



**Design and Development of an
Intelligent Embedded System for
Bearing fault Detection in
Submersible pumps**



A Project Report

Submitted by

I. RAJARAJESWARI - 71206415005

P - 2313



*in partial fulfillment for the award of the degree
of*

**Master of Engineering
in**

Power Electronics and Drives

**DEPARTMENT OF ELECTRICAL & ELECTRONICS
ENGINEERING**

**KUMARAGURU COLLEGE OF TECHNOLOGY
COIMBATORE - 641 006**

ANNA UNIVERSITY: CHENNAI 600 025

2007 - 2008

BONAFIDE CERTIFICATE

Certified that this project report entitled “Design and development of an intelligent embedded system for bearing fault detection in submersible pumps” is the bonafide work of

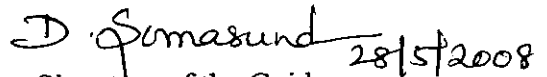
Miss I. Rajarajeswari - Register No. 71206415005

Who carried out the project work under my supervision.



Signature of the Head of the Department

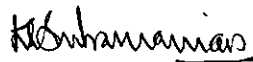
Prof.K.Regupathy Subramanian



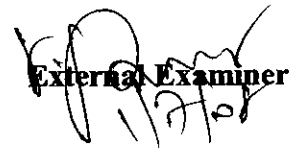
Signature of the Guide

Mrs. D. Somasundareswari

Certified that the candidate with university Register No: 71206415005 was examined in Project viva- voce examination held on ...1..7..2008..



Internal Examiner



External Examiner

**DEPARTMENT OF ELECTRICAL & ELECTRONICS
ENGINEERING
KUMARAGURU COLLEGE OF TECHNOLOGY
COIMBATORE - 641 006**



HINDUSTHAN COLLEGE OF ENGINEERING AND TECHNOLOGY
COIMBATORE -32



DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING
NATIONAL CONFERENCE ON EMERGING TRENDS IN COMMUNICATION SYSTEMS

2nd April 2008

CERTIFICATE OF PARTICIPATION

This is to certify that ~~Mr.~~Ms. I. RAJARAJESWARI, P. G. SCHOKAR

KUMARAGURU COLLEGE OF TECHNOLOGY, COIMBATORE

has participated in NCETCS-2008 and presented a paper entitled, "DESIGN AND DEVELOPMENT OF AN INTELLIGENT MEMBERSHIP FOR BEARING PAULT DETECTION IN SUPERGRIDS" Co-Authors. Ms. D. SOMASUNDARESWARI, Dr. V. DURAISAMY

N. Mohan
Prof. N. Mohanansundram
Coordinator

Dr. V. Duraisamy
Principal

Mahendra Submersible Pumps (P) Ltd.

498-5, Kalabatty Road, Coimbatore - 641 014

Ph: 0426-283007, 2827097 Fax: 0426-2313177

E-Mail: mahendra.pumps@eth.net Web Site: www.mahendrapumps.com

Date : 17.5.2008

TO WHOMEVER IT MAY CONCERN

This is to certify that the following final year M.E., (Power Electronics & Drives) student of KUMARAGURU COLLEGE OF TECHNOLOGY, COIMBATORE, has completed a project in ***DESIGN AND DEVELOPMENT OF AN INTELLIGENT EMBEDDED SYSTEM FOR BEARING FAULT DETECTION IN SUBMERSIBLE PUMPS*** in our concern for a period from Dec 2007 to Apr 2008.

I. RAJARAJESWARI

71206415005

During this period, she has shown interest in conducting the tests required and has done a good work towards completion of the project.

Mahendra Submersible Pumps (P) Ltd



Jayakumar Ramdass

Managing Director

**From parched brown to fertile green ...**

Sales Branches at : Bangalore, Calicut, Indore, Jabalpur, Jaipur, Kollam, Nagpur, Pune, Raipur, Ranchi, Secunderabad, Trichur, Vijayawada.

ABSTRACT

Submersible pumps are used on both open well and in bore wells for agricultural purposes. In particular during summer season, the yield from both open well as well as bore well pumps will be poor. But they are operated frequently without knowing the water level present in the well. This will cause winding fault and bearing fault in the motor. These faults if left undetected, may lead to the degradation and eventual failure of motors. The results of many studies show that bearing problem account for over 40% of all machine failures. Hence it is always essential to check the motor condition from time to time. This can be done by various conventional methods, but these methods require down time of motor. Among the methods that can be employed for fault detection, intelligent embedded techniques offer a good solution due to their capacity of handling numerical and heuristic information.

This project aims at the implementation of intelligent embedded techniques for monitoring motor current patterns in a submersible pump. The fault signature extracted from the motor current patterns by using Fast Fourier Transformation is used for the prediction of motor bearing condition. In addition, the motor is protected against abnormal voltage and current conditions. The insulation condition is also checked. Soft computing techniques like neural network and fuzzy logic are implemented to simulate the pump bearing fault diagnosis based on the extracted information features. The experimental results are presented. Since this scheme detects the faults at their earlier stage, the maintenance can be carried out in organized manner, which reduces the down time and repairing cost. This approach is validated in a 1 HP 230V 50HZ 2880-rpm single-phase submersible pump.

ஆய்வுச்சுருக்கம்

சப்மர்சிபிள் பம்புகள் பெரும்பாலும் விவசாயத்திற்காக திறந்தவெளி மற்றும் அழ்துளை கிணறுகளில் பயன்படுத்தப்படுகிறது. குறிப்பாக கோடை காலங்களில் இவற்றிலிருந்து கிடைக்கும் நீரின் அளவு மிகவும் குறைந்து விடுகிறது. ஆனால் சப்மர்சிபிள் பம்புகள் இதனை அறியாமல் இயக்கப்படுகிறது. இவற்றால் வைண்டிங் மற்றும் பேரிங்குகள் பழுதாகிறது. இப்பழுதானது கண்டறியப்படாவிட்டால், மோட்டாரை செயலிழக்கச்செய்து விடும். மெஷின்களில் ஏற்படும் பெரும்பாலான பழுதுகளில் 40% விழுக்காடு பேரிங் பழுதினால் ஏற்படுகிறது என்று பல்வேறு ஆய்வுகளின் மூலம் உறுதிசெய்யப்பட்டுள்ளது. எனவே அவ்வப்போது மோட்டாரின் தன்மையை சோதிப்பது அவசியமாகிறது. இவற்றை பலமுறைகளில் செயல்படுத்தும்போது பழுதைக் கண்டறிய தாமதமாகிறது. இவ்வாறான பழுதுகளை கண்டறியும் முறைகளில், இன்டெலிஜென்ட் எம்பட்ட் முறையானது சிறந்த பலனைத்தருகிறது.

இந்த ஆய்வில் மோட்டார் மின் அதிர்வுகளை கண்டறிய இன்டெலிஜென்ட் எம்பட்ட் முறை பயன்படுத்தப்பட்டுள்ளது. பழுதினால் ஏற்படும் மின் அதிர்வுகளை பாஸ்ட் ஸ்போரியர் மாற்றுதல் மூலம் கண்டறிப்படுவதால் பேரிங் பழுதினை எளிதாக கணிக்கமுடிகிறது. மேலும் இம்முறையில் மோட்டாரானது அசாதாரண மின் அழுத்தம் மற்றும் கரண்ட் நிலையிலிருந்தும் பாதுகாக்கப்படுகிறது. மோட்டாரின் இன்சுலேசன் தன்மையையும் சோதிக்கலாம். பழுதுகளை ஆய்வு செய்ய நியூரல் வலைப்பின்னல் மற்றும் ஃபஸ்ஸி முறையும் பயன்படுத்தப்பட்டுள்ளது. இம்முறையால் பழுதுதினை முன்கூட்டியே அறிய முடிவதால் பராமரிப்பு செலவு மிகவும் குறைக்கப்படுகிறது. இந்த ஆய்வானது ஒரு குதிரை திறன் கொண்ட சப்மர்சிபிள் பம்பின் மூலம் செயல்படுத்தப்பட்டுள்ளது.

ACKNOWLEDGEMENT

The satisfaction that accompanies the successful completion of any task would be incomplete without the mention of people who made it possible, whose constant guidance and encouragement crowns all efforts with success.

I express my deep sense of gratitude to our beloved Principal **Dr. Joseph V. Thanikal**, B.E., M.E., Ph.D., PDF, CEPIT, for providing all the facilities to carry out this project in a successful manner.

I take pleasure in thanking **Prof. K. Regupathy Subramanian**, B.E (Hons), M.Sc., MIEEE, IES, Head of the Department, Electrical and Electronics Engineering, Kumaraguru College of Technology, for his constant encouragement throughout the project.

I am highly indebted to my guide **Mrs. D. Somasundareswari**, M.E., AMIE, MSSSI, MISTE, Senior Lecturer, Department of Electrical and Electronics Engineering, Kumaraguru College of Technology, who guided me throughout the project with her invaluable suggestions and encouraged successfully for the completion of this project.

I express my sincere thanks to **Dr.V.Duraisamy**, M.E., Ph.D., MISTE, AMIE, MIEEE, Principal, Hindusthan College of Engineering and Technology, Coimbatore, for providing me valuable suggestions and immense support through out this project.

I wish to express my sincere thanks to “**The Institution of Engineers – India**”, **Kolkata** for sponsoring to this project under R & D Grant. Grant No.SCK/T-R&D/26/2007-2008.

I wish to express my sincere thanks to **Sri. Jayakumar Ramdas**, Managing Director, M/s. Mahendra Submersible Pumps Pvt Ltd., for providing the necessary testing facilities. I also thank **Mr.S.Ramakrishnan**, Electrical Engineer, M/s. Mahendra Submersible Pumps Pvt Ltd., for providing technical support.

I am also grateful to the dynamic support of all our staff and friends, who helped me in many ways during the course of this project.

CONTENTS

Title	Page No.
Bonafide certificate	ii
Proof of publication	iii
Company Certificate	iv
Abstract in English	v
Abstract in Tamil	vi
Acknowledgement	vii
Contents	viii
List of Tables	xi
List of Figures	xii
Abbreviations	xiv

CHAPTER1 INTRODUCTION

1.1 Bearing Faults in Submersible Pumps	1
1.2 Need for a Monitoring System	1
1.3 Objective	2
1.4 Bearing Structural Defects	2
1.5 Literature Survey	3
1.6 Organisation of the thesis	4

CHAPTER 2 EXPERIMENTAL SETUP FOR DATA ACQUISITION

2.1 Fault Detection Scheme	6
2.2 Experimental Hardware Setup	6
2.3 Experimental Results	7
2.3.1 Monitoring stator current spectrum	7
2.3.2 Monitoring stator current and rotor speed	21

CHAPTER 3 NEURAL NETWORK BASED FAULT DIAGNOSIS

3.1 Introduction to Neural Network	22
3.1.1 Neuron Model	23

3.1.2	Activation Functions	24
3.1.3	Learning Rules	26
3.2	Back-Propagation Neural Network	26
3.2.1	Choice of Parameters for Network Training	29
3.2.2	Learning Rate	29
3.2.3	Momentum Factor	30
3.3	Structure of BP Network for Fault Detection	30
3.3.1	Fault Detection by Monitoring Stator Current Spectrum	30
3.3.2.	Simulation Results	31
3.3.3.	Fault Detection by Monitoring Stator Current and Rotor Speed	32
3.3.4	Simulation Results	33
CHAPTER 4	FUZZY LOGIC BASED FAULT DIAGNOSIS	
4.1	Introduction	34
4.2	Mamdani Fuzzy Logic Inference System	35
4.2.1	Fuzzifier	36
4.2.2	Fuzzy Rules	37
4.2.3	Inference Engine	38
4.2.4	Defuzzifier	38
4.3	Fuzzy Logic Based Fault Detection Method	39
4.4	Simulation Results	42
4.5	Comparison of Neural Network and Fuzzy Logic Based Fault Diagnosis System	42
CHAPTER 5	HARDWARE IMPLEMENTATION	
5.1	Hardware Description	45
5.2	Block Diagram	46
5.3	Voltage Sensing Circuit	46
5.4	Current Sensing Circuit	47
5.5	Leakage Current Sensing Circuit	47
5.6	Speed Sensing Unit	48
5.7	Relay and Buzzer Circuit	49
5.8	Introduction to dsPIC	49

5.8.1 dsPIC30f4013	49
5.8.2 High-Performance Modified RISC CPU	49
5.8.3 DSP Features	50
5.8.4 Peripheral Features	50
5.8.5 Analog Features	51
5.8.6 Special Micro controller Features	51
5.8.7 CMOS Technology	51
5.8.8 dsPIC30F4013 – Pin Configuration	52
5.8.9 CPU Architecture	52
5.8.10 Status Register	53
5.8.11 Program Counter	53
5.8.12 Memory Organization	53
5.8.13 Flash Program Memory	53
5.8.14 I/O Ports	54
5.8.15 Interrupts	54
5.8.16 Timer2/3 Module	54
5.8.17 PC Module	54
5.8.18 12-Bit Analog-To-Digital Converter Module	54
5.8.19 SPI Module	55
5.8.20 LCD Module	55
5.8.21 MPLAB ICD 2 In-Circuit Debugger	55
5.8.22 MPLAB C18 and MPLAB C30C Compilers	56
5.8.23 MPLAB ASM30 Assembler, Linker and Librarian	56
5.8.24 MPLAB SIM Software Simulator	56
5.9 Software Description	58
5.9.1 Flow chart	58
CONCLUSION	59
REFERENCES	60
APPENDIX A Development Board	62
APPENDIX B dsPIC Programming	69

LIST OF TABLES

Table	Title	Page No.
2.1	Submersible pump specification	7
2.2	Experimental results for healthy bearings	10
2.3	Experimental results for faulty bearings	15
2.4	Experimental results by monitoring stator current and speed	21
3.1	Simulation results	33
4.1	Fuzzy Rules	39
4.2	Membership functions	40
4.3	Simulation results	42
4.4	Comparison of Neural network and fuzzy logic based fault diagnosis	43

LIST OF FIGURES

Figure	Title	Page No.
1.1	Faults in submersible pumps	1
1.2	Bush bearing in healthy condition	3
1.3	Thrust bearing in healthy condition	3
1.4	Thrust bearing in faulty condition	3
2.1	Experimental setup for data acquisition	6
2.2	Photographic view of the setup	7
2.3	Voltage spectrum for healthy bearing	8
2.4	Current spectrum for healthy bearings	8
2.5	Current spectrum for faulty bearings	8
2.6	FFT waveform for healthy bearings	9
2.7	FFT waveform for healthy bearings	9
2.8	FFT waveform for faulty bearings	9
3.1	Structure of biological neuron	22
3.2	Single – Input Neuron without Bias	23
3.3	Single Input Neuron with Bias	24
3.4	Hard Limit Activation Function	24
3.5	Linear Activation Function	25
3.6	Log-Sigmoid Activation Function	25
3.7	Architecture of BP Neural Network	27
3.8	Structure of BP Network for Fault Detection	31
3.9	Epoch Vs Error Characteristics for healthy bearings	31
3.10	Epoch Vs Error characteristics for faulty bearings	32
3.11	Structure of BP Network for Fault Detection	32
3.12	Epoch Vs Error characteristics	33
4.1	Mamdani Fuzzy Logic Inference Systems	35
4.2	Triangular membership functions	37
4.3	Graphical interpretation of fuzzification, inference	38
4.4	Centroid Defuzzification Method	39

4.5	Input Membership Functions for current	40
4.6	Input Membership Functions for speed	41
4.7	Output Membership Functions for temperature	41
4.8	Surface Viewer	41
5.1	Schematic diagram	44
5.2	+5V and +12V power supply	45
5.3	+9V power supply	45
5.4	Fault diagnostic module	46
5.5	Voltage sensing circuit	47
5.6	Current sensing circuit	47
5.7	Leakage current sensing circuit	48
5.8	Speed sensing unit	48
5.9	Relay and Buzzer circuit	49
5.10	Pin Configuration	52
5.11	Photograph of the fault diagnostic module	57
5.12	Flow chart	58

ABBREVIATIONS

dsPIC	digital signal Peripheral Interface Controller
dsc	digital signal controller
FFT	Fast Fourier Transform
PC	Personal Computer
MCSA	Motor Current Signature Analysis
CT	Current Transformer
PT	Potential Transformer
RPM	Revolutions per Minute
HP	Horse power
Hz	Hertz
LCD	Liquid Crystal Display
AC	Alternating Current
DC	Direct Current
A	Amperes
LED	Light Emitting Diode
ADC	Analog to Digital Converter
NNFD	Neural Network based Fault Diagnosis
FFD	Fuzzy Logic based Fault Diagnosis
Sub	Submersible pump
trimf	Triangular Membership Function
smf	S-type Membership Function
zmf	Z-type Membership Function

CHAPTER 1

INTRODUCTION

Submersible pumps find tremendous applications in the field of irrigation engineering and in dewatering. In submersible pump sets, both pump and motor are installed deep inside the tube well. There are many techniques and commercially available tools to monitor submersible pumps to insure a high degree of reliability uptime. Environmental, duty, and installation issues may combine to accelerate motor and pump failure far sooner than the designed lifetimes. It is apparent then that a failure monitoring system should be capable of extracting, in a consistent manner, the evidence of many possible failures from measurements from physically different sensors. Therefore condition-monitoring schemes have concentrated on specific failure modes such as the stator, the rotor or the bearings. Most of the researches are suggested towards electrical monitoring of the motor through stator current.

1.1 BEARING FAULTS IN SUBMERSIBLE PUMPS

Bearings play an important role in the reliability and performances of all pump sets. Most of the faults arising in pumps are related to bearing faults. It is stated that about 40% of faults occurring in submersible pumps are due to bearing failures. The different types of faults occurring and its percentage are shown in the figure 1.1

PERCENTAGE OF FAULTS

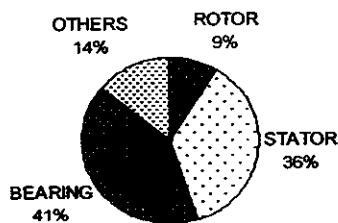


Figure 1.1 Faults in submersible pumps

1.2 NEED FOR A MONITORING SYSTEM

Condition monitoring in submersible pumps is gaining importance due to the need to increase reliability and to decrease the production loss due to breakdown. By comparing the stator current spectrum of a submersible pump running under healthy and faulty conditions, detection of faults like mass imbalance, rotor rub, shaft misalignment and bearing defects is possible. These signals can also be used to detect the incipient failures of the machine

components, through the online monitoring system, reducing the possibility of catastrophic damage and the downtime. Although often the visual inspection of the frequency domain features of the measured signals is adequate to identify the faults, there is a need for a reliable, fast, and automated procedure of diagnostics. Artificial intelligence techniques like Neural Fuzzy techniques can be implemented in the system for automated detection and diagnosis of machine conditions.

1.3 OBJECTIVE

To design an intelligent embedded fault diagnostic module for on line condition monitoring of submersible pumps.

1.4 BEARING STRUCTURAL DEFECTS

In submersible pump sets, both pump and motor are installed deep inside the tube well. The complete unit consisting of pump and motor is suspended in the bore vertically from the discharge rising pipe. Two types of bearings are used in submersible pumps. They are 1) Bush bearings 2) thrust bearing. The motor is a squirrel cage induction type with water lubricated bush bearings. They consist of well-insulated windings with pure clean cold water surrounding the windings. The axial thrust of the pump is taken up by the thrust bearing. The pump bearings are water lubricated.

In particular during summer season, the yield from both open well as well as bore well pumps will be poor. But they are operated frequently without knowing the water level present in the well. This will cause winding fault and bearing fault in the motor. These faults if left undetected, may lead to the degradation and eventual failure of motors. There are many other ways, which reduce the time of bearing failure. These include contamination, corrosion, improper lubrication and improper installation.

Bearing corrosion is produced by the presence of sand or by careless handling during installations. Installation problems are often caused by improperly forcing the bearing onto the shaft or in the housing. This produces physical damage, which leads to premature failure. Misalignment of the bearing is also a common result of defective bearing installation. Figures 1.2 and 1.3 show the bush bearing and thrust bearing in healthy condition. Figures 1.4 shows the thrust bearing in faulty conditions.

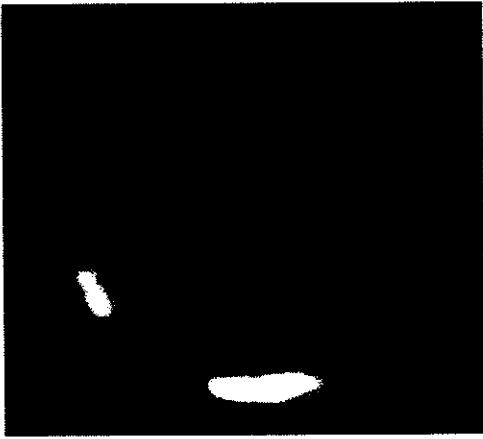


Figure 1.2 Bush bearing in healthy condition

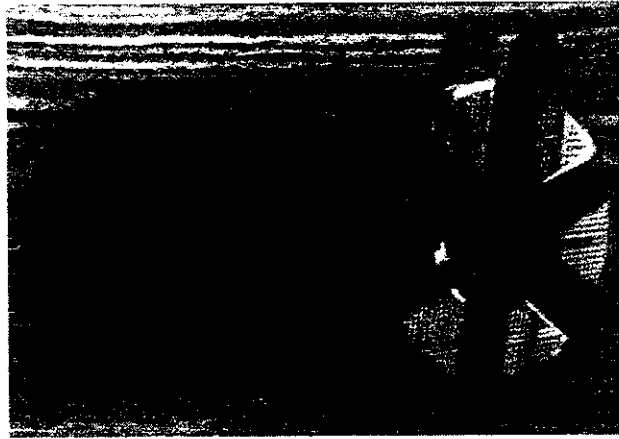


Figure 1.3 Thrust bearing in healthy condition

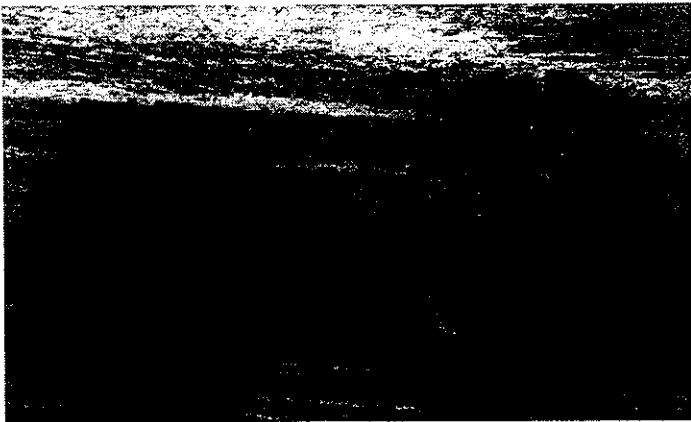


Figure 1.4 Thrust bearing in faulty condition

1.5 LITERATURE SURVEY

Induction machines are widely used in submersible pumps from fractional horsepower to many horsepower ratings. On line monitoring of electrical machines in critical applications has been increasingly necessary to improve their reliability and to minimize failure. These machines are operated at different environmental conditions. This causes incipient faults in the rotor winding and bearing. Bearings play an important role in the reliability and performance of all motor system. As pointed out by Peter Vas (1993), various types of AI-based bearing fault monitoring systems in induction motors are used. They are

- Monitoring stator current (r.m.s value) and the rotor speed;
- Vibration signals;
- Monitoring stator current spectrum

There are many published techniques and many commercially available tools to monitor induction motors to insure a high degree of reliability uptime. In spite of these tools, many companies are still faced with unexpected system failures and reduced motor lifetime. Most of the faults arising in motors are often linked with bearing faults. In general, condition-monitoring schemes have concentrated on sensing specific failure modes in one of three phase induction motor components: the stator, the rotor, or the bearings. Even though thermal and vibration monitoring have been utilized for decades, most of the recent research has been directed toward electrical monitoring of the motor with emphasis on inspecting the stator current of the motor.

Randy R.scheon et al, (1995) have proposed the initial step of investigating the efficacy of current monitoring fir bearing fault detection by correlating the relationship between vibration and current frequencies caused by incipient bearing failures. Mohamed El Hachemi Benbouzid (2000) has proposed the method of motor current signature analysis, which utilizes the results of spectral analysis of the stator current for fault detection in induction motors. Bo Li et al, (2000) have proposed an approach for motor rolling bearing fault diagnosis using neural networks and time/frequency domain bearing vibration analysis

Many papers have been published in international journals in this area. Much of the work done in motor fault detection has focused on the mechanical engineering area with the vibration analysis technique. Those however are delicate and expensive. Steele et al (1982) have proposed that the stator current monitoring can provide the same indications without requiring access to the motor. However, in the last several years, motor fault detection and diagnosis have attracted the attention of electrical engineers because of emerging technologies such as embedded systems, neural networks and fuzzy logic. Therefore this project aims at developing an intelligent embedded system for bearing fault detection in submersible pumps.

1.6 ORGANIZATION OF THE THESIS

This report presents about the bearing faults occurring in submersible pumps and the techniques implemented to detect those faults at an early stage. Chapter 1 tells about the defects that are occurring in bearings, need for a monitoring system etc., Chapter 2 briefs the experimental setup used for fault detection and experimental results obtained by FFT. Chapter 3 details the hardware implementation of the fault diagnostic module, the process flow chart and dsPIC programming. Chapter 4 describes the neural network based fault detection method and simulation results. Chapter 5 describes the fuzzy logic based fault

detection method with simulation results and comparison of the two techniques for bearing fault detection where shown.

CHAPTER 2

EXPERIMENTAL SETUP FOR DATA ACQUISTION

2.1 FAULT DETECTION SCHEME

The purpose of the monitoring system is to measure the submersible pump stator current spectrum and to analyze these data to determine the bearing condition. The stator current is sensed through the phase terminal that is connected to the submersible pump by a current transformer and an equivalent voltage signal is given to the dsPIC dsc. It converts the sampled signal to the frequency domain using Fast Fourier Transformation (FFT). To illustrate the fault detection scheme, a 1 HP, submersible pump is used. Experiments were conducted by using faulty thrust and bush bearings. Fault detection is done by using two methods:

- 1) By monitoring stator current (r.m.s value) and the rotor speed.
- 2) By monitoring the stator current spectrum.

2.2 EXPERIMENTAL HARDWARE SETUP

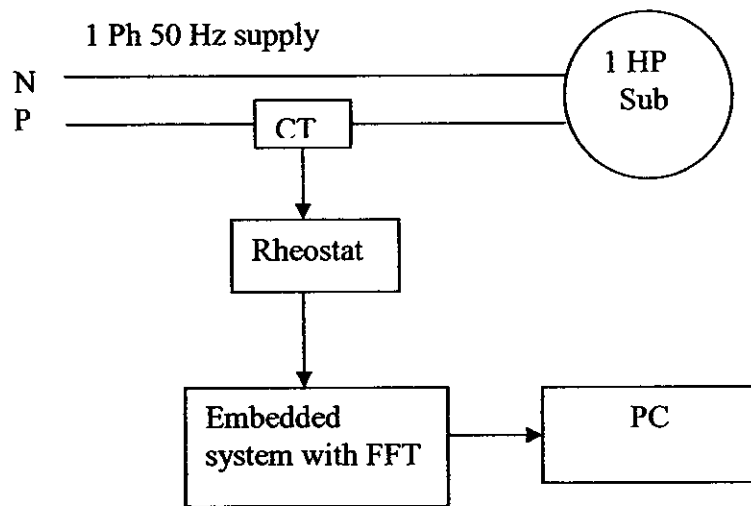


Figure 2.1 Experimental setup for data acquisition

The Figure 2.1 shows the experimental setup used in this project. The 1 ph supply is given to the submersible pump. The phase terminal is connected to the current transformer, which steps down the current value. A rheostat is connected which converts the current to an equivalent voltage. It is then given to the dsPIC processor for fast Fourier transformation. The output is connected to a PC, where the current waveforms and the obtained FFT spectrum can be displayed. Separate bearings with healthy and faulty conditions are used and the data are

obtained. Neural network and fuzzy logic are also used to detect the bearing fault. Figure 2.2 shows the photographic view of the setup.

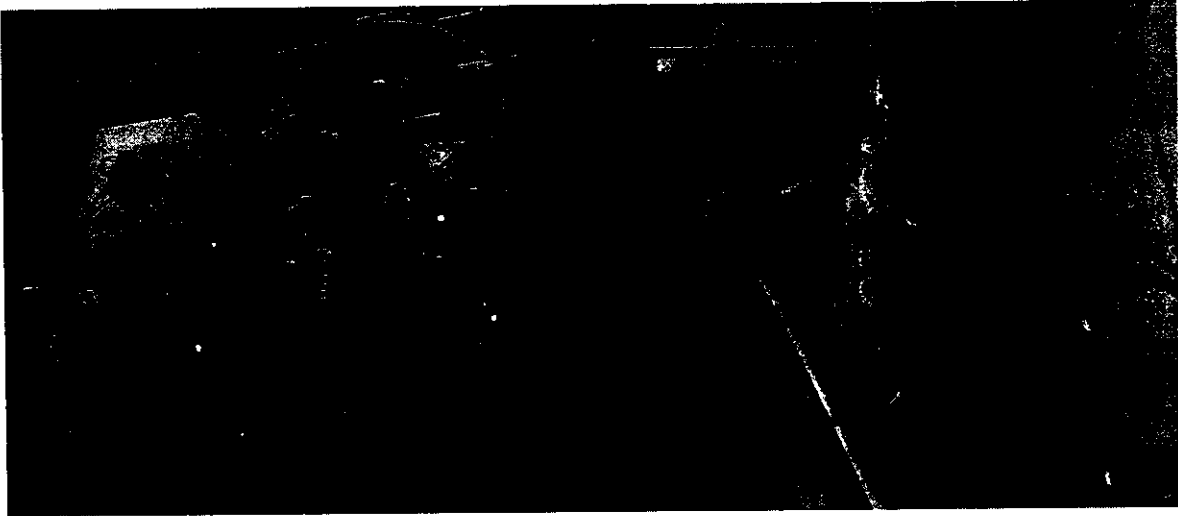


Figure 2.2 Photographic view of the setup

Table 2.1 shows the specification of the submersible pump used for fault detection.

Table 2.1 Submersible pump specification

Type	Submersible pump
Power Rating	1 HP
Voltage	230 V
Frequency	50 Hz
Current	6 A
Speed	2880 rpm
No. Of poles	2

2.3 EXPERIMENTAL RESULTS

The experimental results for bearing fault detection were obtained separately by

- i) Monitoring the stator current spectrum.
- ii) Monitoring the stator current (r.m.s value) and rotor speed.

2.3.1 Monitoring stator current spectrum

The voltage and current spectrum obtained for healthy and faulty bearings is shown in Figures 2.3 – 2.5. The Fast Fourier Transformation (FFT) is done on the obtained voltage and current spectrum and the frequency components are obtained. FFT waveforms are shown in Figures 2.6 – 2.8.

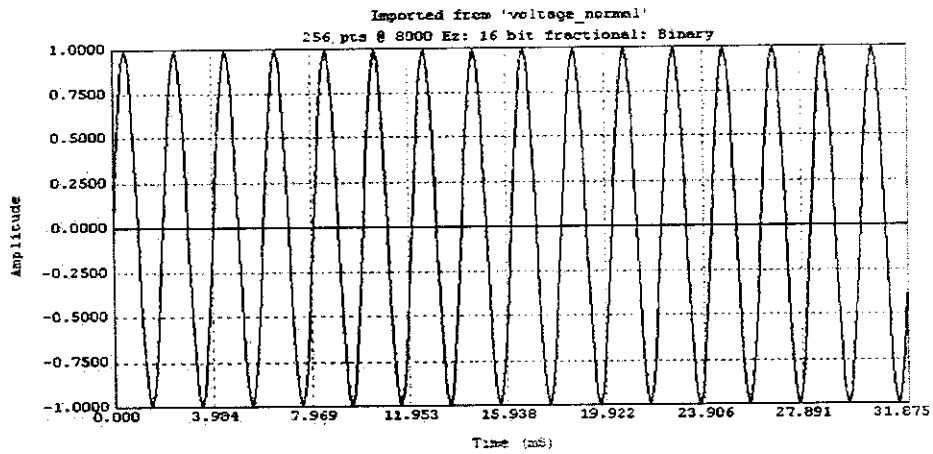


Figure 2.3. Voltage spectrum for healthy bearing

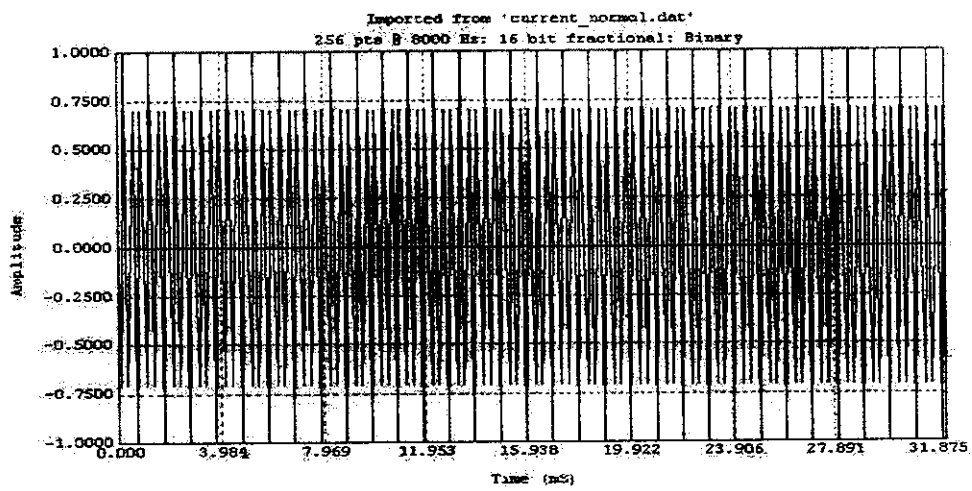


Figure 2.4. Current spectrum for healthy bearings

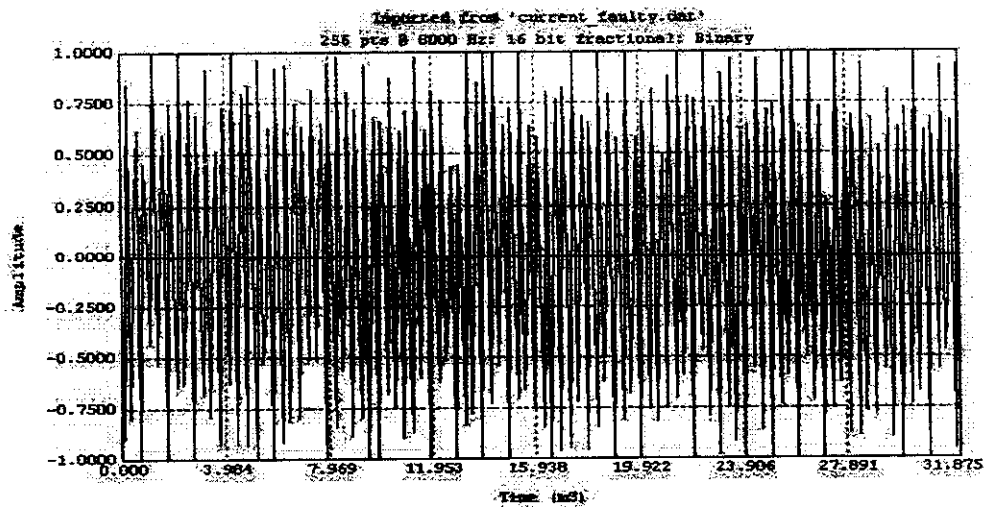


Figure 2.5 Current spectrum for faulty bearings

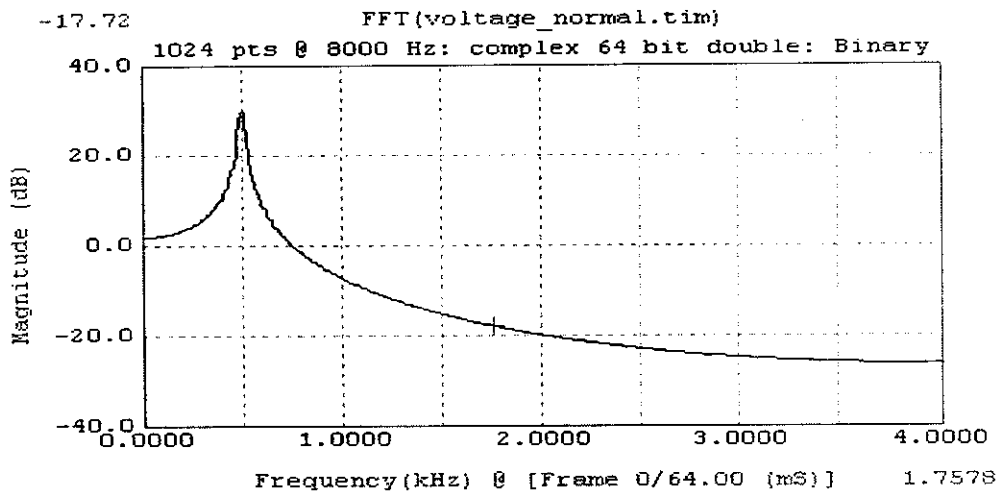


Figure 2.6 FFT waveform for healthy bearings

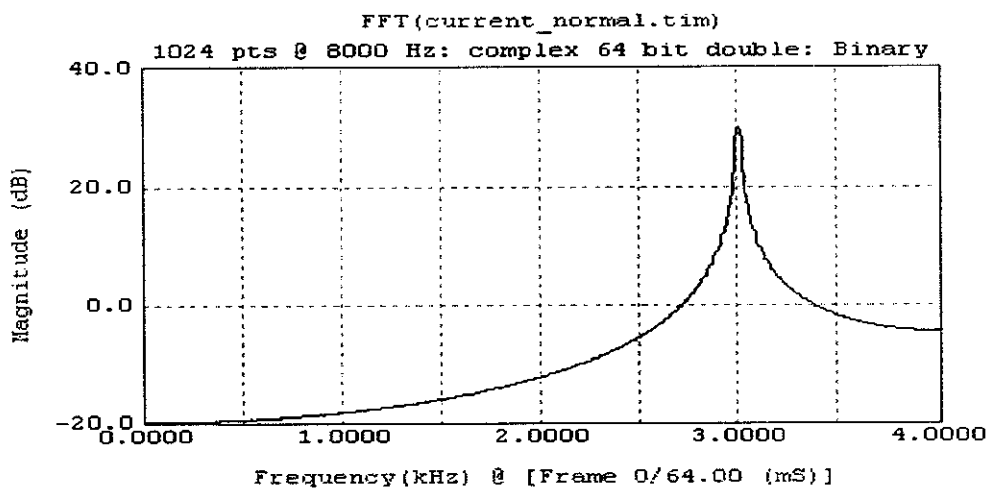


Figure 2.7 FFT waveform for healthy bearings

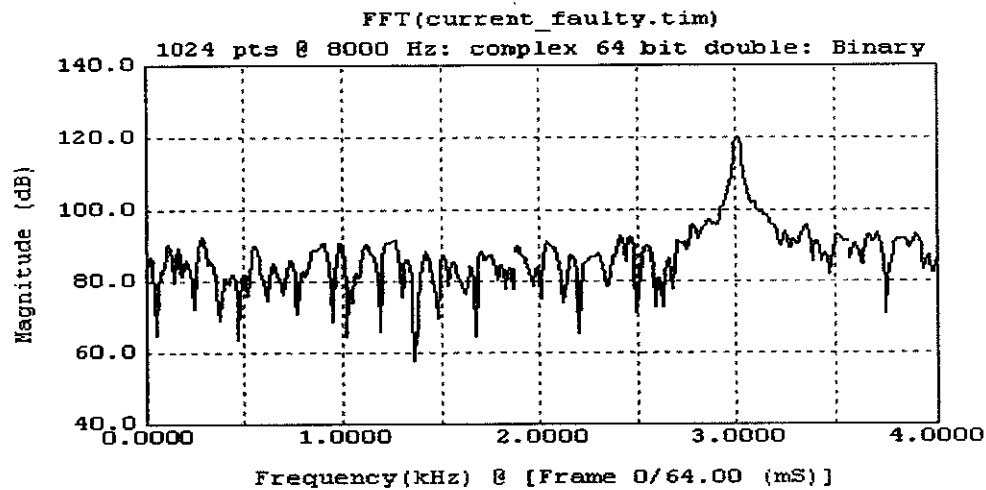


Figure 2.8 FFT waveform for faulty bearings

The data obtained for healthy and faulty bearings by monitoring the stator current spectrum are shown in Table 2.2 & 2.3.

Table 2.2 Experimental results for healthy bearings

S.No	Voltage	Current (Normal)	Speed in rpm	Temperature in °C
1	0	0	2820	43
2	12539	-23169	2820	43
3	23169	32767	2820	43
4	30272	-23169	2820	43
5	32767	0	2820	43
6	30272	23169	2820	43
7	23169	-32767	2820	43
8	12539	23169	2820	43
9	0	0	2820	43
10	-12539	-23169	2820	43
11	-23169	32767	2820	43
12	-30272	-23169	2820	43
13	-32767	0	2820	43
14	-30272	23169	2820	43
15	-23169	-32767	2820	43
16	-12539	23169	2820	43
17	0	0	2820	43
18	12539	-23169	2820	43
19	23169	32767	2820	43
20	30272	-23169	2820	43
21	32767	0	2820	43
22	30272	23169	2820	43
23	23169	-32767	2820	43
24	12539	23169	2820	43
25	0	0	2820	43
26	-12539	-23169	2820	43
27	-23169	32767	2820	43
28	-30272	-23169	2820	43
29	-32767	0	2820	43
30	-30272	23169	2820	43
31	-23169	-32767	2820	43
32	-12539	23169	2820	43
33	0	0	2820	43
34	12539	-23169	2820	43
35	23169	32767	2820	43
36	30272	-23169	2820	43
37	32767	0	2820	43
38	30272	23169	2820	43
39	23169	-32767	2820	43
40	12539	23169	2820	43
41	0	0	2820	43
42	-12539	-23169	2820	43
43	-23169	32767	2820	43
44	-30272	-23169	2820	43

45	-32767	0	2820	43
46	-30272	23169	2820	43
47	-23169	-32767	2820	43
48	-12539	23169	2820	43
49	0	0	2820	43
50	12539	-23169	2820	43
51	23169	32767	2820	43
52	30272	-23169	2820	43
53	32767	0	2820	43
54	30272	23169	2820	43
55	23169	-32767	2820	43
56	12539	23169	2820	43
57	0	0	2820	43
58	-12539	-23169	2820	43
59	-23169	32767	2820	43
60	-30272	-23169	2820	43
61	-32767	0	2820	43
62	-30272	23169	2820	43
63	-23169	-32767	2820	43
64	-12539	23169	2820	43
65	0	0	2820	43
66	12539	-23169	2820	43
67	23169	32767	2820	43
68	30272	-23169	2820	43
69	32767	0	2820	43
70	30272	23169	2820	43
71	23169	-32767	2820	43
72	12539	23169	2820	43
73	0	0	2820	43
74	-12539	-23169	2820	43
75	-23169	32767	2820	43
76	-30272	-23169	2820	43
77	-32767	0	2820	43
78	-30272	23169	2820	43
79	-23169	-32767	2820	43
80	-12539	23169	2820	43
81	0	0	2820	43
82	12539	-23169	2820	43
83	23169	32767	2820	43
84	30272	-23169	2820	43
85	32767	0	2820	43
86	30272	23169	2820	43
87	23169	-32767	2820	43
88	12539	23169	2820	43
89	0	0	2820	43
90	-12539	-23169	2820	43
91	-23169	32767	2820	43
92	-30272	-23169	2820	43
93	-32767	0	2820	43



P-23B

94	-30272	23169	2820	43
95	-23169	-32767	2820	43
96	-12539	23169	2820	43
97	0	0	2820	43
98	12539	-23169	2820	43
99	23169	32767	2820	43
100	30272	-23169	2820	43
101	32767	0	2820	43
102	30272	23169	2820	43
103	23169	-32767	2820	43
104	12539	23169	2820	43
105	0	0	2820	43
106	-12539	-23169	2820	43
107	-23169	32767	2820	43
108	-30272	-23169	2820	43
109	-32767	0	2820	43
110	-30272	23169	2820	43
111	-23169	-32767	2820	43
112	-12539	23169	2820	43
113	0	0	2820	43
114	12539	-23169	2820	43
115	23169	32767	2820	43
116	30272	-23169	2820	43
117	32767	0	2820	43
118	30272	23169	2820	43
119	23169	-32767	2820	43
120	12539	23169	2820	43
121	0	0	2820	43
122	-12539	-23169	2820	43
123	-23169	32767	2820	43
124	-30272	-23169	2820	43
125	-32767	0	2820	43
126	-30272	23169	2820	43
127	-23169	-32767	2820	43
128	-12539	23169	2820	43
129	0	0	2820	43
130	12539	-23169	2820	43
131	23169	32767	2820	43
132	30272	-23169	2820	43
133	32767	0	2820	43
134	30272	23169	2820	43
135	23169	-32767	2820	43
136	12539	23169	2820	43
137	0	0	2820	43
138	-12539	-23169	2820	43
139	-23169	32767	2820	43
140	-30272	-23169	2820	43
141	-32767	0	2820	43
142	-30272	23169	2820	43

143	-23169	-32767	2820	43
144	-12539	23169	2820	43
145	0	0	2820	43
146	12539	-23169	2820	43
147	23169	32767	2820	44
148	30272	-23169	2820	44
149	32767	0	2820	44
150	30272	23169	2820	44
151	23169	-32767	2820	44
152	12539	23169	2818	44
153	0	0	2818	44
154	-12539	-23169	2818	44
155	-23169	32767	2818	44
156	-30272	-23169	2818	44
157	-32767	0	2818	44
158	-30272	23169	2818	44
159	-23169	-32767	2818	44
160	-12539	23169	2818	44
161	0	0	2818	44
162	12539	-23169	2818	44
163	23169	32767	2818	44
164	30272	-23169	2818	44
165	32767	0	2818	44
166	30272	23169	2818	44
167	23169	-32767	2818	44
168	12539	23169	2818	44
169	0	0	2818	44
170	-12539	-23169	2818	44
171	-23169	32767	2818	44
172	-30272	-23169	2818	44
173	-32767	0	2818	44
174	-30272	23169	2818	44
175	-23169	-32767	2818	44
176	-12539	23169	2818	44
177	0	0	2818	44
178	12539	-23169	2818	44
179	23169	32767	2818	44
180	30272	-23169	2818	44
181	32767	0	2818	44
182	30272	23169	2818	44
183	23169	-32767	2818	44
184	12539	23169	2818	44
185	0	0	2818	44
186	-12539	-23169	2818	44
187	-23169	32767	2818	44
188	-30272	-23169	2818	44
189	-32767	0	2818	44
190	-30272	23169	2818	44
191	-23169	-32767	2818	44

192	-12539	23169	2818	44
193	0	0	2818	44
194	12539	-23169	2818	44
195	23169	32767	2818	44
196	30272	-23169	2818	44
197	32767	0	2818	44
198	30272	23169	2818	44
199	23169	-32767	2818	44
200	12539	23169	2818	44
201	0	0	2818	44
202	-12539	-23169	2818	44
203	-23169	32767	2818	44
204	-30272	-23169	2818	44
205	-32767	0	2818	44
206	-30272	23169	2818	44
207	-23169	-32767	2818	44
208	-12539	23169	2818	44
209	0	0	2818	44
210	12539	-23169	2818	44
211	23169	32767	2818	44
212	30272	-23169	2818	44
213	32767	0	2818	44
214	30272	23169	2818	44
215	23169	-32767	2818	44
216	12539	23169	2818	44
217	0	0	2818	44
218	-12539	-23169	2818	44
219	-23169	32767	2818	44
220	-30272	-23169	2818	44
221	-32767	0	2818	44
222	-30272	23169	2818	44
223	-23169	-32767	2818	44
224	-12539	23169	2818	44
225	0	0	2818	44
226	12539	-23169	2818	44
227	23169	32767	2818	44
228	30272	-23169	2818	44
229	32767	0	2818	44
230	30272	23169	2818	44
231	23169	-32767	2818	44
232	12539	23169	2818	44
233	0	0	2818	44
234	-12539	-23169	2818	44
235	-23169	32767	2818	44
236	-30272	-23169	2818	44
237	-32767	0	2818	44
238	-30272	23169	2818	44
239	-23169	-32767	2818	44
240	-12539	23169	2818	44

241	0	0	2818	44
242	12539	-23169	2818	44
243	23169	32767	2818	44
244	30272	-23169	2818	44
245	32767	0	2818	44
246	30272	23169	2818	44
247	23169	-32767	2818	44
248	12539	23169	2818	44
249	0	0	2818	44
250	-12539	-23169	2818	44
251	-23169	32767	2818	44
252	-30272	-23169	2818	44
253	-32767	0	2818	44
254	-30272	23169	2818	44
255	-23169	-32767	2818	44
256	-12539	23169	2818	44

Table.2.3 Experimental results for faulty bearings

S.No	Voltage	Current (Faulty)	Speed in rpm	Temperature in °C
1	0	-3157	2604	73
2	12539	-29324	2604	73
3	23169	27757	2604	73
4	30272	-26036	2604	73
5	32767	5443	2604	73
6	30272	20277	2604	73
7	23169	-32768	2604	73
8	12539	14981	2604	73
9	0	149	2604	73
10	-12539	-14124	2604	73
11	-23169	32767	2604	73
12	-30272	-17366	2604	73
13	-32767	1845	2604	73
14	-30272	19625	2604	73
15	-23169	-32768	2604	73
16	-12539	24754	2604	73
17	0	-3726	2604	73
18	12539	-21074	2604	73
19	23169	32767	2604	73
20	30272	-20613	2604	73
21	32767	-1914	2604	73
22	30272	25244	2604	73
23	23169	-32768	2604	73
24	12539	22873	2604	73
25	0	-1013	2604	73
26	-12539	-22380	2604	73
27	-23169	30236	2604	73

28	-30272	-25884	2604	73
29	-32767	2519	2604	73
30	-30272	17180	2604	73
31	-23169	-30358	2604	73
32	-12539	24193	2604	73
33	0	-2314	2604	73
34	12539	-20455	2604	73
35	23169	32767	2604	73
36	30272	-30321	2604	73
37	32767	-484	2604	73
38	30272	26368	2604	73
39	23169	-30504	2604	73
40	12539	27589	2604	73
41	0	420	2604	73
42	-12539	-32623	2604	73
43	-23169	31635	2604	73
44	-30272	-17118	2604	73
45	-32767	-8375	2604	73
46	-30272	20782	2604	73
47	-23169	-32768	2604	73
48	-12539	30269	2604	73
49	0	-4371	2604	73
50	12539	-29872	2604	73
51	23169	30880	2604	73
52	30272	-26612	2604	73
53	32767	4085	2604	73
54	30272	24670	2604	73
55	23169	-26171	2604	73
56	12539	21127	2604	73
57	0	-2475	2604	73
58	-12539	-16634	2604	73
59	-23169	26784	2604	73
60	-30272	-16679	2604	73
61	-32767	7209	2604	73
62	-30272	21371	2604	73
63	-23169	-32768	2604	73
64	-12539	31023	2604	73
65	0	169	2604	73
66	12539	-27660	2604	73
67	23169	32133	2604	73
68	30272	-18396	2604	73
69	32767	-2002	2604	73
70	30272	26420	2604	73
71	23169	-28999	2604	73
72	12539	23683	2604	73
73	0	-2664	2604	73
74	-12539	-26030	2604	73
75	-23169	30992	2604	73
76	-30272	-32425	2604	73

77	-32767	-6365	2604	73
78	-30272	22205	2604	73
79	-23169	-32768	2604	73
80	-12539	21802	2604	73
81	0	5391	2604	73
82	12539	-22113	2604	73
83	23169	28864	2604	73
84	30272	-24532	2604	73
85	32767	-6031	2604	73
86	30272	20211	2604	73
87	23169	-29403	2604	73
88	12539	23651	2604	73
89	0	-16949	2604	73
90	-12539	-28318	2604	73
91	-23169	32056	2604	73
92	-30272	-19711	2604	73
93	-32767	1605	2604	73
94	-30272	20462	2604	73
95	-23169	-32768	2604	73
96	-12539	26737	2604	73
97	0	-4273	2604	73
98	12539	-20287	2604	73
99	23169	25159	2604	73
100	30272	-16825	2604	73
101	32767	-4924	2604	73
102	30272	14448	2604	73
103	23169	-32768	2604	73
104	12539	14928	2604	73
105	0	-299	2604	73
106	-12539	-27300	2604	73
107	-23169	32767	2604	73
108	-30272	-25423	2604	73
109	-32767	-460	2604	73
110	-30272	28065	2604	73
111	-23169	-32768	2604	73
112	-12539	32767	2604	73
113	0	3612	2604	73
114	12539	-23685	2604	73
115	23169	32767	2604	73
116	30272	-17752	2604	73
117	32767	-2341	2604	73
118	30272	21160	2604	73
119	23169	-32768	2604	73
120	12539	23903	2604	73
121	0	-177	2604	73
122	-12539	-22989	2604	73
123	-23169	32767	2604	73
124	-30272	-21681	2604	73
125	-32767	-3958	2604	73

126	-30272	21131	2604	73
127	-23169	-24538	2604	73
128	-12539	19289	2604	73
129	0	-3626	2604	73
130	12539	-27685	2604	73
131	23169	26406	2604	73
132	30272	-32622	2604	73
133	32767	2893	2604	73
134	30272	25271	2604	73
135	23169	-31428	2604	73
136	12539	27251	2604	73
137	0	-885	2604	73
138	-12539	-31013	2604	73
139	-23169	32767	2604	73
140	-30272	-23654	2604	73
141	-32767	-2103	2604	73
142	-30272	22520	2604	73
143	-23169	-31260	2604	73
144	-12539	21541	2604	73
145	0	-858	2604	73
146	12539	-27650	2604	73
147	23169	32767	2604	73
148	30272	-20176	2604	73
149	32767	1987	2604	73
150	30272	26007	2604	73
151	23169	-32768	2604	73
152	12539	19063	2604	73
153	0	1820	2604	73
154	-12539	-26484	2604	73
155	-23169	32767	2604	73
156	-30272	-22281	2604	73
157	-32767	-5025	2602	74
158	-30272	19341	2602	74
159	-23169	-32768	2602	74
160	-12539	24839	2602	74
161	0	-3008	2602	74
162	12539	-24793	2602	74
163	23169	26831	2602	74
164	30272	-26581	2602	74
165	32767	-11	2602	74
166	30272	16548	2602	74
167	23169	-24207	2602	74
168	12539	29037	2602	74
169	0	341	2602	74
170	-12539	-23071	2602	74
171	-23169	32767	2602	74
172	-30272	-19208	2602	74
173	-32767	-5623	2602	74
174	-30272	25649	2602	74

175	-23169	-32768	2602	74
176	-12539	25352	2602	74
177	0	-568	2602	74
178	12539	-15186	2602	74
179	23169	32767	2602	74
180	30272	-25886	2602	74
181	32767	-2235	2602	74
182	30272	23906	2602	74
183	23169	-32072	2602	74
184	12539	29304	2602	74
185	0	-83	2602	74
186	-12539	-24414	2602	74
187	-23169	31621	2602	74
188	-30272	-29863	2602	74
189	-32767	-3366	2602	74
190	-30272	19950	2602	74
191	-23169	-32768	2602	74
192	-12539	21172	2602	74
193	0	6108	2602	74
194	12539	-18193	2602	74
195	23169	31646	2602	74
196	30272	-28256	2602	74
197	32767	4906	2602	74
198	30272	23636	2602	74
199	23169	-32768	2602	74
200	12539	24556	2602	74
201	0	866	2602	74
202	-12539	-23890	2602	74
203	-23169	32767	2602	74
204	-30272	-19276	2602	74
205	-32767	-1031	2602	74
206	-30272	32767	2602	74
207	-23169	-32768	2602	74
208	-12539	19611	2602	74
209	0	-8589	2602	74
210	12539	-15708	2602	74
211	23169	32767	2602	74
212	30272	-23282	2602	74
213	32767	3478	2602	74
214	30272	24082	2602	74
215	23169	-32768	2602	74
216	12539	9354	2602	74
217	0	-372	2602	74
218	-12539	-24501	2602	74
219	-23169	32767	2602	74
220	-30272	-20451	2602	74
221	-32767	-3100	2602	74
222	-30272	20513	2602	74
223	-23169	-28613	2602	74

224	-12539	22538	2602	74
225	0	8900	2602	74
226	12539	-29026	2602	74
227	23169	31016	2602	74
228	30272	-24998	2602	74
229	32767	-2001	2602	74
230	30272	22147	2602	74
231	23169	-25539	2602	74
232	12539	17820	2602	74
233	0	-396	2602	74
234	-12539	-18302	2602	74
235	-23169	26664	2602	74
236	-30272	-29293	2602	74
237	-32767	-1339	2602	74
238	-30272	20723	2602	74
239	-23169	-32768	2602	74
240	-12539	23982	2602	74
241	0	-459	2602	74
242	12539	-23895	2602	74
243	23169	32767	2602	74
244	30272	-21794	2602	74
245	32767	-57	2602	74
246	30272	20214	2602	74
247	23169	-32225	2602	74
248	12539	22258	2602	74
249	0	5211	2602	74
250	-12539	-18419	2602	74
251	-23169	30627	2602	74
252	-30272	-15209	2602	74
253	-32767	832	2602	74
254	-30272	21901	2602	74
255	-23169	-31126	2602	74
256	-12539	30711	2602	74

2.3.2 Monitoring stator current and rotor speed

The data obtained by monitoring the stator current (r.m.s value) and rotor speed is shown in Table 2.4

Table 2.4 Experimental results by monitoring stator current and speed

S.No	Current	Speed	Temperature
1	6.2	2655	48
2	6.5	2640	48
3	7.0	2610	53
4	8.5	2590	57
5	10.0	2600	72
6	11.0	2500	73

CHAPTER 3

NEURAL NETWORK BASED FAULT DIAGNOSIS

3.1 INTRODUCTION TO NEURAL NETWORK

An artificial neural network is an information processing system that has certain performance characteristics in common with biological neural networks. Laurene Fausett (2004) explains that the artificial neural networks have been developed as generalization of mathematical models of human cognition or neural biology, based on assumptions that:

- Information processing occurs at many simple elements called neurons.
- Signals are passed between neurons over connection links.
- Each connection link has an associated weight, which, in a typical neural net, multiplies the signal transmitted.
- Each neuron applies an activation function (usually nonlinear) to its net input (sum of weighted input signals) to determine its output signal.

A biological neuron has three types of components that are of particular interest in understanding an artificial neuron: its dendrites, soma and axon. Dendrites receive signal from other neurons. The signals are electrical impulses that are transmitted across a synaptic gap by means of a chemical process. The soma or cell body sums the incoming signals. When sufficient input is received, the cell fires; that is, it transmits a signal over its axon to other cells. Figure 3.1 shows the structure of biological neuron.

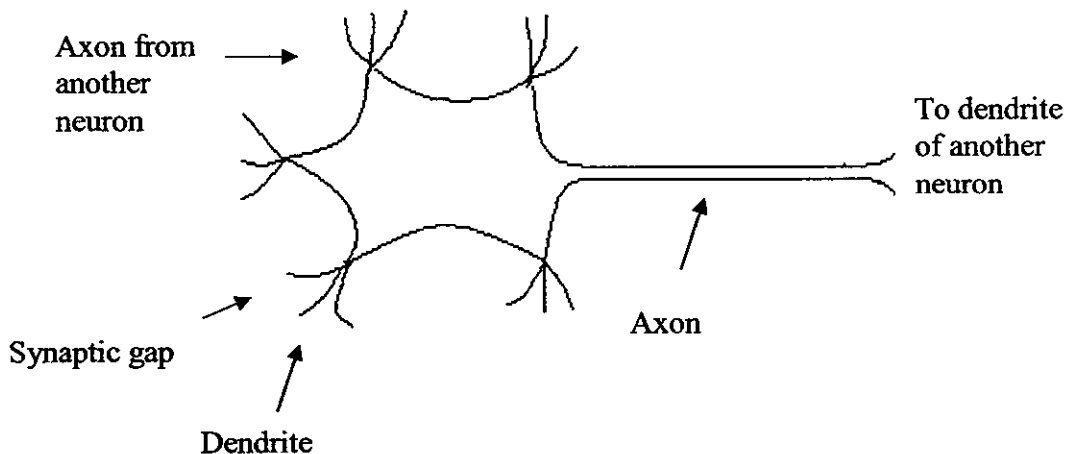


Figure 3.1 Structure of biological neuron

An Artificial Neural Network is characterized by,

- Its pattern of connections between the neurons (called its architecture)

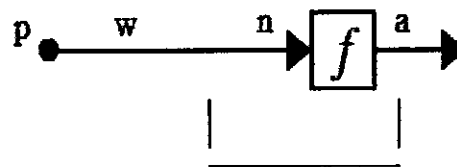
- Its method of determining the weights on the connections (called its training or learning, algorithm), and
- Its activation function

The network function is determined largely by the connections between elements. Therefore, a neural network can be trained to perform a particular function by adjusting the values of the connections (weight) between the elements commonly neural networks are adjusted, or trained, so that a particular input leads to a specific target output.

The network weight is adjusted based on a comparison of the output and the target, until the network output matches the target.

3.1.1 Neuron model

Figure 3.2 shows a neuron with a single scalar input with no bias. The scalar input p , is transmitted through a connection that multiplies its strength by the scalar weight w , to form the product wp , again a scalar. Here the weighted input wp is the only argument of the activation function f , which produces the scalar output a .

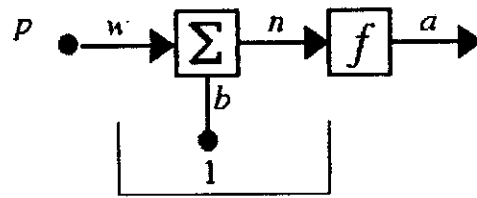


Neuron without bias

$$a = f(wp)$$

Figure 3.2 Single – Input Neuron without Bias

Figure 3.3 shows a neuron with a scalar input, with scalar bias. The bias is much like a weight, except that it has a constant input of 1. The activation function net input n , again a scalar, is a sum of the weighted input wp and the bias b ., this sum is the argument of the activation function f . f is an activation function, typically a step function or a sigmoid function, that takes the argument n and produces the output a . w and b are both adjustable parameters of the neuron.



Neuron with bias

$$a = f(wp + b)$$

Figure 3.3 Single Input Neuron with Bias

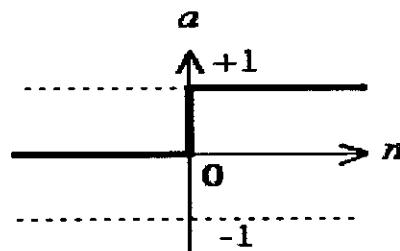
The central idea of neural networks is that such parameters can be adjusted so that the network exhibits some desired or interesting behavior. Thus, we can train the network to do a particular job by adjusting the weight or bias parameters, or perhaps the network itself will adjust these parameters to achieve some desired end.

3.1.2 Activation functions

An activation function may be linear or a non-linear function of an. A particular activation function is chosen to satisfy some specification of a problem that the neuron is attempting to solve. There are three most commonly used activation function. They are

- (a) Hard limit activation function
- (b) Linear activation function
- (c) Log-sigmoid activation function

(a) Hard limit activation function:



$$a = \text{hardlim}(n)$$

Figure 3.4 Hard Limit Activation Function

Figure 3.4 shows the graphical representation of the hard limit activation function. The hard limit activation function sets the output of the neuron to 0 if the function argument is less than 0, or 1 if its argument is greater than or equal to 0.

(b) Linear activation function:

The output of a linear activation function is equal to its input. The output (a) versus input (p) characteristic of a single-input linear neuron is shown in Figure 3.5.

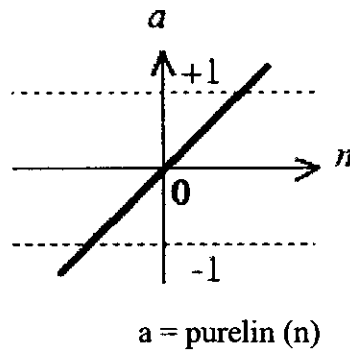


Figure 3.5 Linear Activation Function

(c) Log-sigmoid activation function:

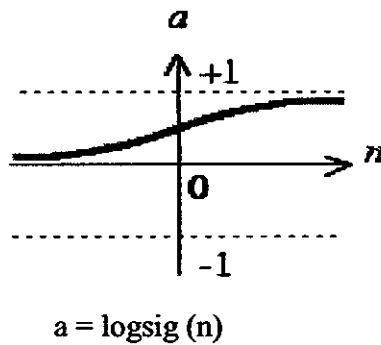


Figure 3.6 Log-Sigmoid Activation Function

Figure 3.6 shows the log-sigmoid activation function. This activation function takes the input (which may have any value between plus and minus infinity) and squashes the output into the range 0 to 1, according to expression

$$a = 1 / (1 + e^{-n}) \quad (3.1)$$

This activation function is commonly used in multilayer networks that are trained using the back-propagation algorithm, in part because this function is differentiable.

3.1.3 Learning rules

The weights and biases of the network can be modified by means of 'learning rule'. This procedure may also be referred to as a training algorithm. The purpose of the learning rule is to train the network to perform some task. Neural networks can be trained to solve problem that are difficult for conventional computers or human beings. There are many types of neural network learning rules. They fall into three broad categories: supervised learning, unsupervised learning and reinforcement (or graded) learning.

(a) **Supervised learning:** In supervised learning, the network is provided with inputs and the corresponding correct output. As the inputs are applied to the network, the network outputs are compared to the targets. The learning rule is then used to adjust the weights and biases of the network in order to move the network outputs closer to the targets. An example for the supervised learning is the perceptron-learning rule.

(b) **Reinforcement learning:** This is similar to supervised learning, except that, instead of being provided with the correct output for each network input, the algorithm is only given a grade. The grade is a measure of the network performance over some sequence of inputs. This type of learning is currently much less common than supervised learning.

(c) **Unsupervised learning:** In unsupervised learning, the weights and biases are modified in response to network inputs only. There are no target outputs available. The network learns to categorize the input patterns into a finite number of classes. An example for unsupervised learning algorithm is Adaptive Resonance Theory.

3.2 BACK-PROPAGATION NEURAL NETWORK

In supervised learning, the first learning rule is perception-learning rule, in which the learning rule is provided with a set of examples of proper network behavior. As each input is applied to the network, the learning rule adjusts the network parameters so that the network output will move closer to the target. The perception learning rule is very simple, but it is also quite powerful. This rule will always converge to a correct solution, if such a solution exists. The perception-learning rule forms the basis for understanding the more complex networks. As with the perceptron rule, the Least Mean Square (LMS) algorithm is an example of supervised training. The LMS algorithm will adjust the weights and biases to minimize the mean square error, where the error is the difference between the target output and the network output. The perceptron-net is incapable of implementing certain elementary functions. These limitations were overcome with improved (multilayer) perceptron networks. Performance

learning is another important class of learning law, in which the network parameters are adjusted to optimize the performance of the network. Back propagation (BP) algorithm can be used to train multilayer networks. As with the LMS learning law, BP is an approximate steepest descent algorithm, in which the performance index is mean square error. The difference between the LMS algorithm and back propagation is only in the way in which the derivatives are calculated. The single-layer perceptron like networks are only able to solve linearly separable classification problems. Multilayer perceptron, trained by BP algorithm were developed to overcome these limitations and is currently the most widely used neural network. In addition, multi-layer networks can be used as universal function approximators. A two-layer network, with sigmoid-type activation functions in the hidden layer, can approximate any practical function, with enough neurons in the hidden layer. The Figure 3.7 shows the Architecture of BP Neural Network.

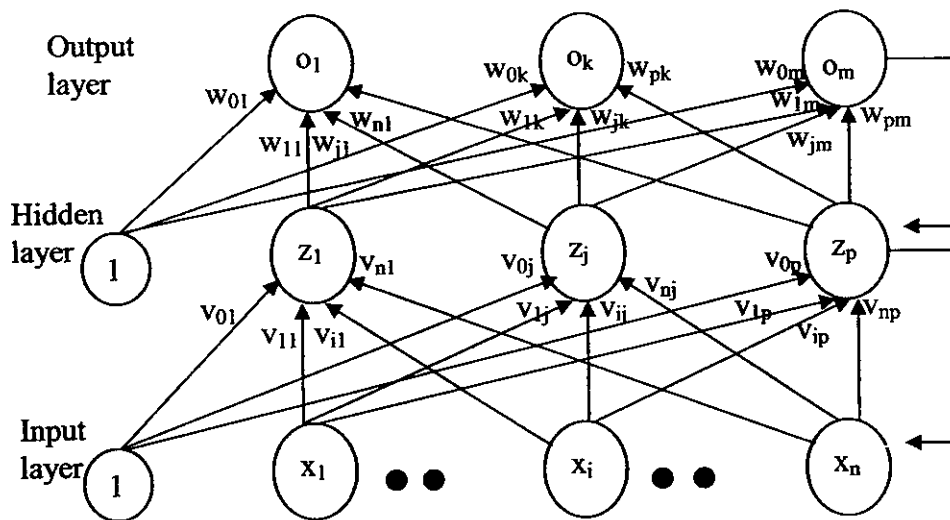


Figure 3.7 Architecture of BP Neural Network

The BP algorithm uses the chain rule in order to compute the derivatives of the squared error with respect to the weights and biases in the hidden layers. It is called BP because the derivatives are computed first at the last layer of the network, and then propagated backward through the network, using the chain rule, to compute the derivatives in the hidden layers.

The BP training algorithm is an interactive gradient algorithm designed to minimize the mean square error between the actual output of a feed-forward net and the desired output.

- Step 0: Initialize weights.
- Step 1: While stopping condition is false, do steps 2-9,
- Step 2: For each training pair, do steps 3-8,
Feed forward:
- Step 3: Each input unit ($X_i, i=1 \dots n$) receives input signal x_i and broadcasts this signal to all units in the layer above (the hidden units).
- Step 4: Each hidden unit ($Z_j, j=1 \dots n$) sums its weighted input signals,

$$Z_in_j = v_{oj} + \sum_{i=1}^n x_i v_{ij},$$
 Applies its activation function to compute its output signals,
 $Z_j = f(z_in_j)$, and sends this signal to all units in the layer above .
- Step 5: Each output unit ($y_k, k=1 \dots m$) sums its weighted input signals,

$$Y_In_k = W_{ok} = \sum_{j=1}^p Z_j W_{jk}$$
 And applies its activation function to compute its signals,

$$Y_k = f(y_ink).$$
 Back propagation of error:
- Step 6: Each output unit ($y_k, k=1 \dots m$) receives a target pattern corresponding to input training pattern, computes its error information term ,

$$\delta_k = (t_k - y_k) f'(y_ink),$$
 Calculates its weight correction term (used to update w_{jk} later),

$$\Delta w_{jk} = \alpha \delta_k z_j,$$
 Calculates its bias correction term (used to update W_{ok} later)

$$\Delta w_{ok} = \alpha \delta_k$$
- Step 7: Each hidden unit ($Z_j, j=1 \dots p$) sums its delta inputs

$$\delta_inj = \sum_{k=1}^m \delta_k w_{jk},$$
 multiplies by the derivative of its activation function to calculate its error information term

$$\delta_j = \delta_inj f'(z_inj),$$
 Calculates its weight correction term (used to update V_{ij} later),

$$\Delta v_{ij} = \alpha \delta_j x_i,$$
 And calculates its bias correction term (used to update V_{oj} later),

$$\Delta v_{oj} = \alpha \delta_j.$$

Update weights and biases:

Step 8: Each output unit ($Y_k, k=1 \dots m$) updates its bias and weights ($j=0 \dots p$):

$$W_{jk}(\text{new}) = w_{jk}(\text{old}) + \Delta w_{jk},$$

Each hidden unit ($Z_j, j=1 \dots p$) updates its bias and weights ($i=0 \dots n$):

$$V_{ij}(\text{new}) = v_{ij}(\text{old}) + \Delta v_{ij}$$

Step 9: Test stopping condition.

3.2.1 Choice of parameters for network training

When the basic BP algorithm is applied to a practical problem the training may take days or weeks of computer time. This has encouraged considerable research on methods to accelerate the convergence of the algorithm. The research on faster algorithms falls roughly into two categories; the first category involves the development of heuristic techniques, which arises out of a study of the distinctive performance of the standard BP algorithm. These heuristic techniques include such ideas as varying the learning rate, using momentum and rescaling variables. Another category of research has focused on standard numerical optimization techniques.

3.2.2 Learning rate

The speed of training the BP network is improved by changing the learning rate during training. Increasing the learning rate on flat surfaces and then decreasing the learning rate when slope increases can increase the process of convergence. If the learning rate is too large, it leads to unstable learning. And if it is too small, it leads to incredibly long training times. Hence care has to be taken while deciding learning rate. There are many different approaches for varying the learning rate. The learning rate is varied according to the performance of the algorithm. The rules of the variable learning rate BP algorithm are:

1. If the squared error increases by more than some set percentage α (typically one to five percent) after weight update, then the weight update is discarded, the learning rate is multiplied by some factor $0 < \rho < 1$, and the momentum coefficient β (if it is used) is set to zero.

2. If the squared error decreases after a weight update, then the weight update is accepted and the learning rate is multiplied by some factor $\eta > 1$. If η has been previously set to zero, it is reset to its original value.
3. If the squared error increases by less than η then the weight update is accepted but the learning rate is unchanged. If η has been previously set to zero, it is reset to its original value.

3.2.3 Momentum factor

In BP with momentum, the weight change is in a direction that is a combination of the current gradient and the previous gradient. This is a modification of gradient descent whose advantage arises chiefly when some training data are very different from the majority of the data. By the use of momentum larger training rate can be used, while maintaining the stability of the algorithm. Another feature of momentum is that it tends to accelerate convergence when the trajectory is moving in a consistent direction. The larger the value of η , the more the momentum the trajectory has. The momentum coefficient is maintained with the range $[0, 1]$.

3.3 STRUCTURE OF BP NETWORK FOR FAULT DETECTION

An artificial neural network is composed of neurons with a deterministic activation function. The neural network is trained by adjusting the numerical value of the weights will contain the non-linearity of the desired mapping, so that difficulties in the mathematical modeling can be avoided. The BP training algorithm is used to adjust the numerical values of the weights and the internal threshold of each neuron. The network is trained by, initially selecting small random weights and internal threshold and then presenting all training data. Weights and thresholds are adjusted after every training example is presented to the network; until the weight converges or the error is reduced to acceptable value. The fault detection scheme is implemented by

- 1) Monitoring the stator current spectrum
- 2) Monitoring the stator current and rotor speed

3.3.1 Fault detection by monitoring stator current spectrum

Fault detection neural network consists of two layers: hidden layer and the output layer. The inputs to the neural network are voltage, current and speed. The hidden layer consists of three neurons. The output layer has one neuron that corresponds to bearing

condition. The transfer function used for hidden layer is log-sigmoid and for output layer, tan-sigmoid is used. The learning rate is 0.0155085 and the momentum factor is 0.8 respectively.

Figure 3.8 shows the structure of BP Network for Fault Detection by monitoring the stator current spectrum.

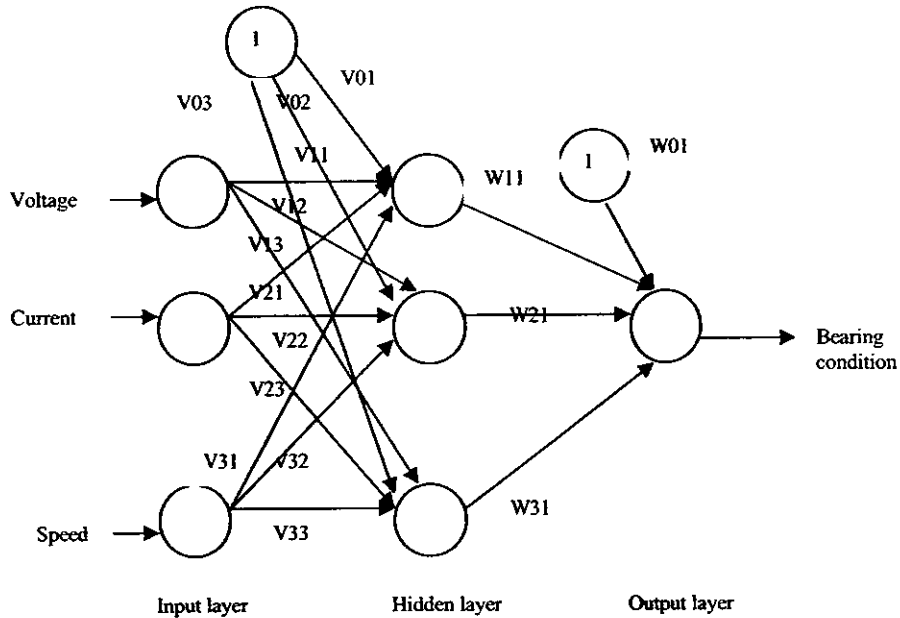


Figure 3.8 Structure of BP Network for Fault Detection

The network is trained separately for healthy bearings and faulty bearings.

3.3.2 Simulation results

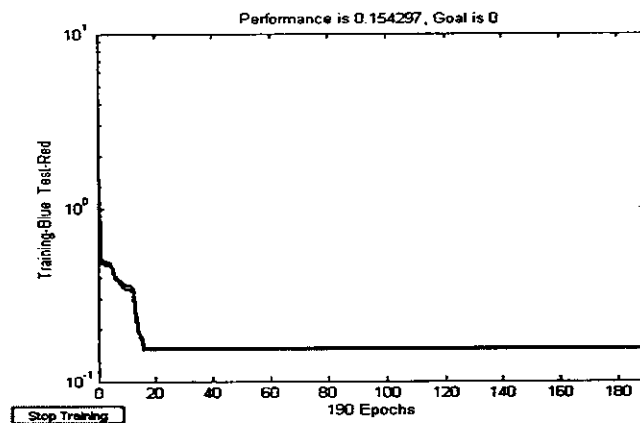


Figure 3.9 Epoch Vs Error Characteristics for healthy bearings

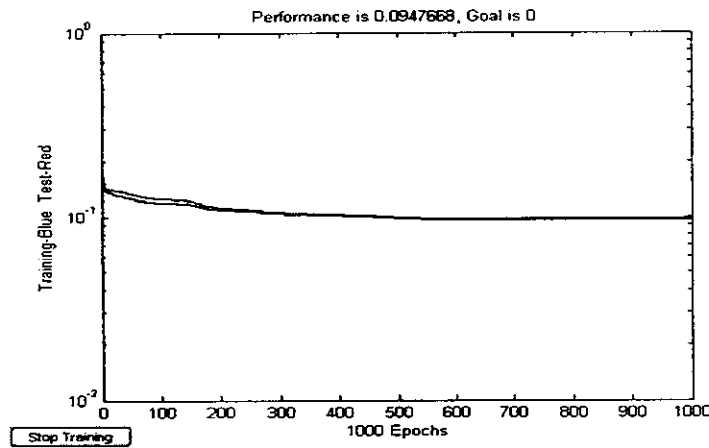


Figure 3.10 Epoch Vs Error characteristics for faulty bearings

3.3.3. Fault detection by monitoring stator current and rotor speed

Fault detection neural network consists of two layers: hidden layer and the output layer. The inputs to the neural network are current and speed. The hidden layer consists of three neurons. The output layer has one neuron that corresponds to bearing condition. The transfer function used for hidden layer is log-sigmoid and for output layer, tan-sigmoid is used. The learning rate is 0.01477 and the momentum factor is 0.85 respectively.

Figure 3.11 shows the structure of BP Network for Fault Detection.

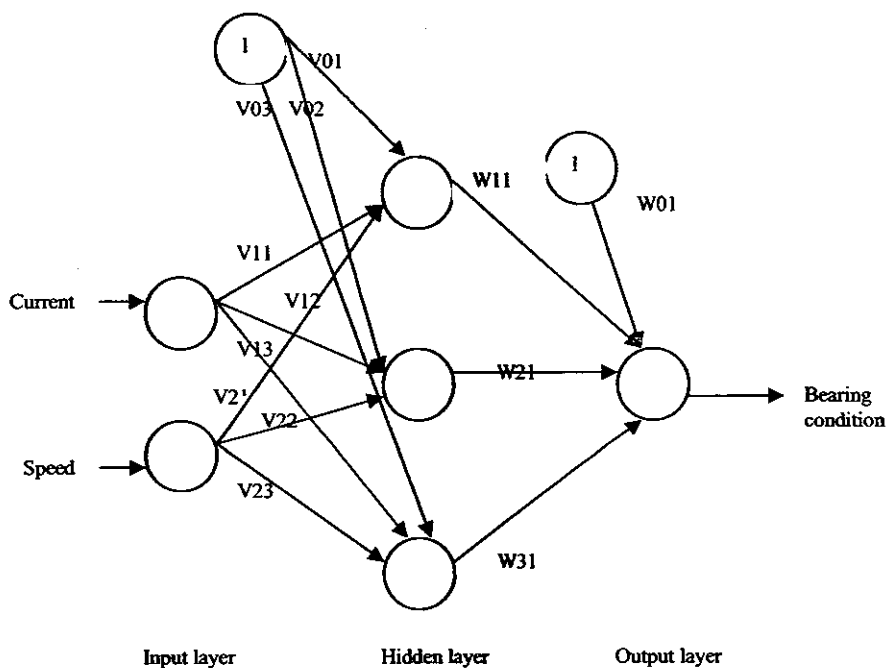


Figure 3.11 Structure of BP Network for Fault Detection

3.3.4 Simulation results

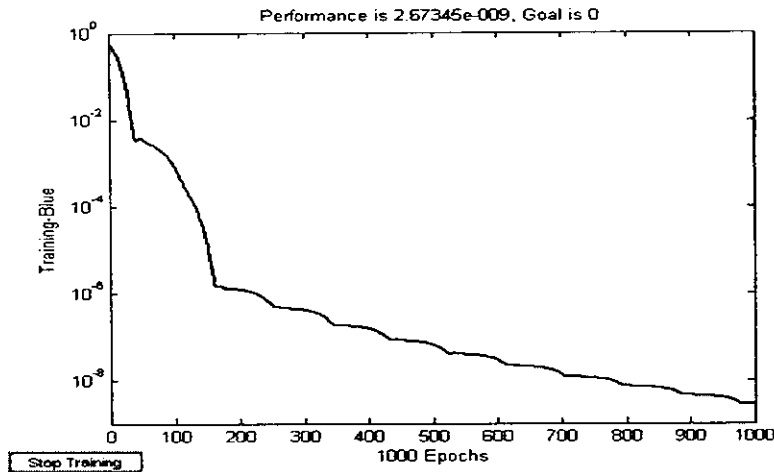


Figure 3.12 Epoch Vs Error characteristics

The Figure 3.12 shows the Epoch Vs Error characteristics by monitoring the stator current (r.m.s value) and rotor speed. The results obtained are shown in Table 3.1. The error is calculated by the formula,

$$\frac{|\text{Actual value} - \text{observed value}|}{\text{Actual value}} * 100$$

The actual value is obtained from experimental setup and the observed value is obtained from NN simulation.

Table 3.1 Simulation results

S.No.	Current	Speed	Temperature	% Error Obtained in NNFD
1	6.2	2655	48	0.79
2	6.5	2640	48	1.40
3	7.0	2610	53	0.03
4	8.5	2590	57	0
5	10.0	2600	72	0.02
6	11.0	2500	73	0.02
Average percentage error obtained				0.376

CHAPTER 4

FUZZY LOGIC BASED FAULT DIAGNOSIS

4.1 INTRODUCTION

Problems in the real world quite often turn out to be complex owing to an element of uncertainty either in the parameters, which define the problem, or in the situations in which the problem occurs.

The uncertainty may arise due to partial information about the problem, or due to information which is not fully reliable, or due to inherent imprecision in the language with which the problem is defined, or due to receipt of information from more than one source about the problem which is conflicting. It is in such situations that fuzzy set theory exhibits immense potential for effective solving of the uncertainty in the problem. Fuzziness means 'vagueness'. Fuzzy set theory is an excellent mathematical tool to handle the uncertainty arising due to vagueness.

Fuzzy logic systems are universal function approximators. In general, the goal of the fuzzy logic system is to yield a set of outputs for given inputs in a non-linear system, without using any mathematical model, but by using linguistic rules. It has many advantages. They are

- Fuzzy logic is conceptually easy to understand. The mathematical concepts behind fuzzy reasoning are very simple. What makes fuzzy better is the "Naturalness" of its approach and not its far-reaching complexity.
- Fuzzy logic is flexible. With any given system, it's easy to massage it or layer more functionality on top of it without starting again from scratch.
- Fuzzy logic is tolerant of imprecise data. Everything is imprecise if you look closely enough, but more than that, most things are imprecise even on careful inspection. Fuzzy reasoning builds this understanding into the process rather than tacking it onto the end.
- Fuzzy logic can model nonlinear functions of arbitrary complexity. You can create a fuzzy system to match any set of input-output data. This process is made particularly easy by adaptive techniques like Adaptive Neuro-Fuzzy Inference Systems (ANFIS), which are available in the Fuzzy Logic Toolbox.
- Fuzzy logic can be built on top of the experience of experts. In direct contrast

to neural networks, which take training data and generate opaque, impenetrable models, fuzzy logic lets you rely on the experience of people who already understand your system.

- Fuzzy logic can be blended with conventional control techniques. Fuzzy systems don't necessarily replace conventional control methods. In many cases fuzzy systems augment them and simplify their implementation.
- Fuzzy logic is based on natural language. The basis for fuzzy logic is the basis for human communication. This observation underpins many of the other statements about fuzzy logic.

4.2 MAMDANI FUZZY LOGIC INFERENCE SYSTEM

Mamdani-type of fuzzy logic controller contains four main parts, two of which perform transformations. The four parts are

- Fuzzifier (transformation 1)
- Fuzzy rule base
- Inference engine (fuzzy reasoning, decision-making logic)
- Defuzzifier (transformation 2)

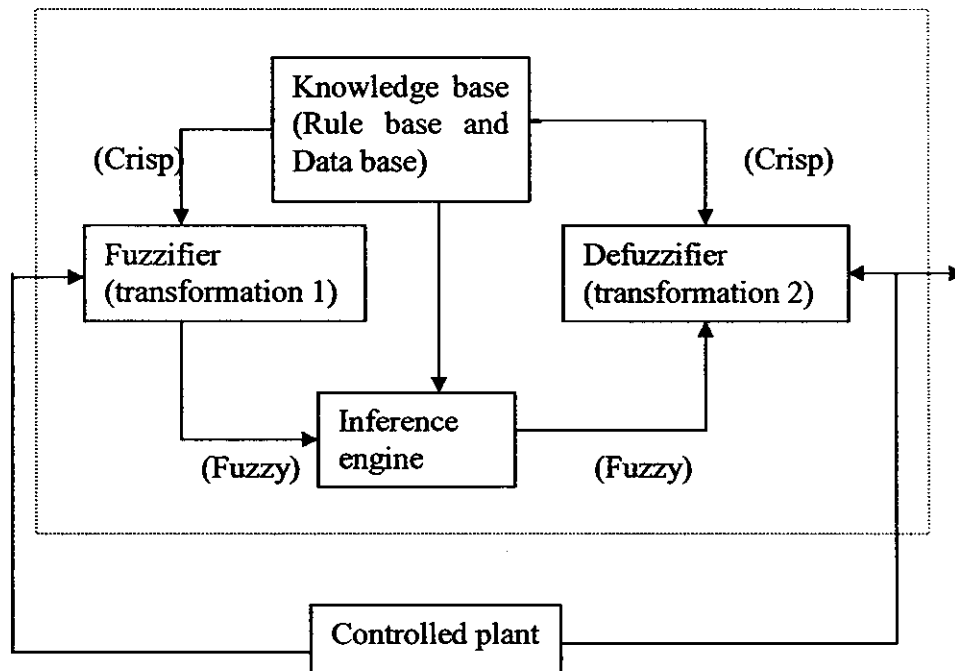


Figure 4.1 Mamdani Fuzzy Logic Inference Systems

4.2.1 Fuzzifier

The fuzzifier performs measurement of the input variables (input signals, real variables), scale mapping and fuzzification (transformation 1). Thus all the monitoring input signals are scaled and fuzzification means that the measured signals (crisp input quantities which have numerical values) are transformed into fuzzy quantities. This transformation is performed by using membership functions. In a conventional fuzzy logic controller, the number of membership functions and the shapes of these are initially determined by the user. A membership function has a value between 0 and 1, and it indicates the degree of belongingness of a quantity to a fuzzy set. If it is absolutely certain that the quantity belongs to the fuzzy set, then its value is 1 (it is 100% certain that the quantity belongs to this set), but if it is absolutely certain that it does not belong to this set then its value is 0. Similarly if for example the quantity belongs to the fuzzy set to an extent of 50%, then the membership function is 0.5.

There are many types of different membership functions, piecewise linear or continuous. Some of these are smooth membership functions, e.g. bell-shaped, sigmoid, Gaussian etc. and others are non-smooth, e.g. triangular, trapezoidal etc. the choice of the type of membership function used in a specific problem is not unique. Thus it is reasonable to specify parameterized membership functions, which can be fitted to a practical problem. If the number of elements in the universe X is very large or if a continuum is used for X then it is useful to have a parameterized membership function, where the parameters are adjusted according to the given problem. Parameterized membership functions play an important role in adaptive fuzzy systems, but are also useful for digital implementation. Due to their simple forms and high computational efficiency, simple membership functions, which contain straight-line segments, are used extensively in various implementations. Obviously, the triangular membership function is a special case of the trapezoidal one.

$$\mu_A'(x;a,b,c) = \begin{cases} 0 & x < a \\ (x-a)/(b-a) & a \leq x \leq b \\ (c-x)/(c-b) & b \leq x \leq c \\ 0 & x > c \end{cases} \quad (4.1)$$

Triangular membership function depends on three parameters a, b, c and can be described as follows by considering four regions.

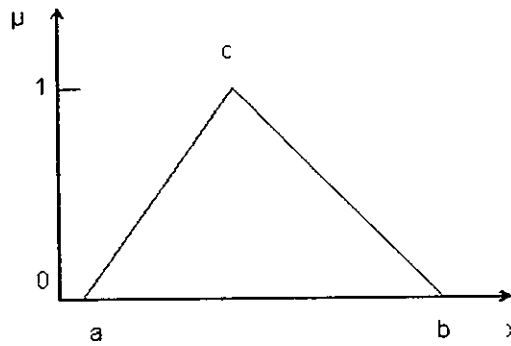


Figure 4.2 Triangular membership functions

A triangular membership function is shown in Figure 4.2 are used for both the input and output variable and the points a, b, c are also denoted. Alternatively, it is possible to give a more compact form

$$\mu^A(x; a, b, c) = \max \{ \min [(x-a) / (b-a), (c-x) / (c-b)], 0 \} \quad (4.2)$$

The detection of bearing fault severity is considered by utilizing Mamdani-style fuzzy inference and using as input variables are speed and current .Low, medium, and high are the three membership functions used for the input variables. Poor, fair, and good are the membership functions used for output variable.

4.2.2 Fuzzy rules

The knowledge base consists of the database and the linguistic control rule base. The database provides the information, which is used to define the linguistic control rules and the fuzzy data manipulation in the fuzzy logic controller. The rule base specifies the control goal actions by means of a set of linguistic control rules. In other words, the rule base contains rules such as would be provided by an expert. The fuzzy logic controller looks at the input signals and by using the expert rules determines the appropriate output signals (control actions). The rule base contains a set of if-then rules. The main methods of developing a rule base are:

- Using the experience and knowledge of an expert for the application and the control goals;
- Modeling the control action of the operator;
- Modeling the process;
- Using a self-organized fuzzy controller;
- Using artificial neural networks;

When the initial rules are obtained by using expert physical considerations, these can be formed by considering that the three main objectives to be achieved by the fuzzy logic controller are:

- Removal of any significant errors in the process output by suitable adjustment of the control output;
- Ensuring a smooth control action near the reference value (small oscillations in the process output are not transmitted to the control input);
- Preventing the process output exceeding user specified values;

By considering the two dimensional matrix of the input variables, each subspace is associated with a fuzzy output situation.

4.2.3 Inference engine

It is the kernel of a fuzzy logic controller and has the capability both of simulating human decision-making based on fuzzy concepts and of inferring fuzzy control actions by using fuzzy implication and fuzzy logic rules of inference as shown in Figure 4.3. In other words, once all the monitored input variables are transformed into their respective linguistic variables, the inference engine evaluates the set of if-then rules and thus result is obtained which is again a linguistic value for the linguistic variable. This linguistic result has to be then transformed into a crisp output value of the fuzzy logic control.

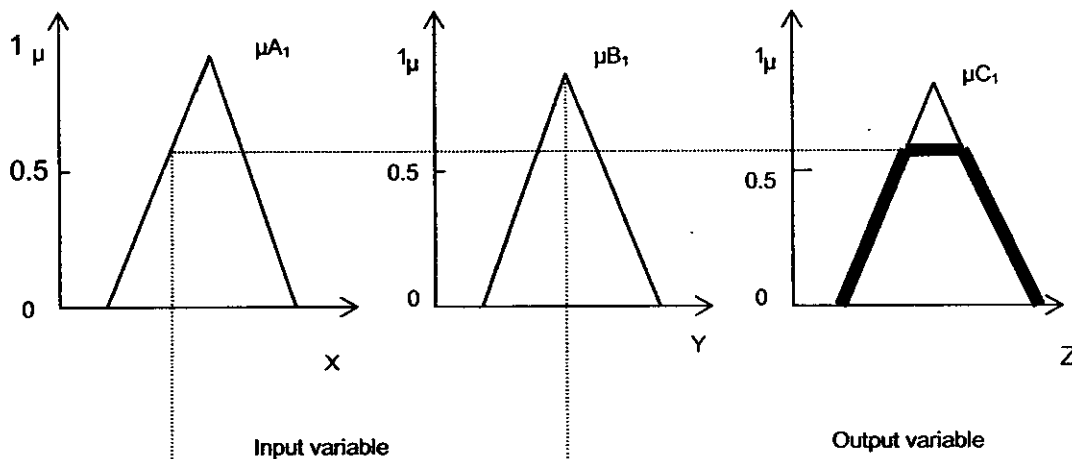


Figure 4.3 Graphical interpretation of fuzzification, inference

4.2.4 Defuzzifier

The second transformation is performed by the defuzzifier, which performs scale mapping as well as defuzzification. The defuzzifier yields a non-fuzzy, crisp control action from the inferred fuzzy control action by using the consequent membership functions of the

rules. There are many defuzzification techniques. They are centroid method, centre of gravity method, height method, mean of maxima method, first of maxima method, sum of maxima. In this project centroid method defuzzification technique is used as shown in Figure 4.4. It follows that the output value is

$$z^* = \int \mu_c(z) \cdot z \, dz / \int \mu_c(z) \, dz \quad (4.3)$$

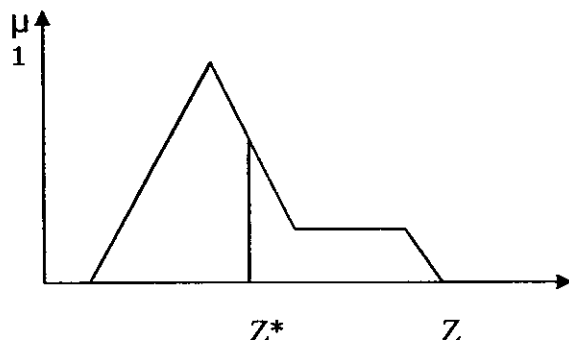


Figure 4.4 Centroid Defuzzification Method

4.3 FUZZY LOGIC BASED FAULT DETECTION METHOD

The fuzzy inference system used is Mamdani. Hybrid membership function is used for both the input speed and current and for the output temperature. Three membership functions for the input variables as well as for output variable are selected.

Table 4.1 Fuzzy Rules

CURRENT	SPEED	TEMPERATURE
LOW	MEDIUM	FAIR
LOW	HIGH	GOOD
MEDIUM	LOW	FAIR
MEDIUM	MEDIUM	FAIR
MEDIUM	HIGH	GOOD
HIGH	MEDIUM	BAD
HIGH	LOW	BAD

Table 4.1 gives the seven fuzzy rules used for bearing fault detection. The hybrid membership function is used for fuzzification. Table 4.2 shows the name, type and ranges of the membership functions used for the input and output variables.

Table 4.2 Membership functions

Variable	Name of the mf	Range	Type of the mf
Current	Low	6A – 8.3A	smf
	Medium	8.1A – 9.8A	Trimf
	High	9.5A – 12A	zmf
Speed	Low	2220rpm – 2400rpm	smf
	Medium	2390rpm – 2600rpm	trimf
	High	2570rpm – 2720rpm	zmf
Temperature	Good	40°C - 59°C	smf
	Fair	57°C - 70°C	trimf
	Bad	68°C - 75°C	zmf

The membership functions used for simulation are shown in Figures (Figure4.5– Figure 4.7). The Figure 4.8 shows the surface viewer of the FFD.

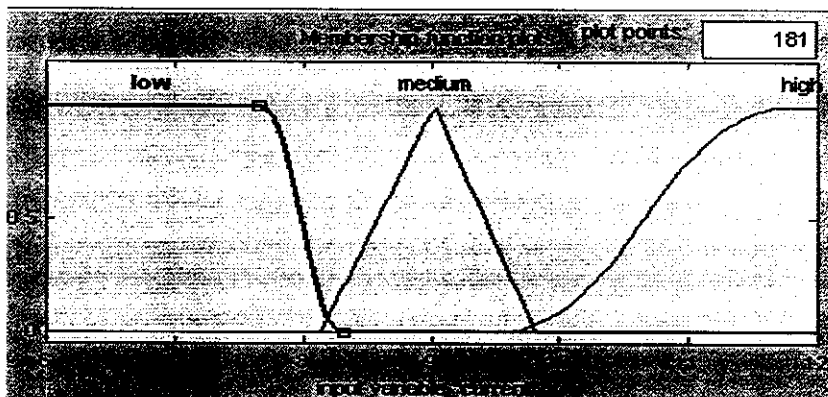


Figure 4.5 Input Membership Functions for current

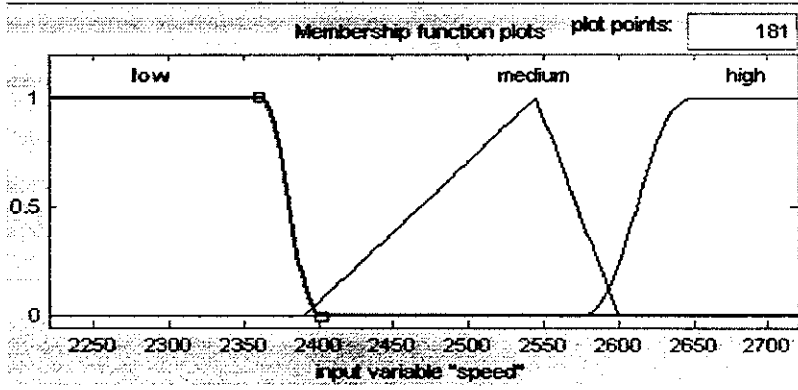


Figure 4.6 Input Membership Functions for speed

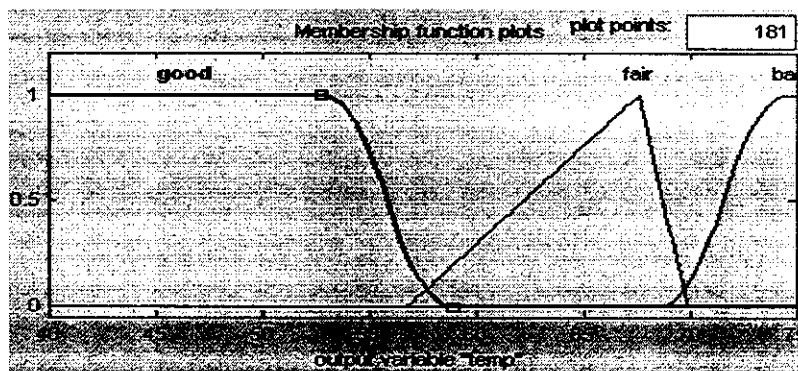


Figure 4.7 Output Membership Functions for temperature

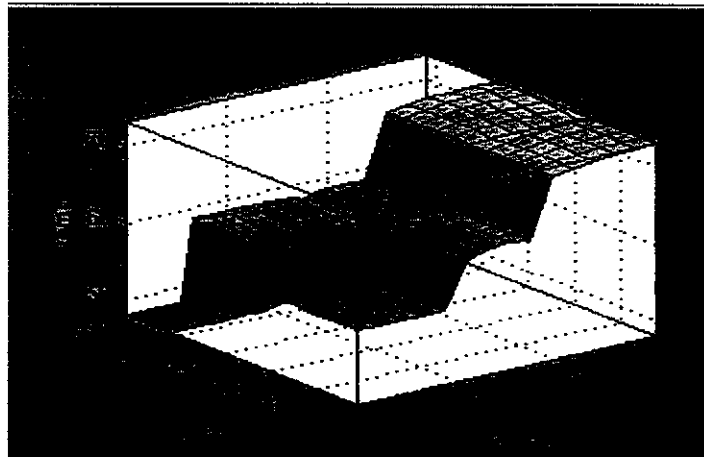


Figure 4.8 Surface Viewer

4.4 SIMULATION RESULTS

The error percentage is calculated for different inputs. The results obtained are shown in Table 4.3. The error is calculated by the formula,

$$\frac{|\text{Actual value} - \text{observed value}|}{\text{Actual value}} * 100$$

The actual value is obtained from experimental setup and the observed value is obtained from fuzzy logic simulation.

Table 4.3 Simulation results

S.No.	Current	Speed	Temperature	% Error Obtained in FFD
1	6.2	2655	48	0.208
2	6.5	2640	48	0
3	7.0	2610	53	0
4	8.5	2590	57	0
5	10.0	2600	72	0
6	11.0	2500	73	0.136
Average percentage error obtained				0.057

4.5 COMPARISON OF NEURAL NETWORK AND FUZZY LOGIC BASED FAULT DIAGNOSIS SYSTEM

Neural network approach is a black box approach, where the expert knowledge is hidden in the black box system in the form of weights and biases of the neural network. However, in fuzzy logic based system the actions of a human expert are clearly present in the rule base. Comparison of the neural network and fuzzy based fault diagnoses for monitoring the stator current and rotor speed is given in Table 4.4.

Table 4.4 Comparison of Neural network and fuzzy logic based fault diagnosis

S.No	Method	% Error
1	Neural Network based Diagnosis	0.376
2	Fuzzy Logic based Diagnosis	0.057

From the above table, it is inferred that the fuzzy logic based fault diagnosis gives reduced error compared with neural network based fault diagnosis

CHAPTER 5 HARDWARE IMPLEMENTATION

The hardware implementation is done using a dsPIC digital signal controller that uses the MCSA method for detecting the bearing fault. This method monitors the frequency components of stator current spectrum. Figure 5.1 shows the schematic diagram of the hardware implementation.

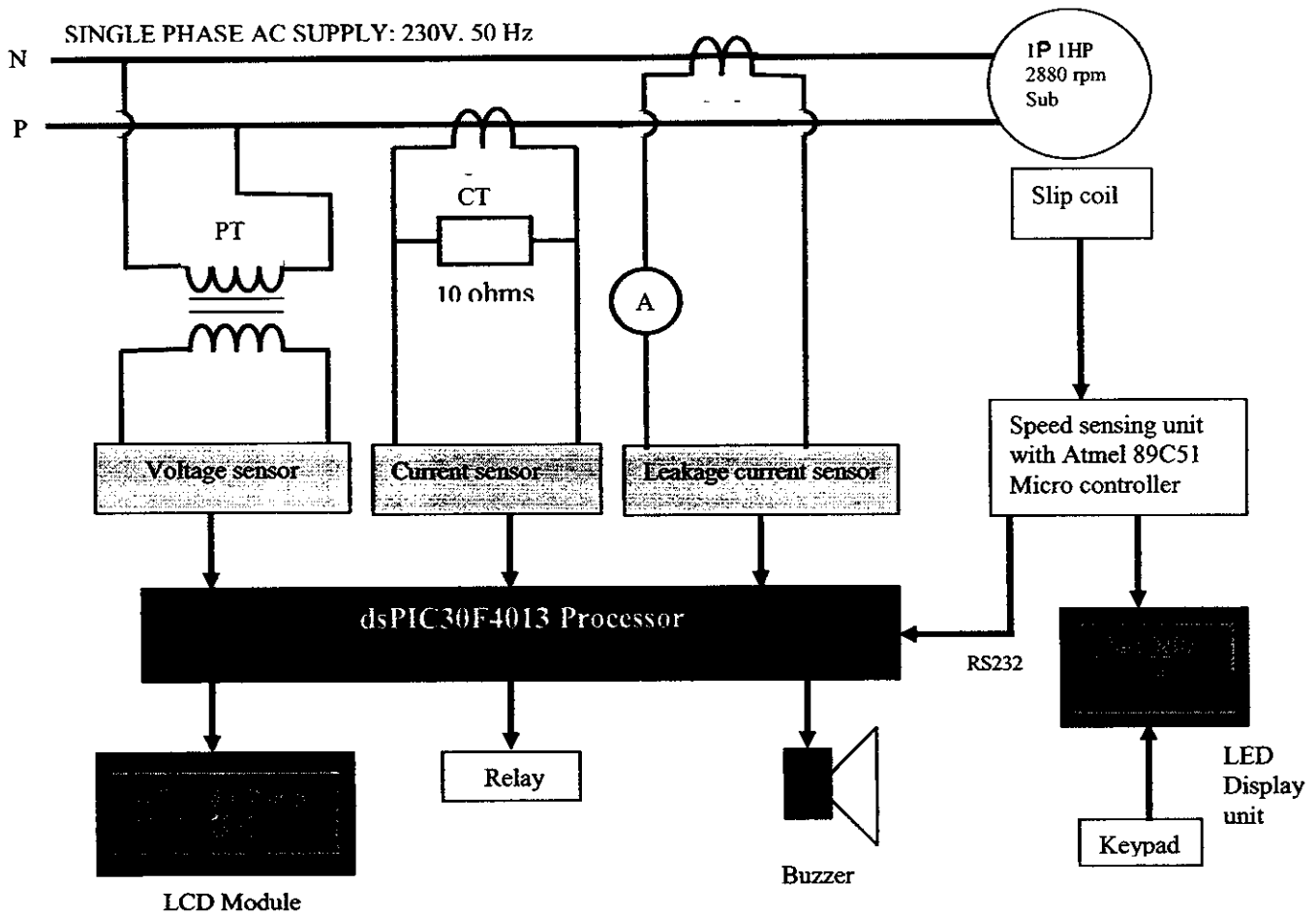


Figure 5.1 schematic diagram

5.1 HARDWARE DESCRIPTION

The hardware consists of several modules for measuring supply voltage, load current, speed and leakage current. The main part is the dsPIC dsc. These modules require dc supply in the range of +5V, +9V and +12V respectively for their operation. Figures 5.2 shows the power supply circuit for +5V and +12V. Figure 5.3 shows the power supply circuit for obtaining +9V.

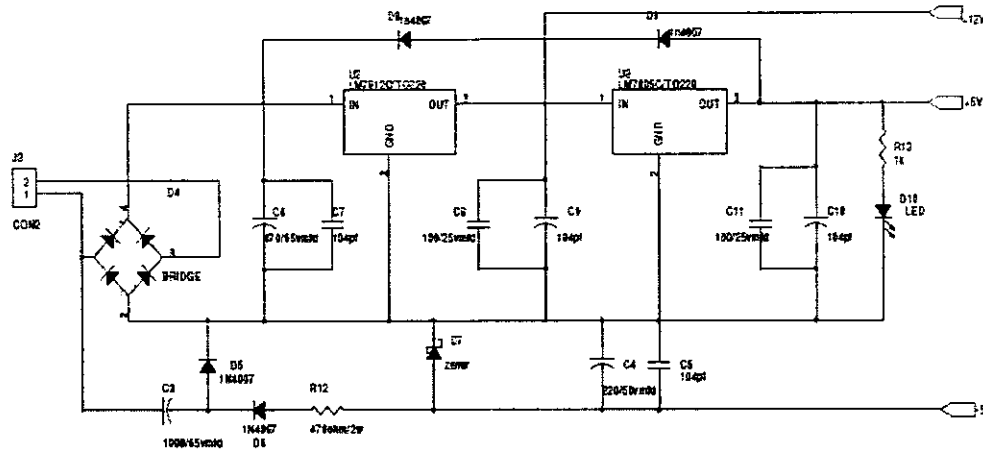


Figure 5.2 +5V and +12V power supply

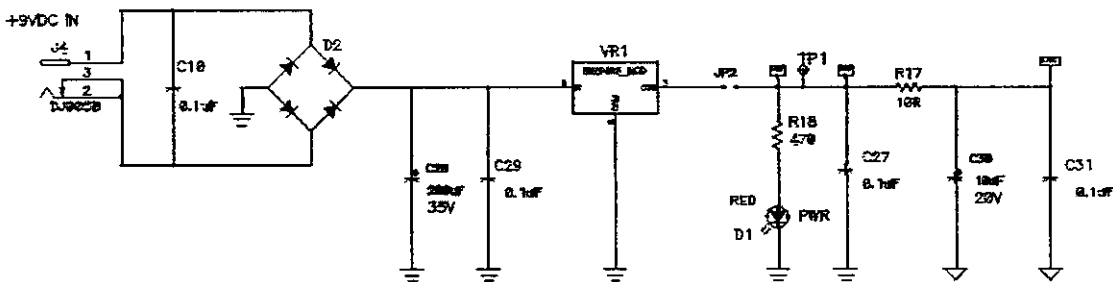


Figure 5.3 +9V power supply

The available AC input voltage is 230 V, 50 Hz. It is stepped down to 12 V AC by a step down transformer. It is rectified using a full bridge rectifier, which consists of four IN4007 diodes. The output obtained is filtered using capacitors to get a smooth DC. The obtained DC voltage is given through voltage regulators to regulate the voltage that varies due to changes in load or fluctuates due to AC input voltage. LM 7812 and LM 7805 are used to provide the required regulated supply respectively. The +9 V power supply is provided for giving supply to the dsPIC processor board.

5.2 BLOCK DIAGRAM

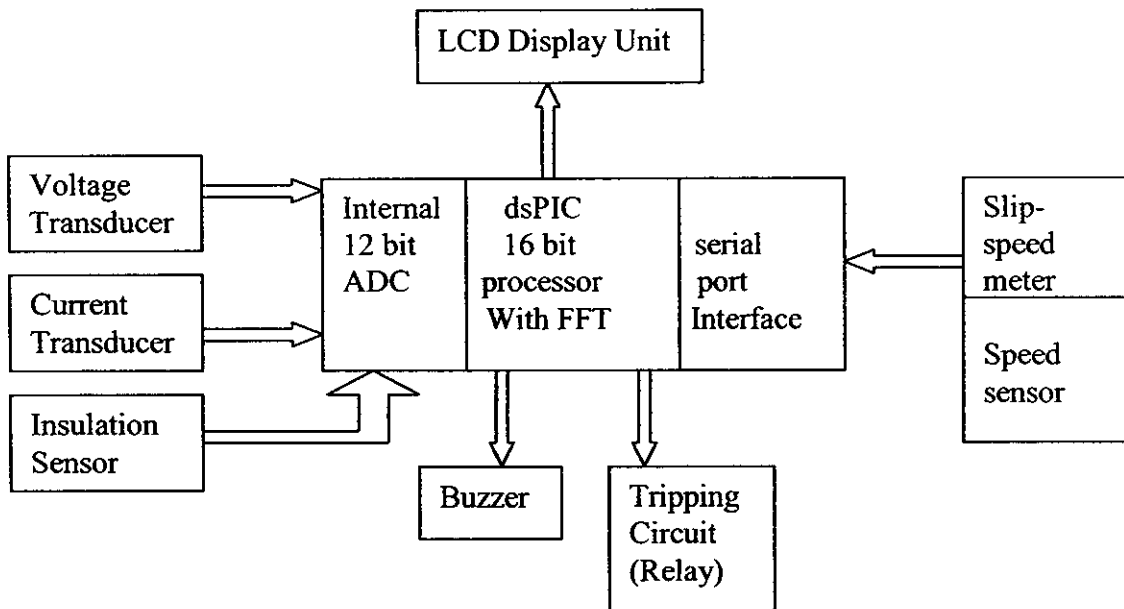


Figure 5.4 Fault diagnostic module

The block diagram of the fault diagnostic module is shown in Figure 5.4. It shows the various sensing parameters such as voltage, load current, leakage current and speed. These signals are processed in the dsPIC dsc for fault detection and given to the LCD and LED display units for monitoring purpose. Relay is provided for tripping purpose and buzzer is used as indication of a fault. The speed-sensing unit consists of a slip coil for sensing the speed and an ATMEL 89C51 micro controller is used for computing the speed. The speed sensing unit is connected to the dsPIC dsc using a RS-232 serial port communication.

5.3 VOLTAGE SENSING CIRCUIT

The voltage sensing circuit is shown in Figure 5.5. The PT is connected in parallel to the single-phase supply. The rating is 300V/6V. The output voltage of PT is stepped down and given to the inverting input terminal of the op-amp LM 358. For positive half cycle, diode D3 conducts and for negative half cycle diode D2 conducts. The obtained DC output is given to pin AN2 of the dsPIC processor. This circuit is called the precision diode and it is capable of rectifying input signals of the order of milli volt. By placing a diode in the feedback loop of the op-amp, the cut-in voltage can be eliminated by the open loop gain of the op-amp

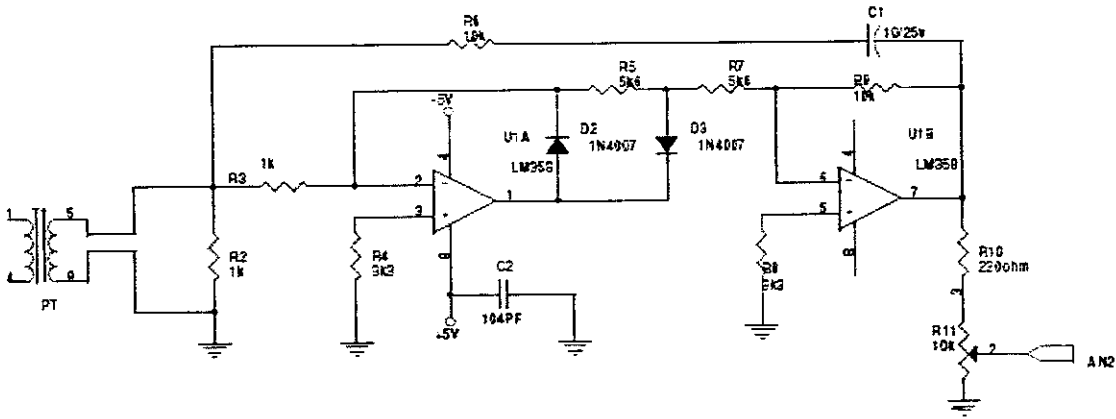


Figure 5.5 Voltage sensing circuit

5.4 CURRENT SENSING CIRCUIT

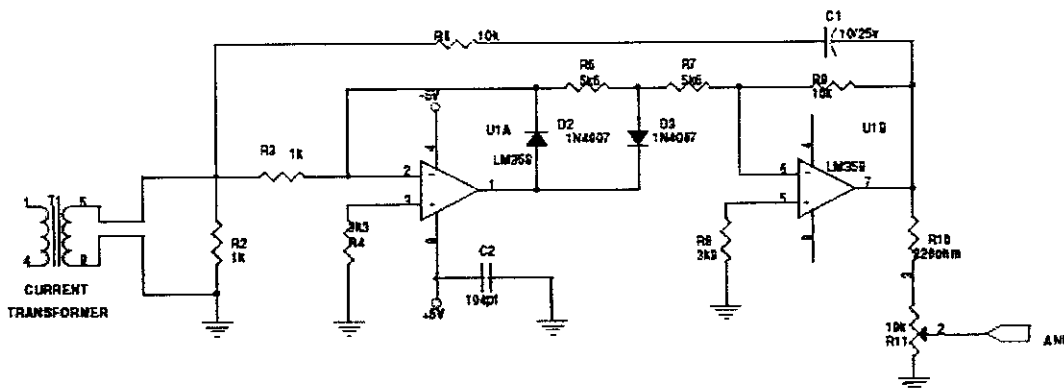


Figure 5.6 Current sensing circuit

The current sensing circuit is shown in Figure 5.6. The CT is connected in series to the input single-phase supply for monitoring the load current. The rating of the current sensing circuit is 25 A. The CT ratio is 5:1. The output from the CT is given as input to the inverting input terminal of the op-amp. As mentioned in the previous section, for positive half cycle diode D3 conducts and for negative half cycle diode D2 conducts. The obtained DC output is to the pin AN0 of the dsPIC processor.

5.5 LEAKAGE CURRENT SENSING CIRCUIT

The leakage current sensing circuit is shown in Figure 5.7. For measuring leakage current, the CT is connected to the neutral terminal before the neutral is earthed. The secondary winding of the CT is connected to the leakage current sensing unit via an ammeter. When there is leakage current, the current will flow through the neutral terminal and the CT senses this current. It is indicated in the ammeter. For submersible pumps the leakage current

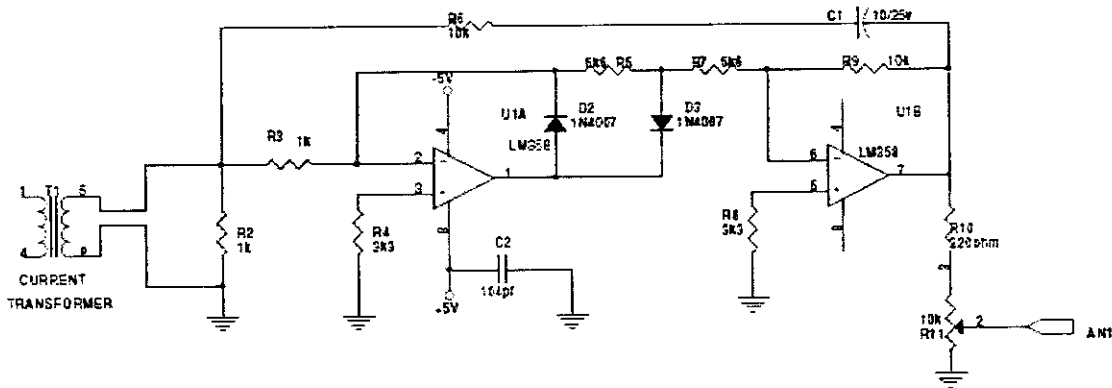


Figure 5.7 Leakage current sensing unit

should not exceed 160mA. Below this value, the insulation condition will be good. The CT ratio is 1:1. The output from the CT is given as input to the inverting input terminal of the op-amp. As mentioned in the previous section, for positive half cycle diode D3 conducts and for negative half cycle diode D2 conducts. The obtained DC output is to the pin AN1 of the dsPIC processor.

5.6 SPEED SENSING UNIT

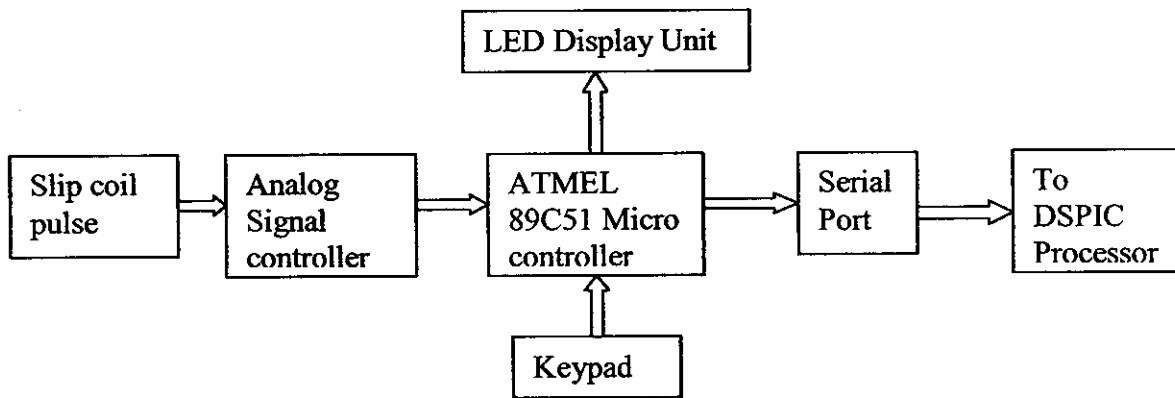


Figure 5.8 Speed sensing unit

The speed-sensing unit is shown in Figure 5.8. Slip coil pulse is given to the analog signal controller. Analog signal controller is used to control the amplitude of the input signal to the micro controller. ATMEL 89C51 micro controller is used for computing the speed proportional to the input pulse. It is connected to the DSPIC processor through a serial port. A LED display unit is provided for displaying the speed. Keypad option is also provided to select frequency, slip, or speed to display on the LED display.

5.7 RELAY AND BUZZER CIRCUIT

The relay and buzzer circuit is shown in Figure 5.9. If a bearing fault occurs, the current will be high. It will be displayed on the LCD unit. At the same time, the trip circuit is activated which stops the pump. Buzzer indication is also provided. The relay and buzzer are connected to the pin RB0 of the dsPIC processor.

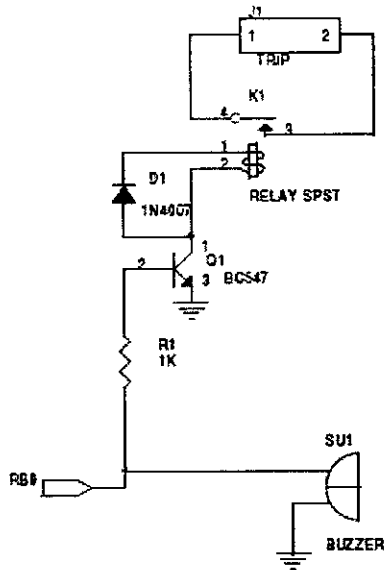


Figure 5.9 Relay and Buzzer circuit

5.8 INTRODUCTION TO dsPIC

The micro controller used in this project is an advanced processor of the microchip family of devices. They contain extensive Digital Signal Processor (DSP) functionality within high-performance 16-bit micro controller architecture with modified RISC CPU and CMOS technology. Advantages are low power consumption and high speed.

5.8.1 dsPIC30F4013:

The digital signal controller used in this project is dsPIC30F4013. It is a 40-pin PDIP with modified Harvard architecture. Flash technology is used so that the data can be retained even when the power is switched off.

5.8.2 High-Performance Modified RISC CPU:

- Modified Harvard architecture
- C compiler optimized instruction set architecture
- Flexible addressing modes
- 83 base instructions

- 24-bit wide instructions, 16-bit wide data path
- Up to 48 Kbytes on-chip Flash program space
- 2 Kbytes of on-chip data RAM
- 1 Kbytes of nonvolatile data EEPROM
- 16 x 16-bit working register array
- Up to 30 MIPS operation:
 - DC to 40 MHz external clock input
 - 4 MHz-10 MHz oscillator input with PLL active (4x, 8x, 16x)
- Up to 33 interrupt sources:
 - 8 user selectable priority levels
 - 3 external interrupt sources
 - 4 processor traps

5.8.3 DSP Features:

- Dual data fetch
- Modulo and Bit-Reversed modes
- Two 40-bit wide accumulators with optional saturation logic
- 17-bit x 17-bit single-cycle hardware fractional/integer multiplier
- All DSP instructions are single cycle
 - Multiply-Accumulate (MAC) operation
- Single-cycle ± 16 shift

5.8.4 Peripheral Features:

- High-current sink/source I/O pins: 25-mA/25 mA
- Up to five 16-bit timers/counters; optionally pair up 16-bit timers into 32-bit timer modules
- Up to four 16-bit Capture input functions
- Up to four 16-bit Compare/PWM output functions
- Data Converter Interface (DCI) supports common audio Codec protocols, including I2S and AC'97
- 3-wire SPI module (supports 4 Frame modes)
- I2C™ module supports Multi-Master/Slave mode and 7-bit/10-bit addressing
- Up to two addressable UART modules with FIFO buffers
- CAN bus module compliant with CAN 2.0B standard

5.8.5 Analog Features:

- 12-bit Analog-to-Digital Converter (ADC) with:
 - 200 ksps conversion rate
 - Up to 13 input channels
 - Conversion available during Sleep and Idle
- Programmable Low-Voltage Detection (PLVD)
- Programmable Brown-out Reset

5.8.6 Special Micro controller Features:

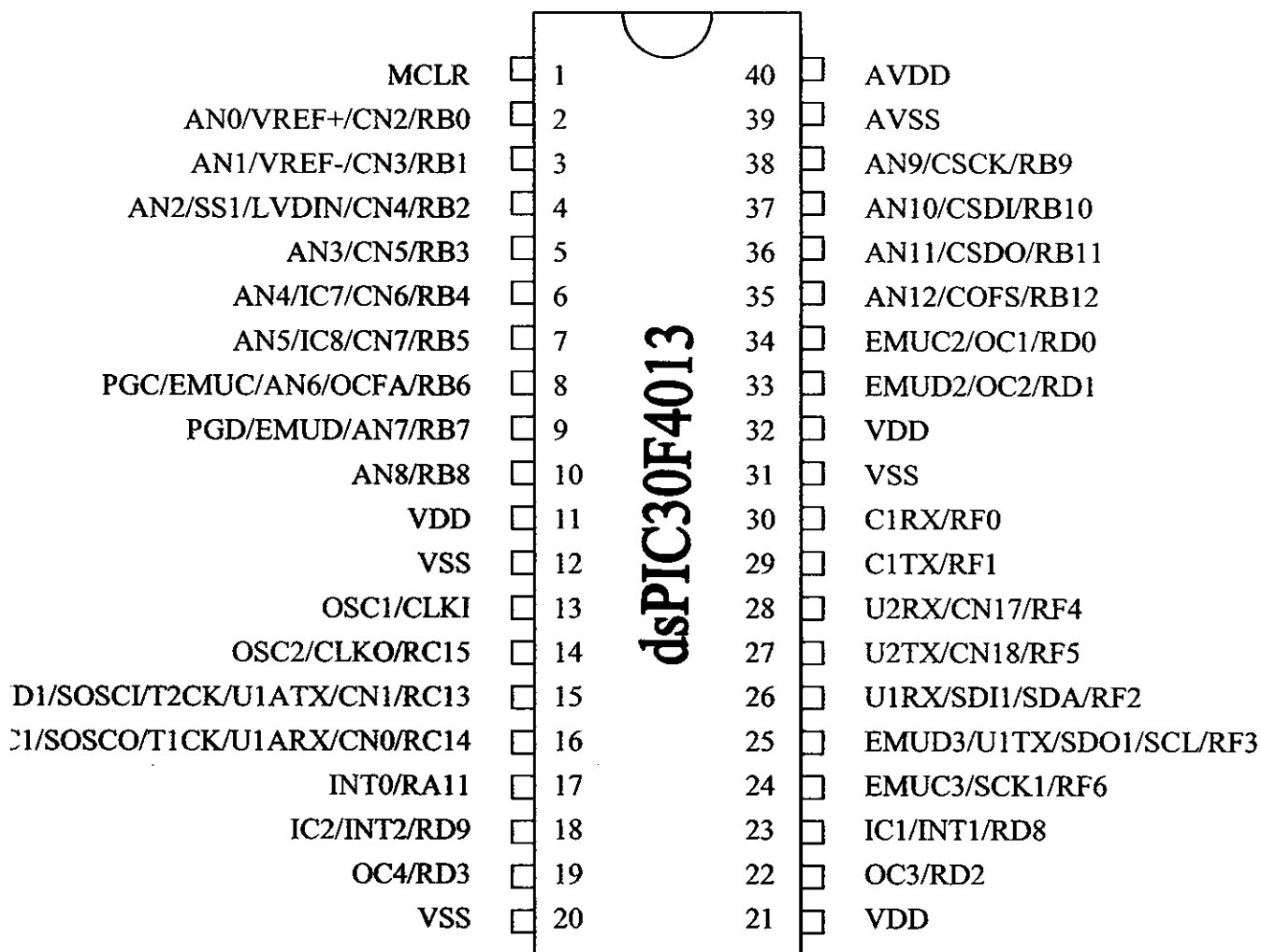
- Enhanced Flash program memory:
 - 10,000-erase/write cycle (min.) for industrial temperature range, 100K (typical)
- Data EEPROM memory:
 - 100,000-erase/write cycle (min.) for industrial temperature range, 1M (typical)
- Self-reprogrammable under software control
- Power-on Reset (POR), Power-up Timer (PWRT) and Oscillator Start-up Timer (OST)
- Flexible Watchdog Timer (WDT) with on-chip low-power RC oscillator for reliable operation
- Fail-Safe Clock Monitor operation:
 - Detects clock failure and switches to on-chip low-power RC oscillator
- Programmable code protection
- In-Circuit Serial Programming™ (ICSP™)
- Selectable Power Management modes:
 - Sleep, Idle and Alternate Clock modes

5.8.7 CMOS Technology:

- Low-power, high-speed Flash technology
- Wide operating voltage range (2.5V to 5.5V)
- Industrial and Extended temperature ranges
- Low-power consumption

5.8.8 dsPIC30F4013 – PIN CONFIGURATION

The pin out diagram for DSP30F4013 is shown in the Figure 5.10



5.8.9 CPU ARCHITECTURE

The core has a 24-bit instruction word. The Program Counter (PC) is 23 bits wide with the Least Significant bit (LSb) always clear, and the Most Significant bit (MSb) is ignored during normal program execution, except for certain specialized instructions. An instruction prefetch mechanism is used to help maintain throughput. The working register array consists of 16-bit x 16-bit registers, each of which can act as data, address or offset registers. One working register (W15) operates as software Stack Pointer for interrupts and calls. The data space is 64 Kbytes (32K words) and is split into two blocks, referred to as X and Y data memory. Each block has its own independent Address Generation Unit (AGU).

5.8.10 STATUS REGISTER

The dsPIC DSC core has a 16-bit STATUS register (SR), the Least Significant Byte (LSB) of which is referred to as the SR Low byte (SRL) and the Most Significant Byte (MSB) as the SR High byte (SRH). The upper byte of the STATUS register contains the DSP adder/subtractor Status bits, the DO Loop Active bit (DA) and the Digit Carry (DC) Status bit.

5.8.11 PROGRAM COUNTER

The program counter is 23 bits wide; bit 0 is always clear. Therefore, the PC can address up to 4M instruction words.

5.8.12 MEMORY ORGANIZATION

Program Address Space

The program address space is 4M instruction words. It is addressable by a 24-bit value from either the 23-bit PC, table instruction Effective Address (EA), or data space EA, when program space is mapped into data space the program space address is incremented by two between successive program words in order to provide compatibility with data space addressing. User program space access is restricted to the lower 4M instruction word address range (0x000000 to 0x7FFFFE)

Data Address Space

The core has two data spaces. The data spaces can be considered either separate (for some DSP instructions), or as one unified linear address range (for MCU instructions). The data spaces are accessed using two Address Generation Units (AGUs) and separate data paths.

5.8.13 FLASH PROGRAM MEMORY

The dsPIC30F family of devices contains internal program Flash memory for executing user code. There are two methods by which the user can program this memory:

1. Run-Time Self-Programming (RTSP)
2. In-Circuit Serial Programming™ (ICSP™)

In-Circuit Serial Programming (ICSP)

dsPIC30F devices can be serially programmed while in the end application circuit. This is simply done with two lines for Programming Clock and Programming Data (which are named PGC and PGD, respectively), and three other lines for Power (VDD), Ground (VSS) and Master Clear (MCLR).

Run-Time Self-Programming (RTSP)

RTSP is accomplished using TBLRD (table read) and TBLWT (table write) instructions. With RTSP, the user may erase program memory, 32 instructions (96 bytes) at a time and can write program memory data, 32 instructions (96 bytes) at a time.

5.8.14 I/O PORTS

All of the device pins (except VDD, VSS, MCLR and OSC1/CLKI) are shared between the peripherals and the parallel I/O ports. All I/O input ports feature Schmitt Trigger inputs for improved noise immunity. All port pins have three registers directly associated with the operation of the port pin. The Data Direction register (TRISx) determines whether the pin is an input or an output. Reads from the latch (LATx), read the latch. Writes to the latch, write the latch (LATx). Reads from the port (PORTx), read the port pins and writes to the port pins, write the latch (LATx). The use of the ADPCFG and TRIS registers control the operation of the A/D port pins.

5.8.15 INTERRUPTS

The dsPIC30F sensor and general purpose families have up to 41 interrupt sources and 4 processor exceptions (traps), which must be arbitrated, based on a priority scheme.

5.8.16 TIMER2/3 MODULE

The dsPIC30F4013 dsc has five timers namely Timer1, Timer2/3 and Timer 4/5. Timer 2/3 is used in this project. It is a 32-bit timer (which can be configured as two 16-bit timers) with selectable operating modes.

5.8.17 I²C MODULE

The Inter-Integrated Circuit (I²CTM) module provides complete hardware support for both Slave and Multi-Master modes of the I²C serial communication standard, with a 16-bit interface. I²C REGISTERS I2CCON and I2CSTAT are control and STATUS registers, respectively. The I2CCON register is readable and writeable. The lower 6 bits of I2CSTAT are read-only. The remaining bits of the I2CSTAT are read/write. I2CRSR is the shift register used for shifting data, whereas I2CRCV is the buffer register to which data bytes are written, or from which data bytes are read. The I2CADD register holds the slave address.

5.8.18 12-BIT ANALOG-TO-DIGITAL CONVERTER (ADC) MODULE

The 12-bit Analog-to-Digital Converter (ADC) allows conversion of an analog input signal to a 12-bit digital number. This module is based on Successive Approximation Register (SAR) architecture and provides a maximum sampling rate of 200 ksp/s. The A/D module has up to 16 analog inputs, which are multiplexed into a sample and hold amplifier.

The output of the sample and hold is the input into the converter, which generates the result. The analog reference voltage is software selectable to either the device supply voltage (AVDD/AVSS) or the voltage level on the (VREF+/VREF-) pin. The A/D converter has a unique feature of being able to operate while the device is in Sleep mode with RC oscillator selection.

The A/D module has six 16-bit registers:

- A/D Control Register 1 (ADCON1)
- A/D Control Register 2 (ADCON2)
- A/D Control Register 3 (ADCON3)
- A/D Input Select Register (ADCHS)
- A/D Port Configuration Register (ADPCFG)
- A/D Input Scan Selection Register (ADCSSL)

The ADCON1, ADCON2 and ADCON3 registers control the operation of the A/D module. The ADCHS register selects the input channels to be converted. The ADPCFG register configures the port pins as analog inputs or as digital I/O. The ADCSSL register selects inputs for scanning.

5.8.19 SPI MODULE

The Serial Peripheral Interface (SPI) module is a synchronous serial interface. It is useful for communicating with other peripheral devices, such as EEPROMs, shift registers, display drivers and A/D converters, or other micro controllers.

5.8.20 LCD MODULE

A 2x16 ASCII-text LCD is provided on the dsPICDEM2 Development Board. dsPIC30F devices installed on the dsPICDEM2 development board may use this LCD to display characters. The interface to the LCD is via a 2-wire Serial Peripheral Interface (SPI™).

5.8.21 MPLAB ICD 2 In-Circuit Debugger

Microchip's In-Circuit Debugger, MPLAB ICD 2, is a powerful, low-cost, run-time development tool, connecting to the host PC via an RS-232 or high-speed USB interface. This tool is based on the Flash PIC MCUs and can be used to develop for these and other PIC MCUs and dsPIC DSCs. The MPLAB ICD 2 utilizes the in-circuit debugging capability built into the Flash devices. This feature, along with Microchip's In-Circuit Serial Programming™ (ICSPTM) protocol, offers cost effective, in-circuit flash debugging from the graphical user interface of the MPLAB Integrated Development Environment. The user using

configuration options in MPLAB IDE may select one of four pairs of Debug I/O pins. These pin pairs are named EMUD/EMUC, EMUD1/EMUC1, EMUD2/EMUC2 and MUD3/EMUC3.

5.8.22 MPLAB C18 and MPLAB C30C Compilers

The MPLAB C18 and MPLAB C30 Code Development Systems are complete ANSI C compilers for Microchip's PIC18 family of micro controllers and the dsPIC30, dsPIC33 and PIC24 family of digital signal controllers. These compilers provide powerful integration capabilities, superior code optimization and ease of use not found with other compilers. For easy source level debugging, the compilers provide symbol information that is optimized to the MPLAB IDE debugger.

5.8.23 MPLAB ASM30 Assembler, Linker and Librarian

MPLAB ASM30 Assembler produces relocatable machine code from symbolic assembly language for dsPIC30F devices. MPLAB C30 C Compiler uses the assembler to produce its object file. The assembler generates relocatable object files that can then be archived or linked with other relocatable object files and archives to create an executable file.

5.8.24 MPLAB SIM Software Simulator

The MPLAB SIM Software Simulator allows code development in a PC-hosted environment by simulating the PIC MCUs and dsPIC® DSCs on an instruction level. The MPLAB SIM Software Simulator fully supports symbolic debugging using the MPLAB C18 and MPLAB C30 C Compilers, and the MPASM and MPLAB ASM30 Assemblers. The software simulator offers the flexibility to develop and debug code outside of the hardware laboratory environment, making it an excellent, economical software development tool.

The photograph of the fault diagnostic module is shown in Figure 5.11

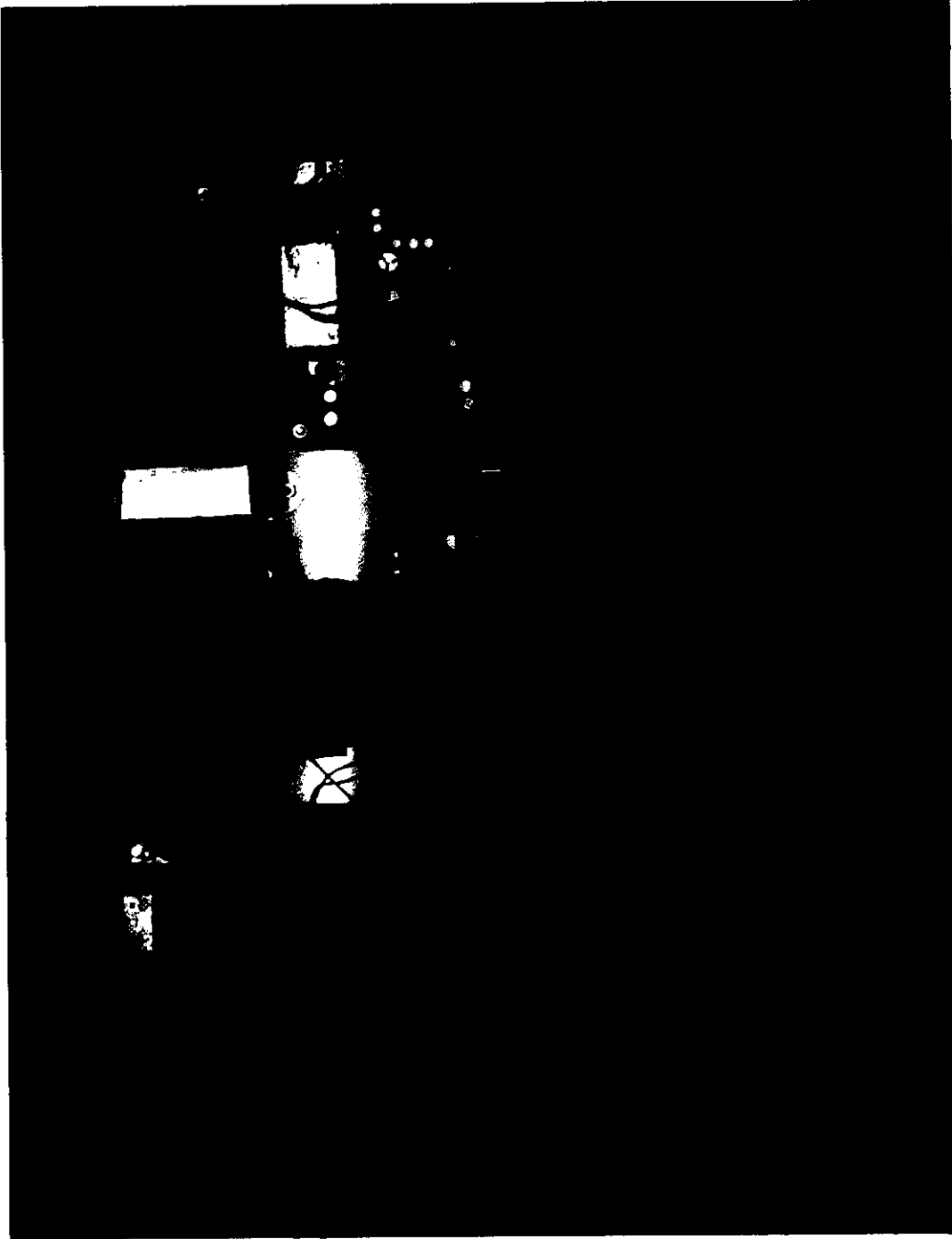


Figure 5.11 photograph of the fault diagnostic module

5.9 SOFTWARE DESCRIPTION

The software part contains the flowchart for the monitoring system and the program of the dsPIC processor. Figure 5.12 shows the flowchart.

5.9.1 FLOW CHART

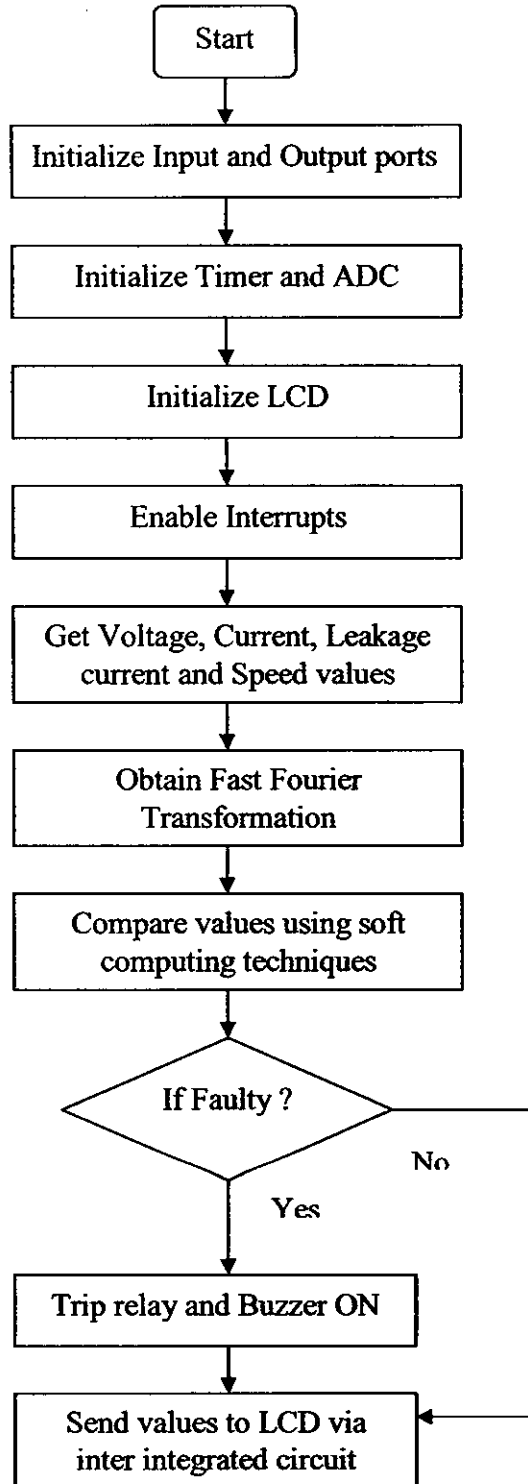


Figure 5.12 Flow chart

CONCLUSION

The project has investigated the feasibility of detecting bearing faults using a spectrum of the stator current for a single-phase 1HP submersible pump. Since bearings support the rotor, a bearing defect also produces variations in air gap length of the machine. These variations generate noticeable changes in the stator current spectrum. Measured current spectrums were presented separately for healthy and faulty bearings to verify the fault. The frequency components of the stator current pattern have been experimentally analyzed by Fast Fourier Transformation. A hardware implementation using embedded processor is developed for on-line monitoring and experimental results are obtained. Neural networks and fuzzy logic based fault detection have been used to perform pump bearing fault diagnosis based on the extracted information features. The performance of the neural network and fuzzy logic based fault detection is compared in terms of error percentage. From the simulation results, it is inferred that the fuzzy logic based fault detection gives the reduced percentage error than the neural network based fault detection. Thus, this work suggests that non-invasive diagnostic system of motor current signature analysis monitoring can lead to an improvement in the reliability of diagnosis of bearings.

REFERENCES

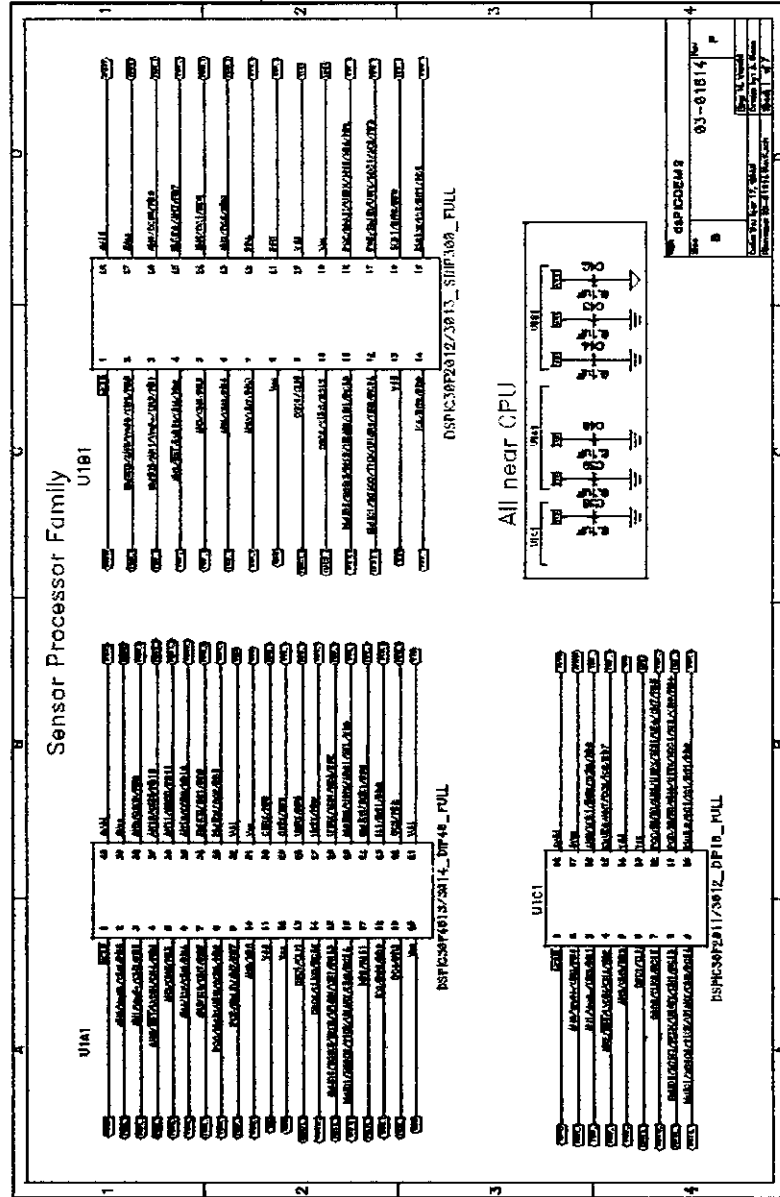
- [1] M.E. Steele, R.A Ashen, and L.G.Knight, "An electrical method for condition monitoring of motors", in int.conf.Elec.Mach-Design and applicat. , no 213, pp 231-235, July 1982.
- [2] " Methods of motor current signature analysis", Elec.Mach.Power sys, vol.20. No.5, pp 463-474, Sept, 1992.
- [3] M.E.H Benbouzid, "A review of Induction motors signature analysis as a medium for faults detection", IEEE Trans.ind.elect, Vol 47, No 5, Oct 2000.
- [4] R.R.Schoen et al, " Motor bearing damage detection using stator current monitoring", IEEE Trans.ind.applicat, Vol 31, pp 1274-1279, Nov/Dec 1995.
- [5] Bo Li et al, " Neural network based motor rolling bearing fault diagnosis", IEEE Trans.ind.elect, Vol 47, No 5, Oct 2000.
- [6] R.C.Kryter et al., " Condition monitoring of machinery using motor current signature analysis," sound Vib, pp 14-21, Sept 1989.
- [7] J Penman et al., " Condition monitoring of electrical drives," Proc.inst.Elect.Eng, pt.b Vol. 133, pp 142-148, May 1986.
- [8] S. Chen et al., "A new approach to motor condition monitoring in induction motor drives,"IEEE Trans.Ind.Applicat., Vol.30, pp. 905-911 July/Aug. 1994.
- [9] Fiorenzo fillipetti, Giovanni Franceschini, and Peter vas, "Recent development of induction motor drives fault diagnosis using AI techniques ", IEEE Trans. Industrial Electronics., vol.47,pp.994-1003.
- [10] Rajasekaran .S and Vijayalaksmi Pai.G.A, ' Neural networks, Fuzzy Logic and Genetic Algorithm Synthesis and Applications', Prentice Hall of India, 1995.

- [11] Linear Integrated Circuits, D.Rai Choudary and Shail Jain, Wiley Eastern Ltd., 1991.
- [12] Artificial Intelligence-Based Electrical Machines and Drives', Peter Vas 1999, Oxford University.
- [13] www.microchip.com
- [14] www.atmel.com

APPENDIX A

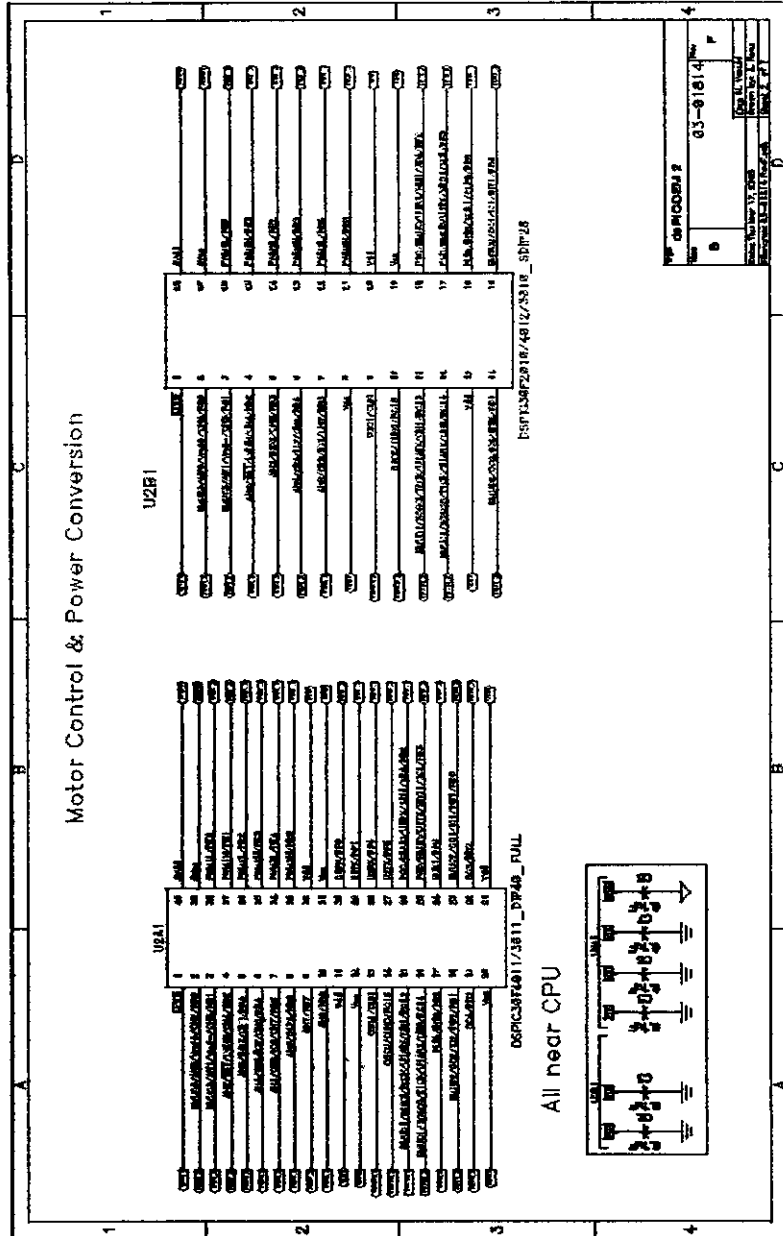
dsPICDEM 2 Development Board User's Guide

FIGURE A-2: dsPICDEM™ 2 DEVELOPMENT BOARD SCHEMATIC (SHEET 1 OF 7)



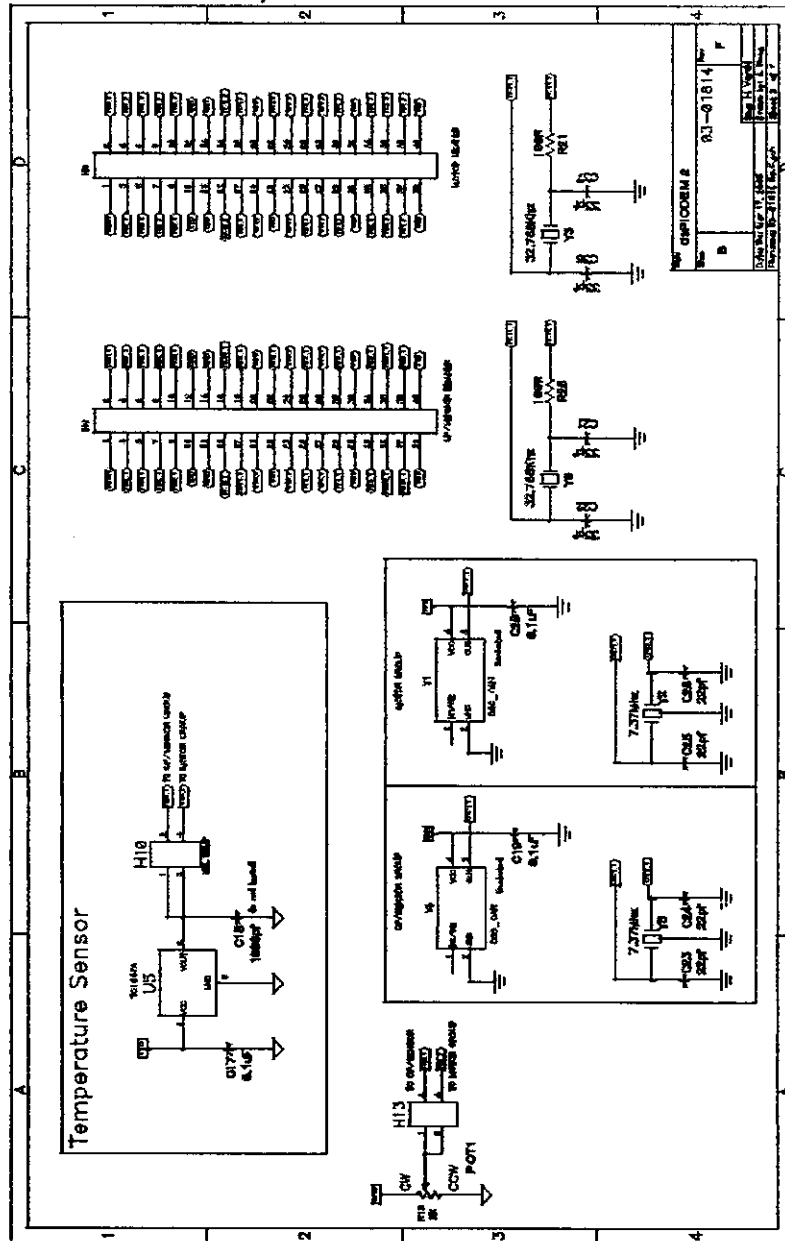
Hardware Drawings and Schematics

FIGURE A-3: dsPICDEM™ 2 DEVELOPMENT BOARD SCHEMATIC (SHEET 2 OF 7)



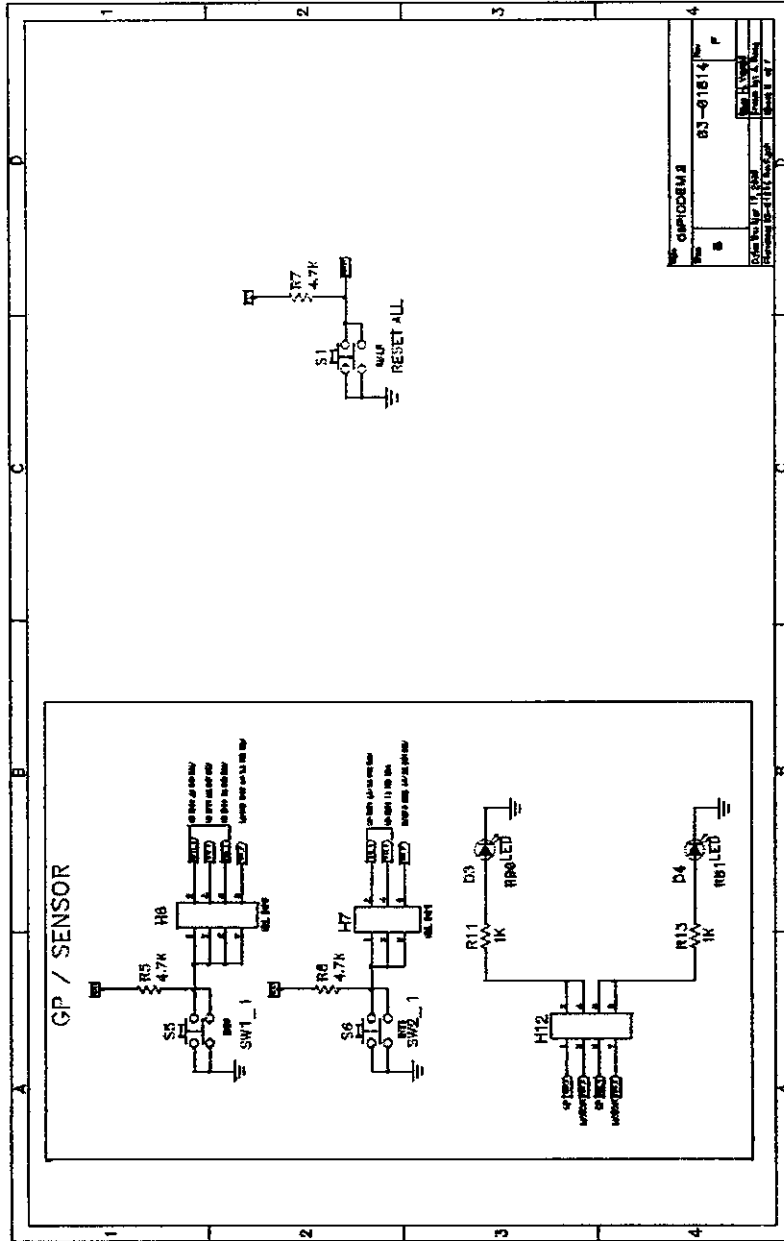
dsPICDEM 2 Development Board User's Guide

FIGURE A-6: dsPICDEM™ 2 DEVELOPMENT BOARD SCHEMATIC (SHEET 5 OF 7)



Hardware Drawings and Schematics

FIGURE A-7: dsPICDEM™ 2 DEVELOPMENT BOARD SCHEMATIC (SHEET 6 OF 7)



APPENDIX B: dsPIC PROGRAMMING

//Pre-Processor Directives:

//Provide pre-processor directives to include Device header files and

//any application specific header files. Path locations to the Device

//header files are set up within the MPLAB Project>>BuildOptions>>Project

//dialog box. The path should point to X:\MPLAB C30\support\h\, where

//"X:" is the folder where the MPLAB C30 tools are installed.

#include <p30fxxxx.h> // "p30fxxxx.h" is a generic header file for dsPIC30F

//devices. This file will in turn select the actual

//device header file, based on the device selected

//by the MPLAB workspace/project. Using the p30fxxxx.h

//file makes it easy to migrate a project from one

//dsPIC device to another.

#include "system.h" // "system.h" is a header file defined for this demo

//application.

//Macros for Configuration Fuse Registers:

//Invoke macros to set up device configuration fuse registers.

//The fuses will select the oscillator source, power-up timers, watch-dog

//timers, BOR characteristics etc. The macros are defined within the device

//header files. The configuration fuse registers reside in Flash memory.

_FOSC(CSW_FSCM_OFF & XT_PLL8); //Run this project using an external crystal

//routed via the PLL in 8x multiplier mode

//For the 7.3728 MHz crystal we will derive a

//throughput of $7.3728e+6 * 8 / 4 = 14.74$ MIPS(Fcy)

//, ~67nanoseconds instruction cycle time(Tcy).

_FWDT(WDT_OFF); //Turn off the Watch-Dog Timer.

_FBORPOR(MCLR_EN & PWRT_OFF); //Enable MCLR reset pin and turn off the

//power-up timers.

_FGS(CODE_PROT_OFF); //Disable Code Protection

//Declaration to Link External Functions & Variables:

//Declare functions being used that are defined outside this file, or

//elsewhere in this MPLAB project.

```

extern SPI_Init(void);
extern UpdateDisplayBuffer(void);
extern WriteUART_to_RS232(void);
extern WriteSPI_to_LCD(void);
extern UART_Init(void);
extern ADC_Init(void);
extern INTx_Init(void);
extern Timer1_Init(void);
extern Timer2_Init(void);

//Functions and Variables with Global Scope:
//Declare functions in this file that have global scope.
int main (void);

//Code execution automatically reaches the main() function after
//two events have occurred;
//1. A Reset event triggered by hardware or software
//2. The execution of the C Start up library functions, present
// in the crt0.o file in the libpic30-coff.a library file
int main (void)
{
    ADPCFG = 0xFFFF;    //After reset all port pins multiplexed
                        //with the A/D converter are configred analog.
                        //We will reconfigure them to be digital
                        //by writing all 1's to the ADPCFG register.
                        //Note: All dsPIC registers are memory mapped.
                        //The address of ADPCFG and other registers
                        //are defined in the device linker script
                        //in your project.

    //Function Delay5ms() available in file, Delay.s
    Delay5ms(100);    //Provide 500ms delay for the LCD to start-up.

```

```

//Function SPI_Init() available in file, SPI_for_LCD.c
SPI_Init();      //Initialize the SPI module to communicate with
                 //the LCD.

//Function UART_Init() available in file, UART.c
UART_Init();     //Initialize the UART module to communicate
                 //with the COM port on the Host PC via an
                 //RS232 cable and the DB9 connector.

//Function ADC_Init() available in file, A_to_D_Converter.c
ADC_Init();      //Initialize the A/D converter to convert
                 //signals from the Temperature Sensor and the
                 //Potentiometer.

//Function INTx_IO_Init() available in file, INTx_IO_pins.c
INTx_IO_Init();  //Initialize the External interrupt pins and
                 //some I/O pins to accept input from the
                 //switches, S5 and S6 and drive the LEDs, D3
                 //and D4.

//Function Timer1_Init() & Timer2_Init() available in file, Timers.c
Timer1_Init();   //Initialize Timer1 to provide "blinking" time
                 //for the LEDs.
Timer2_Init();   //Initialize Timer2 and Timer3 as a 32-bit
                 //Timer to be used for updating data sent to
                 //the LCD(SPI) and COM(UART) interfaces.

while (1)        //Main Loop of Code Executes forever
{
    while (IFS0bits.T3IF == 1) //Wait until 32-bit Timer
    {
        //interrupt flag bit is set.
        IFS0bits.T3IF = 0; //Clear 32-bit timer interrupt
        //flag bit
    }
}

```

```

T2CONbits.TON = 0;    //Stop 32-bit Timer

//Function UpdateDisplayBuffer() available
//in file, DisplayRoutines.c
UpdateDisplayBuffer(); //Write the most recent
                        //temperature and potentiometer
                        //values into display buffer

//Function WriteUART_to_RS232() available in UART.c
WriteUART_to_RS232(); //Update RS232 via UART

//Function WriteSPI_to_LCD() in file, SPI_for_LCD.c
WriteSPI_to_LCD();    //Update the LCD via SPI

T2CONbits.TON = 1;    //Start 32-bit Timer again
    }
}
return 0;             //Code never reaches here!
}
#include <p30fxxxx.h>

//Defines for System Clock Timing -
//For oscillator configuration XT x PLL8 mode,
//Device Throughput in MIPS = Fcy = 7372800*8/4 = ~14.74 MIPS
//Instruction Cycle time = Tcy = 1/(Fcy) = ~68 nanoseconds
#define XTFREQ      7372800    //On-board Crystal frequency
#define PLLMODE     8         //On-chip PLL setting
#define FCY         XTFREQ*PLLMODE/4    //Instruction Cycle Frequency

//Defines that equate Switches on board to specific interrupt pins on device
#define Switch_S5   PORTAbits.RA11 //Switch S5 is connected to RA11
#define Switch_S6   PORTDbits.RD8  //Switch S6 is connected to RD8

```

```

//The Port A structures defined in certain device header files provided
//in the MPLAB C30 v1.30 compiler do not contain definition for port pin RA11.
//Please use the device header file in MPLAB C30 compiler v1.32 or later.
//Pre-Processor Directives:
#include <p30fxxx.h>
#include "system.h"

//Declaration to Link External Functions & Variables:
extern int  current;;

//Functions and Variables with Global Scope:
void Timer1_Init(void);
void Timer2_Init(void);
void __attribute__((__interrupt__)) _T1Interrupt(void);

//Timer1_Init() sets up Timer1 to count up to the maximum 16-bit value, 0xFFFF,
//and interrupt the CPU. ISR processing is enabled for Timer 1.
void Timer1_Init(void)
{
    T1CON = 0x0020;    //Timer1 set up to count on instruction cycle
                    //edge with 1:64 prescaler applied initially.
    PR1 = 0xFFFF;    //Period Register, PR1, set to maximum count
    IFS0bits.T1IF = 0; //Clear the Timer1 Interrupt Flag
    IEC0bits.T1IE = 1; //Enable Timer1 Interrupt Service Routine
    T1CONbits.TON=1;  //Start Timer 1
}

//Timer2_Init() sets up Timer2 and Timer3 to count up to a 32-bit value,
//0x003FFFFFFF with a prescaler of 1:1. 32-bit Timer ISR is disabled.
void Timer2_Init(void)
{

```

```

T2CON = 0x0000;    //32-bit Timer3:Timer2 pair set up
T2CONbits.T32 = 1; //to increment every instruction cycle
PR3 = 0x003F;
PR2 = 0xFFFF;    //Period Register, PR3:PR2, set to 0x003FFFFF
IFS0bits.T3IF = 0; //Clear the Timer3 Interrupt Flag
IEC0bits.T3IE = 0; //Disable Timer3 Interrup Service Routine
T2CONbits.TON=1;  //Start 32-bit timer, setting Timer 2 ON bit
}

//_T1Interrupt() is the Timer1 Interrupt Service Routine
//The routine must have global scope in order to be an ISR.
//The ISR name is the same name provided for the module in the device linker
//script.
//At every Timer1 interrupt event, the state of the two LEDs, D3 and D4,
//is toggled/flipped.
void __attribute__((__interrupt__)) _T1Interrupt(void)
{

//    if(current > 820) // faulty
//    {
//        LATBbits.LATB0 = 1;
//    }
//    else
//        //LATBbits.LATB0 = 0;

//    LATBbits.LATB0 =~ LATBbits.LATB0;    //Toggle LATB0 (LED D3)
//    LATBbits.LATB1 =~ LATBbits.LATB1;    //Toggle LATB1 (LED D4)
    IFS0bits.T1IF = 0;        //Clear Timer1 Interrupt Flag
}

#include "p30fxxx.h"

void __attribute__((__interrupt__)) _OscillatorFail(void);
void __attribute__((__interrupt__)) _AddressError(void);

```



```

void __attribute__((__interrupt__)) _StackError(void);
void __attribute__((__interrupt__)) _MathError(void);
void __attribute__((__interrupt__)) _AltOscillatorFail(void);
void __attribute__((__interrupt__)) _AltAddressError(void);
void __attribute__((__interrupt__)) _AltStackError(void);
void __attribute__((__interrupt__)) _AltMathError(void);

```

```
/*
```

Primary Exception Vector handlers:

These routines are used if `INTCON2bits.ALTIPT = 0`.

All trap service routines in this file simply ensure that device continuously executes code within the trap service routine. Users may modify the basic framework provided here to suit to the needs of their application.

```
*/
```

```

void __attribute__((__interrupt__)) _OscillatorFail(void)
{
    INTCON1bits.OSCFAIL = 0;    //Clear the trap flag
    while (1);
}

```

```

void __attribute__((__interrupt__)) _AddressError(void)
{
    INTCON1bits.ADDRERR = 0;    //Clear the trap flag
    while (1);
}

```

```

void __attribute__((__interrupt__)) _StackError(void)
{
    INTCON1bits.STKERR = 0;    //Clear the trap flag
    while (1);
}

```

```

void __attribute__((__interrupt__)) _MathError(void)

```

```

{
    INTCON1bits.MATHERR = 0;    //Clear the trap flag
    while (1);
}

/*
Alternate Exception Vector handlers:
These routines are used if INTCON2bits.ALTIVT = 1.
All trap service routines in this file simply ensure that device
continuously executes code within the trap service routine. Users
may modify the basic framework provided here to suit to the needs
of their application.
*/

void __attribute__((__interrupt__)) _AltOscillatorFail(void)
{
    INTCON1bits.OSCFAIL = 0;
    while (1);
}

void __attribute__((__interrupt__)) _AltAddressError(void)
{
    INTCON1bits.ADDRERR = 0;
    while (1);
}

void __attribute__((__interrupt__)) _AltStackError(void)
{
    INTCON1bits.STKERR = 0;
    while (1);
}

void __attribute__((__interrupt__)) _AltMathError(void)

```

```

{
    INTCON1bits.MATHERR = 0;
    while (1);
}
//Pre-Processor Directives:
#include <p30fxxx.h>
#include "system.h"

//Functions and Variables with Global Scope:
void INTx_IO_Init(void);
void __attribute__((__interrupt__)) _INT0Interrupt(void);
void __attribute__((__interrupt__)) _INT1Interrupt(void);

//Functions:
//INTx_IO_Init() sets up the INT0 and INT1 pins connected to the
//switches S5 and S6, on the dsPICDEM2 board. It also sets port
//pins, RB0 and RB1, as output pins to drive LEDs, D3 and D4.
void INTx_IO_Init(void)
{
    //Set up Switches S5 and S6 as external interrupt input pins
    //S5 connected to INT0
    //S6 connected to INT1
    INTCON2 = 0x0003;    //Setup INT0-1 pins to interupt on falling edge
    IFS0bits.INT0IF = 0; //Reset INT0 interrupt flag
    IEC0bits.INT0IE = 1; //Enable INT0 Interrupt Service Routine
    IFS1bits.INT1IF = 0; //Reset INT1 interrupt flag
    IEC1bits.INT1IE = 1; //Enable INT1 Interrupt Service Routine

    //Set up port pins RB0 and RB1 to drive the LEDs, D3 and D4
    //RB0 connected to D3
    //RB1 connected to D4
    LATBbits.LATB0 = 0; //Clear Latch bit for RB0 port pin

```

```

    TRISBbits.TRISB0 = 0; //Set the RB0 pin direction to be an output
    LATBbits.LATB1 = 0; //Clear Latch bit for RB1 port pin
    TRISBbits.TRISB1 = 0; //Set the RB1 pin direction to be an output
}

```

```

//_INT0Interrupt() is the INT0 interrupt service routine (ISR).
//The routine must have global scope in order to be an ISR.
//The ISR name is chosen from the device linker script.
//On every INT0 interrupt event, the rate at which the LEDs blink is changed
//by varying the prescaler on Timer1 between the states 1:1, 1:8, 1:64 & 1:256

```

```

void __attribute__((__interrupt__)) _INT0Interrupt(void)
{
    while (Switch_S5 == 1); //Wait in ISR until Switch S5 is released

        //Change Timer1 Prescaler to next value
        //in the sequence 1:1, 1:8, 1:64, 1:256
        if (T1CONbits.TCKPS == 3) T1CONbits.TCKPS = 0;
        else T1CONbits.TCKPS +=1;

    IFS0bits.INT0IF = 0; //Clear the INT0 interrupt flag or else
        //the CPU will keep vectoring back to the ISR
}

```

```

//_INT1Interrupt() is the INT1 interrupt service routine (ISR).
//The routine must have global scope in order to be an ISR.
//The ISR name is chosen from the device linker script.
//On every other INT1 interrupt event, the state of LED D3 is flipped so that
//the two LEDs are NOT ON simultaneously. On every other INT1 interrupt event,
//the state of LED D3 is flipped back so that the two LEDs are ON simultaneously.

```

```

void __attribute__((__interrupt__)) _INT1Interrupt(void)
{
    while (Switch_S6 == 1); //Wait in ISR until Switch S5 is released
//    LATBbits.LATB0 =~ LATBbits.LATB0; //Toggle LED D3
}

```

```

    IFS1bits.INT1IF = 0; //Clear the INT0 interrupt flag or else
                        //the CPU will keep vectoring back to the ISR
}
//Pre-Processor Directives:
#include <p30fxxx.h>
#include "system.h"

//Functions and Variables with Global Scope:
void ADC_Init(void);
void __attribute__((__interrupt__)) _ADCInterrupt(void);

int TempSensor = 0;
int Potentiometer = 0;
int leakage = 0;

//Functions:

//ADC_Init() is used to configure A/D to scan and convert 2 input channels
//per interrupt. The A/D is set up for a total sampling rate of 8KHz
//or 4KHz per channel. The internal counter in the A/D is used to provide
//Acquisition time delay. The input pins being scanned are AN2 and AN3.
//AN2 and AN3 are connected to the Temperature Sensor and the Potentiometer
//on the dsPICDEM2 board.
void ADC_Init(void)
{
    //ADCON1 Register
    //Set up A/D for Automatic Sampling, Auto-Convert
    //All other bits to their default state
    ADCON1bits.SSRC = 7;
    ADCON1bits.ASAM = 1;

    //ADCON2 Register

```

```

//Set up A/D for interrupting after 3 samples get filled in the buffer
//Also, enable Channel scanning
//All other bits to their default state
ADCON2bits.SMPI = 2;
ADCON2bits.CSCNA = 1;

//ADCON3 Register
//Set up Acquisition time (Tacq) for 31 Tad periods
//where, Tad = A/D conversion clock time.
//Set up Tad period to be 20.5 Tcy (Tcy = instruction cycle time)
//Given that each conversion takes 14*Tad (=Tconv) periods,
//Total Sample Time = Acquisition Time + Conversion Time
// = (31 + 14)*Tad = 45*Tad periods
// = 45 * 20.5 * Tcy = 922.5*Tcy periods
//At 7.3728 MIPS, Tcy = 135 ns = Instruction Cycle Time
//So Tsamp = Tacq + Tconv = 45*Tad(in this example)= 125.1 microseconds
//So Fsamp = Sampling Rate ≈ 8 KHz
//All other bits to their default state
ADCON3bits.SAMC = 31;
ADCON3bits.ADCS = 40;

//ADCHS Register
//When Channel scanning is enabled (ADCON2bits.CSCNA=1)
//AND Alternate mux sampling is disabled (ADCON2bits.ALTS=0)
//then ADCHS is a "don't care"
ADCHS = 0x0000;

//ADCSSL Register
//Scan channels AN2, AN3, AN4 as part of scanning sequence
ADCSSL = 0x001C;

//ADPCFG Register
//Set up channels AN2, AN3 as analog inputs and leave rest as digital

```

```

//Recall that we configured all A/D pins as digital when code execution
//entered main() out of reset
ADPCFGbits.PCFG2 = 0;
ADPCFGbits.PCFG3 = 0;
ADPCFGbits.PCFG4 = 0;

//Clear the A/D interrupt flag bit
IFS0bits.ADIF = 0;

//Set the A/D interrupt enable bit
IEC0bits.ADIE = 1;

//Turn on the A/D converter
//This is typically done after configuring other registers
ADCON1bits.ADON = 1;

}

//_ADCInterrupt() is the A/D interrupt service routine (ISR).
//The routine must have global scope in order to be an ISR.
//The ISR name is chosen from the device linker script.
void __attribute__((__interrupt__)) _ADCInterrupt(void)
{
    //Copy the A/D conversion results from ADCBUF $n$  to variables-
    // "Potentiometer" and "TempSensor".
    //Since ADCON2bits.SMPI = 1, only the first two (i.e. SMPI+1) ADCBUF $n$ 
    //locations are used by the module to store conversion results
    Potentiometer = ADCBUF0 + 20;
    TempSensor = ADCBUF1*2;
    leakage = ADCBUF2*2;

    //Clear the A/D Interrupt flag bit or else the CPU will
    //keep vectoring back to the ISR

```

```

    IFS0bits.ADIF = 0;

}

//Pre-Processor Directives:
#include <p30fxxxx.h>
#include "system.h"

#define BAUDRATE    9600           //Desired Baud Rate
#define BRGVAL      ((FCY/BAUDRATE)/16)-1 //Formula for U1BRG register
                                     //from dsPIC30F Family
                                     //Reference Manual

//Declaration to Link External Functions & Variables:
extern unsigned char DisplayData[];

//Functions and Variables with Global Scope:
void UART_Init (void);
void WriteUART_to_RS232(void);
void __attribute__((__interrupt__)) _U1TXInterrupt(void);

unsigned char *UARTCharPtr;

//Functions

//UART_Init() sets up the UART for a 8-bit data, No Parity, 1 Stop bit
//at 9600 baud with transmitter interrupts enabled
void UART_Init (void)
{
    U1MODE = 0x0000;    //Clear UART1 registers
    U1STA = 0x0000;

```



```

//Since the SPI1 (SCK1, SDO1, SDI1) pins are multiplexed with the UART
//(UITX and U1RX ) pins on this device this demonstration program
//will use alternate UART1 pins (U1ATX and U1ARX)
//The SPI1 module is used to communicate to the LCD Controller while
//UART1 module is used to communicate with the RS232 port on the PC
    U1MODEbits.ALTIO = 1; //Enable U1ATX and U1ARX instead of
        //UITX and U1RX pins

    U1MODEbits.UARTEN = 1; //Enable UART1 module
    U1BRG = BRGVAL;      //Load UART1 Baud Rate Generator

    IFS0bits.U1RXIF = 0; //Clear UART1 Receiver Interrupt Flag
    IFS0bits.U1TXIF = 0; //Clear UART1 Transmitter Interrupt Flag
    IEC0bits.U1RXIE = 0; //Disable UART1 Receiver ISR
    IEC0bits.U1TXIE = 1; //Enable UART1 Transmitter ISR
    U1STAbits.UTXISEL = 1; //Setup UART1 transmitter to interrupt
        //when a character is transferred to the
        //Transmit Shift register and as result,
        //the transmit buffer becomes empty.

    U1STAbits.UTXEN = 1; //Enable UART1 transmitter
    UARTCharPtr = &DisplayData[0]; //Initialize UARTCharPtr to point
        //to the first character in the Display buffer
}

//WriteUART_to_RS232() triggers interrupt-driven UART communication by writing
//the first character in the Display buffer to the UART Transmit register
void WriteUART_to_RS232(void)
{
    if ((UARTCharPtr > &DisplayData[0]) &&
        (UARTCharPtr < &DisplayData[38])) return;
    else
    {

```

```

    UARTCharPtr = &DisplayData[0]; //Re-Initialize UART display
        //buffer pointer to point to
        //the first character
    U1TXREG = *UARTCharPtr++; //Load the UART transmit
        //register with first character
    }
}

//_UITXInterrupt() is the UART1 Interrupt Service Routine.
//The routine must have global scope in order to be an ISR.
//The ISR name is the same name provided for the module in the device linker
//script.
//The UART1 ISR loads the UART1 4-deep FIFO buffers with the next
//4 characters in the Display buffer unless it encounters a null character.
void __attribute__((__interrupt__)) _UITXInterrupt(void)
{
    int i = 0;
    while ((*UARTCharPtr != '\0') && (i < 4))
    {
        U1TXREG = *UARTCharPtr++;
        i++;
    }
    IFS0bits.U1TXIF = 0; //Clear the UART1 transmitter interrupt flag
}

//Pre-Processor Directives:
#include <p30fxxx.h>
#include "system.h"

#define LCDLINE1CMD 0x80 //Command to set LCD display to Line 1
#define LCDLINE2CMD 0xC0; //Command to set LCD display to Line 2

```

```

//Declaration to Link External Functions & Variables:
extern Delay5ms (int);
extern unsigned char DisplayData[];

//Functions and Variables with Global Scope:
void SPI_Init(void);
void WriteSPI_to_LCD(void);

unsigned char *LCDCharPtr;

//SPI_Init Function:
//SPI1 module set up to communicate with the LCD controller on-board.
//Note that when SPI1 is enabled on this device, the UART1 pins will not be
//available due to peripheral multiplexing. So this project utilizes
//alternate UART1 pins, U1ATX and U1ARX
void SPI_Init(void)
{
    SPI1STAT = 0x0000;
    SPI1CON = 0x0020 ; //Set the SPI1 module to 8-bit Master mode
    IFS0bits.SPI1IF = 0; //Clear the SPI1 Interrupt Flag
    IEC0bits.SPI1IE = 0; //SPI1 ISR processing is not enabled.
        //SPI1 will be used in polling-mode
    SPI1STATbits.SPIEN = 1; //Turn on the SPI1 module
}

//WriteSPI_to_LCD() Function:
//WriteSPI_to_LCD() writes 32 characters to the 2x16 LCD
//using the SPI1 interface in a polling fashion.
//After each byte is written, the Interrupt Flag bit is polled and the
//next character is written after the interrupt flag bit is set.
void WriteSPI_to_LCD(void)
{
    int temp, i;

```

```

i=0;
temp = SPI1BUF;
SPI1STATbits.SPIROV = 0;
IFS0bits.SPI1IF = 0;
Delay5us(50);
SPI1BUF = LCDLINE1CMD;    //First write the command to set cursor
                          //to Line1 on LCD
LCDCharPtr = &DisplayData[0]; //Set up LCD character pointer to point
                          //to the Display buffer
while(i < 16)
{
    while (IFS0bits.SPI1IF=0); //Now write 16 characters
    temp = SPI1BUF;           //to Line 1 of the LCD
    IFS0bits.SPI1IF = 0;
    SPI1STATbits.SPIROV = 0;
    Delay5ms(1);
    SPI1BUF = *LCDCharPtr++;
    i++;
}
while (IFS0bits.SPI1IF==0);
temp = SPI1BUF;
IFS0bits.SPI1IF = 0;
SPI1STATbits.SPIROV = 0;
temp = *LCDCharPtr++; //Some characters in the Display buffer
temp = *LCDCharPtr++; //are skipped while writing to LCD. CR
Delay5us(50);         //and LF are for writing to RS2322 UART
SPI1BUF = LCDLINE2CMD; //Next, write the command to set cursor
                          //to Line2 on LCD

i = 0;
while(i < 16)
{
    while (IFS0bits.SPI1IF==0); //Now write 16 characters
    temp = SPI1BUF;           //to Line 2 of the LCD

```

```

        IFS0bits.SPI1IF = 0;
        SPI1STATbits.SPIROV = 0;
        Delay5us(50);
        SPI1BUF = *LCDCharPtr++;
        i++;
    }
}
//Pre-Processor Directives:
#include <p30fxxxx.h>
#include "system.h"

//Declaration to Link External Functions & Variables:
extern int TempSensor;
extern int Potentiometer;
extern int leakage;

int current,temperature;
//Functions and Variables with Global Scope:
void UpdateDisplayBuffer(void);
void ADCResult2Decimal(unsigned int ADRES);

unsigned char DisplayData[] = "I=00.0A ..... \r\nT=+00C Vol=0.00V\r\n\r\n0";
unsigned char adones=0;
unsigned char adtenths=0;
unsigned char adhundredths=0;

//UpdateDisplayBuffer() Function:
//This function writes new A/D readings to the Display Buffer.
//The display buffer stores a string to be displayed on both the RS232 and
//the LCD.
void UpdateDisplayBuffer(void)
{
    int temp = 0;           //Convert TC1047A reading to Deg

```

```

                //Celsius
/*  temp = TempSensor - 0x199;    //Determine if the temperature is
if (temp > 0)                    //negative, in order to write
{
    //either a + or - sign to display areas
    DisplayData[20] = '+';
}
else
{
    temp = 0x199 - TempSensor;
    DisplayData[20] = '-';
}

*/

                // leakage current
ADCResult2Decimal(leakage);    //Convert the hex value to decimal
DisplayData[13] = adones;    //Update the display buffer characters
DisplayData[14] = adtenths;    //Update the display buffer characters
DisplayData[15] = adhundredths;

                // TempSensor is connected with Current measuring
                current = TempSensor;
ADCResult2Decimal(current*2);    //Convert the hex value to decimal
DisplayData[2] = adones;    //Update the display buffer characters
DisplayData[3] = adtenths;    //Update the display buffer characters
DisplayData[5] = adhundredths;

temperature = TempSensor - 100;
if(temperature < 350)
temperature = 350;

if(temperature > 850)
temperature = 850;

```

```

ADCResult2Decimal(temperature);    //Convert the hex value to decimal
//  DisplayData[21] = adones;    //Update the display buffer characters
DisplayData[21] = adtenths;    //Update the display buffer characters
DisplayData[22] = adhundredths;

if(TempSensor > 820) // faulty
{
    DisplayData[8] = 'F';
    DisplayData[9] = 'a';
    DisplayData[10] = 'u';
    DisplayData[11] = 'l';
    DisplayData[12] = 't';

    LATBbits.LATB0 = 1;
}
else
{
    DisplayData[8] = 'G';
    DisplayData[9] = 'o';
    DisplayData[10] = 'o';
    DisplayData[11] = 'd';
    DisplayData[12] = ' ';

    LATBbits.LATB0 = 0;
}

ADCResult2Decimal(Potentiometer); //Convert the A/D hex value to
DisplayData[30] = adones;    //decimal and update the display buffer
DisplayData[31] = adtenths;
DisplayData[32] = adhundredths;

```

```
}
```

```
//ADCResult2Decimal() Function:
```

```
//This function converts a 12-bit A/D result in hexadecimal to decimal
```

```
//assuming a 5 volt reference voltage.
```

```
void ADCResult2Decimal(unsigned int ADRES)
```

```
{
```

```
    adones = 0;        //reset values
```

```
    adtenths = 0;
```

```
    adhundredths = 0;
```

```
    while (ADRES > 0x8)
```

```
    {
```

```
        if(ADRES > 0x333)        //test for 1 volt or greater
```

```
        {
```

```
            adones++;           //increment 1 volt counter
```

```
            ADRES -= 0x334;     //subtract 1 volt
```

```
        }
```

```
        else if(ADRES > 0x51 && ADRES <= 0x333) //test for 0.1 volt or greater
```

```
        {
```

```
            if (adtenths < 9)
```

```
            {
```

```
                adtenths++;     //increment tenths
```

```
            }
```

```
            else
```

```
            {
```

```
                adones++;       //tenths has rolled over
```

```
                adtenths = 0;    //so increment ones and reset tenths
```

```
            }
```

```
            ADRES -= 0x52;
```

```
        }
```

```
        else if(ADRES > 0x8 && ADRES <= 0x51) //test for 0.01 volt or greater
```

```
        {
```

```
            if (adhundredths < 9)
```



```

    {
        adhundredths++;    //increment hundreths
    }
    else
    {

        adhundredths = 0;    //reset hundredths
        if (adtenths < 9)
        {
            adtenths++;        //and increment tenths
        }
        else
        {
            adones++;          //unless tenths has rolled over
            adtenths = 0;      //so increment ones and reset tenths
        }
    }
    ADRES -= 0x9;
}

```

```

}
adones += 0x30;
adtenths += 0x30;
adhundredths += 0x30;

```

```

}

```