b_Value;
}

if (ssIsContinuousTask(S, tid)) {    /* Sample time: [0.0, 0.0] */

 /* Clock: '<S22>/Clock' */
 rtb_temp145 = ssGetT(S);

 /* Lookup: '<S22>/Look-Up Table' */

 SteadyStateSystem_B->Look_Up_Table_a =
  rt_Lookup(SteadyStateSystem_P->Look_Up_Table_a_XData, 3, rtb_temp145,
   SteadyStateSystem_P->Look_Up_Table_a_YData);

 /* Switch: '<S3>/Switch3' */
 if (SteadyStateSystem_B->C4_b >= SteadyStateSystem_P->Switch3_a_Threshold) {
```

```
  rtb_Switch3_a = SteadyStateSystem_B->com_a;
 } else {
  rtb_Switch3_a = SteadyStateSystem_B->Look_Up_Table_a;
 }
}

if (ssIsSampleHit(S, 1, tid)) {      /* Sample time: [0.0, 1.0] */

 /* Constant: '<S3>/Constant1' */
 SteadyStateSystem_B->Constant1_c = SteadyStateSystem_P->Constant1_c_Value;

 /* Constant: '<S3>/Constant5' */
 SteadyStateSystem_B->Constant5_a = SteadyStateSystem_P->Constant5_a_Value;
}

if (ssIsContinuousTask(S, tid)) {    /* Sample time: [0.0, 0.0] */

 /* Clock: '<S23>/Clock' */
 rtb_temp145 = ssGetT(S);

 /* Lookup: '<S23>/Look-Up Table' */

 SteadyStateSystem_B->Look_Up_Table_b =
  rt_Lookup(SteadyStateSystem_P->Look_Up_Table_b_XData, 7, rtb_temp145,
  SteadyStateSystem_P->Look_Up_Table_b_YData);
}

if (ssIsContinuousTask(S, tid)) {    /* Sample time: [0.0, 0.0] */

 /* Switch: '<S19>/Switch3' */
 if (SteadyStateSystem_B->C4_a) {

  {
   /* simstruct variables */
   SteadyStateSystem_BlockIO *SteadyStateSystem_B =
    (SteadyStateSystem_BlockIO *) _ssGetBlockIO(S);
   SteadyStateSystem_Parameters *SteadyStateSystem_P =
    (SteadyStateSystem_Parameters *) ssGetDefaultParam(S);

   /* Switch: '<S3>/Switch' */
   if (SteadyStateSystem_B->Constant1_c >=
    SteadyStateSystem_P->Switch_a_Threshold) {
    SteadyStateSystem_B->Switch_a = rtb_Switch3_a;
   } else {
    SteadyStateSystem_B->Switch_a = SteadyStateSystem_B->Constant5_a;
   }
  }

  SteadyStateSystem_B->Switch3_b = SteadyStateSystem_B->Switch_a;
 } else {
  SteadyStateSystem_B->Switch3_b = SteadyStateSystem_B->Look_Up_Table_b;
 }

 /* Derivative Block: <S23>/Derivative */
 {
  real_T t = ssGetTaskTime(S,tid);
  real_T timeStampA = SteadyStateSystem_DWork->Derivative_a_RWORK.TimeStampA;
  real_T timeStampB = SteadyStateSystem_DWork->Derivative_a_RWORK.TimeStampB;

  if (timeStampA >= t && timeStampB >= t) {
   SteadyStateSystem_B->Derivative_a = 0.0;
  } else {
   real_T deltaT;
   real_T *lastBank =
    &SteadyStateSystem_DWork->Derivative_a_RWORK.TimeStampA;
   if (timeStampA < timeStampB) {
    if (timeStampB < t) {
     lastBank += 2;
    }
   } else if (timeStampA >= t) {
    lastBank += 2;
```

```
    }
    deltaT = t - *lastBank++;
    SteadyStateSystem_B->Derivative_a =
      (SteadyStateSystem_B->Look_Up_Table_b - *lastBank++) / deltaT;
   }
  }
 }

 if (ssIsSampleHit(S, 1, tid)) {      /* Sample time: [0.0, 1.0] */

  /* Constant: '<S20>/Constant1' */
  SteadyStateSystem_B->Constant1_d = SteadyStateSystem_P->Constant1_d_Value;

  /* Constant: '<S20>/C4' */
  SteadyStateSystem_B->C4_c = SteadyStateSystem_P->C4_c_Value;

  /* Constant: '<S3>/Constant2' */
  SteadyStateSystem_B->Constant2_b = SteadyStateSystem_P->Constant2_b_Value;
 }

 if (ssIsContinuousTask(S, tid)) {    /* Sample time: [0.0, 0.0] */

  /* Clock: '<S24>/Clock' */
  rtb_temp145 = ssGetT(S);

  /* Lookup: '<S24>/Look-Up Table' */

  SteadyStateSystem_B->Look_Up_Table_c =
    rt_Lookup(SteadyStateSystem_P->Look_Up_Table_c_XData, 7, rtb_temp145,
    SteadyStateSystem_P->Look_Up_Table_c_YData);
 }

 if (ssIsContinuousTask(S, tid)) {    /* Sample time: [0.0, 0.0] */

  /* Switch: '<S20>/Switch3' */
  if (SteadyStateSystem_B->C4_c) {

   {
    /* simstruct variables */
    SteadyStateSystem_BlockIO *SteadyStateSystem_B =
      (SteadyStateSystem_BlockIO *) _ssGetBlockIO(S);
    SteadyStateSystem_Parameters *SteadyStateSystem_P =
      (SteadyStateSystem_Parameters *) ssGetDefaultParam(S);

    /* Switch: '<S3>/Switch1' */
    if (SteadyStateSystem_B->Constant2_b >=
     SteadyStateSystem_P->Switch1_a_Threshold) {
     SteadyStateSystem_B->Switch1_a = rtb_Switch3_a;
    } else {
     SteadyStateSystem_B->Switch1_a = SteadyStateSystem_B->Constant5_a;
    }
   }

   SteadyStateSystem_B->Switch3_c = SteadyStateSystem_B->Switch1_a;
  } else {
   SteadyStateSystem_B->Switch3_c = SteadyStateSystem_B->Look_Up_Table_c;
  }

  /* Derivative Block: <S24>/Derivative */
  {
   real_T t = ssGetTaskTime(S,tid);
   real_T timeStampA = SteadyStateSystem_DWork->Derivative_b_RWORK.TimeStampA;
   real_T timeStampB = SteadyStateSystem_DWork->Derivative_b_RWORK.TimeStampB;

   if (timeStampA >= t && timeStampB >= t) {
    SteadyStateSystem_B->Derivative_b = 0.0;
   } else {
    real_T deltaT;
    real_T *lastBank =
     &SteadyStateSystem_DWork->Derivative_b_RWORK.TimeStampA;
    if (timeStampA < timeStampB) {
     if (timeStampB < t) {
```

```
        lastBank += 2;
       }
      } else if (timeStampA >= t) {
       lastBank += 2;
      }
      deltaT = t - *lastBank++;
      SteadyStateSystem_B->Derivative_b =
       (SteadyStateSystem_B->Look_Up_Table_c - *lastBank++) / deltaT;
    }
   }
  }

  if (ssIsSampleHit(S, 1, tid)) {      /* Sample time: [0.0, 1.0] */

   /* Constant: '<S21>/Constant1' */
   SteadyStateSystem_B->Constant1_e = SteadyStateSystem_P->Constant1_e_Value;

   /* Constant: '<S21>/C4' */
   SteadyStateSystem_B->C4_d = SteadyStateSystem_P->C4_d_Value;

   /* Constant: '<S3>/Constant3' */
   SteadyStateSystem_B->Constant3_b = SteadyStateSystem_P->Constant3_b_Value;
  }

  if (ssIsContinuousTask(S, tid)) {    /* Sample time: [0.0, 0.0] */

   /* Clock: '<S25>/Clock' */
   rtb_temp145 = ssGetT(S);

   /* Lookup: '<S25>/Look-Up Table' */

   SteadyStateSystem_B->Look_Up_Table_d =
    rt_Lookup(SteadyStateSystem_P->Look_Up_Table_d_XData, 7, rtb_temp145,
     SteadyStateSystem_P->Look_Up_Table_d_YData);
  }

  if (ssIsContinuousTask(S, tid)) {    /* Sample time: [0.0, 0.0] */

   /* Switch: '<S21>/Switch3' */
   if (SteadyStateSystem_B->C4_d) {

    {
     /* simstruct variables */
     SteadyStateSystem_BlockIO *SteadyStateSystem_B =
      (SteadyStateSystem_BlockIO *) _ssGetBlockIO(S);
     SteadyStateSystem_Parameters *SteadyStateSystem_P =
      (SteadyStateSystem_Parameters *) ssGetDefaultParam(S);

     /* Switch: '<S3>/Switch2' */
     if (SteadyStateSystem_B->Constant3_b >=
      SteadyStateSystem_P->Switch2_a_Threshold) {
      SteadyStateSystem_B->Switch2_a = rtb_Switch3_a;
     } else {
      SteadyStateSystem_B->Switch2_a = SteadyStateSystem_B->Constant5_a;
     }
    }

    SteadyStateSystem_B->Switch3_d = SteadyStateSystem_B->Switch2_a;
   } else {
    SteadyStateSystem_B->Switch3_d = SteadyStateSystem_B->Look_Up_Table_d;
   }

   /* Derivative Block: <S25>/Derivative */
   {
    real_T t = ssGetTaskTime(S,tid);
    real_T timeStampA = SteadyStateSystem_DWork->Derivative_c_RWORK.TimeStampA;
    real_T timeStampB = SteadyStateSystem_DWork->Derivative_c_RWORK.TimeStampB;

    if (timeStampA >= t && timeStampB >= t) {
     SteadyStateSystem_B->Derivative_c = 0.0;
    } else {
     real_T deltaT;
```

```
    real_T *lastBank =
      &SteadyStateSystem_DWork->Derivative_c_RWORK.TimeStampA;
    if (timeStampA < timeStampB) {
      if (timeStampB < t) {
        lastBank += 2;
      }
    } else if (timeStampA >= t) {
      lastBank += 2;
    }
    deltaT = t - *lastBank++;
    SteadyStateSystem_B->Derivative_c =
      (SteadyStateSystem_B->Look_Up_Table_d - *lastBank++) / deltaT;
  }
 }

 /* Derivative Block: <S22>/Derivative */
 {
  real_T t = ssGetTaskTime(S,tid);
  real_T timeStampA = SteadyStateSystem_DWork->Derivative_d_RWORK.TimeStampA;
  real_T timeStampB = SteadyStateSystem_DWork->Derivative_d_RWORK.TimeStampB;

  if (timeStampA >= t && timeStampB >= t) {
    SteadyStateSystem_B->Derivative_d = 0.0;
  } else {
    real_T deltaT;
    real_T *lastBank =
      &SteadyStateSystem_DWork->Derivative_d_RWORK.TimeStampA;
    if (timeStampA < timeStampB) {
      if (timeStampB < t) {
        lastBank += 2;
      }
    } else if (timeStampA >= t) {
      lastBank += 2;
    }
    deltaT = t - *lastBank++;
    SteadyStateSystem_B->Derivative_d =
      (SteadyStateSystem_B->Look_Up_Table_a - *lastBank++) / deltaT;
  }
 }
}

if (ssIsSampleHit(S, 1, tid)) {    /* Sample time: [0.0, 1.0] */

 /* Constant: '<S26>/Constant1' */
 SteadyStateSystem_B->Constant1_f = SteadyStateSystem_P->Constant1_f_Value;

 /* Constant: '<S26>/C4' */
 SteadyStateSystem_B->C4_e = SteadyStateSystem_P->C4_e_Value;

 /* Constant: '<S4>/com' */
 SteadyStateSystem_B->com_b = SteadyStateSystem_P->com_b_Value;

 /* Constant: '<S4>/C4' */
 SteadyStateSystem_B->C4_f = SteadyStateSystem_P->C4_f_Value;
}

if (ssIsContinuousTask(S, tid)) {    /* Sample time: [0.0, 0.0] */

 /* Clock: '<S29>/Clock' */
 rtb_temp145 = ssGetT(S);

 /* Lookup: '<S29>/Look-Up Table' */

 SteadyStateSystem_B->Look_Up_Table_e =
  rt_Lookup(SteadyStateSystem_P->Look_Up_Table_e_XData, 3, rtb_temp145,
   SteadyStateSystem_P->Look_Up_Table_e_YData);

 /* Switch: '<S4>/Switch3' */
 if (SteadyStateSystem_B->C4_f >= SteadyStateSystem_P->Switch3_e_Threshold) {

  rtb_Switch3_e = SteadyStateSystem_B->com_b;
 } else {
```

```
  rtb_Switch3_e = SteadyStateSystem_B->Look_Up_Table_e;
 }
}

if (ssIsSampleHit(S, 1, tid)) {      /* Sample time: [0.0, 1.0] */

 /* Constant: '<S4>/Constant1' */
 SteadyStateSystem_B->Constant1_g = SteadyStateSystem_P->Constant1_g_Value;

 /* Constant: '<S4>/Constant5' */
 SteadyStateSystem_B->Constant5_b = SteadyStateSystem_P->Constant5_b_Value;
}

if (ssIsContinuousTask(S, tid)) {    /* Sample time: [0.0, 0.0] */

 /* Clock: '<S30>/Clock' */
 rtb_temp145 = ssGetT(S);

 /* Lookup: '<S30>/Look-Up Table' */

 SteadyStateSystem_B->Look_Up_Table_f =
  rt_Lookup(SteadyStateSystem_P->Look_Up_Table_f_XData, 7, rtb_temp145,
  SteadyStateSystem_P->Look_Up_Table_f_YData);
}

if (ssIsContinuousTask(S, tid)) {    /* Sample time: [0.0, 0.0] */

 /* Switch: '<S26>/Switch3' */
 if (SteadyStateSystem_B->C4_e) {

  {
   /* simstruct variables */
   SteadyStateSystem_BlockIO *SteadyStateSystem_B =
    (SteadyStateSystem_BlockIO *) _ssGetBlockIO(S);
   SteadyStateSystem_Parameters *SteadyStateSystem_P =
    (SteadyStateSystem_Parameters *) ssGetDefaultParam(S);

   /* Switch: '<S4>/Switch' */
   if (SteadyStateSystem_B->Constant1_g >=
    SteadyStateSystem_P->Switch_b_Threshold) {
    SteadyStateSystem_B->Switch_b = rtb_Switch3_e;
   } else {
    SteadyStateSystem_B->Switch_b = SteadyStateSystem_B->Constant5_b;
   }
  }

  SteadyStateSystem_B->Switch3_f = SteadyStateSystem_B->Switch_b;
 } else {
  SteadyStateSystem_B->Switch3_f = SteadyStateSystem_B->Look_Up_Table_f;
 }

 /* Derivative Block: <S30>/Derivative */
 {
  real_T t = ssGetTaskTime(S,tid);
  real_T timeStampA = SteadyStateSystem_DWork->Derivative_e_RWORK.TimeStampA;
  real_T timeStampB = SteadyStateSystem_DWork->Derivative_e_RWORK.TimeStampB;

  if (timeStampA >= t && timeStampB >= t) {
   SteadyStateSystem_B->Derivative_e = 0.0;
  } else {
   real_T deltaT;
   real_T *lastBank =
    &SteadyStateSystem_DWork->Derivative_e_RWORK.TimeStampA;
   if (timeStampA < timeStampB) {
    if (timeStampB < t) {
     lastBank += 2;
    }
   } else if (timeStampA >= t) {
    lastBank += 2;
   }
   deltaT = t - *lastBank++;
   SteadyStateSystem_B->Derivative_e =
```

```
      (SteadyStateSystem_B->Look_Up_Table_f - *lastBank++) / deltaT;
    }
  }
}

if (ssIsSampleHit(S, 1, tid)) {      /* Sample time: [0.0, 1.0] */

 /* Constant: '<S27>/Constant1' */
 SteadyStateSystem_B->Constant1_h = SteadyStateSystem_P->Constant1_h_Value;

 /* Constant: '<S27>/C4' */
 SteadyStateSystem_B->C4_g = SteadyStateSystem_P->C4_g_Value;

 /* Constant: '<S4>/Constant2' */
 SteadyStateSystem_B->Constant2_c = SteadyStateSystem_P->Constant2_c_Value;
}

if (ssIsContinuousTask(S, tid)) {    /* Sample time: [0.0, 0.0] */

 /* Clock: '<S31>/Clock' */
 rtb_temp145 = ssGetT(S);

 /* Lookup: '<S31>/Look-Up Table' */

 SteadyStateSystem_B->Look_Up_Table_g =
  rt_Lookup(SteadyStateSystem_P->Look_Up_Table_g_XData, 7, rtb_temp145,
  SteadyStateSystem_P->Look_Up_Table_g_YData);
}

if (ssIsContinuousTask(S, tid)) {    /* Sample time: [0.0, 0.0] */

 /* Switch: '<S27>/Switch3' */
 if (SteadyStateSystem_B->C4_g) {

  {
   /* simstruct variables */
   SteadyStateSystem_BlockIO *SteadyStateSystem_B =
    (SteadyStateSystem_BlockIO *) _ssGetBlockIO(S);
   SteadyStateSystem_Parameters *SteadyStateSystem_P =
    (SteadyStateSystem_Parameters *) ssGetDefaultParam(S);

   /* Switch: '<S4>/Switch1' */
   if (SteadyStateSystem_B->Constant2_c >=
    SteadyStateSystem_P->Switch1_b_Threshold) {
    SteadyStateSystem_B->Switch1_b = rtb_Switch3_e;
   } else {
    SteadyStateSystem_B->Switch1_b = SteadyStateSystem_B->Constant5_b;
   }
  }

  SteadyStateSystem_B->Switch3_g = SteadyStateSystem_B->Switch1_b;
 } else {
  SteadyStateSystem_B->Switch3_g = SteadyStateSystem_B->Look_Up_Table_g;
 }

 /* Derivative Block: <S31>/Derivative */
 {
  real_T t = ssGetTaskTime(S,tid);
  real_T timeStampA = SteadyStateSystem_DWork->Derivative_f_RWORK.TimeStampA;
  real_T timeStampB = SteadyStateSystem_DWork->Derivative_f_RWORK.TimeStampB;

  if (timeStampA >= t && timeStampB >= t) {
   SteadyStateSystem_B->Derivative_f = 0.0;
  } else {
   real_T deltaT;
   real_T *lastBank =
    &SteadyStateSystem_DWork->Derivative_f_RWORK.TimeStampA;
   if (timeStampA < timeStampB) {
    if (timeStampB < t) {
     lastBank += 2;
    }
   } else if (timeStampA >= t) {
```

```
      lastBank += 2;
    }
    deltaT = t - *lastBank++;
    SteadyStateSystem_B->Derivative_f =
      (SteadyStateSystem_B->Look_Up_Table_g - *lastBank++) / deltaT;
  }
 }
}

if (ssIsSampleHit(S, 1, tid)) {      /* Sample time: [0.0, 1.0] */

 /* Constant: '<S28>/Constant1' */
 SteadyStateSystem_B->Constant1_i = SteadyStateSystem_P->Constant1_i_Value;

 /* Constant: '<S28>/C4' */
 SteadyStateSystem_B->C4_h = SteadyStateSystem_P->C4_h_Value;

 /* Constant: '<S4>/Constant3' */
 SteadyStateSystem_B->Constant3_c = SteadyStateSystem_P->Constant3_c_Value;
}

if (ssIsContinuousTask(S, tid)) {    /* Sample time: [0.0, 0.0] */

 /* Clock: '<S32>/Clock' */
 rtb_temp145 = ssGetT(S);

 /* Lookup: '<S32>/Look-Up Table' */

 SteadyStateSystem_B->Look_Up_Table_h =
  rt_Lookup(SteadyStateSystem_P->Look_Up_Table_h_XData, 7, rtb_temp145,
   SteadyStateSystem_P->Look_Up_Table_h_YData);
}

if (ssIsContinuousTask(S, tid)) {    /* Sample time: [0.0, 0.0] */

 /* Switch: '<S28>/Switch3' */
 if (SteadyStateSystem_B->C4_h) {

  {
   /* simstruct variables */
   SteadyStateSystem_BlockIO *SteadyStateSystem_B =
    (SteadyStateSystem_BlockIO *) _ssGetBlockIO(S);
   SteadyStateSystem_Parameters *SteadyStateSystem_P =
    (SteadyStateSystem_Parameters *) ssGetDefaultParam(S);

   /* Switch: '<S4>/Switch2' */
   if (SteadyStateSystem_B->Constant3_c >=
    SteadyStateSystem_P->Switch2_b_Threshold) {
    SteadyStateSystem_B->Switch2_b = rtb_Switch3_e;
   } else {
    SteadyStateSystem_B->Switch2_b = SteadyStateSystem_B->Constant5_b;
   }
  }

  SteadyStateSystem_B->Switch3_h = SteadyStateSystem_B->Switch2_b;
 } else {
  SteadyStateSystem_B->Switch3_h = SteadyStateSystem_B->Look_Up_Table_h;
 }

 /* Derivative Block: <S32>/Derivative */
 {
  real_T t = ssGetTaskTime(S,tid);
  real_T timeStampA = SteadyStateSystem_DWork->Derivative_g_RWORK.TimeStampA;
  real_T timeStampB = SteadyStateSystem_DWork->Derivative_g_RWORK.TimeStampB;

  if (timeStampA >= t && timeStampB >= t) {
   SteadyStateSystem_B->Derivative_g = 0.0;
  } else {
   real_T deltaT;
   real_T *lastBank =
    &SteadyStateSystem_DWork->Derivative_g_RWORK.TimeStampA;
   if (timeStampA < timeStampB) {
```

```
    if (timeStampB < t) {
      lastBank += 2;
    }
   } else if (timeStampA >= t) {
    lastBank += 2;
   }
   deltaT = t - *lastBank++;
   SteadyStateSystem_B->Derivative_g =
    (SteadyStateSystem_B->Look_Up_Table_h - *lastBank++) / deltaT;
  }
 }

 /* Derivative Block: <S29>/Derivative */
 {
  real_T t = ssGetTaskTime(S,tid);
  real_T timeStampA = SteadyStateSystem_DWork->Derivative_h_RWORK.TimeStampA;
  real_T timeStampB = SteadyStateSystem_DWork->Derivative_h_RWORK.TimeStampB;

  if (timeStampA >= t && timeStampB >= t) {
   SteadyStateSystem_B->Derivative_h = 0.0;
  } else {
   real_T deltaT;
   real_T *lastBank =
    &SteadyStateSystem_DWork->Derivative_h_RWORK.TimeStampA;
   if (timeStampA < timeStampB) {
    if (timeStampB < t) {
     lastBank += 2;
    }
   } else if (timeStampA >= t) {
    lastBank += 2;
   }
   deltaT = t - *lastBank++;
   SteadyStateSystem_B->Derivative_h =
    (SteadyStateSystem_B->Look_Up_Table_e - *lastBank++) / deltaT;
  }
 }
}

if (ssIsSampleHit(S, 1, tid)) {      /* Sample time: [0.0, 1.0] */

 /* Constant: '<S5>/Constant' */
 SteadyStateSystem_B->Constant_i = SteadyStateSystem_P->Constant_i_Value;

 /* Constant: '<S5>/Constant1' */
 SteadyStateSystem_B->Constant1_j = SteadyStateSystem_P->Constant1_j_Value;

 /* Constant: '<S5>/Constant2' */
 SteadyStateSystem_B->Constant2_d = SteadyStateSystem_P->Constant2_d_Value;

 /* Constant: '<S5>/Constant3' */
 SteadyStateSystem_B->Constant3_d = SteadyStateSystem_P->Constant3_d_Value;

 /* Constant: '<S5>/N' */
 SteadyStateSystem_B->N_b = SteadyStateSystem_P->N_b_Value;

 /* Constant: '<S6>/Constant' */
 SteadyStateSystem_B->Constant_j = SteadyStateSystem_P->Constant_j_Value;

 /* Constant: '<S6>/Constant1' */
 SteadyStateSystem_B->Constant1_k = SteadyStateSystem_P->Constant1_k_Value;

 /* Constant: '<S6>/Constant2' */
 SteadyStateSystem_B->Constant2_e = SteadyStateSystem_P->Constant2_e_Value;

 /* Constant: '<S6>/Simulation method' */
 SteadyStateSystem_B->Simulation_method_a =
  SteadyStateSystem_P->Simulation_method_a_Value;
}

/* SubSystem: '<S6>/Phasor Measurements' */
SteadyStateSystem_Phasor(S, tid, SteadyStateSystem_B->Simulation_method_a,
 &SteadyStateSystem_B->State_Space[8], &SteadyStateSystem_B->State_Space[18],
```

```
&SteadyStateSystem_DWork->Phasor_Mea_a, (rtP_SteadyStateSystem_Phasor *)
&SteadyStateSystem_P->Phasor_Mea_a);

if (ssIsSampleHit(S, 1, tid)) {      /* Sample time: [0.0, 1.0] */

 /* Constant: '<S7>/Constant' */
 SteadyStateSystem_B->Constant_k = SteadyStateSystem_P->Constant_k_Value;

 /* Constant: '<S7>/Constant1' */
 SteadyStateSystem_B->Constant1_l = SteadyStateSystem_P->Constant1_l_Value;

 /* Constant: '<S7>/Constant2' */
 SteadyStateSystem_B->Constant2_f = SteadyStateSystem_P->Constant2_f_Value;

 /* Constant: '<S7>/Simulation method' */
 SteadyStateSystem_B->Simulation_method_b =
   SteadyStateSystem_P->Simulation_method_b_Value;
}

/* SubSystem: '<S7>/Phasor Measurements' */
SteadyStateSystem_Phasor(S, tid, SteadyStateSystem_B->Simulation_method_b,
 &SteadyStateSystem_B->State_Space[11], &SteadyStateSystem_B->State_Space[21],
 &SteadyStateSystem_DWork->Phasor_Mea_b, (rtP_SteadyStateSystem_Phasor *)
 &SteadyStateSystem_P->Phasor_Mea_b);

if (ssIsSampleHit(S, 1, tid)) {      /* Sample time: [0.0, 1.0] */

 /* Constant: '<S8>/Constant' */
 SteadyStateSystem_B->Constant_l = SteadyStateSystem_P->Constant_l_Value;

 /* Constant: '<S8>/Constant1' */
 SteadyStateSystem_B->Constant1_m = SteadyStateSystem_P->Constant1_m_Value;

 /* Constant: '<S8>/Constant2' */
 SteadyStateSystem_B->Constant2_g = SteadyStateSystem_P->Constant2_g_Value;

 /* Constant: '<S8>/Simulation method' */
 SteadyStateSystem_B->Simulation_method_c =
   SteadyStateSystem_P->Simulation_method_c_Value;
}

/* SubSystem: '<S8>/Phasor Measurements' */
SteadyStateSystem_Phasor(S, tid, SteadyStateSystem_B->Simulation_method_c,
 &SteadyStateSystem_B->State_Space[14], &SteadyStateSystem_B->State_Space[24],
 &SteadyStateSystem_DWork->Phasor_Mea_c, (rtP_SteadyStateSystem_Phasor *)
 &SteadyStateSystem_P->Phasor_Mea_c);

if (ssIsSampleHit(S, 1, tid)) {      /* Sample time: [0.0, 1.0] */

 /* Constant: '<S10>/Simulation method' */
 SteadyStateSystem_B->Simulation_method_d =
   SteadyStateSystem_P->Simulation_method_d_Value;
}

/* SubSystem: '<S10>/Phasor Measurements' */

/* Output and update for enable system: '<S10>/Phasor Measurements' */
{
 /* simstruct variables */
 SteadyStateSystem_BlockIO *SteadyStateSystem_B = (SteadyStateSystem_BlockIO
   *) _ssGetBlockIO(S);
 SteadyStateSystem_D_Work *SteadyStateSystem_DWork =
   (SteadyStateSystem_D_Work *) ssGetRootDWork(S);
 SteadyStateSystem_Parameters *SteadyStateSystem_P =
   (SteadyStateSystem_Parameters *) ssGetDefaultParam(S);

 /* detect enable/disable transitions */
 if (ssIsSampleHit(S, 1, tid)) {
   EnableStates prevEnableState = (EnableStates)
     SteadyStateSystem_DWork->Phasor_Mea_d_MODE[0];
   EnableStates enableState;
```

Appendix

```
if (ssIsSampleHit(S, 1, tid)) {  /* Sample time: [0.0, 1.0] */
 SteadyStateSystem_DWork->Phasor_Mea_d_MODE[1] =
  (SteadyStateSystem_B->Simulation_method_d > 0.0) ? SUBSYS_ENABLED :
   SUBSYS_DISABLED;
}

enableState = (EnableStates) SteadyStateSystem_DWork->Phasor_Mea_d_MODE[1];
if (enableState == SUBSYS_ENABLED) {
 if (prevEnableState == SUBSYS_DISABLED) {
  /* SUBSYS_BECOMING_ENABLED */

  if ( ssGetT(S) != ssGetTStart(S) ) {
   ssSetSolverNeedsReset(S);
  }
  /* (system enable function is empty) */
  SteadyStateSystem_DWork->Phasor_Mea_d_MODE[0] = (int_T) SUBSYS_ENABLED;
 }
} else {
 if (prevEnableState == SUBSYS_ENABLED) {
  /* SUBSYS_BECOMING_DISABLED */
  ssSetSolverNeedsReset(S);

  /* (system disable function is empty) */
  SteadyStateSystem_DWork->Phasor_Mea_d_MODE[0] = (int_T)
   SUBSYS_DISABLED;
 }
}

/* run blocks if enabled */
if (SteadyStateSystem_DWork->Phasor_Mea_d_MODE[0] == SUBSYS_ENABLED) {
 if (ssIsContinuousTask(S, tid)) {

  /* ComplexToMagnitudeAngle: '<S51>/Complex to Magnitude-Angle' */
  rtb_Complex_to_Magnitude_Angle =
   fabs(SteadyStateSystem_B->State_Space[17]);
  if (SteadyStateSystem_B->State_Space[17] >= 0.0) {
   rtb_temp144 = 0.0;
  } else {
   rtb_temp144 = RT_PI;
  }

  /* Gain: '<S51>/180//pi'
   *
   * Regarding '<S51>/180//pi':
   *  Gain value: SteadyStateSystem_P->pi_Gain
   */
  rtb_temp144 *= SteadyStateSystem_P->pi_Gain;
 }}
}

if (ssIsContinuousTask(S, tid)) {   /* Sample time: [0.0, 0.0] */

 /* Fcn: '<S57>/Fcn2'
  *
  * Regarding '<S57>/Fcn2':
  *  Expression: (u[4]*(2*u[1]+u[2]) + (sqrt3*u[2]*u[3])) * one_third
  */
 rtb_Fcn2 = ( rtb_Elementary_Math1 * ( 2.0 *
  SteadyStateSystem_B->State_Space[6] +
  SteadyStateSystem_B->State_Space[7] ) + 1.7320508075688772E+000 *
  SteadyStateSystem_B->State_Space[7] * rtb_Elementary_Math ) *
  3.3333333333333331E-001;

 /* Fcn: '<S57>/Fcn3'
  *
  * Regarding '<S57>/Fcn3':
  *  Expression: (u[3]*(2*u[1]+u[2]) + (-sqrt3*u[2]*u[4])) * one_third
  */
 rtb_Fcn3 = ( rtb_Elementary_Math * ( 2.0 *
  SteadyStateSystem_B->State_Space[6] +
  SteadyStateSystem_B->State_Space[7] ) + ( -1.7320508075688772E+000) *
```

```
SteadyStateSystem_B->State_Space[7] * rtb_Elementary_Math1 ) *
3.3333333333333331E-001;

/* Gain: '<S59>/Gain'
 *
 * Regarding '<S59>/Gain':
 *  Gain value: SteadyStateSystem_P->Gain_b_Gain
 */
SteadyStateSystem_B->Gain_b = rtb_Int * SteadyStateSystem_P->Gain_b_Gain;

/* Sum: '<S61>/Sum' incorporates:
 *  Gain: '<S61>/R//Ld'
 *  Gain: '<S61>/1//Ld'
 *  Gain: '<S61>/Lq//Ld'
 *  Product: '<S61>/Product'
 *
 * Regarding '<S61>/R//Ld':
 *  Gain value: SteadyStateSystem_P->R_Ld_Gain
 *
 * Regarding '<S61>/1//Ld':
 *  Gain value: SteadyStateSystem_P->Ld_Gain
 *
 * Regarding '<S61>/Lq//Ld':
 *  Gain value: SteadyStateSystem_P->Lq_Ld_Gain
 */
SteadyStateSystem_B->Sum_b = - (rtb_id_a * SteadyStateSystem_P->R_Ld_Gain)
  + (rtb_Fcn3 * SteadyStateSystem_P->Ld_Gain)
  + ((SteadyStateSystem_B->Gain_b * rtb_iq) *
  SteadyStateSystem_P->Lq_Ld_Gain);

/* Sum: '<S62>/Sum1' incorporates:
 *  Gain: '<S62>/R//Lq'
 *  Gain: '<S62>/Ld//Lq'
 *  Product: '<S62>/Product1'
 *  Gain: '<S62>/lam//Lq'
 *  Gain: '<S62>/1//Lq'
 *
 * Regarding '<S62>/R//Lq':
 *  Gain value: SteadyStateSystem_P->R_Lq_Gain
 *
 * Regarding '<S62>/Ld//Lq':
 *  Gain value: SteadyStateSystem_P->Ld_Lq_Gain
 *
 * Regarding '<S62>/lam//Lq':
 *  Gain value: SteadyStateSystem_P->lam_Lq_Gain
 *
 * Regarding '<S62>/1//Lq':
 *  Gain value: SteadyStateSystem_P->Lq_Gain
 */
SteadyStateSystem_B->Sum1 = - (rtb_iq * SteadyStateSystem_P->R_Lq_Gain)
  - ((rtb_id_a * SteadyStateSystem_B->Gain_b) *
  SteadyStateSystem_P->Ld_Lq_Gain)
  - (SteadyStateSystem_B->Gain_b * SteadyStateSystem_P->lam_Lq_Gain)
  + (rtb_Fcn2 * SteadyStateSystem_P->Lq_Gain);

/* Gain: '<S59>/Gain1'
 *
 * Regarding '<S59>/Gain1':
 *  Gain value: SteadyStateSystem_P->Gain1_b_Gain
 */
rtb_temp143 *= SteadyStateSystem_P->Gain1_b_Gain;

/* Gain: '<S59>/Gain2' incorporates:
 *  Sum: '<S59>/Sum'
 *  Gain: '<S59>/Gain3'
 *
 * Regarding '<S59>/Gain2':
 *  Gain value: SteadyStateSystem_P->Gain2_Gain
 *
 * Regarding '<S59>/Gain3':
 *  Gain value: SteadyStateSystem_P->Gain3_Gain
 */
```

```
 SteadyStateSystem_B->Gain2 = (-SteadyStateSystem_B->Gain1_a +
  SteadyStateSystem_B->Te - (rtb_Int * SteadyStateSystem_P->Gain3_Gain)) *
  SteadyStateSystem_P->Gain2_Gain;
}

if (ssIsSampleHit(S, 1, tid)) {      /* Sample time: [0.0, 1.0] */

 /* Level2 S-Function Block: <S64>/S function (sfun_psbbreaker) */
 /* Call into Simulink for MEX-version of S-function */
 ssCallAccelRunBlock(S, 10, 158, SS_CALL_MDL_OUTPUTS);
}

if (ssIsContinuousTask(S, tid)) {    /* Sample time: [0.0, 0.0] */

 /* Gain: '<S65>/R_device' incorporates:
  *  Sum: '<S65>/Vak-Vf-Ron*Iak'
  *
  * Regarding '<S65>/R_device':
  *  Gain value: SteadyStateSystem_P->R_device_Gain
  */
 {
  int_T i1;

  real_T *y0 = &SteadyStateSystem_B->R_device[0];

  for (i1=0; i1 < 6; i1++) {
   y0[i1] = (SteadyStateSystem_B->status[i1] -
    SteadyStateSystem_B->Integrator[i1]) *
    SteadyStateSystem_P->R_device_Gain;
  }
 }
}

if (ssIsSampleHit(S, 1, tid)) {      /* Sample time: [0.0, 1.0] */

 /* Constant: '<S10>/Constant' */
 SteadyStateSystem_B->Constant_m = SteadyStateSystem_P->Constant_m_Value;

 /* Constant: '<S72>/Constant' */
 SteadyStateSystem_B->Constant_n = SteadyStateSystem_P->Constant_n_Value;

 /* Constant: '<S73>/Constant' */
 SteadyStateSystem_B->Constant_o = SteadyStateSystem_P->Constant_o_Value;

 /* Constant: '<S74>/Constant' */
 SteadyStateSystem_B->Constant_p = SteadyStateSystem_P->Constant_p_Value;

 /* Constant: '<S75>/padding' */
 {
  int_T i1;

  real_T *y0 = &SteadyStateSystem_B->padding[0];
  const real_T *p_padding_Value = &SteadyStateSystem_P->padding_Value[0];

  for (i1=0; i1 < 10; i1++) {
   y0[i1] = p_padding_Value[i1];
  }
 }

 /* Constant: '<S75>/lastStatus' */
 {
  int_T i1;

  real_T *y0 = &SteadyStateSystem_B->lastStatus[0];
  const real_T *p_lastStatus_Value =
   &SteadyStateSystem_P->lastStatus_Value[0];

  for (i1=0; i1 < 15; i1++) {
   y0[i1] = p_lastStatus_Value[i1];
  }
 }
```

```
  /* Constant: '<S77>/g1' */
  SteadyStateSystem_B->g1_a = SteadyStateSystem_P->g1_a_Value;

  /* Constant: '<S77>/g2' */
  SteadyStateSystem_B->g2_a = SteadyStateSystem_P->g2_a_Value;

  /* Constant: '<S77>/g3' */
  SteadyStateSystem_B->g3_a = SteadyStateSystem_P->g3_a_Value;

  /* Constant: '<S78>/g1' */
  SteadyStateSystem_B->g1_b = SteadyStateSystem_P->g1_b_Value;

  /* Constant: '<S78>/g2' */
  SteadyStateSystem_B->g2_b = SteadyStateSystem_P->g2_b_Value;

  /* Constant: '<S78>/g3' */
  SteadyStateSystem_B->g3_b = SteadyStateSystem_P->g3_b_Value;

  /* Constant: '<S79>/g1' */
  SteadyStateSystem_B->g1_c = SteadyStateSystem_P->g1_c_Value;

  /* Constant: '<S79>/g2' */
  SteadyStateSystem_B->g2_c = SteadyStateSystem_P->g2_c_Value;

  /* Constant: '<S79>/g3' */
  SteadyStateSystem_B->g3_c = SteadyStateSystem_P->g3_c_Value;
 }
}

/* Update for root system: '<Root>' */
#define MDL_UPDATE
static void mdlUpdate(SimStruct *S, int_T tid)
{

 /* simstruct variables */
 SteadyStateSystem_BlockIO *SteadyStateSystem_B = (SteadyStateSystem_BlockIO *)
  _ssGetBlockIO(S);
 SteadyStateSystem_ContinuousStates *SteadyStateSystem_X =
  (SteadyStateSystem_ContinuousStates*) ssGetContStates(S);
 SteadyStateSystem_D_Work *SteadyStateSystem_DWork = (SteadyStateSystem_D_Work
  *) ssGetRootDWork(S);
 SteadyStateSystem_Parameters *SteadyStateSystem_P =
  (SteadyStateSystem_Parameters *) ssGetDefaultParam(S);

 if (ssIsContinuousTask(S, tid)) {    /* Sample time: [0.0, 0.0] */

  /* Limited Integrator Block: <S65>/Integrator */
  {
   enum {INTG_NORMAL, INTG_LEAVING_UPPER_SAT, INTG_LEAVING_LOWER_SAT,
    INTG_UPPER_SAT, INTG_LOWER_SAT };

   {
    int_T i1;

    const real_T *u0 = &SteadyStateSystem_B->R_device[0];
    int_T *mode = &SteadyStateSystem_DWork->Integrator_MODE[0];
    real_T *xc = &SteadyStateSystem_X->Integrator_CSTATE[0];

    for (i1=0; i1 < 6; i1++) {
     if (xc[i1] == SteadyStateSystem_P->Integrator_UpperSat) {
      switch(mode[i1]) {
       case INTG_UPPER_SAT:
        if (u0[i1] < 0.0) {
         ssSetSolverNeedsReset(S);
         mode[i1] = INTG_LEAVING_UPPER_SAT;
        }
        break;
       case INTG_LEAVING_UPPER_SAT:
        if (u0[i1] >= 0.0) {
         mode[i1] = INTG_UPPER_SAT;
         ssSetSolverNeedsReset(S);
        }
```

```
      break;
     default:
      ssSetSolverNeedsReset(S);
      if (u0[i1] < 0.0) {
       mode[i1] = INTG_LEAVING_UPPER_SAT;
      } else {
       mode[i1] = INTG_UPPER_SAT;
      }
      break;
    }
   } else if (xc[i1] == SteadyStateSystem_P->Integrator_LowerSat) {
    switch(mode[i1]) {
     case INTG_LOWER_SAT:
      if (u0[i1] > 0.0) {
       ssSetSolverNeedsReset(S);
       mode[i1] = INTG_LEAVING_LOWER_SAT;
      }
      break;
     case INTG_LEAVING_LOWER_SAT:
      if (u0[i1] <= 0.0) {
       mode[i1] = INTG_LOWER_SAT;
       ssSetSolverNeedsReset(S);
      }
      break;
     default:
      ssSetSolverNeedsReset(S);
      if (u0[i1] > 0.0) {
       mode[i1] = INTG_LEAVING_LOWER_SAT;
      } else {
       mode[i1] = INTG_LOWER_SAT;
      }
      break;
     }
    } else {
     mode[i1] = INTG_NORMAL;
    }
   }
  }
 }

 /* Derivative Block: <S23>/Derivative */
 {
  real_T timeStampA = SteadyStateSystem_DWork->Derivative_a_RWORK.TimeStampA;
  real_T timeStampB = SteadyStateSystem_DWork->Derivative_a_RWORK.TimeStampB;
  real_T *lastBank = &SteadyStateSystem_DWork->Derivative_a_RWORK.TimeStampA;

  if (timeStampA != rtInf) {
   if (timeStampB == rtInf) {
    lastBank += 2;
   } else if (timeStampA >= timeStampB) {
    lastBank += 2;
   }
  }
  *lastBank++ = ssGetTaskTime(S,tid);
  *lastBank++ = SteadyStateSystem_B->Look_Up_Table_b;
 }

 /* Derivative Block: <S24>/Derivative */
 {
  real_T timeStampA = SteadyStateSystem_DWork->Derivative_b_RWORK.TimeStampA;
  real_T timeStampB = SteadyStateSystem_DWork->Derivative_b_RWORK.TimeStampB;
  real_T *lastBank = &SteadyStateSystem_DWork->Derivative_b_RWORK.TimeStampA;

  if (timeStampA != rtInf) {
   if (timeStampB == rtInf) {
    lastBank += 2;
   } else if (timeStampA >= timeStampB) {
    lastBank += 2;
   }
  }
  *lastBank++ = ssGetTaskTime(S,tid);
  *lastBank++ = SteadyStateSystem_B->Look_Up_Table_c;
```

Appendix

```
}

/* Derivative Block: <S25>/Derivative */
{
 real_T timeStampA = SteadyStateSystem_DWork->Derivative_c_RWORK.TimeStampA;
 real_T timeStampB = SteadyStateSystem_DWork->Derivative_c_RWORK.TimeStampB;
 real_T *lastBank = &SteadyStateSystem_DWork->Derivative_c_RWORK.TimeStampA;

 if (timeStampA != rtInf) {
  if (timeStampB == rtInf) {
   lastBank += 2;
  } else if (timeStampA >= timeStampB) {
   lastBank += 2;
  }
 }
 *lastBank++ = ssGetTaskTime(S,tid);
 *lastBank++ = SteadyStateSystem_B->Look_Up_Table_d;
}

/* Derivative Block: <S22>/Derivative */
{
 real_T timeStampA = SteadyStateSystem_DWork->Derivative_d_RWORK.TimeStampA;
 real_T timeStampB = SteadyStateSystem_DWork->Derivative_d_RWORK.TimeStampB;
 real_T *lastBank = &SteadyStateSystem_DWork->Derivative_d_RWORK.TimeStampA;

 if (timeStampA != rtInf) {
  if (timeStampB == rtInf) {
   lastBank += 2;
  } else if (timeStampA >= timeStampB) {
   lastBank += 2;
  }
 }
 *lastBank++ = ssGetTaskTime(S,tid);
 *lastBank++ = SteadyStateSystem_B->Look_Up_Table_a;
}

/* Derivative Block: <S30>/Derivative */
{
 real_T timeStampA = SteadyStateSystem_DWork->Derivative_e_RWORK.TimeStampA;
 real_T timeStampB = SteadyStateSystem_DWork->Derivative_e_RWORK.TimeStampB;
 real_T *lastBank = &SteadyStateSystem_DWork->Derivative_e_RWORK.TimeStampA;

 if (timeStampA != rtInf) {
  if (timeStampB == rtInf) {
   lastBank += 2;
  } else if (timeStampA >= timeStampB) {
   lastBank += 2;
  }
 }
 *lastBank++ = ssGetTaskTime(S,tid);
 *lastBank++ = SteadyStateSystem_B->Look_Up_Table_f;
}

/* Derivative Block: <S31>/Derivative */
{
 real_T timeStampA = SteadyStateSystem_DWork->Derivative_f_RWORK.TimeStampA;
 real_T timeStampB = SteadyStateSystem_DWork->Derivative_f_RWORK.TimeStampB;
 real_T *lastBank = &SteadyStateSystem_DWork->Derivative_f_RWORK.TimeStampA;

 if (timeStampA != rtInf) {
  if (timeStampB == rtInf) {
   lastBank += 2;
  } else if (timeStampA >= timeStampB) {
   lastBank += 2;
  }
 }
 *lastBank++ = ssGetTaskTime(S,tid);
 *lastBank++ = SteadyStateSystem_B->Look_Up_Table_g;
}

/* Derivative Block: <S32>/Derivative */
{
```

```
  real_T timeStampA = SteadyStateSystem_DWork->Derivative_g_RWORK.TimeStampA;
  real_T timeStampB = SteadyStateSystem_DWork->Derivative_g_RWORK.TimeStampB;
  real_T *lastBank = &SteadyStateSystem_DWork->Derivative_g_RWORK.TimeStampA;

  if (timeStampA != rtInf) {
   if (timeStampB == rtInf) {
    lastBank += 2;
   } else if (timeStampA >= timeStampB) {
    lastBank += 2;
   }
  }
  *lastBank++ = ssGetTaskTime(S,tid);
  *lastBank++ = SteadyStateSystem_B->Look_Up_Table_h;
 }

 /* Derivative Block: <S29>/Derivative */
 {
  real_T timeStampA = SteadyStateSystem_DWork->Derivative_h_RWORK.TimeStampA;
  real_T timeStampB = SteadyStateSystem_DWork->Derivative_h_RWORK.TimeStampB;
  real_T *lastBank = &SteadyStateSystem_DWork->Derivative_h_RWORK.TimeStampA;

  if (timeStampA != rtInf) {
   if (timeStampB == rtInf) {
    lastBank += 2;
   } else if (timeStampA >= timeStampB) {
    lastBank += 2;
   }
  }
  *lastBank++ = ssGetTaskTime(S,tid);
  *lastBank++ = SteadyStateSystem_B->Look_Up_Table_e;
 }
 }
}

/* Derivatives for root system: '<Root>' */
#define MDL_DERIVATIVES
static void mdlDerivatives(SimStruct *S)
{
 /* simstruct variables */
 SteadyStateSystem_BlockIO *SteadyStateSystem_B = (SteadyStateSystem_BlockIO *)
  _ssGetBlockIO(S);
 SteadyStateSystem_StateDerivatives *SteadyStateSystem_Xdot =
  (SteadyStateSystem_StateDerivatives*) ssGetdX(S);
 SteadyStateSystem_StateDisabled *SteadyStateSystem_Xdis =
  (SteadyStateSystem_StateDisabled*) ssGetContStateDisabled(S);
 SteadyStateSystem_D_Work *SteadyStateSystem_DWork = (SteadyStateSystem_D_Work
  *) ssGetRootDWork(S);

 /* Integrator Block: <S59>/Int */
 {

  SteadyStateSystem_Xdot->Int_CSTATE = SteadyStateSystem_B->Gain2;
 }

 /* Integrator Block: <S59>/Int1 */
 {

  SteadyStateSystem_Xdot->Int1_CSTATE = SteadyStateSystem_B->Gain_b;
 }

 /* Integrator Block: <S62>/iq */
 {

  SteadyStateSystem_Xdot->iq_CSTATE = SteadyStateSystem_B->Sum1;
 }

 /* Integrator Block: <S61>/id */
 {

  SteadyStateSystem_Xdot->id_a_CSTATE = SteadyStateSystem_B->Sum_b;
 }
```

Appendix

```
/* Limited Integrator Block: <S65>/Integrator */
{
 enum {INTG_NORMAL, INTG_LEAVING_UPPER_SAT, INTG_LEAVING_LOWER_SAT,
  INTG_UPPER_SAT, INTG_LOWER_SAT};

 {
  int_T i1;

  const real_T *u0 = &SteadyStateSystem_B->R_device[0];
  int_T *mode = &SteadyStateSystem_DWork->Integrator_MODE[0];
  real_T *xdot = &SteadyStateSystem_Xdot->Integrator_CSTATE[0];
  boolean_T *xdis = &SteadyStateSystem_Xdis->Integrator_CSTATE[0];

  for (i1=0; i1 < 6; i1++) {
   if ((mode[i1] != INTG_UPPER_SAT)
    && (mode[i1] != INTG_LOWER_SAT) ) {
    xdot[i1] = u0[i1];
    xdis[i1] = FALSE;
   } else {
    /* in saturation */
    xdot[i1] = 0.0;
    if ((mode[i1] == INTG_UPPER_SAT) || (mode[i1] == INTG_LOWER_SAT)) {
     xdis[i1] = TRUE;
    }
   }
  }
 }

 /* Level2 S-Function Block: <S52>/State-Space (sfun_psbcontc) */
 /* Call into Simulink for MEX-version of S-function */
 ssCallAccelRunBlock(S, 10, 25, SS_CALL_MDL_DERIVATIVES);
}

/* ZeroCrossings for root system: '<Root>' */
#define MDL_ZERO_CROSSINGS
static void mdlZeroCrossings(SimStruct *S)
{
 /* simstruct variables */
 SteadyStateSystem_BlockIO *SteadyStateSystem_B = (SteadyStateSystem_BlockIO *)
  _ssGetBlockIO(S);
 SteadyStateSystem_ContinuousStates *SteadyStateSystem_X =
  (SteadyStateSystem_ContinuousStates*) ssGetContStates(S);
 SteadyStateSystem_D_Work *SteadyStateSystem_DWork = (SteadyStateSystem_D_Work
  *) ssGetRootDWork(S);
 SteadyStateSystem_Parameters *SteadyStateSystem_P =
  (SteadyStateSystem_Parameters *) ssGetDefaultParam(S);
 SteadyStateSystem_NonsampledZCs *SteadyStateSystem_NonsampledZC =
  (SteadyStateSystem_NonsampledZCs *) ssGetNonsampledZCs(S);

 /* Limited Integrator Block: <S65>/Integrator */
 {
  enum {INTG_NORMAL, INTG_LEAVING_UPPER_SAT, INTG_LEAVING_LOWER_SAT,
   INTG_UPPER_SAT, INTG_LOWER_SAT};

  /* zero crossings for limited integrator */
  {
   int_T i1;

   real_T *nszc = &SteadyStateSystem_NonsampledZC->Integrator_NSZC[0];
   real_T *xc = &SteadyStateSystem_X->Integrator_CSTATE[0];

   int_T *mode = &SteadyStateSystem_DWork->Integrator_MODE[0];

   for (i1=0; i1 < 6; i1++) {
    if (mode[i1] == INTG_LEAVING_UPPER_SAT &&
     xc[i1] >= SteadyStateSystem_P->Integrator_UpperSat) {
     nszc[(2 * i1) + 0] = 0.0;
    } else {
     nszc[(2 * i1) + 0] = xc[i1] - SteadyStateSystem_P->Integrator_UpperSat;
    }
    if (mode[i1] == INTG_LEAVING_LOWER_SAT &&
```

```
    xc[i1] <= SteadyStateSystem_P->Integrator_LowerSat) {
     nszc[(2 * i1) + 1] = 0.0;
    } else {
     nszc[(2 * i1) + 1] = xc[i1] - SteadyStateSystem_P->Integrator_LowerSat;
    }
   }
  }

/* zero crossings for input of limited integrator */
  {

   {
    int_T i1;

    const real_T *u0 = &SteadyStateSystem_B->R_device[0];
    real_T *nszc = &SteadyStateSystem_NonsampledZC->Integrator_NSZC[0];
    int_T *mode = &SteadyStateSystem_DWork->Integrator_MODE[0];

    for (i1=0; i1 < 6; i1++) {
     if ((mode[i1] == INTG_UPPER_SAT) ||
      (mode[i1] == INTG_LOWER_SAT)) {
       nszc[i1 + 12] = u0[i1];
      } else {
       nszc[i1 + 12] = 0.0;
      }
     }
    }
   }
  }

/* RelationalOperator Block: <S65>/Relational Operator1 */
  {
   int_T i1;

   const real_T *u0 = &SteadyStateSystem_B->Integrator[0];
   real_T *nszc = &SteadyStateSystem_NonsampledZC->Relational_Operator1_NSZC[0];

   for (i1=0; i1 < 6; i1++) {
    nszc[i1] = u0[i1] - SteadyStateSystem_B->id_b;
   }
  }

/* HitCross Block: '<S23>/Hit  Crossing' */
SteadyStateSystem_NonsampledZC->Hit_Crossing_a_NSZC =
  SteadyStateSystem_B->Derivative_a -
  SteadyStateSystem_P->Hit_Crossing_a_Offset;

/* HitCross Block: '<S24>/Hit  Crossing' */
SteadyStateSystem_NonsampledZC->Hit_Crossing_b_NSZC =
  SteadyStateSystem_B->Derivative_b -
  SteadyStateSystem_P->Hit_Crossing_b_Offset;

/* HitCross Block: '<S25>/Hit  Crossing' */
SteadyStateSystem_NonsampledZC->Hit_Crossing_c_NSZC =
  SteadyStateSystem_B->Derivative_c -
  SteadyStateSystem_P->Hit_Crossing_c_Offset;

/* HitCross Block: '<S22>/Hit  Crossing' */
SteadyStateSystem_NonsampledZC->Hit_Crossing_d_NSZC =
  SteadyStateSystem_B->Derivative_d -
  SteadyStateSystem_P->Hit_Crossing_d_Offset;

/* HitCross Block: '<S30>/Hit  Crossing' */
SteadyStateSystem_NonsampledZC->Hit_Crossing_e_NSZC =
  SteadyStateSystem_B->Derivative_e -
  SteadyStateSystem_P->Hit_Crossing_e_Offset;

/* HitCross Block: '<S31>/Hit  Crossing' */
SteadyStateSystem_NonsampledZC->Hit_Crossing_f_NSZC =
  SteadyStateSystem_B->Derivative_f -
  SteadyStateSystem_P->Hit_Crossing_f_Offset;
```

Appendix

```
/* HitCross Block: '<S32>/Hit  Crossing' */
SteadyStateSystem_NonsampledZC->Hit_Crossing_g_NSZC =
  SteadyStateSystem_B->Derivative_g -
  SteadyStateSystem_P->Hit_Crossing_g_Offset;

/* HitCross Block: '<S29>/Hit  Crossing' */
SteadyStateSystem_NonsampledZC->Hit_Crossing_h_NSZC =
  SteadyStateSystem_B->Derivative_h -
  SteadyStateSystem_P->Hit_Crossing_h_Offset;

/* Level2 S-Function Block: <S64>/S function (sfun_psbbreaker) */
/* Call into Simulink for MEX-version of S-function */
ssCallAccelRunBlock(S, 10, 158, SS_CALL_MDL_ZERO_CROSSINGS);
}

/* Function to initialize sizes */
static void mdlInitializeSizes(SimStruct *S)
{

  /* checksum */
  ssSetChecksumVal(S, 0, 691194347U);
  ssSetChecksumVal(S, 1, 3836388576U);
  ssSetChecksumVal(S, 2, 4250063101U);
  ssSetChecksumVal(S, 3, 3138451366U);

  /* options */
  ssSetOptions(S, SS_OPTION_EXCEPTION_FREE_CODE);

  /* Accelerator check memory map size match for DWork */
  if (ssGetSizeofDWork(S) != sizeof(SteadyStateSystem_D_Work)) {
    ssSetErrorStatus(S,"Unexpected error: Internal DWork sizes do "
      "not match for accelerator mex file.");
  }

  /* Accelerator check memory map size match for BlockIO */
  if (ssGetSizeofGlobalBlockIO(S) != sizeof(SteadyStateSystem_BlockIO)) {
    ssSetErrorStatus(S,"Unexpected error: Internal BlockIO sizes do "
      "not match for accelerator mex file.");
  }

  /* model parameters */
  _ssSetDefaultParam(S, (real_T *) &SteadyStateSystem_DefaultParameters);

  /* non-finites */
  rtInf = mxGetInf();
  rtMinusInf = -mxGetInf();

  /* Non-finite (run-time) assignments */
  {
    /* simstruct variables */
    SteadyStateSystem_Parameters *SteadyStateSystem_P =
      (SteadyStateSystem_Parameters *) ssGetDefaultParam(S);

    SteadyStateSystem_P->Integrator_LowerSat = rtMinusInf;
  }
}

/* Empty mdlInitializeSampleTimes function (never called) */
static void mdlInitializeSampleTimes(SimStruct *S) { }

/* Empty mdlTerminate function (never called) */
static void mdlTerminate(SimStruct *S) { }

/* MATLAB MEX Glue */
#include "simulink.c"
```