116

A High Impedance Fault Detector

By
Dehua Zheng

A THESIS

Submitted to the Faculty of Graduate Studies
in partial fulfillment of the requirements for the degree of

Master of Science

Department of Electrical and Computer Engineering
The University of Manitoba
Winnipeg, Manitoba, Canada

ISBN 0-612-13596-9

Name ____________________

Dissertation Abstracts International is arranged by broad, general subject categories. Please select the one subject which most nearly describes the content of your dissertation. Enter the corresponding four-digit code in the spaces provided.

Electronics and Electrical — SUBJECT TERM

0 5 4 4 — SUBJECT CODE

U·M·I

Subject Categories

THE HUMANITIES AND SOCIAL SCIENCES

COMMUNICATIONS AND THE ARTS

Architecture	0729
Art History	0377
Cinema	0900
Dance	0378
Fine Arts	0357
Information Science	0723
Journalism	0391
Library Science	0399
Mass Communications	0708
Music	0413
Speech Communication	0459
Theater	0465

EDUCATION

General	0515
Administration	0514
Adult and Continuing	0516
Agricultural	0517
Art	0273
Bilingual and Multicultural	0282
Business	0688
Community College	0275
Curriculum and Instruction	0727
Early Childhood	0518
Elementary	0524
Finance	0277
Guidance and Counseling	0519
Health	0680
Higher	0745
History of	0520
Home Economics	0278
Industrial	0521
Language and Literature	0279
Mathematics	0280
Music	0522
Philosophy of	0998
Physical	0523
Psychology	0525
Reading	0535
Religious	0527
Sciences	0714
Secondary	0533
Social Sciences	0534
Sociology of	0340
Special	0529
Teacher Training	0530
Technology	0710
Tests and Measurements	0288
Vocational	0747

LANGUAGE, LITERATURE AND LINGUISTICS

Language	
General	0679
Ancient	0289
Linguistics	0290
Modern	0291
Literature	
General	0401
Classical	0294
Comparative	0295
Medieval	0297
Modern	0298
African	0316
American	0591
Asian	0305
Canadian (English)	0352
Canadian (French)	0355
English	0593
Germanic	0311
Latin American	0312
Middle Eastern	0315
Romance	0313
Slavic and East European	0314

PHILOSOPHY, RELIGION AND THEOLOGY

Philosophy	0422
Religion	
General	0318
Biblical Studies	0321
Clergy	0319
History of	0320
Philosophy of	0322
Theology	0469

SOCIAL SCIENCES

American Studies	0323
Anthropology	
Archaeology	0324
Cultural	0326
Physical	0327
Business Administration	
General	0310
Accounting	0272
Banking	0770
Management	0454
Marketing	0338
Canadian Studies	0385
Economics	
General	0501
Agricultural	0503
Commerce-Business	0505
Finance	0508
History	0509
Labor	0510
Theory	0511
Folklore	0358
Geography	0366
Gerontology	0351
History	
General	0578
Ancient	0579
Medieval	0581
Modern	0582
Black	0328
African	0331
Asia, Australia and Oceania	0332
Canadian	0334
European	0335
Latin American	0336
Middle Eastern	0333
United States	0337
History of Science	0585
Law	0398
Political Science	
General	0615
International Law and Relations	0616
Public Administration	0617
Recreation	0814
Social Work	0452
Sociology	
General	0626
Criminology and Penology	0627
Demography	0938
Ethnic and Racial Studies	0631
Individual and Family Studies	0628
Industrial and Labor Relations	0629
Public and Social Welfare	0630
Social Structure and Development	0700
Theory and Methods	0344
Transportation	0709
Urban and Regional Planning	0999
Women's Studies	0453

THE SCIENCES AND ENGINEERING

BIOLOGICAL SCIENCES

Agriculture	
General	0473
Agronomy	0285
Animal Culture and Nutrition	0475
Animal Pathology	0476
Food Science and Technology	0359
Forestry and Wildlife	0478
Plant Culture	0479
Plant Pathology	0480
Plant Physiology	0817
Range Management	0777
Wood Technology	0746
Biology	
General	0306
Anatomy	0287
Biostatistics	0308
Botany	0309
Cell	0379
Ecology	0329
Entomology	0353
Genetics	0369
Limnology	0793
Microbiology	0410
Molecular	0307
Neuroscience	0317
Oceanography	0416
Physiology	0433
Radiation	0821
Veterinary Science	0778
Zoology	0472
Biophysics	
General	0786
Medical	0760

EARTH SCIENCES

Biogeochemistry	0425
Geochemistry	0996
Geodesy	0370
Geology	0372
Geophysics	0373
Hydrology	0388
Mineralogy	0411
Paleobotany	0345
Paleoecology	0426
Paleontology	0418
Paleozoology	0985
Palynology	0427
Physical Geography	0368
Physical Oceanography	0415

HEALTH AND ENVIRONMENTAL SCIENCES

Environmental Sciences	0768
Health Sciences	
General	0566
Audiology	0300
Chemotherapy	0992
Dentistry	0567
Education	0350
Hospital Management	0769
Human Development	0758
Immunology	0982
Medicine and Surgery	0564
Mental Health	0347
Nursing	0569
Nutrition	0570
Obstetrics and Gynecology	0380
Occupational Health and Therapy	0354
Ophthalmology	0381
Pathology	0571
Pharmacology	0419
Pharmacy	0572
Physical Therapy	0382
Public Health	0573
Radiology	0574
Recreation	0575
Speech Pathology	0460
Toxicology	0383
Home Economics	0386

PHYSICAL SCIENCES

Pure Sciences

Chemistry	
General	0485
Agricultural	0749
Analytical	0486
Biochemistry	0487
Inorganic	0488
Nuclear	0738
Organic	0490
Pharmaceutical	0491
Physical	0494
Polymer	0495
Radiation	0754
Mathematics	0405
Physics	
General	0605
Acoustics	0986
Astronomy and Astrophysics	0606
Atmospheric Science	0608
Atomic	0748
Electronics and Electricity	0607
Elementary Particles and High Energy	0798
Fluid and Plasma	0759
Molecular	0609
Nuclear	0610
Optics	0752
Radiation	0756
Solid State	0611
Statistics	0463

Applied Sciences

Applied Mechanics	0346
Computer Science	0984
Engineering	
General	0537
Aerospace	0538
Agricultural	0539
Automotive	0540
Biomedical	0541
Chemical	0542
Civil	0543
Electronics and Electrical	0544
Heat and Thermodynamics	0348
Hydraulic	0545
Industrial	0546
Marine	0547
Materials Science	0794
Mechanical	0548
Metallurgy	0743
Mining	0551
Nuclear	0552
Packaging	0549
Petroleum	0765
Sanitary and Municipal	0554
System Science	0790
Geotechnology	0428
Operations Research	0796
Plastics Technology	0795
Textile Technology	0994

PSYCHOLOGY

General	0621
Behavioral	0384
Clinical	0622
Developmental	0620
Experimental	0623
Industrial	0624
Personality	0625
Physiological	0989
Psychobiology	0349
Psychometrics	0632
Social	0451

A HIGH IMPEDANCE FAULT DETECTOR

BY

DEHUA ZHENG

A Thesis submitted to the Faculty of Graduate Studies of the University of Manitoba in partial fulfillment of the requirements of the degree of

MASTER OF SCIENCE

ABSTRACT

In this thesis, a **High Impedance Fault (HIF) Detector** is built for detecting high impedance faults in a power distribution system. The detector consists of an algorithm, a –data acquisition board, and a Windows program incorporating the algorithm.

A HIF simulation circuit was set up in the laboratory at the University of Manitoba. The simulated fault current proved to be a credible fault–current source for simulating HIF in a power–distribution system. Four different kinds of waveforms were collected for faults involving wet soils, dry soils, grassy wet soils and grassy dry soils, in a power–distribution system.

The algorithm proposed in this paper for detecting HIF includes three parts: **Flicker, Asymmetry, and Quarter–Cycle Asymmetry**, all of which are used as indications of HIF. The main part of this algorithm uses the half–cycle current rms value to detect Flicker and Asymmetry when there is a high impedance fault. Since at times there is no obvious Flicker and Asymmetry when a fault happens on wet soil, this algorithm also uses the Quarter–Cycle Asymmetry feature to detect the fault. These algorithms show good performance in identifying the characteristics of HIF waveforms. They have good ***dependability*** (ability to trip when it should). In addition, for high impedance fault–like loads (for example a computer current load), the algorithm also gives quite satisfactory results in terms of the ***security*** (ability to not trip when it should not).

With the Windows program, the detector can be operated in either the **Intermittent** or the **Continuous** mode. It takes 20 seconds for the detector to carry out the whole detection procedure, so the result meets the requirement of real–time detection. In addition, when the detector is running in the continuous mode, it sends out a 5V or 0V output signal every 20 seconds corresponding to the fault or no–fault situation. This signal can be used for alarm or trip purposes.

ACKNOWLEDGEMENTS

First of all I would like to express my sincere gratitude to Professor G. W. Swift, my supervisor, for suggesting this topic, for his excellent advice, continuous encouragement and helpful discussion throughout the course of this research , especially for providing the opportunity of pursing my degree in Canada.

I would like to thank Mr. Erwin Dirks, Mr. Roy Yang, Mr. Ming Zhang, Mr. Yiping Zhuang, Mr. Xiao Jiang and all my colleagues for all the discussion and support they provided in numerous ways in carrying out this research.

Special thanks to my wife Ya–Hao, Son Alan and my parents, my brother and sisters for their patience, constant encouragements and understanding.

Finally, I would like to acknowledge the financial assistance of Manitoba Hydro and Mr. D.J. Fedirchuk who made this research possible.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1 INTRODUCTION

1.1 High Impedance Fault

1.1.1 Fault Problem Statement

When a fallen distribution conductor in a power system is in contact with a high–impedance surface of some material, such as asphalt, soil, sand or trees, the current of this kind of fault is quite often below the trip level of a fault–clearing device and there is often an arcing phenomenon at the point where the conductor touches the material. This kind of fault is considered as a high impedance fault or an arcing high impedance fault [1] [2].

A high impedance fault (HIF) frequently happens when a distribution line or a conductor in a power system faults to the ground, or is contacted by a foreign object, or a pole or pole hardware is broken. Since the power delivery exists, as long as there is a HIF in a power system, the following results could be occurring: energy waste, fire hazard, power supply interruption and property damage. Especially, when a person contacts with an energized conductor, as shown in Fig 1., it could cause injury or death. Therefore, generally speaking, a high impedance fault presents a source of threat to a utility's customers and personnel rather than to the integrity of the power system [3] [4].

1.1.2 Electrical Effects on the Human Body Due to Faults

It is known that the heart of the human body is very vulnerable to electrical current. When a current flows through the human body, it can result in muscle contraction, heart stoppage and skin burns. The effects depend on the amount of current, the length of time, the resistance value of the skin and the current path. Since the human body's skin provides a resistance from 1.5 k–ohm to 5.0 k–ohm, when a person touches an energized power line, the extent of injury could be different in terms of the different skin resistance values [1].

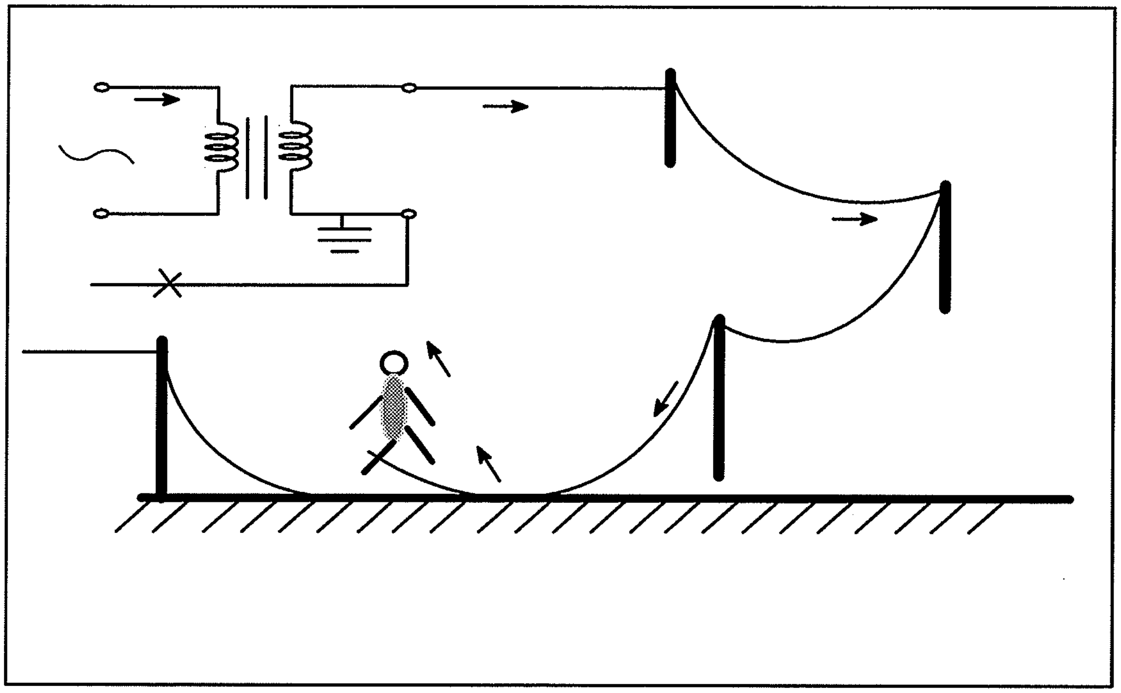


Fig. 1 A Person Contacts the Downed Power Line

1.1.3 V–I Characteristics of HIF

When HIF happens , it can persist for a long period of time because of the lack of a detecting strategy. HIF has a random arc behavior. Also, the V–I characteristic of an arc is entirely different from that of a metal conductor. Generally speaking, the V–I characteristic across a conductor is a linear relationship, that is, the voltage across the conductor is proportional to the current flowing through the conductor. The arcing feature associated with downed power lines deviates from that of conductor–to–conductor faults, or across the circuit breaker poles. Arcing in high impedance faults appears as a largely resistive and nonlinear V– I characteristics as shown in Fig. 2.

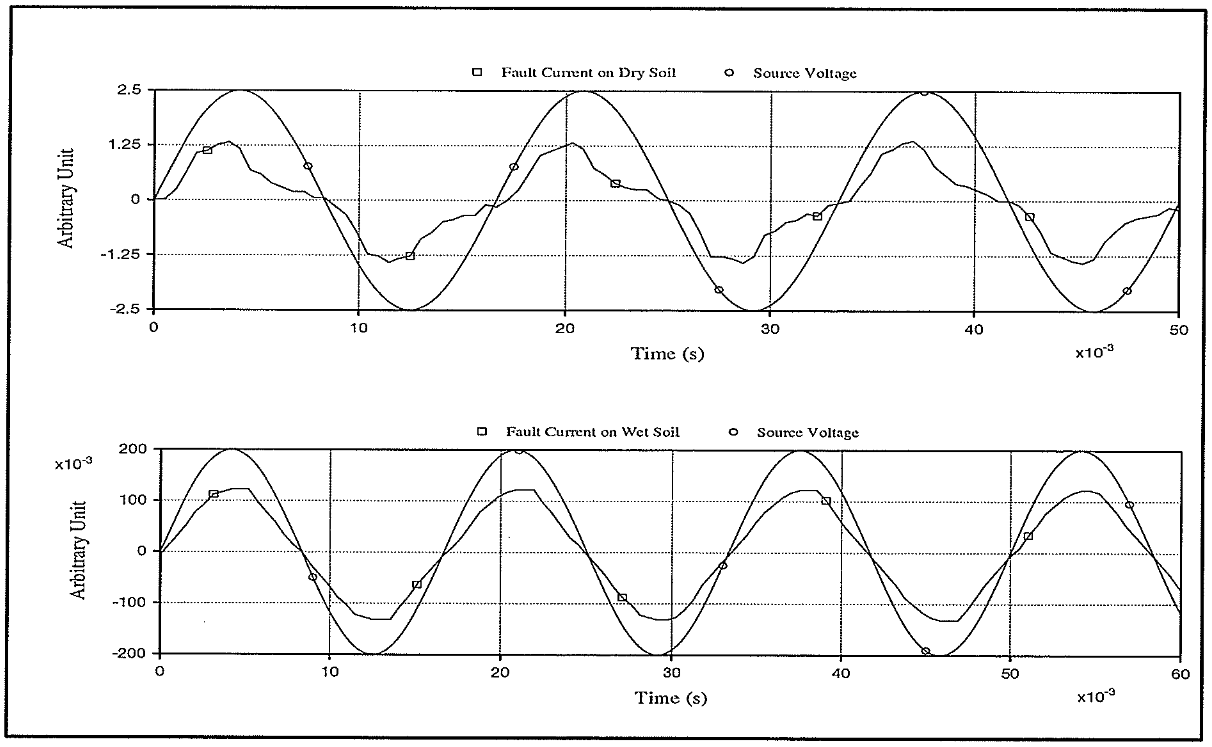


Fig. 2 V – I Characteristics of HIF on Dry and Wet Soil

1.1.4 Difficulties with Detecting HIF

The protection of a primary distribution system is mainly accomplished by conventional overcurrent relays. Unfortunately, the fault current of HIF generally does not show a large enough value to be detected by an overcurrent relay. Therefore, difficulties appear when using this technique to detect HIF. While overcurrent relays interrupt fault currents, they should not trip normal emergency loads like transient overcurrents caused by inrush events or load surges. For this reason, the trip level must be set at a relatively high current value to avoid tripping during the normal operations. From a practical point of view, the trip level of an overcurrent relay is usually set to a value at 125–200% of maximum load current [1] .

Although the conventional overcurrent relays do detect a great number of faults in the distribution lines, they still do not detect many faults with low fault currents in which the magnitude of fault currents is in the range of (0 – 120) % of normal load current; this is illustrated as shown in Fig. 3.

1.2 Previous Research Work in Detecting High Impedance Faults

With the great efforts done by electrical engineers for detecting HIF, many detection schemes have been proposed in the past few years.

1.2.1 Carr

Carr did a theoretical analysis on a grounded – Y– connected systems. His scheme

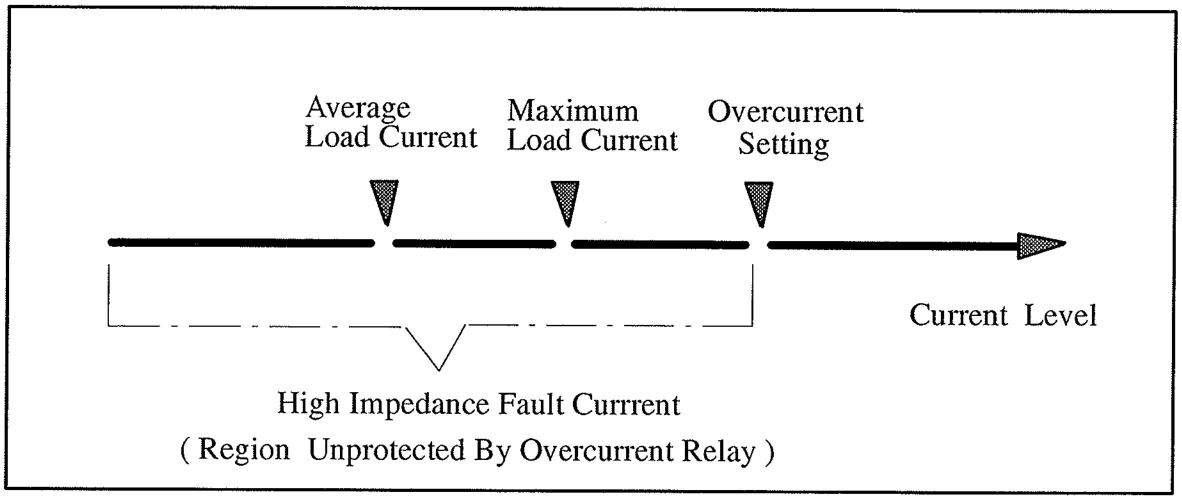

Average Load Current
Maximum Load Current
Overcurrent Setting
Current Level
High Impedance Fault Currrent
(Region Unprotected By Overcurrent Relay)

Fig. 3 Relation of HIF Current to Overcurrent Relay Setting

[5] was based on (1) combining neutral and ground current measurements, a proportional relaying method in which three–phase unbalanced currents are used for the detection of HIF, (2) detecting power frequency harmonics, and (3) sensing sequence voltages at the load side of the line. The negative sequence voltage is preferable to the zero sequence voltage, since the former is less dependent on the zero sequence impedance which is closely related to the grounding systems.

1.2.2 Aucoin and Russell

Aucoin and Russell have been playing a leading role in the detection of arcing high impedance faults by using a harmonic components method. They proposed a HIF detection method [6] [7] [8] based on the increase in a high frequency (2–10 kHz) signal caused by an arcing phenomenon. This high frequency component is due to continuous strikes in the air gaps between the conductor and the surface of the material. Russell and others also proposed a technique [9] in which burst noise signals near 60 Hz and low frequency components caused by distorted fault currents were used for the detection of high impedance faults. Russell also pointed out that even–order components could be useful for the detection of arcing faults.

1.2.3 Jeerings and Linders

According to the results of their research, Jeerings and Linders[10] [11] [12] expressed the characteristics of HIF as being highly resistive and nonlinear ; as a result of this, the currents of low–order harmonics of HIF will appear with current peaks coincident with the voltage peaks. This characteristic is independent of any other single system phenomenon. The third harmonic components of the fault current and voltage were separated from the fundamental and other harmonics. The phasor ratio was defined by the voltage change to the current change and referred to as the sink impedance. The HIF could be detected by the sink impedance . The normal power system current may contain third harmonic voltage and current as well, but the ratio of their changes is different from the one for HIF.

1.2.4 Sultan and Swift

Sultan and Swift proposed an algorithm [13] in which they used Flicker and Asymmetry to detect the arcing high impedance fault, since Flicker and Asymmetry are obvious features of an arcing high impedance fault. The algorithm calculates one cycle of normal load current rms value as reference $I_{rms-ref}$, and then compares the rms values of the following cycles. If any new value is sufficiently different from the reference value, Flicker and Asymmetry are calculated. If two continuous half cycle rms values of the positive side or negative side are different and the differences are changing sign, " Flicker " is defined. In each cycle, if the positive side rms value is different from the negative side rms value and the differences are changing sign comparing to next half cycle, "Asymmetry " is defined. The algorithm showed quite satisfactory performance in identifying HIF as well as discrimination against fault–like loads such as computer loads and some loads due to abnormal events in power systems. One of the salient ideas [14] that they pointed out is that the design of a reliable high impedance fault detector should include two aspects ; not only ***dependability*** (ability to trip when it should) but also ***security*** (ability to not trip when it should not), since most HIF detection schemes proposed so far have been tested for dependability, but few have been tested for security.

They also proposed a HIF detection algorithm [15] which used an artificial neural network, or simply a neural net (NN) [7] . In the algorithm, a feed–forward three layer network was trained by high impedance fault loads, fault–like loads and normal load current patterns, using the back propagation training method. The algorithm was tested by tracing normal load current disturbed by HIF currents on wet and dry soils, an arc welder, computers and fluorescent lights. The investigation outcomes of the algorithm on these loads was able to reach general solutions to the problems.

CHAPTER 2 ALGORITHM DESIGN

2.1 Using the R.M.S. Value of the Fault Current In the Algorithm

Since the waveform identification method is going to be used to detect high impedance faults, the best way is to use half cycle fault current rms values to express Flicker and Asymmetry concepts. In data acquisition (discussed in Appendix B in detail), the sampling rate is set at 32 sample–per–cycle, which is the rate applicable to many modern practical microprocessor based relays. For the discrete sampling values, the half cycle fault current rms value should be;

$$I_{rms} = \sqrt{\frac{1}{16} \times \sum_{j=1}^{16} i_j^2} \qquad \text{Eq. 1}$$

where i_j is the jth sample of the current waveform.

2.2 Algorithm

In accordance with the characteristics of HIF waveforms obtained from the experiments , it was found that there are three obvious features of the fault waveforms: Flicker, Asymmetry and Quarter Cycle Asymmetry. The following is a detailed explanation of the algorithm.

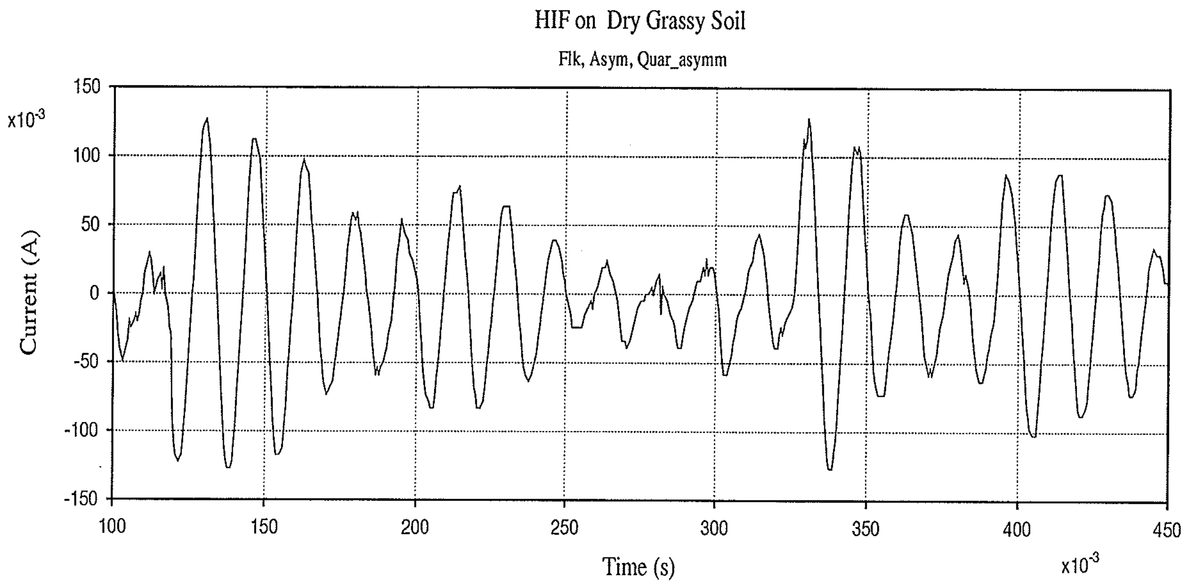


Fig. 4 Fault Current Showing Flicker and Asymmetry

2.2.1 Flicker

We define **Flicker** as the degree to which the signal amplitude various erratically from one cycle to the next.

Mathematically, the positive and negative I_{rms} values are compared in two continuous cycles to determine **Flicker**. Here I_{rms} is the half cycle fault current rms value as shown in Eq. 1. In terms of Fig. 4 and Eq. 2, it can be seen that $I_{rms1[j]}$ means jth cycle positive side half cycle current rms value and $I_{rms2[j]}$ means the same cycle but negative value. On the positive waveform side:

$$ABS\{(I_{rms1[2]}) - (I_{rms1[1]})\} >= C1*(I_{rms1[1]})\ AND$$

$$ABS\{(I_{rms1[3]}) - (I_{rms1[2]})\} >= C1*(I_{rms1[2]})\ AND$$

$$(I_{rms1[3]} - I_{rms1[2]})*(I_{rms1[2]} - I_{rms1[1]}) < 0 \qquad \text{Eq. 2}$$

where C1 is a constant which depends on experimental experience. Generally, C1 = (0.05 – 0.1).

For the negative waveform side, the same feature is required. If Eq. 2 holds for both the sides of the waveform, then it is said that there is Flicker. The third part of the conditions in Eq.2 means that for each side of the waveform the differences in I_{rms} are changing sign.

2.2.2 *Asymmetry*

The positive and negative I_{rms} values are compared in each cycle. It is said that there is **Asymmetry**. if

$$ABS\{ (I_{rms1[1]}) - (I_{rms2[1]}) \} >= C2 * (I_{rms1[1]}) \; AND$$

$$ABS\{ (I_{rms2[1]}) - (I_{rms1[2]}) \} >= C2 * (I_{rms1[2]}) \; AND$$

$$(I_{rms2[1]} - I_{rms1[1]}) * (I_{rms1[2]} - I_{rms2[1]}) < 0 \qquad \text{Eq. 3}$$

where C2 is also a constant which depends on experimental experience. Here, the principal for choosing C2 is same as in **Flicker**.

2.2.3 *Quarter Cycle Asymmetry*

When HIF happens on relatively wet soil, the waveform as shown in Fig. 5, is almost sinusoidal . This kind of fault is very hard to identify if only Flicker and Asymmetry are used as indications of the fault. However, if the waveform is studied carefully, it can be realized that there is a quarter cycle asymmetry feature on the waveform. In fact, the quarter

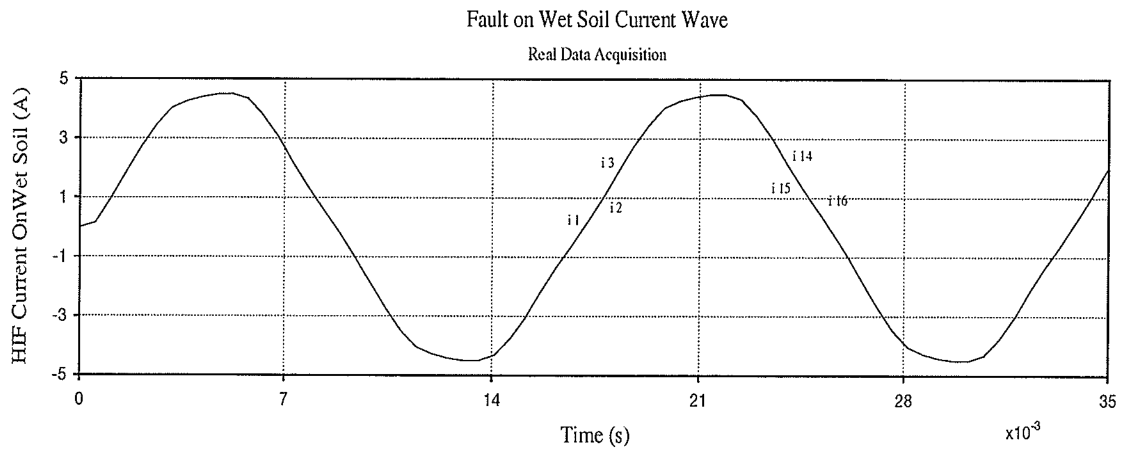


Fig. 5 Quarter Cycle Asymmetry For Decting Fault On Wet Soil

cycle asymmetry characteristic appears on all HIF waveforms obtained. Therefore, this thesis proposes **Quarter Cycle Asymmetry** as one of the important parts of the algorithm. As seen from Fig.5, in order to use the Quarter Cycle Asymmetry algorithm it needs to compare two quarter cycle sampled values for each half cycle, so the following group of variables are defined;

K1 = ABS (i1 – i16) K2 = ABS (i2 – i15)

K3 = ABS (i3 – i14) K4 = ABS (i4 – i13)

– –

K7 = ABS (i7 – i10) K8 = ABS (i8 – i9) Eq. 4

A second group of variables can be defined;

D1 = ABS (K1 – K2) D2 = ABS (K3 – K4)

D3 = ABS (K5 – K6) D4 = ABS (K7 – K8) Eq. 5

The two groups of variables will be used to calculate Quarter Cycle Asymmetry. It is extremely important to find the zero crossing point at the beginning of each half cycle; otherwise, the algorithm will become useless. During data acquisition, there are 16 samples at each half cycle, but it is only a coincidence if the sampled data i1 and i16 are equal to zero. For most cases, they are not zero, so interpolation has to be used to find the zero crossing point in order to realize the algorithm properly.

2.2.3 Interpolation for Finding Zero Point

When data are read from the data file, if two adjacent data values are of opposite sign, the actual zero crossing time can be calculated as follows:

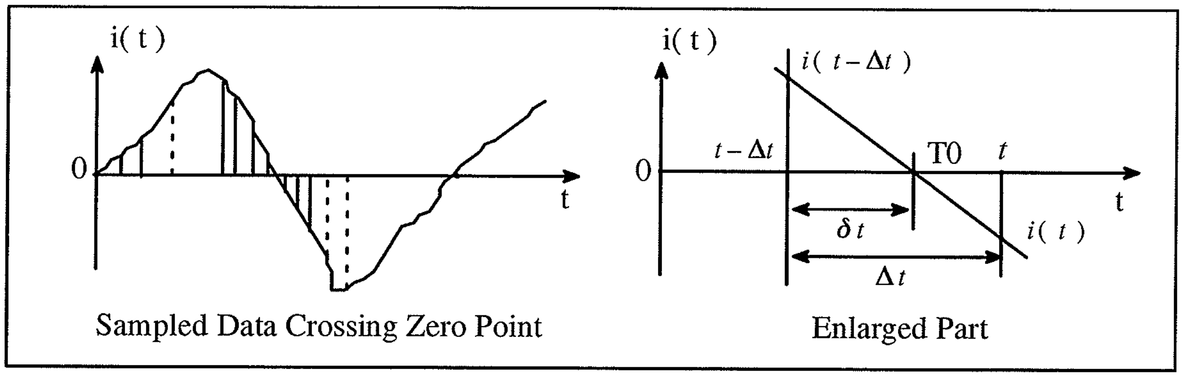


Fig. 6 Interpolation to Find the Zero Point

From Fig. 6, it is easy to derive that

$$\delta_t = \left[\frac{0 - i(t - \Delta t)}{i(t) - i(t - \Delta t)}\right] \times \Delta t \qquad \text{Eq. 6}$$

From Eq. 6, T0, the new time start point can be found

$$T0 = (t - \Delta t) + \delta_t \qquad \text{Eq. 7}$$

at t = T0, i (T0) = 0. Then from T0, the same time interval will still be used to get 15 more sample values of the current in order to get the constants K1,K2, ... , D1,D2,D3,D4. The same strategy is continuously used to deal with the all zero crossing points no matter whether the curve is starting at the rising edge or the falling edge. Now it is further defined:

$$S = \sum_{j=1}^{j=4} D_j \qquad \text{Eq.8}$$

where Dj is defined in Eq. 5. For a sinusoidal waveform, S is much smaller than for the waveform of a fault on wet soil.

A sinusoidal waveform is symmetrical for each quarter cycle , so the sum S in Eq.8 for a fault on wet soil is obviously bigger than for the sinusoidal waveform. This characteristic is used in the algorithm to check the dependability for a fault on wet soil, and security for a sinusoidal waveform. As an indication of the Quarter Cycle Asymmetry, the selection of sum S is 300% of that for sum S for a sinusoidal waveform.

2.2.4 Algorithm Realization

In the algorithm, 15000 samples are collected (it will be discussed in detail later in the data acquisition part in Appendix B) into a data file. Then, three cycles of data are read from the data file each time. Score_Flk, Score_Asym, Score_Quart are calculated using Eq.1 to Eq. 9 as indications of Flicker, Asymmetry, and Quarter Cycle Asymmetry respectively. The sum of the scores are put into an integrator and when the output of the integrator reaches a sufficient level, (set by experience), a trip signal is generated. The integrator output is expressed as follows :

$$\text{Output\_new} = \text{Output\_old} + \text{Score\_Flk} + \text{Score\_Asym} + \text{Score\_Quart} \qquad \text{Eq.9}$$

There are strategies for choosing the ratios of the Score_Flk, Score_Asym, and Score_Quart as parts of the Output. However, the main aim is that the algorithm should perform dependably (ability to trip when it should), and as well as be secure (ability to not trip when it should not). Therefore, when all 15000 data points have been acquired , the outputs should look like Fig. 7 and Fig. 8, in which the X–axis represents time in seconds, the Y–axis is current, in arbitrary units.

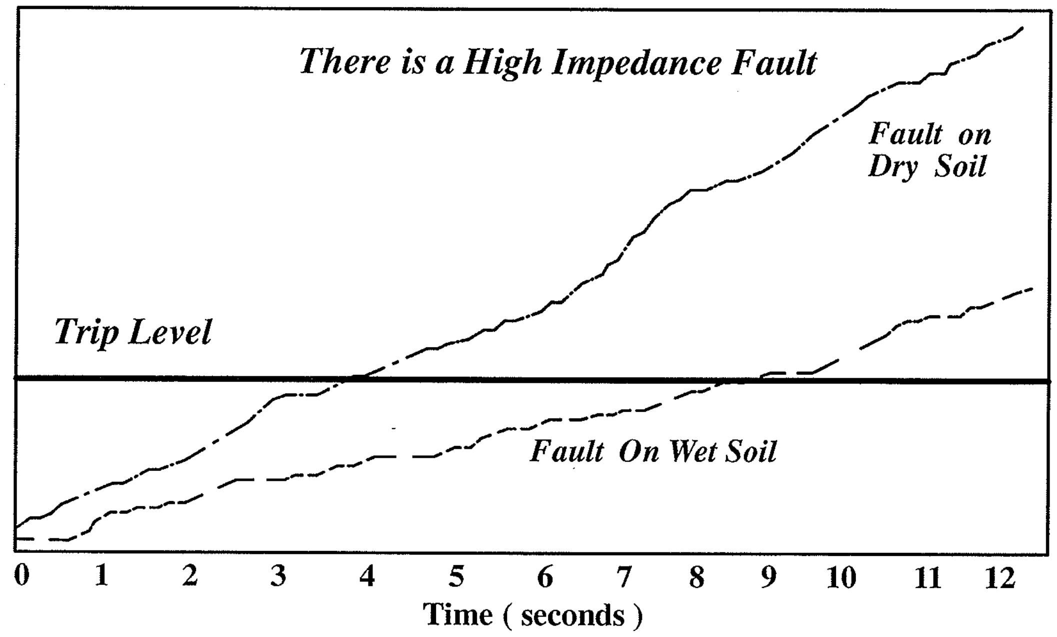


Fig. 7 Outputs of Faults on Dry and Wet Soils

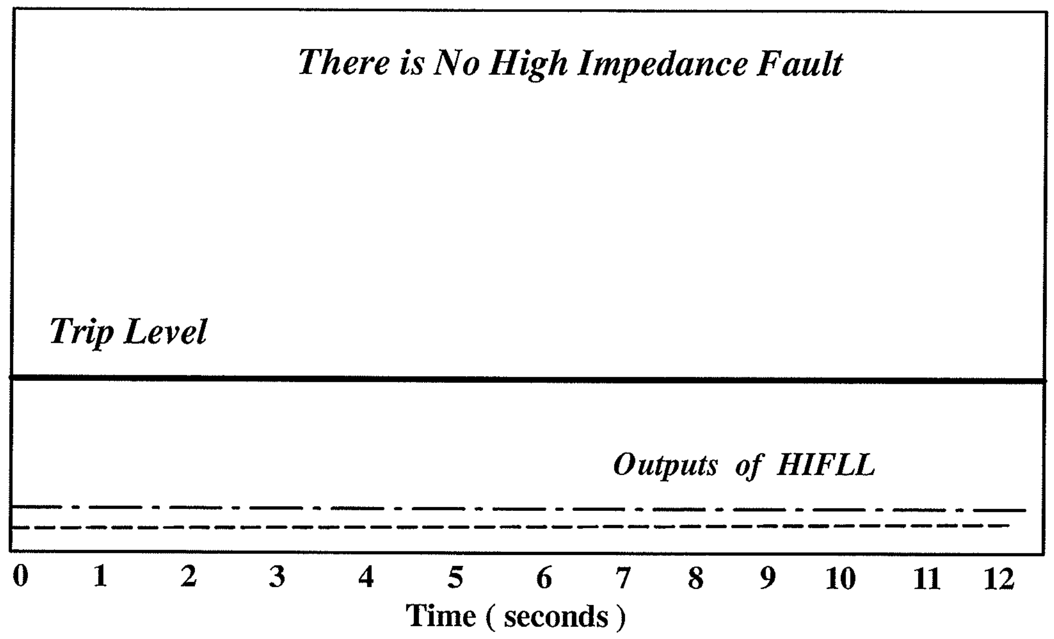


Fig. 8 Outputs of Sinusoidal and Computer Loads

CHAPTER 3 TESTING AND RESULTS

3.1 Apparatus

3.1.1 Circuit Set–up

At the laboratory of the University of Manitoba, a circuit was set up as shown in Fig. 9 to simulate a HIF in a power distribution system. The current is supplied by a 115 V single phase voltage source and an isolation transformer is used to get rid of DC offset voltage from the source.

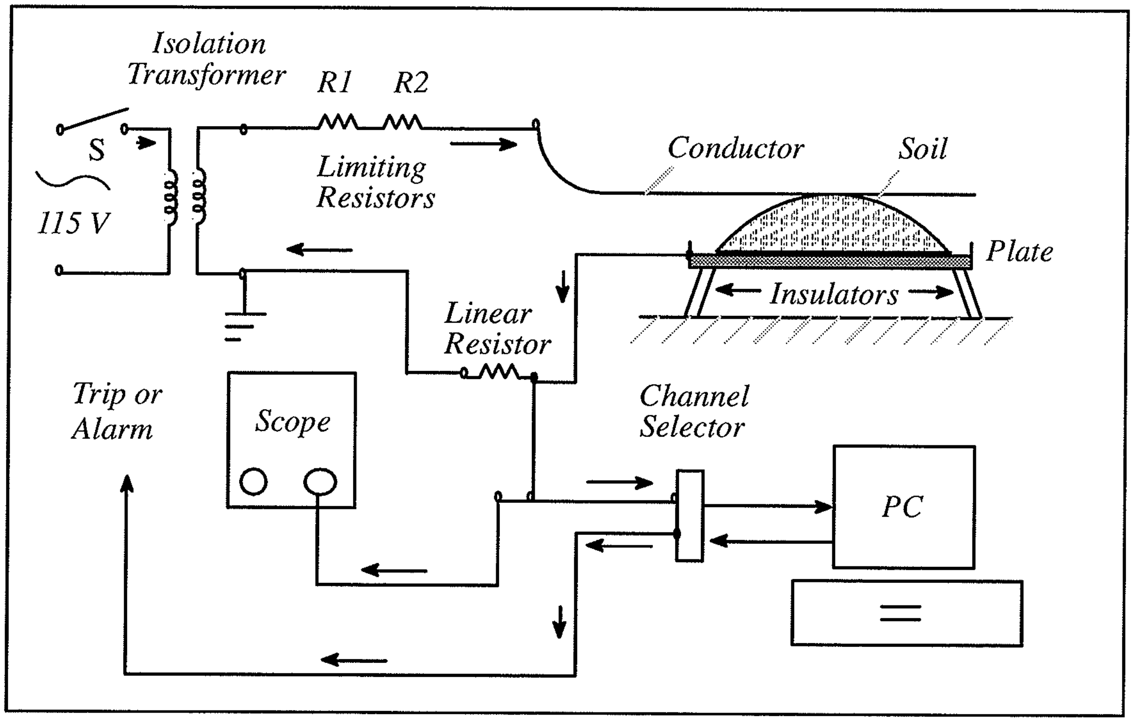


Fig. 9 Schematic of Apparatus for Simulating HIF

Resistors R1 and R2 are used to limit the HIF current level in order to protect the AT–MIO–16 data acquisition board [16] inserted into a PC computer (see detail in Appendix A) and the computer itself. A copper conductor touches the ground, which consists of various kinds of soils contained in a conducting plate. The fault current passes through a linear shunt resistor to generate a voltage proportional to the fault current. Finally, the voltage signal is taken to a scope, and the computer, to carry out data acquisition and analysis.

3.1.2 Initialization of A High Impedance Fault

Before initiating a HIF, it has to to be ensured that all elements can withstand the maximum possible current. Therefore, the fault impedance was temporarily set to zero, to get the worst case. The equivalent circuit for the experiment (with soil impedance equal to zero) is represented in Fig. 10.

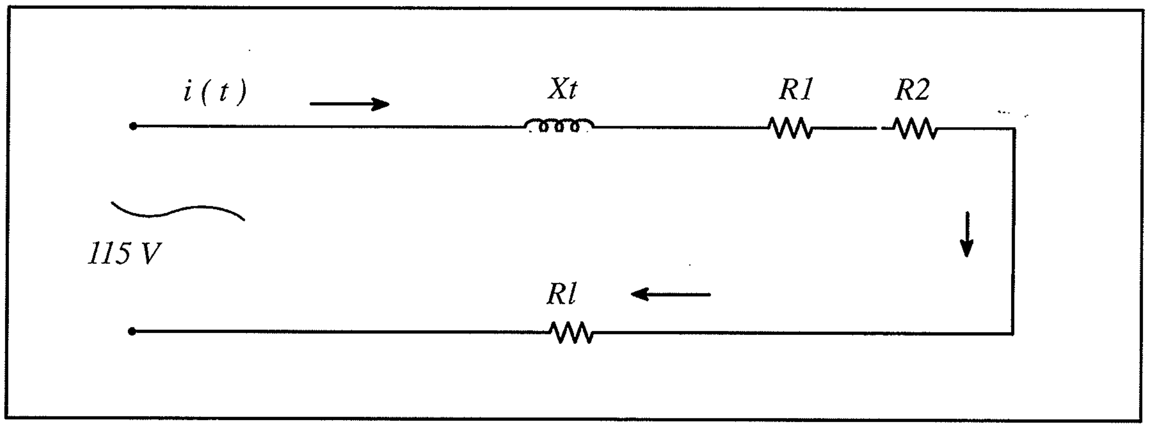


Fig. 10 Equivalent Circuit for Testing Security of Elements

The elements used in the circuit have the following parameters:

Isolation Transformer: its leakage reactance Xt% would be 2% – 2.5%;

$$X_t = 0.023 \times \frac{115 \times 115}{300} = 1.00\ (ohm)$$

Limiting Resistors: they are two 660W, 115 V heaters, so it follows that

$$R1 = R2 = \frac{115 \times 115}{660} = 20.0\ (ohm)$$

Shunt Resistor: RL = 0.5 (*Ohm*), if only one heater is connected into the circuit, then

the impedance of the circuit Z = R + j Xt , here R = R1 +RL = 20.5 (*ohm*).

However, we are only interested in the magnitude of Z, hence,

$$Z_{\min} = \sqrt{R^2 + X_t^2} = 20.5\ (\ ohm\ )$$

Thus, the maximum rms value of the current is:

$$I_{\max} = \frac{V}{Z_{\min}} = \frac{115}{20.5} = 5.61\ (\ A\ )$$

Since the transformer's normal working current is 300/110 about 2.73 A, care must be taken to monitor the current, and only allow this 100% overload for a short time. Normally, the soil resistance will keep the current well below the transformer rating. From Fig. 9, the signal being taken into the scope (and the AT–MIO–16 board) is the voltage across the linear resistor. Even under the extreme case, this voltage is only about 3 V, which is certainly safe because the voltage range of the scope is +/– 20V and the voltage range of the data acquisition board is +/– 10 V.

This initialization of a HIF consists of the closing the switch 'S' in Fig. 9. The equivalent circuit is shown in Fig. 10, except that there is a nonlinear resistor , Rs , representing the soil resistance as shown in Fig. 11.

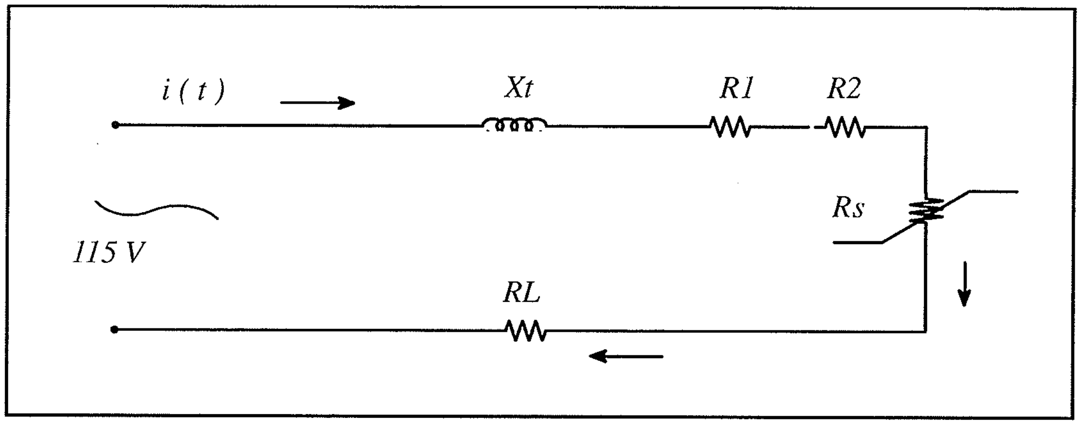


Fig. 11 Equivalent Circuit for Simulating HIF

3.1.3 Signal Generation of Computer and Sinusoidal Loads

As mentioned previously, any algorithm for detecting HIF should have two aspects: dependability and security. In order to check the security of fault–like loads (e.g. a computer load) and for normal sinusoidal loads, proper signals must be generated. It is very easy to get a sinusoidal signal from a signal generator, but for the computer load, a shunt resistor, RL should be connected in series with a computer. Here, the computer becomes Rs of Fig. 11. The voltage across the linear resistor is taken into the scope and the working computer through coaxial cables. The experimental set up is as shown in Fig.12. The voltage across RL can be adjusted by choosing RL with care, such that safe signals are taken into the data acquisition equipment.

See Appendix B for details of the data acquisition procedure.

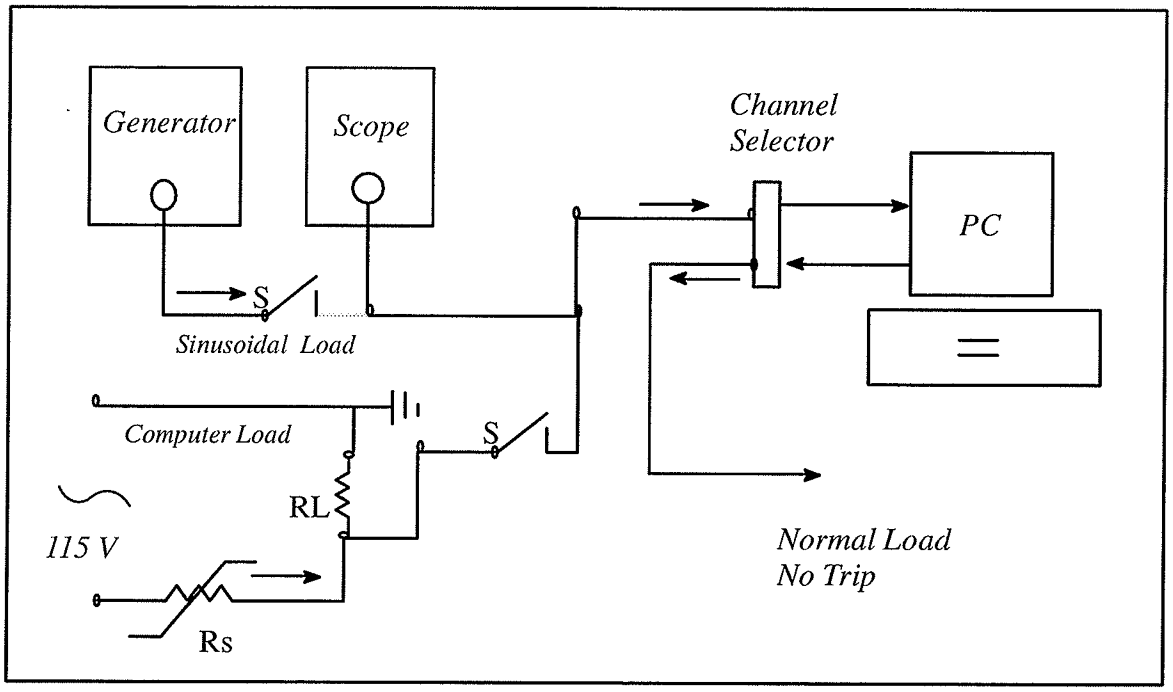
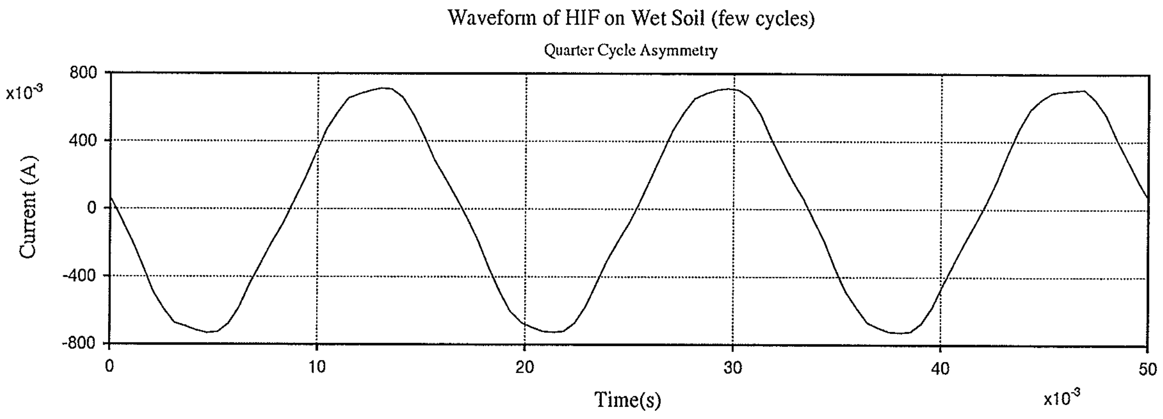


Fig. 12 Schematic of Apparatus for Checking Security

3.2 Waveforms

Based on the circuits described, many waveforms were collected including faults on wet soil, dry soil, grassy wet soil , and grassy dry soil for a dependability check of the algorithm. As a security check, sinusoidal and computer load waveforms were also acquired. These waveforms are illustrated below.

3.2.1 Current Waveform of Fault on Wet Soil



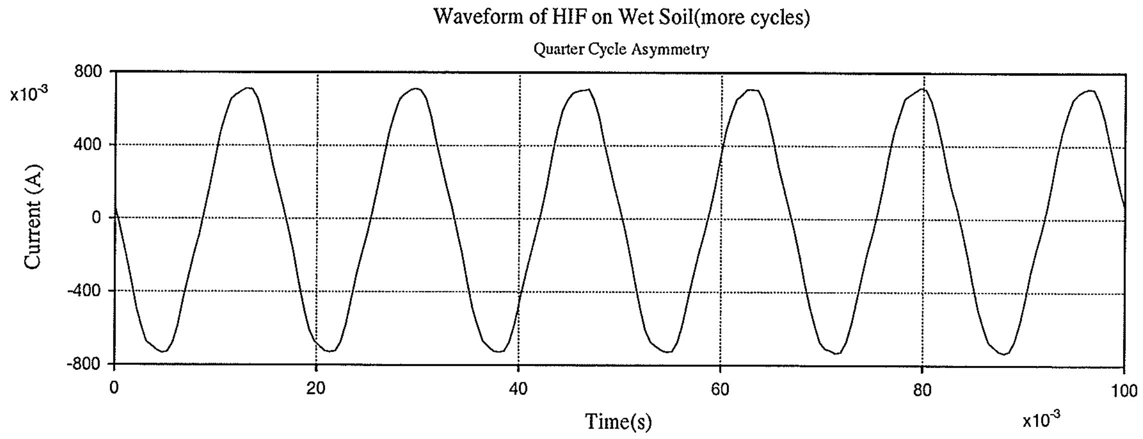
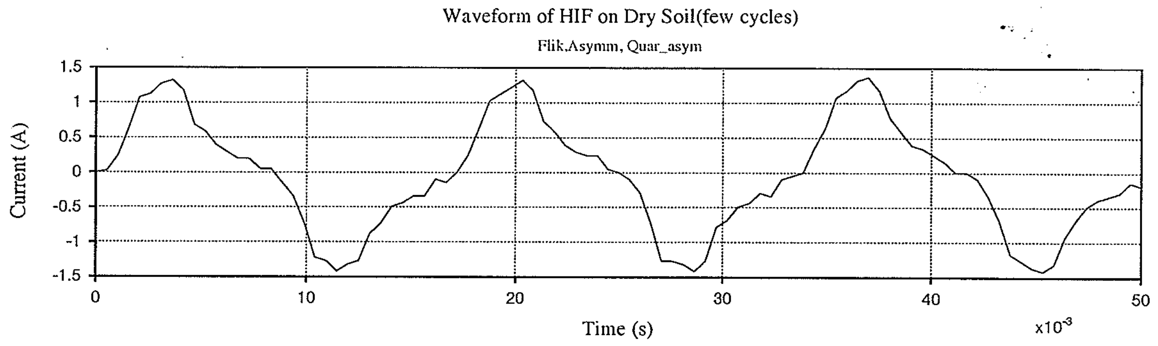


Fig. 13 Current Waveforms of Fault on Wet Soil

From Fig. 13, for wet soil, it is obvious that only the Quarter Cycle Asymmetry part of the algorithm explained in Chapter 2 will be effective.

3.2.2 Current Waveform of Fault on Dry Soil



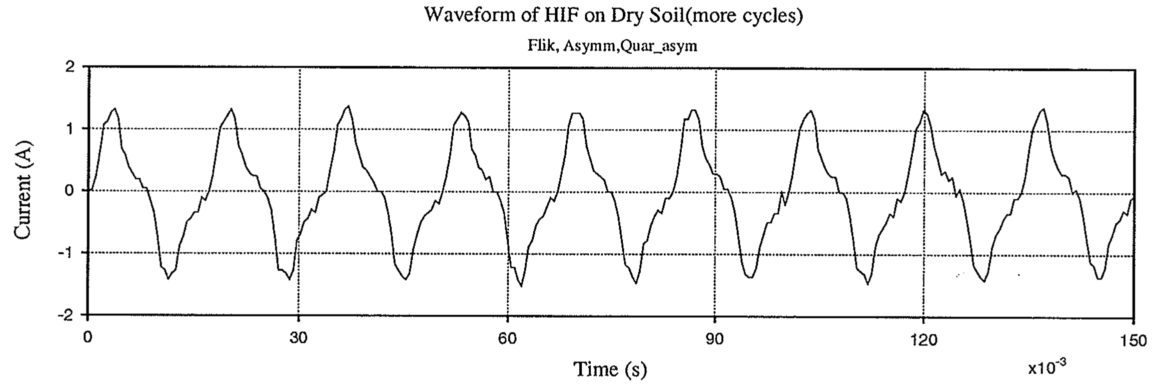
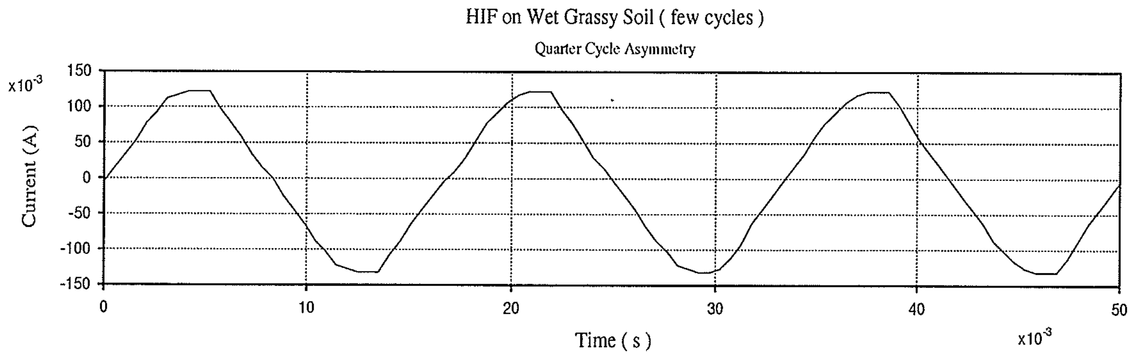


Fig. 14 Current Waveforms of Fault on Dry Soil

In Fig. 14, for dry soil, Quarter Cycle Asymmetry is obviously present in the current waveforms. Flicker and Asymmetry are not as obvious. However, if the waveforms could be observed carefully and the effect of the current's rms value be considered as well, there could be some of all three: Flicker, Asymmetry, and Quarter Cycle Asymmetry.

3.2.3 Current Waveform of Fault on Grassy Wet Soil



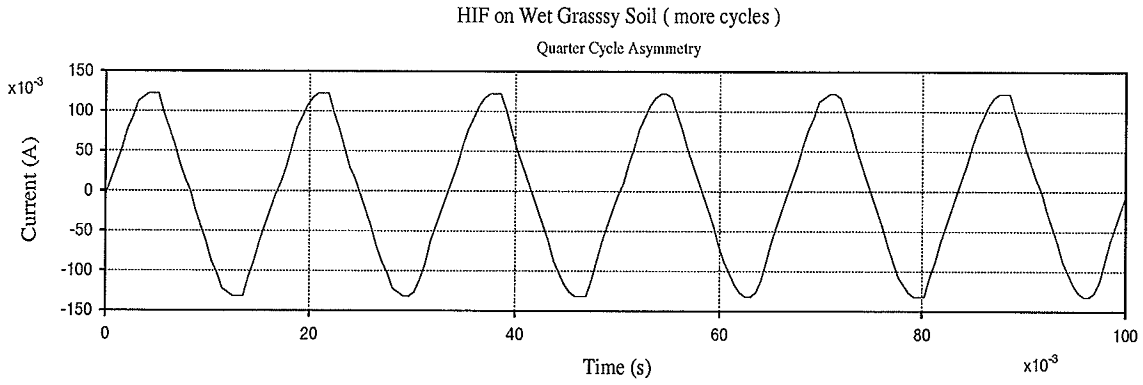
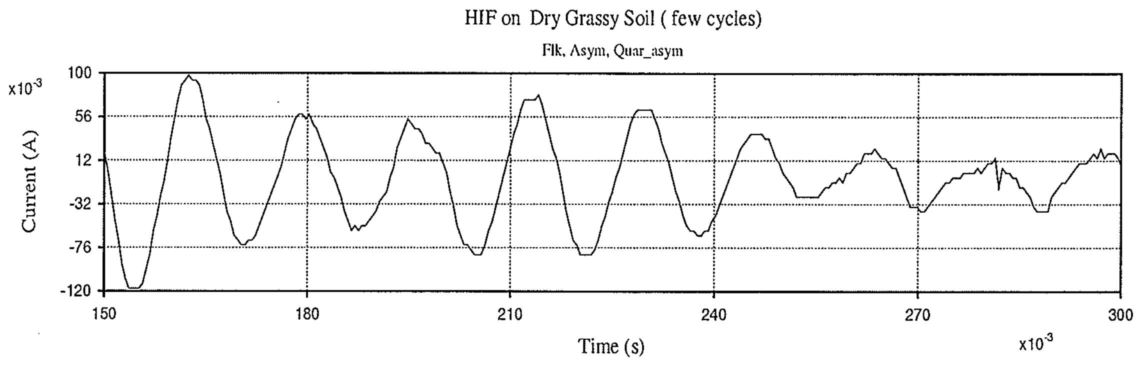


Fig. 15 Current Waveforms of Fault on Grassy Wet Soil

The waveforms in Fig. 15, for grassy wet soil are similar but not identical to the waveforms of faults on wet soil. The above waveforms of Fig.15 have more ripples than those of faults on pure wet soil. However, Quarter Cycle Asymmetry is still a distinct characteristic.

3.2.4 Current Waveform of Fault on Grassy Dry Soil

Figure 16 shows fault current waveforms for grassy dry soil. It is quite obvious that all parts of the algorithm will be relevant.



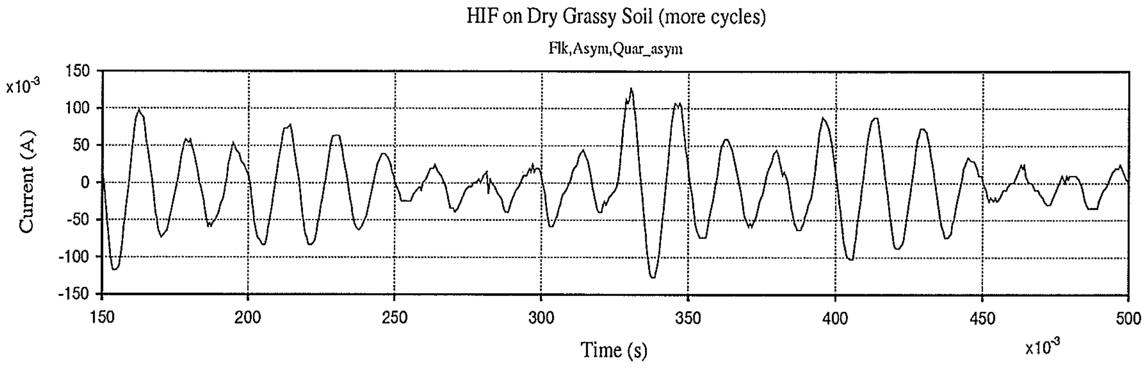


Fig. 16 Current Waveforms of Fault on Grassy Dry Soil

So far four different waveforms of HIF on different kinds of soil materials have been shown. Many more waveforms were obtained than those shown here; however, Fig. 13 to Fig. 16 are enough to illustrate the various types.

3.2.5 Sinusoidal Current Waveform

It is necessary for us to use a sinusoidal waveform in order to guarantee never to trip under normal conditions. Figure 17 shows this case.

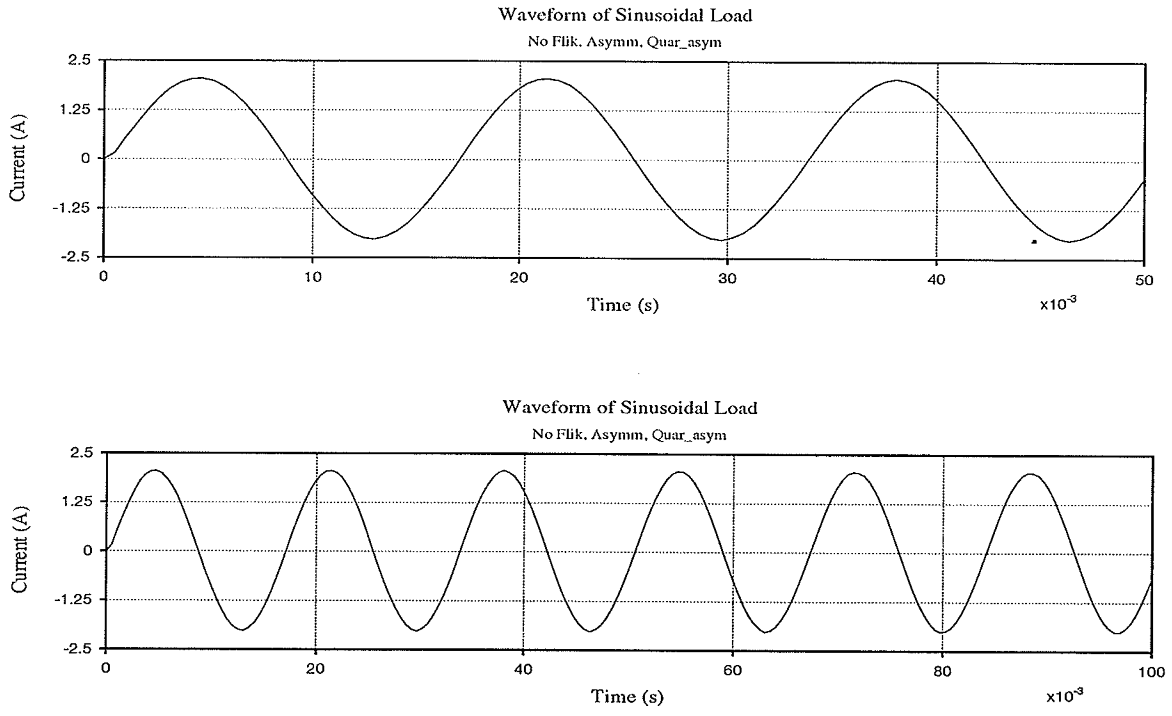
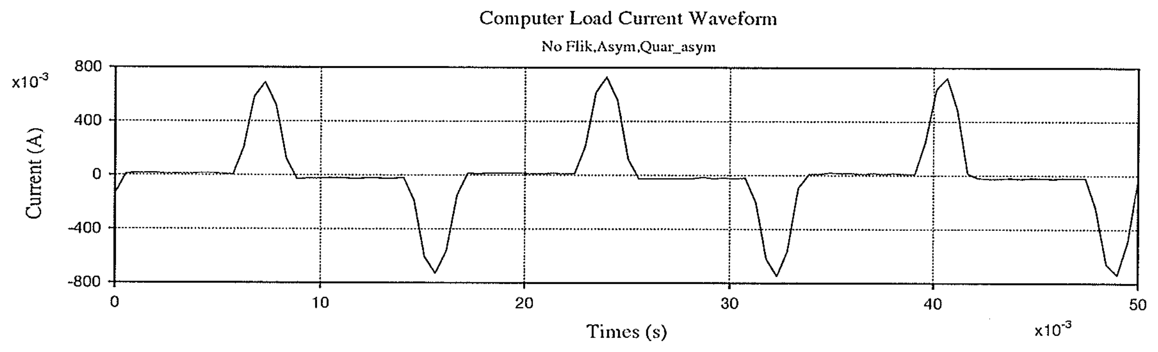


Fig. 17 Sinusoidal Current Waveform

3.2.6 Computer Load Current Waveform

Typical computer current waveforms are shown in Fig. 18. The algorithm must be “secure” (not trip) for this kind of input. It can be easily seen that there are no Flicker and Asymmetry. There are very slight Quarter Cycle Asymmetry characteristics appearing on these waveforms; however, the S for calculating Quarter–Cycle Asymmetry in Eq.8 is generally bigger than for this waveform.



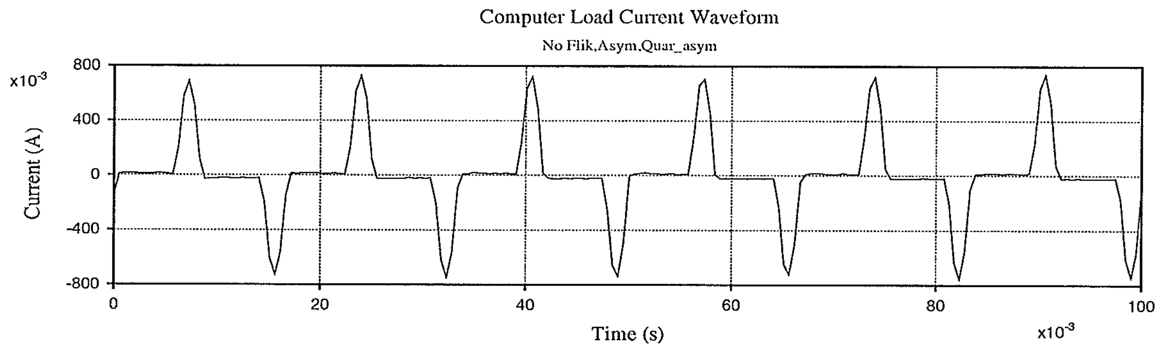


Fig. 18 Computer Load Current Waveforms

3.3 Use of a C++ Program to Realize the Algorithm

Currently, various program languages are being used in electrical engineering areas. The question is which one is better to use to realize the algorithm. Due to the demands of HIF detection, many 'objects' have to be dealt with, such as data acquisition, data processing, real time display, calculation, detection and control. It has been concluded that the C or C++ [21] [22] programming language is the best one to use.

3.3.1 C++ Program Based on the DOS System

C and C++ are very similar languages in general use, but C++ is more powerful when the program becomes large. C++ has two distinct functions C does not have,

1) Object oriented programming ability.

2) Very easy transfer to a Windows program, since all Windows programs coming with the AT–MIO–16 data acquisition board are written in C++ .

This research project deals with several aspects, such as data acquisition, data processing, data files translation from one environment to another environment, algorithm realization, graphic display of the results on the screen, and so on. Therefore, C++ and a Windows program will be the final goal.

The program has too many details and techniques to explain here. The flowchart is shown in Fig. 19. For details of the program, see Appendix C.

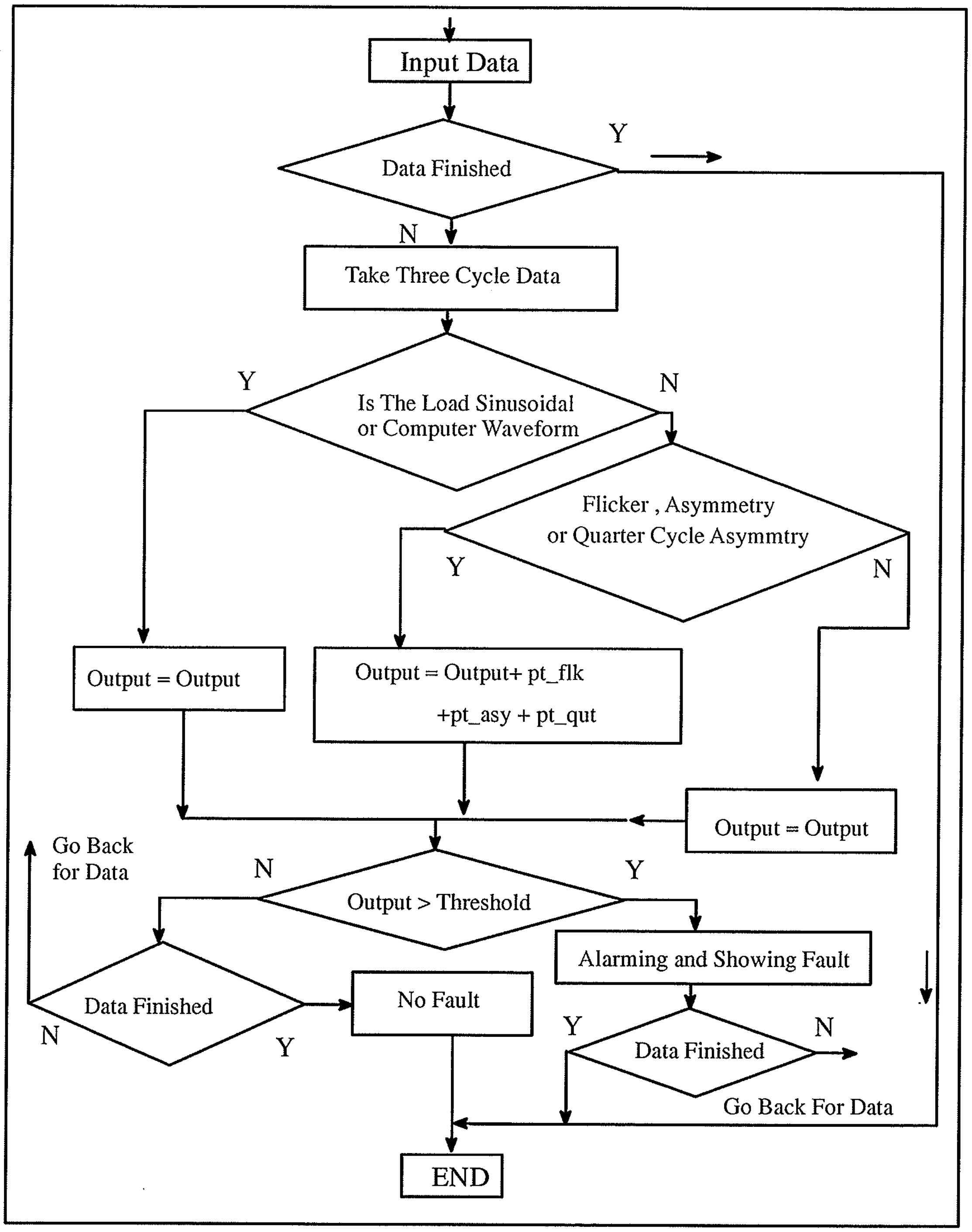


Fig. 19 Flowchart of C++ Program for Detecting HIF Based on DOS

3.3.2 Testing Results and Analysis for Different Waveforms

A facsimile of a test result for a HIFLL waveform is shown in the Fig. 8. For the computer load waveforms, the results always stay the same. Figure 20 illustrates these .

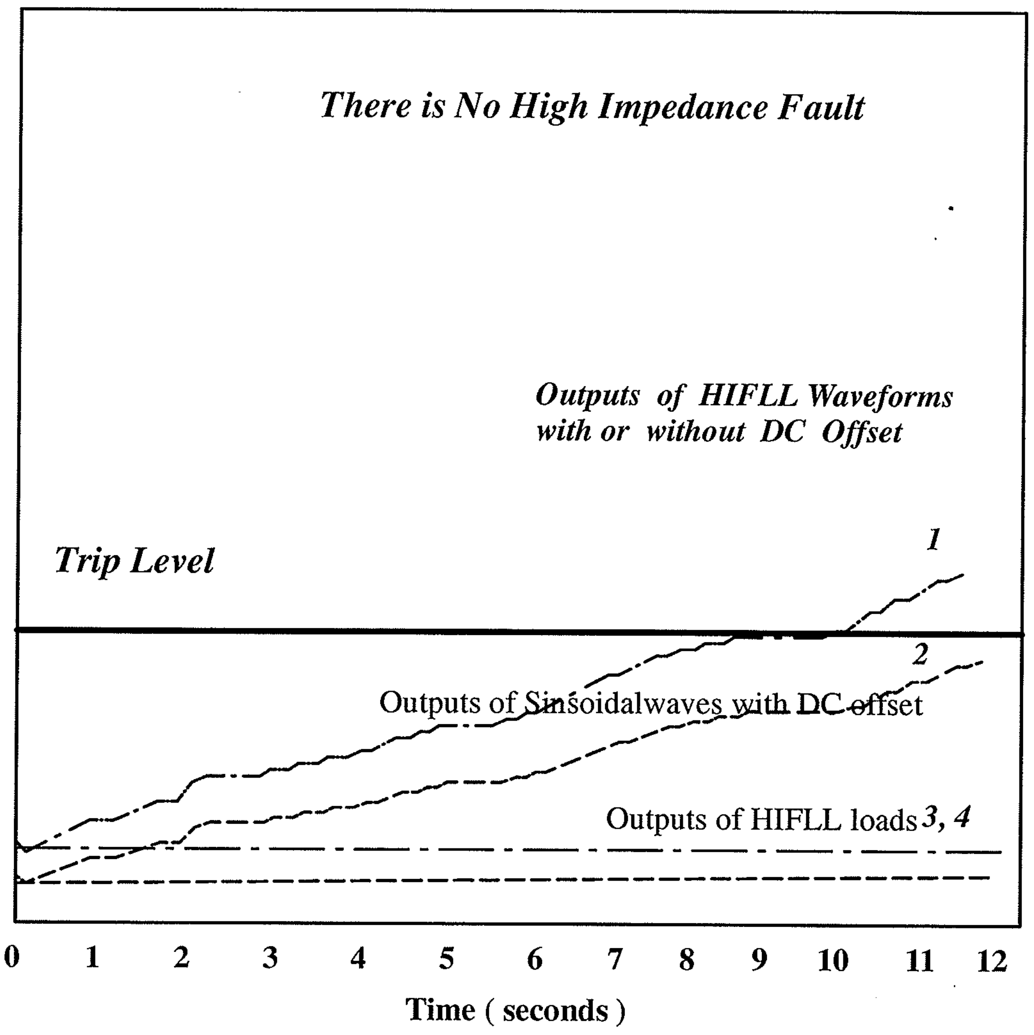


Fig. 20 Outputs of Sinusoidal and Computer Loads

Generally speaking, the outputs of sinusoidal and computer load waveforms through the algorithm should always appear as a flat straight line (see curves 3, 4 above) .

Incidentally, a dc offset in an otherwise sinusoidal input was a convenient check of "Asymmetry": curves 1 and 2 of Fig.20.

The computer load waveform, because it is symmetrical and periodic, will always give excellent output results(i.e."no trip"). Hence, the most important conclusion is that the algorithm and program are **secure** for the HIFLL at least within the range considered.

Returning to the **dependability** of the algorithm and program for HIF: from Fig. 21, it can be seen that the algorithm and program work very well for detecting HIF on wet, dry, grassy wet, and grassy dry soils. In Fig. 21, the four output results could vary slightly depending on random voids within the soils and the extent of the moisture of the soils. Of course, it is impossible to show all the results here, but the algorithm and program have indeed worked well in detecting HIF cases for these kinds of soils. Therefore, the final conclusion is that the algorithm and program have satisfied **dependability** for HIF cases and **security** for HIFLL cases.

Even though quite satisfactory results have been achieved , the research still needs more improvements. During the data acquisition procedure, there were too many manual operations required making the procedure unsuitable for automatic control and protection in power systems. In addition, the outputs based on DOS systems can only be displayed on one screen at a time. This is not convenient from the customer's point of view. To solve this problem, a Windows program is a good answer. Fortunately, the program developed was written by C++ and could be easily transferred to Windows.

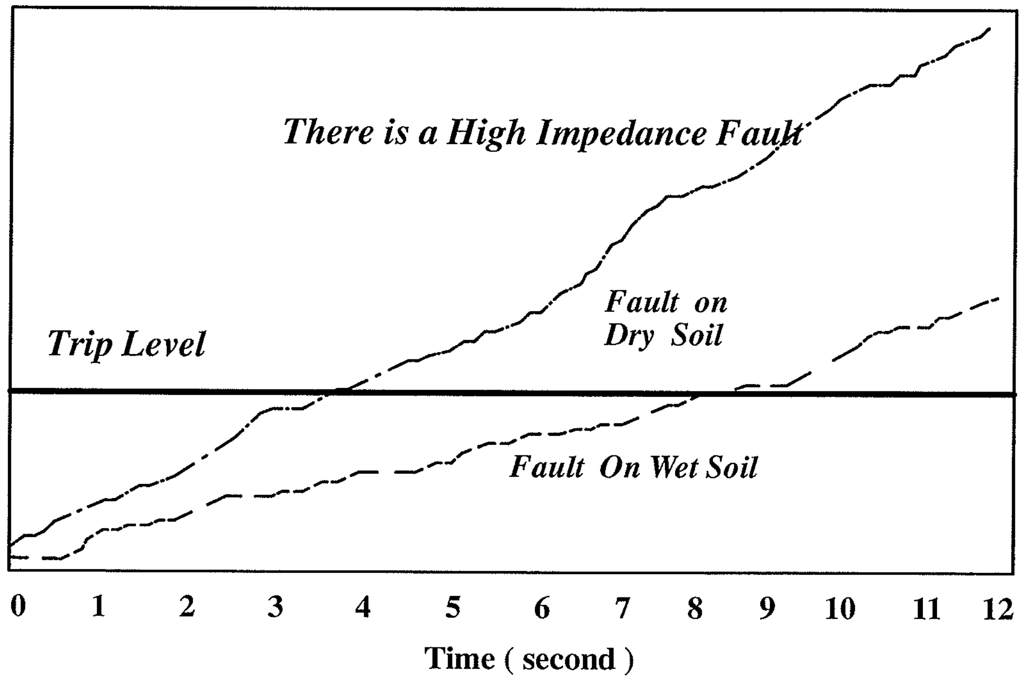


Fig. 21 a   Outputs of Faults on Dry and Wet Soils

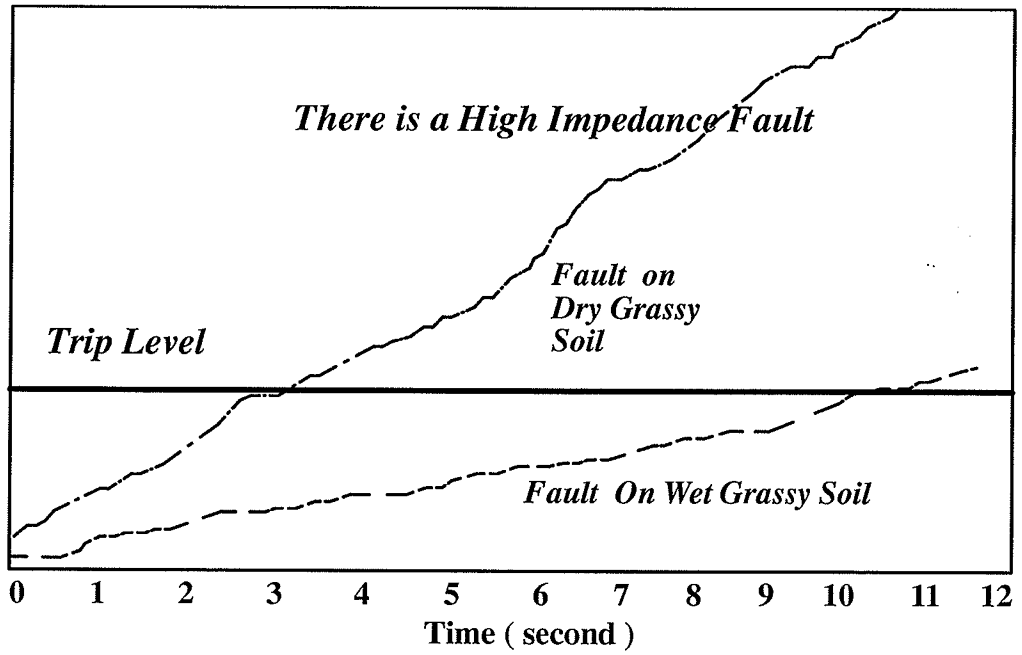


Fig. 21 b   Outputs of Faults on Dry and Wet Grassy Soils

CHAPTER 4 WINDOWS DESIGN AND OPERATION FOR REAL TIME PROTECTION

4.1 Windows Program Statement

Currently, with the development of computer technology, Windows programs are becoming more and more popular. It is known that a computer program based on either DOS or UNIX systems can be reached by using a mouse to click the appropriate button in a Windows display. However, the reason for using a true Windows program here to realize HIF detection is not only because of its popularity, but also its ease of use. As mentioned previously, the program written in C++ has already been developed. A program written in C++ is easily transferred to a Windows program. C++ is also object oriented. It is very suitable for this research project because each problem in the research project can be considered as an individual object model. The interfaces between them can be finished at a later time. In terms of the Windows program, each object can be considered as a different window or button. In addition, a Windows program is considerably more convenient than one based on DOS. A program based on DOS has only one screen as an output device. If different screens are required, different programs have to be run several times. In contrast to DOS, by clicking the buttons using the mouse, many windows can be opened at the same time.

Another important reason for using the Windows program is the AT–MIO–16 board requirements. Much of the software that comes with the board is written for MS–Windows in C++, some of which can carry out data acquisition continuously. Connecting the C++ program based on DOS to the board software in order to realize the real time detection and control is an easier way to do the project.

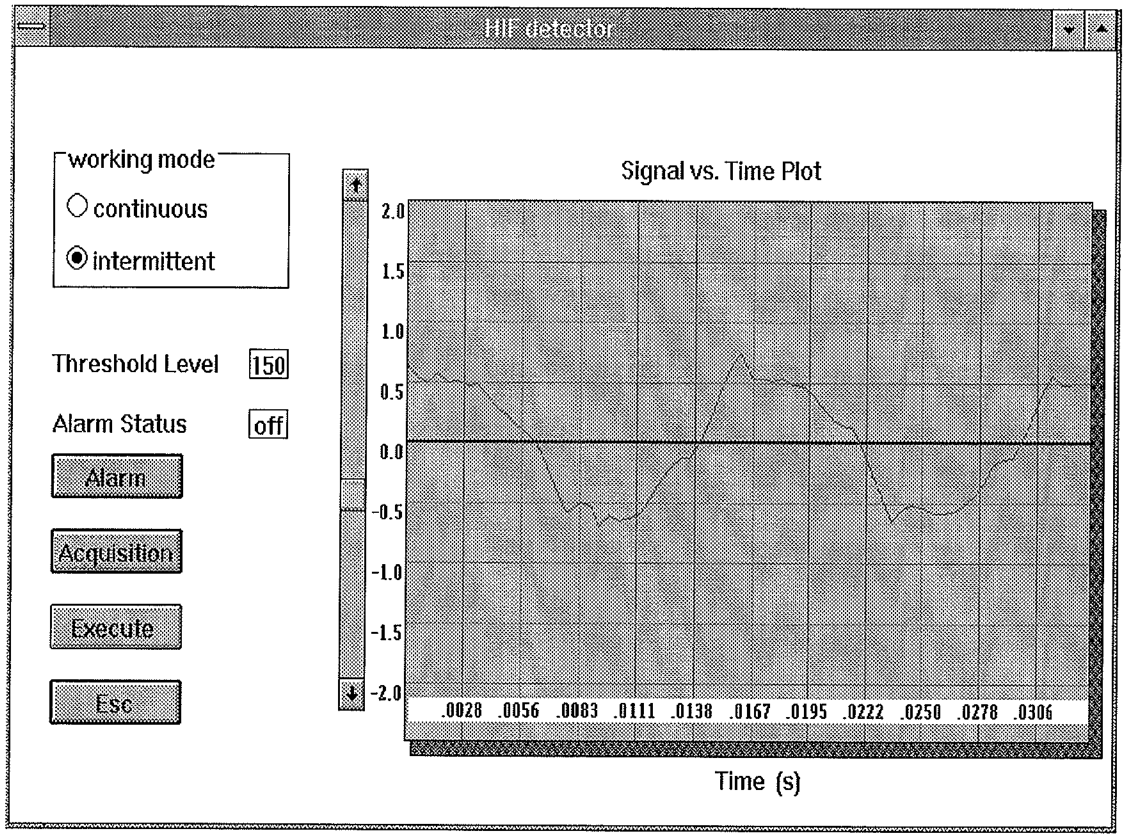
Since the Windows programming [14], [15], [16] took considerable time, it is impractical to explain the details here . The Windows program is included in Appendix C. Only the results of the Windows design which relate to the HIF and HIFLL analysis will be shown in the following sections.

4.2 Main Windows Design

In a Windows program, there are two kinds of windows. One is called the parent window or main window, and the other is called the child window. The parent window does the main job and in the meantime, it can talk to its child windows, control them or be controlled by them.

4.2.1 Signal Acquisition Window Design

In the Windows program, two main windows have been designed . One is a signal acquisition window whose functions include data acquisition, data processing, and displaying the real time signal on the window screen. This window should be used together with an oscilloscope in order to make sure that the signal waveform is being captured correctly. The acquisition window as shown in Fig. 22 has a child window called "**Acquisition**". As a matter of fact, the child window is only a button. If the signal being shown on the scope is needed, it can be acquired using the mouse to click the "**Acquisition**" button. The acquisition window program will start to work, performing data acquisition, processing the data, and then displaying the waveform on the screen. In Fig. 22, on the top bar of the window there is a title "**HIF detector**" meaning the HIF analysis is in process. The title will also appear on another main windows design called algorithm output analysis window, which is to be discussed in the next topic. In Fig. 22, the X–axis represents time in seconds, the Y–axis is current, in arbitrary units.


HIF detector

working mode
continuous
[✓ intermittent]

Threshold Level 150

Alarm Status off

Fig. 22 Signal Acquisition Window Design

On the windows screen, there are only two cycles shown. Many cycles may be displayed if desired.

4.2.2 Algorithm Output Analysis Window Design

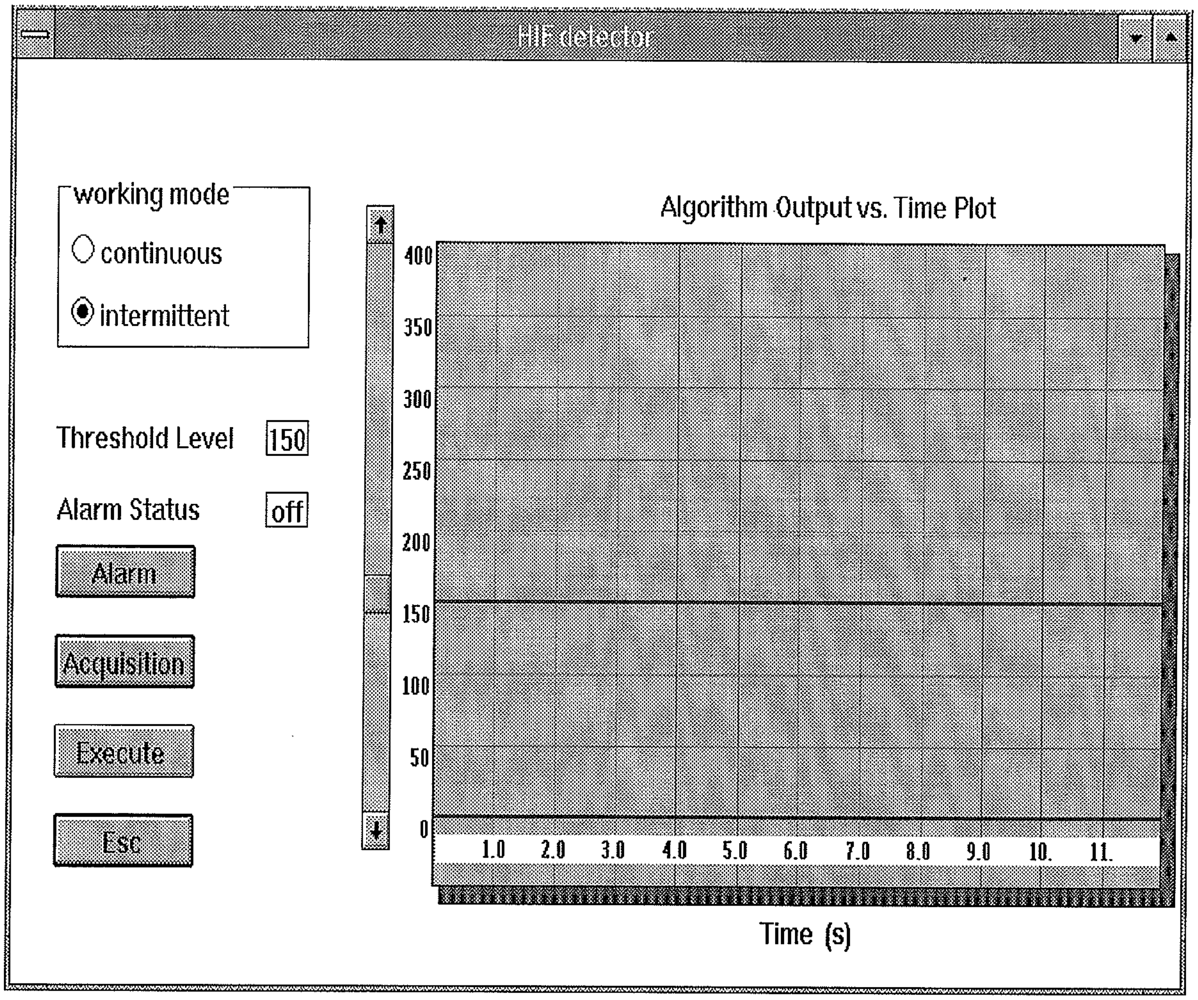
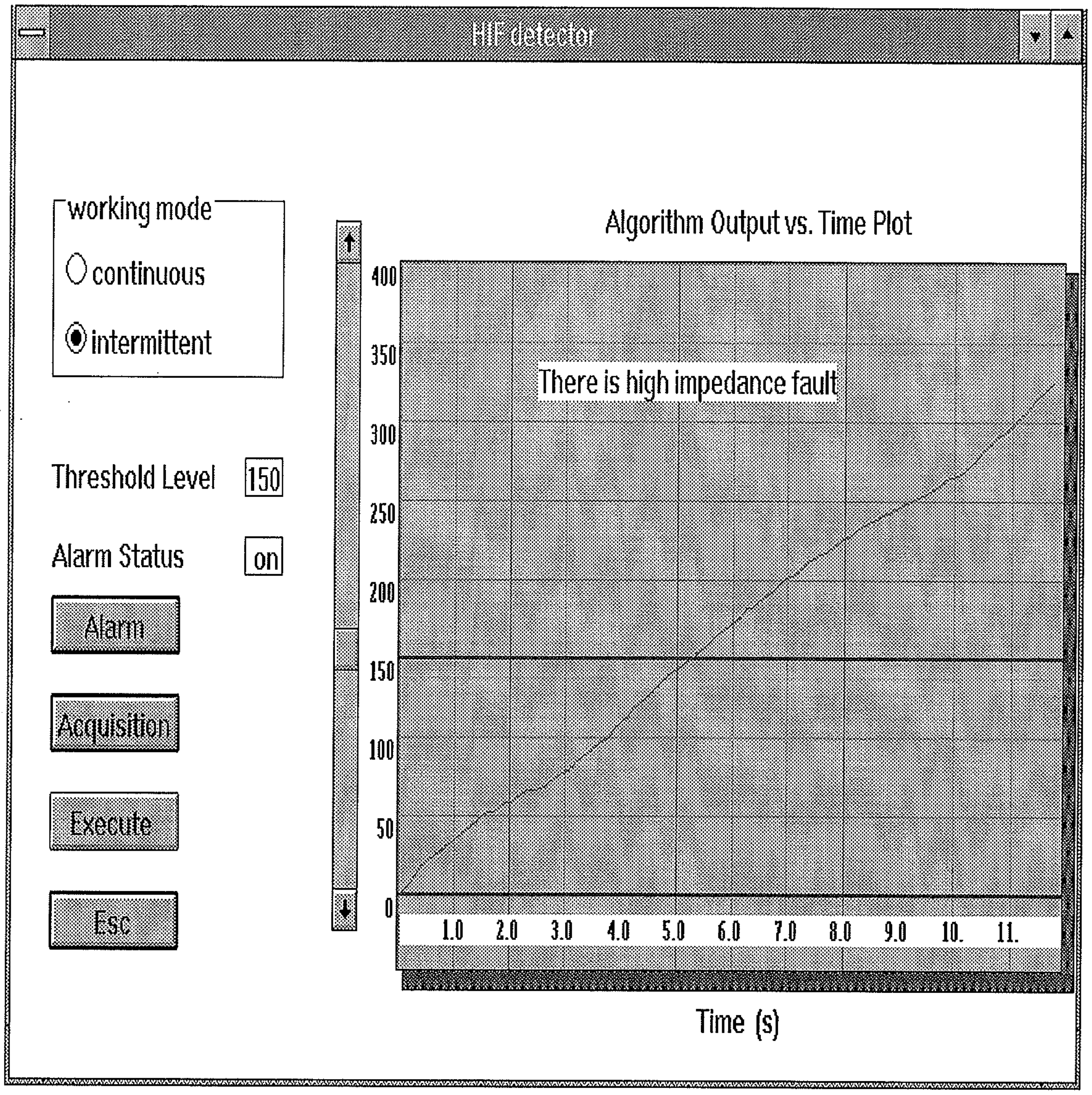


Fig. 23 Algorithm Output Analysis Window Design

This window in Fig. 23 is the most important one among the designed windows, since it covers data acquisition, data processing, algorithm realization, automatic control and protection, dynamic output results displaying, etc. It includes five child windows to talk, control or be controlled. If there is a HIF in the tested system, the dynamic output will look like what Fig. 24 shows. Otherwise, for HIFLL, the results will appear as shown in


HIF detector

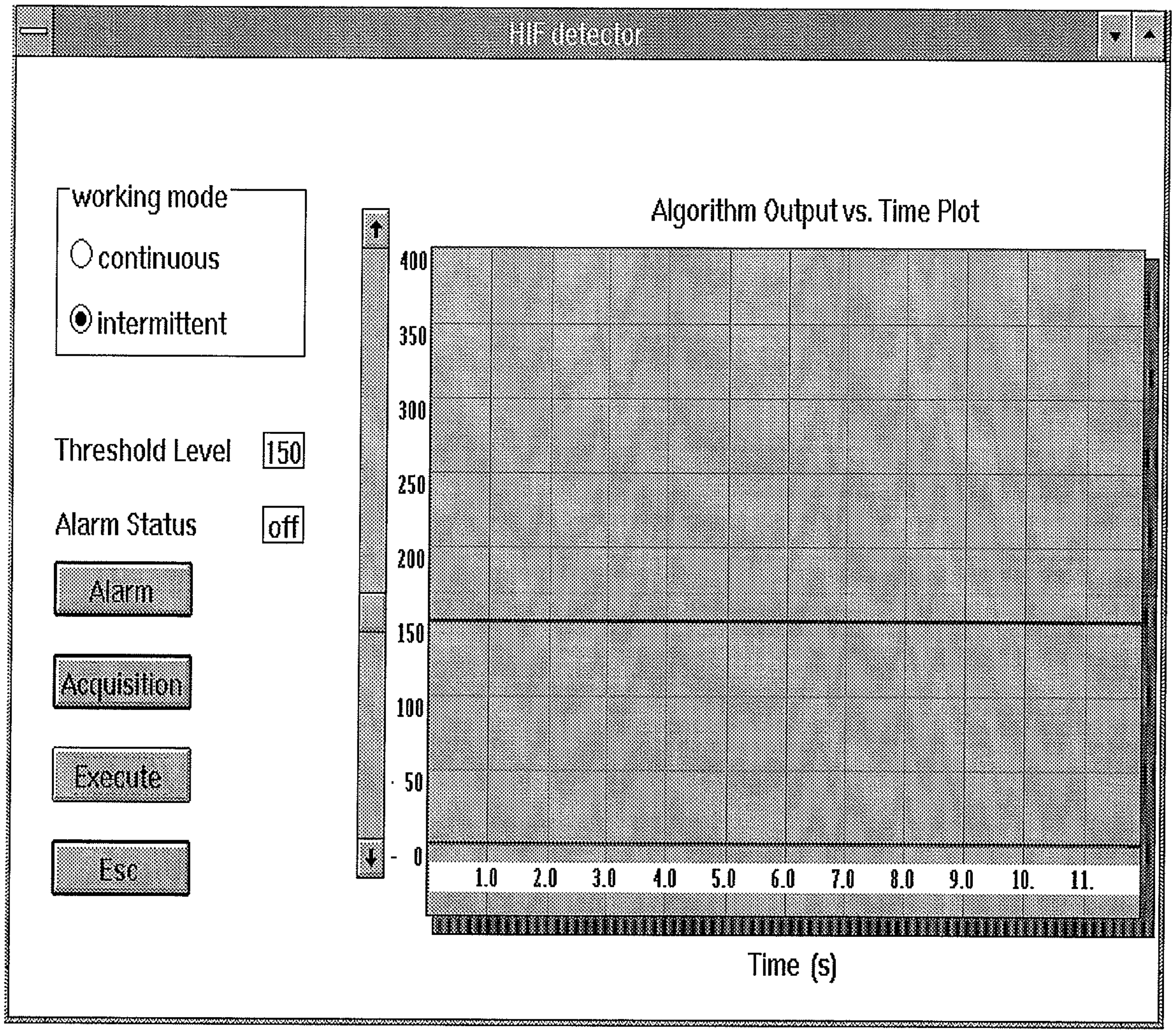
working mode
○ continuous
● intermittent

Threshold Level 150

Alarm Status on

Fig. 24 Window Outputs of HIF

Fig. 25, the only difference with Fig. 24 being the output curve and the **Alarm Status,** which will be in an **off** state representing a zero–volt output.



Threshold Level 150

Alarm Status off

Fig. 25 Window Outputs of HIFLL

4.3 Child's Window Design

4.3.1 Working Mode Windows Design

In Fig. 23, there is a child window called "**Working Mode**" window in which there are two selections: **Continuous** and **Intermittent.**

In the intermittent mode, when the **Execute** button is selected, the program proceeds with data processing, algorithm realization, and display of output results. The Windows program is much more convienent than a DOS program. In the DOS system, too many manual operations are required.

For a power systems field installation, the HIF detection must work continuously and automatically. This is the continuous mode.

When the **Continuous** button is clicked, the Windows program will operate continuously and repeat the whole procedure mentioned above every 20 seconds until the button **Esc** is hit.

4.3.2 Threshold Level and Alarm Windows Design

Assuming it is in the continuous state, as in Fig. 23, there is a **Threshold Level** child window and a vertical bar beside the window. In the bar, there is a button alongside a number which can be moved by the mouse. The setting is in the range of 0 – 400 (arbitrary units). The **Threshold Level** is shown as 150. As previously explained , the setting of the **Threshold Level** really depends on experience, but it is very important that the output of

HIF should always reach this level while the output of HIFLL should not.

The function of the **Alarm Status** child window is to transmit an external signal and give a warning to operators when there is a HIF. The **Alarm Status** is in the **off** state before the computer runs. When the dynamic output on the window screen reaches the **Threshold Level** , the **Alarm Status** changes to the **on** state automatically. At the same time, the sentence **There is High Impedance Fault** appears on the window as shown in Fig. 24. The status will stay in the **on** state until the whole procedure finishes (about 7– 10 seconds).

When the next procedure starts, the **Alarm Status** will be reset to the **off** state again automatically and the sentence **There is High Impedance Fault** disappears. Then the whole procedure will be repeated. Of course, if the output can not reach the **Threshold Level** , the **Alarm Status** will stay in the **off** state as shown in Fig. 25. The **on** or **off** state can also be manually controlled by clicking the button **Alarm** in order to check the alarm output function. Whenever **Alarm Status** is in the **on** state as shown on Fig. 24, a 5V signal is sent through the output channel of the channel selector. Otherwise, the output is zero as shown in Fig. 25. The signal could be used to trip a breaker, turn on a light, make a sound through an alarm or whatever the engineer chooses.

4.4 Real Time Detection and Automatic Control

From a power systems perspective, it is acceptable for a high impedance fault to persist for a relatively long time , possibly around half an hour. Since it only takes 20 seconds for the Windows program to carry out the whole detection procedure, the result meets the requirement of real time detection. In addition, when the Windows program is running in the continuous state, it sends a 5V or 0V output signal every 20 seconds. This signal can be used for various automatic control purposes depending on customers' preference.

CHAPTER 5   RESULTS AND ANALYSIS FOR WINDOWS OUTPUTS

All the results that have been acquired from the Windows program are similar to those shown in Fig. 24 and Fig. 25. For HIF, some results which are shown could be slightly different from those in Fig. 24, depending on the soil features and random characteristics in the HIF. However, the results almost always reach the threshold level as expected. For HIFLL, some results occasionally go up a bit higher than the horizontal line as shown in Fig. 24. This could be caused by fluctuation in the power supply, unstable operation of the signal generator(sinusoidal signal with dc offset) or some other magnetic field disturbances. However, the outputs of the windows program for HIFLL almost never reach the threshold level. This is the goal which is expected.

5.1 Windows Outputs for Sinusoidal Waveforms with Different DC Offsets

As mentioned previously, incidentally, a dc offset in an otherwise sinusoidal input is a convenient check of "Asymmetry". Such a waveform is shown as follows.

5.1.1 Windows Output for an Ideal Sinusoidal Waveform

For an ideal sinusoidal waveform as shown in Fig. 26, the windows output of the algorithm is shown in Fig.27.

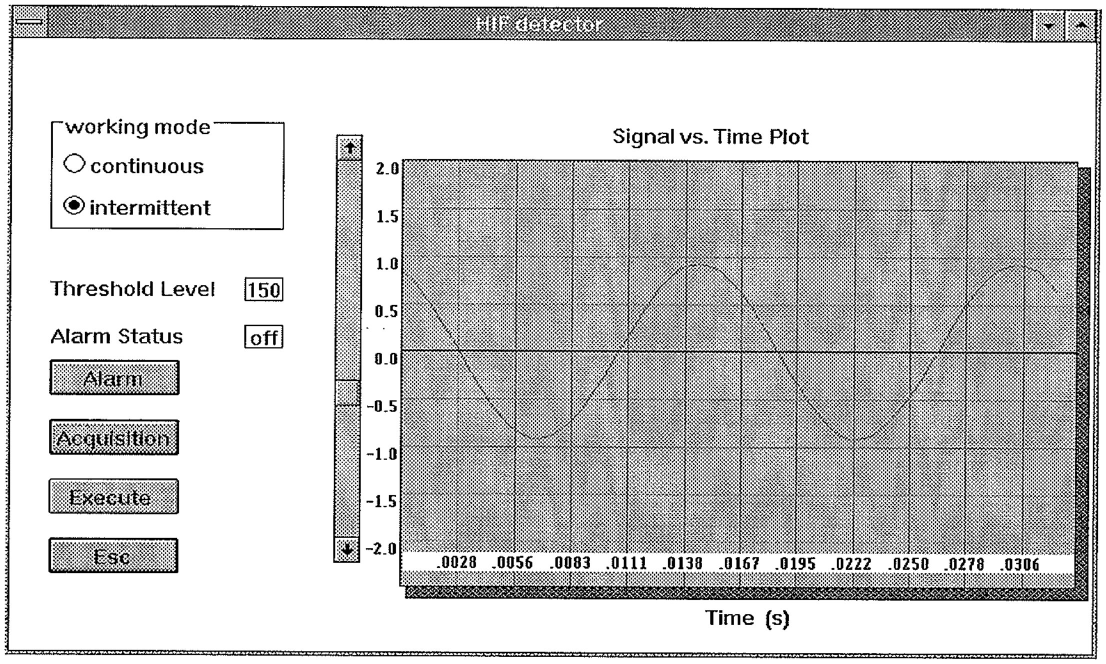


Fig. 26 An Ideal Sinusoidal Waveform

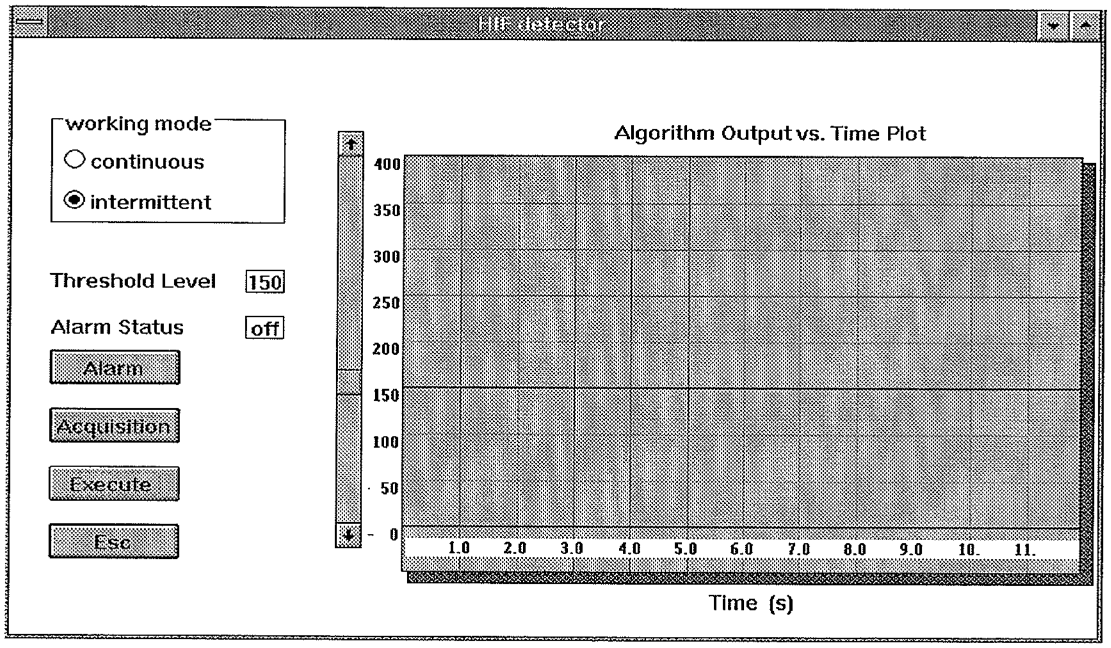
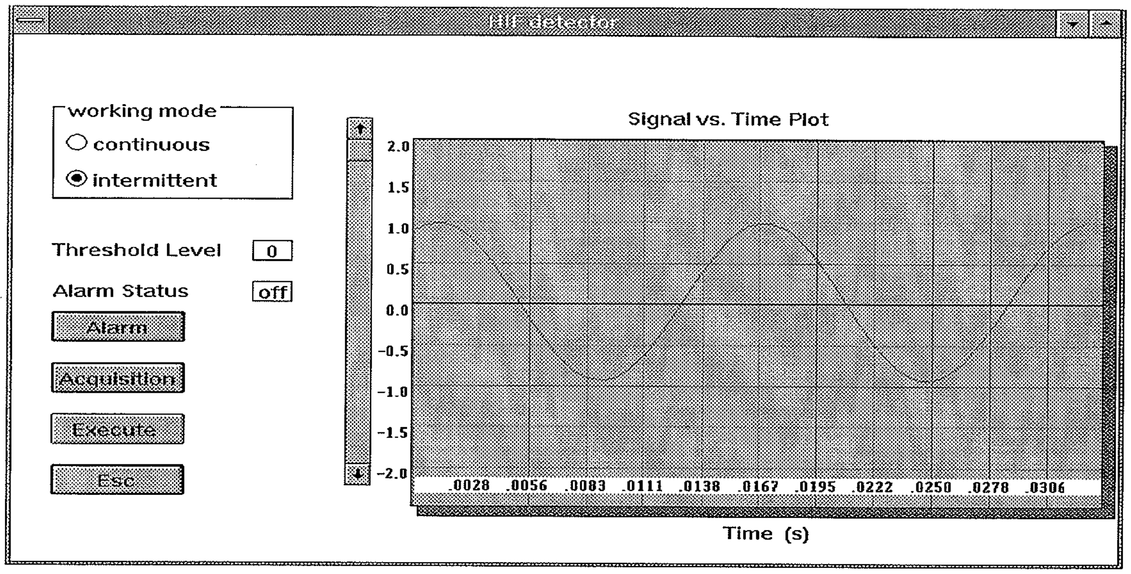


Fig. 27 Windows Output of an Ideal Sinusoidal Waveform

5.1.2 Windows Output of Sinusoidal Waveform with DC Offset

From Fig. 28 and Fig. 29, the windows output goes up because of Asymmetry of the waveform.

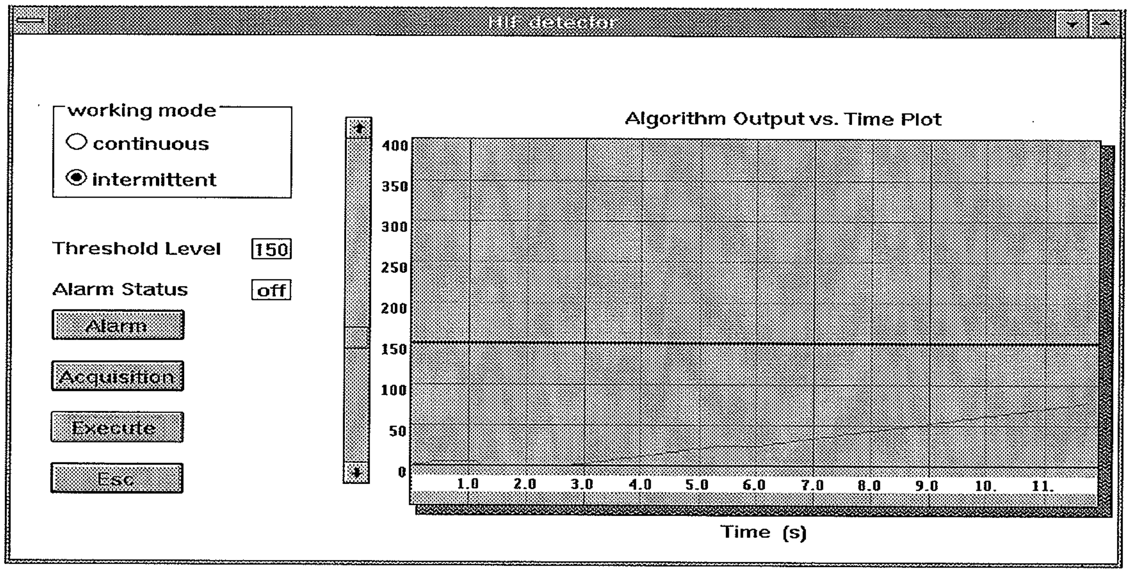


working mode: ○ continuous [✓ ● intermittent]

Threshold Level 0

Alarm Status off

Fig. 28 Sinusoidal Waveform with DC offset



working mode: ○ continuous [✓ ● intermittent]

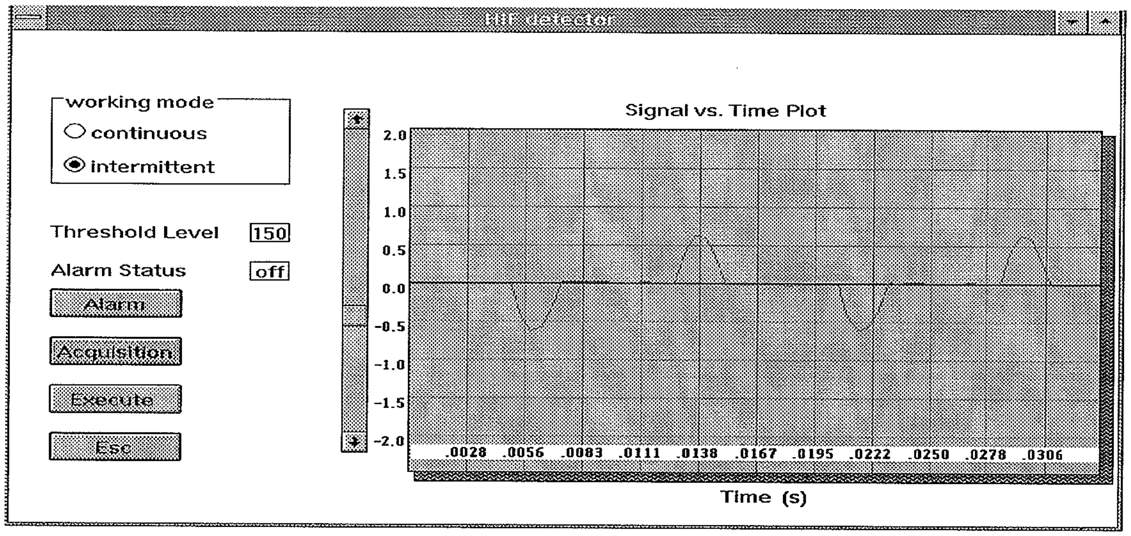
Threshold Level 150

Alarm Status off

Fig. 29 Windows Output of Sinusoidal Waveform with DC Offset

5.2 Windows Output of a Typical Computer Current Load

Since there are no Flicker, Asymmetry, and Quarter Cycle features on the computer waveform as shown in Fig. 30, the windows output is secure as shown in Fig. 31.

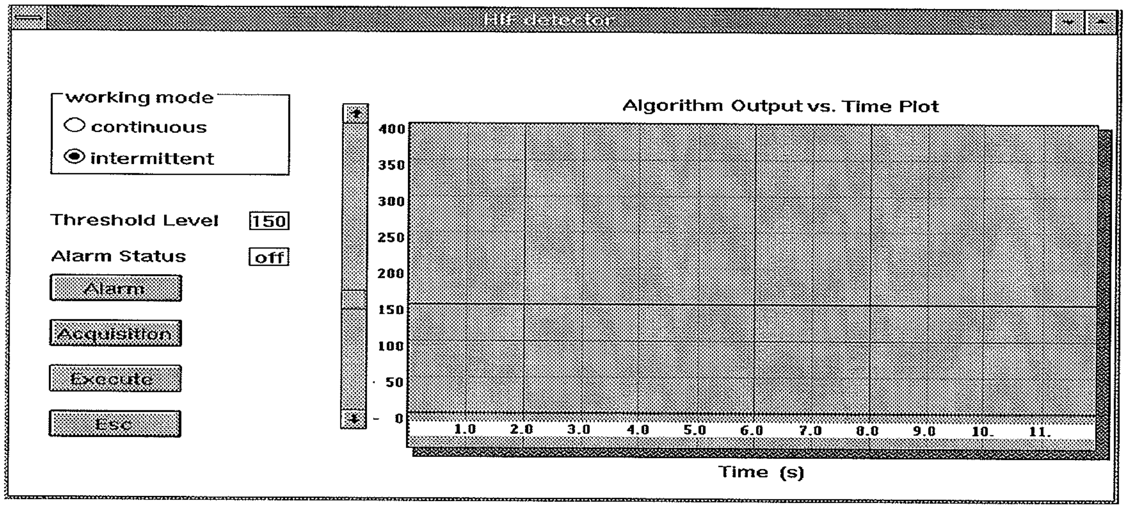


working mode
continuous [✓ intermittent]

Threshold Level 150

Alarm Status off

Fig. 30 A Typical Computer Current Waveform



working mode
continuous [✓ intermittent]

Threshold Level 150

Alarm Status off

Fig. 31 Windows Output for a Computer Current Waveform

5.3 Windows Outputs of Faults on Wet Soil

As explained previously, the Quarter–Cycle Asymmetry part of the algorithm will be effective, the waveform and windows output are shown as in Fig. 32 and Fig. 33.

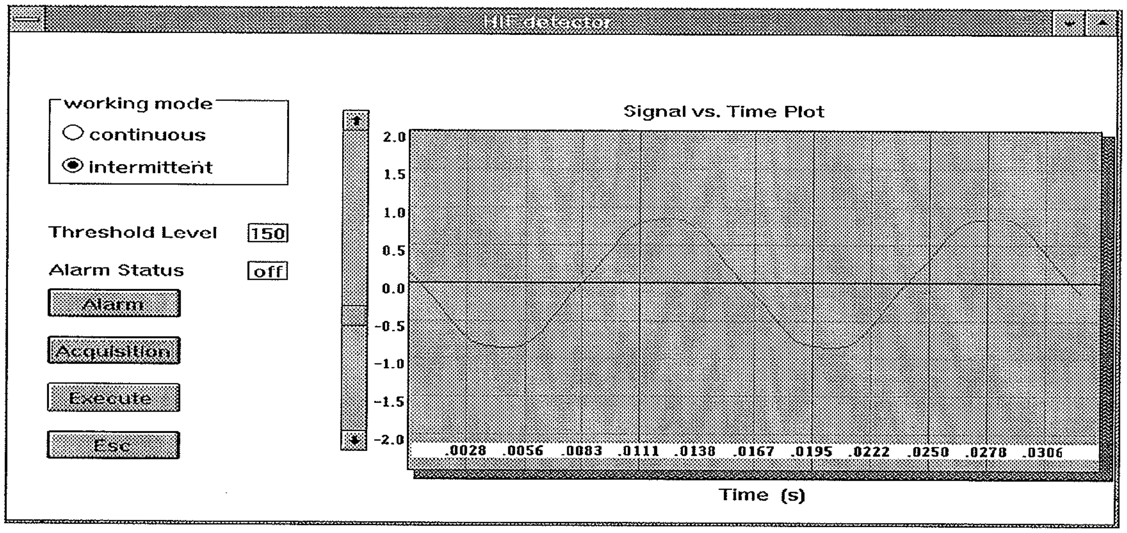


Fig. 32 Current Waveform of Fault on Wet Soil–1

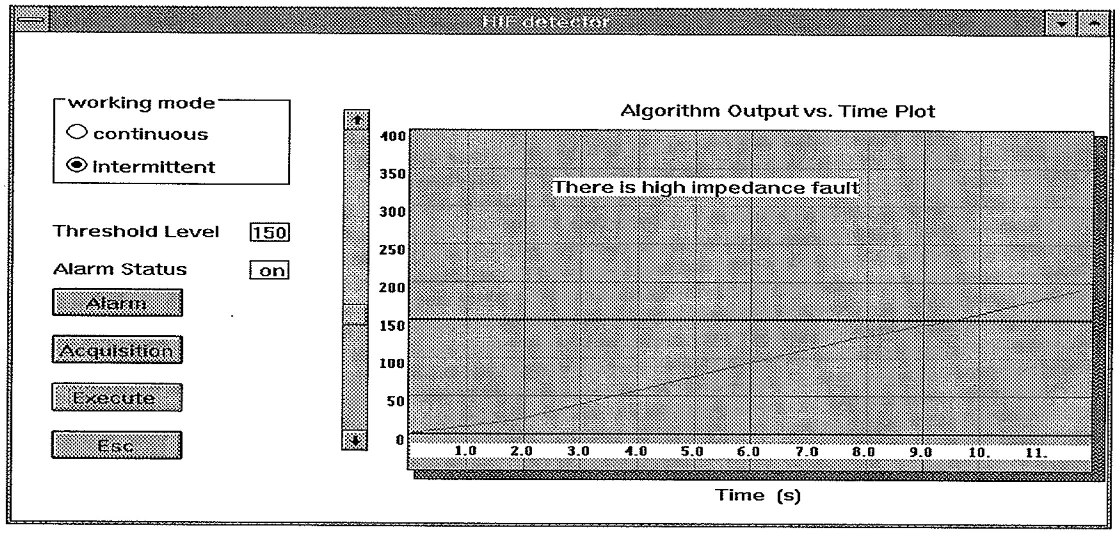
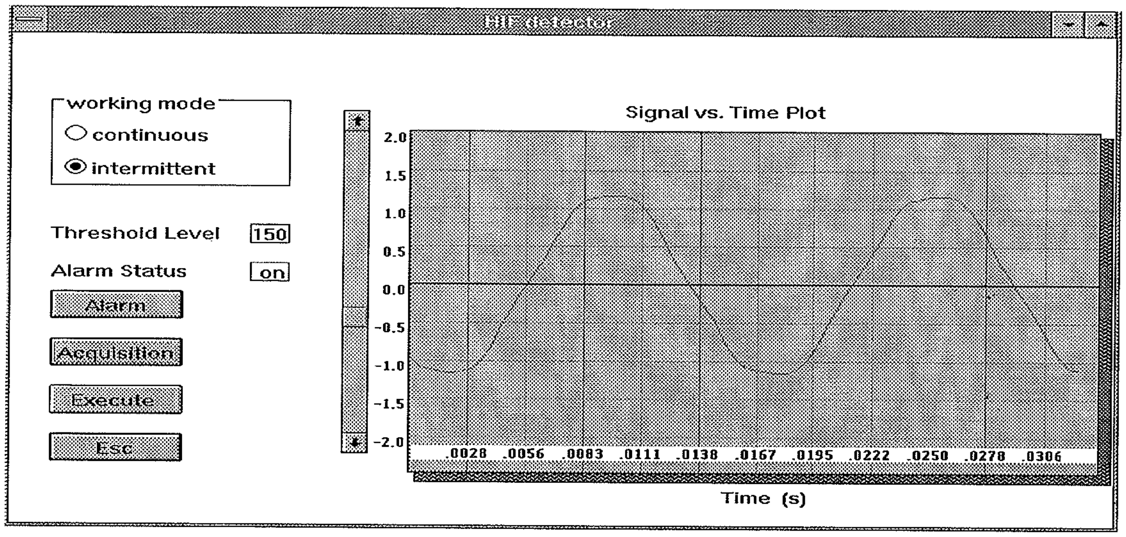


Fig. 33 Windows Output of Fault on Wet Soil–1

Figure 34 and Figure 35 show a different case of fault on wet soil: fault current waveform and windows output respectively. The result is similar to the previous one.

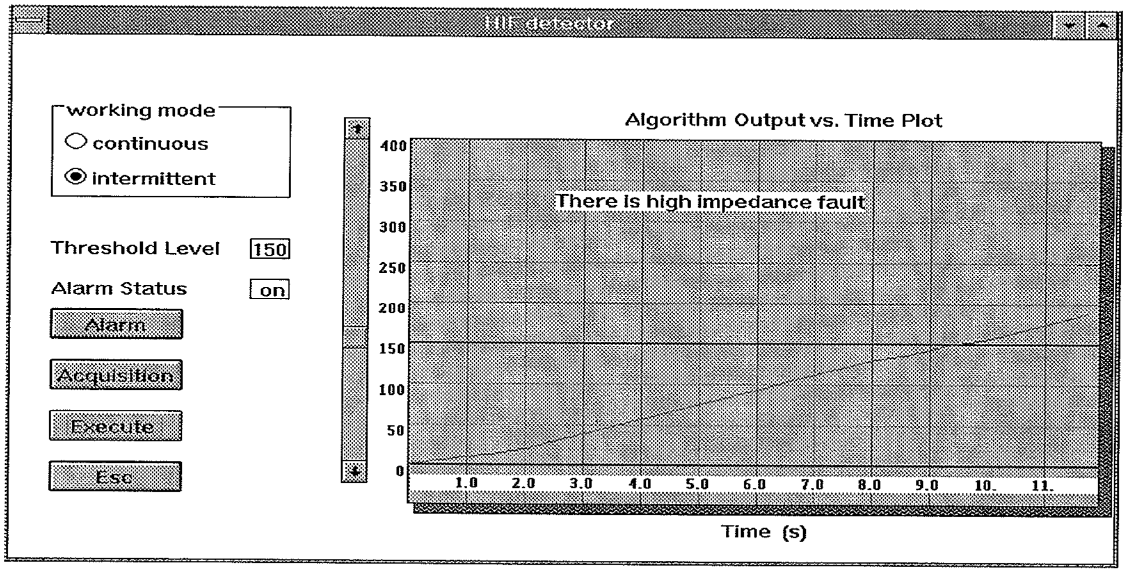

HIF detector

working mode
continuous
[✓ intermittent]

Threshold Level 150

Alarm Status on

Fig. 34 Current Waveform of Fault on Wet soil–2


HIF detector

working mode
continuous
[✓ intermittent]

Threshold Level 150

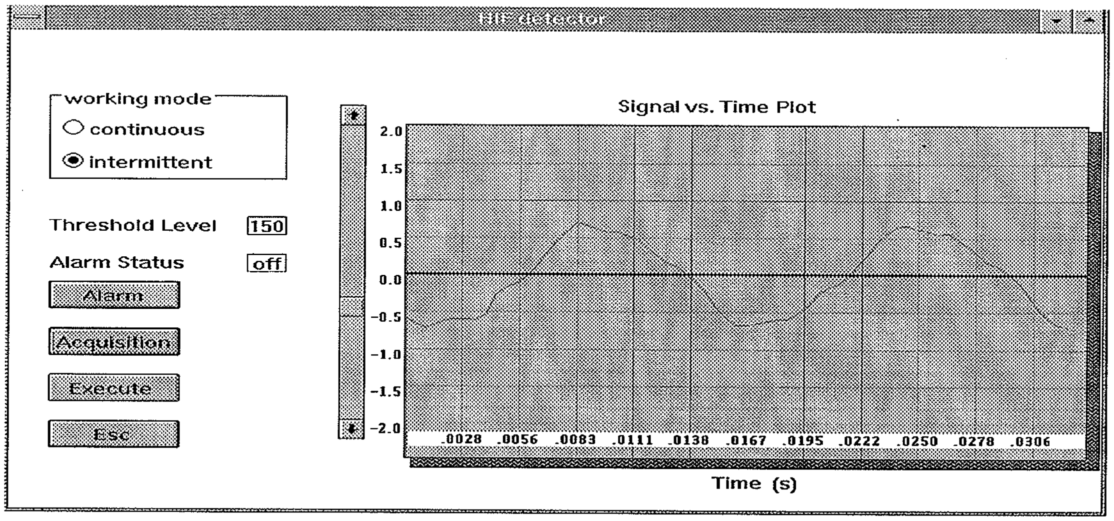
Alarm Status on

There is high impedance fault

Fig. 35 Windows Output of Fault on Wet Soil–2

5.4 *Windows Outputs of Faults on Dry Soil*

As expected, these outputs reach the threshold level faster than those of faults on wet soil, since three parts of the algorithm are effective. Figure 36 to Figure 39 show two examples.

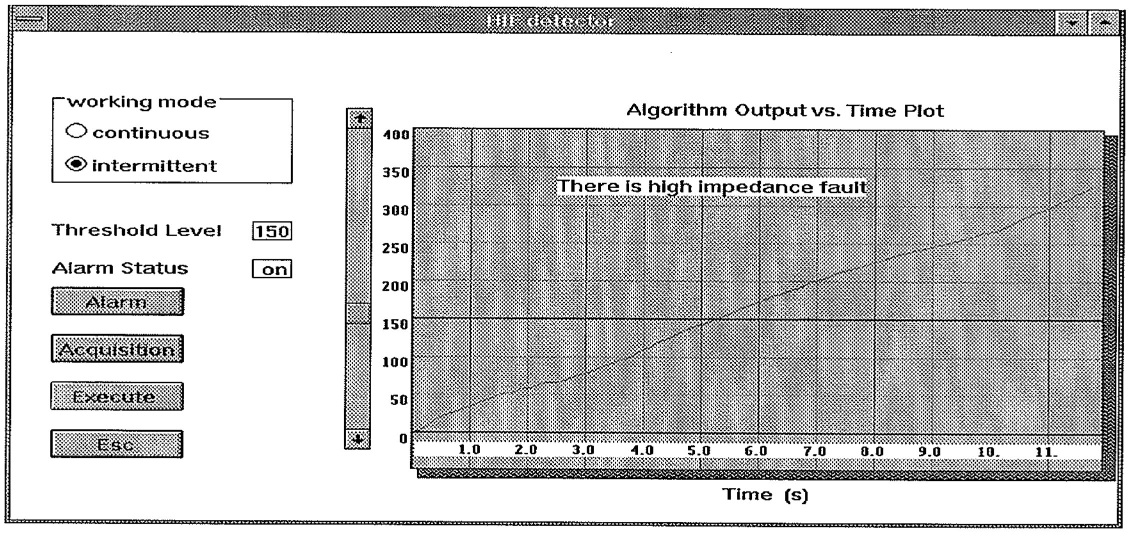


working mode
○ continuous
◉ intermittent

Threshold Level 150

Alarm Status off

Fig. 36 Current Waveform of Fault on Dry Soil–1



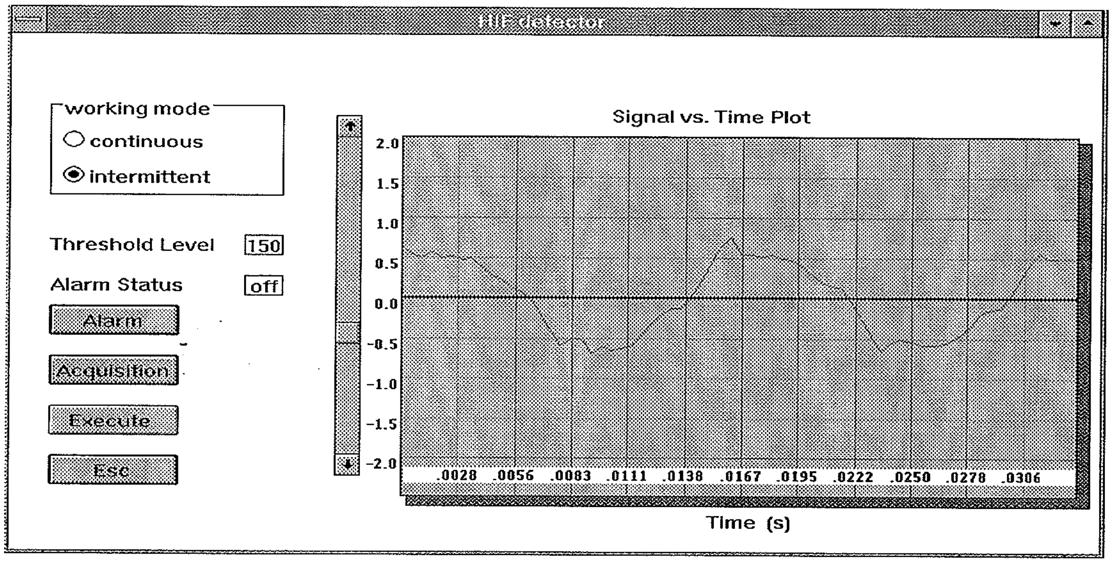
working mode
○ continuous
◉ intermittent

Threshold Level 150

Alarm Status on

There is high impedance fault

Fig. 37 Windows Output of Fault on Dry Soil–1

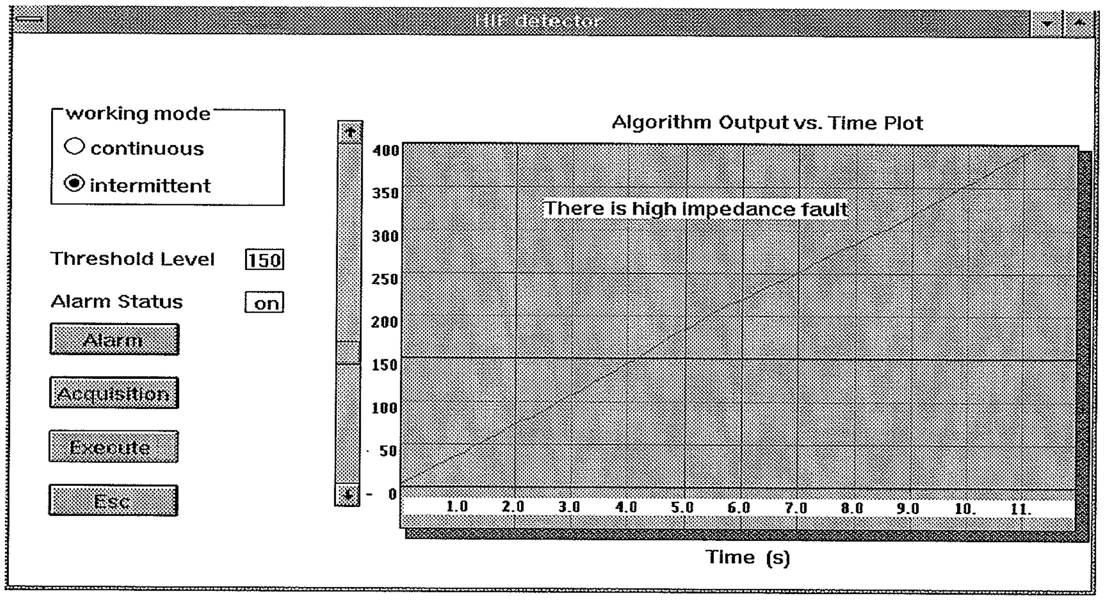

HIF detector

working mode
○ continuous
● intermittent

Threshold Level 150

Alarm Status off

Fig. 38 Current Waveform of Fault on Dry Soil–2


HIF detector

working mode
○ continuous
● intermittent

Threshold Level 150

Alarm Status on

There is high impedance fault

Fig. 39 Windows Output of Fault on Dry Soil–2

5.5 Windows Outputs of Faults on Grassy Wet Soil

In general, current waveforms of faults on grassy wet soil have more ripples than those on pure wet soil. Quarter Cycle Asymmetry is still an obvious characteristic of the faults. Figures 40 to 43, which are in following two pages, show that the Windows program works very well in identifying faults on grassy wet soil as well as on pure wet soil.

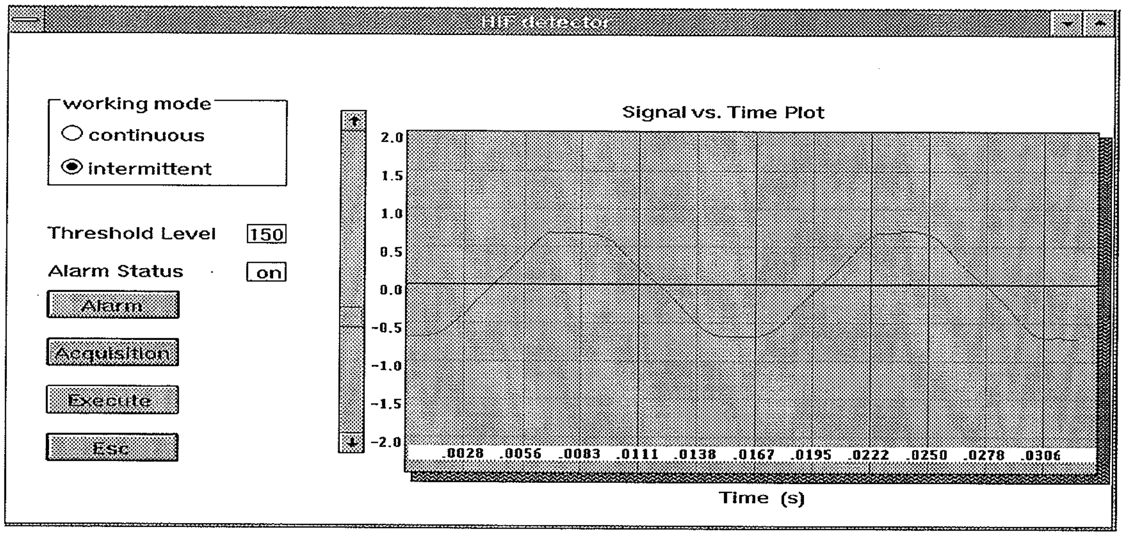


Fig. 40 Current Waveform of Fault on Grassy Wet Soil–1

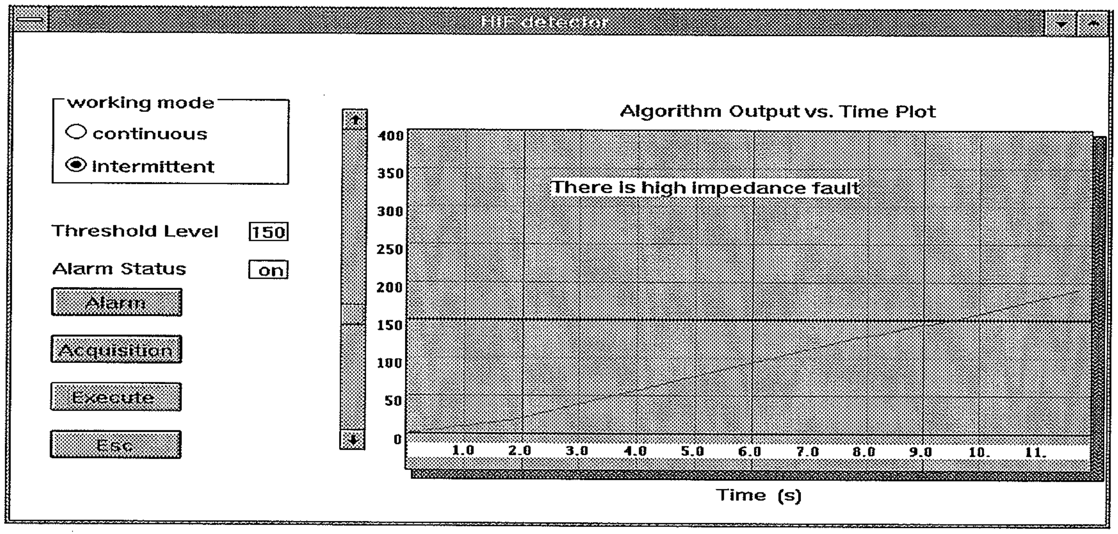
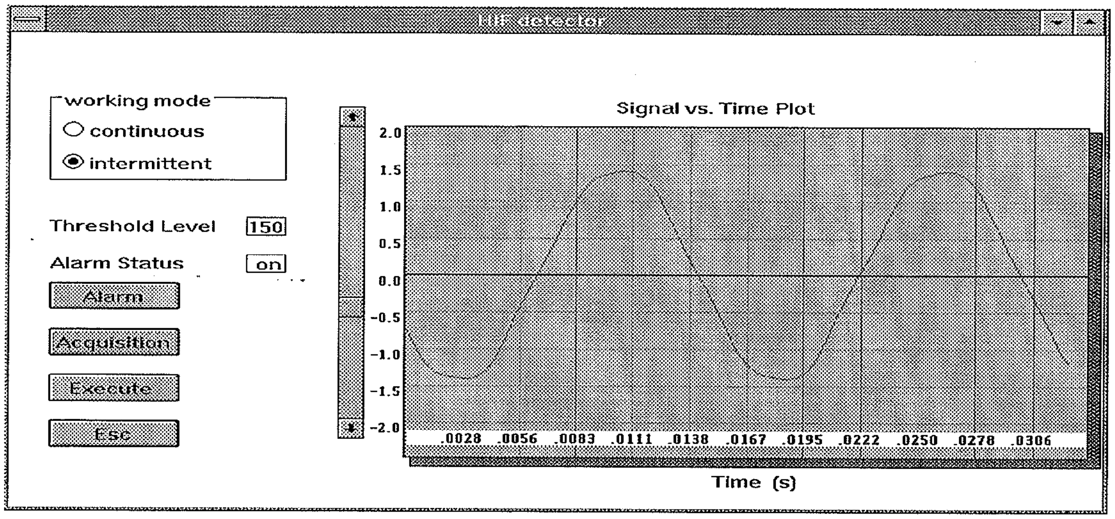


Fig. 41 Windows Output of Fault on Grassy Wet Soil–1

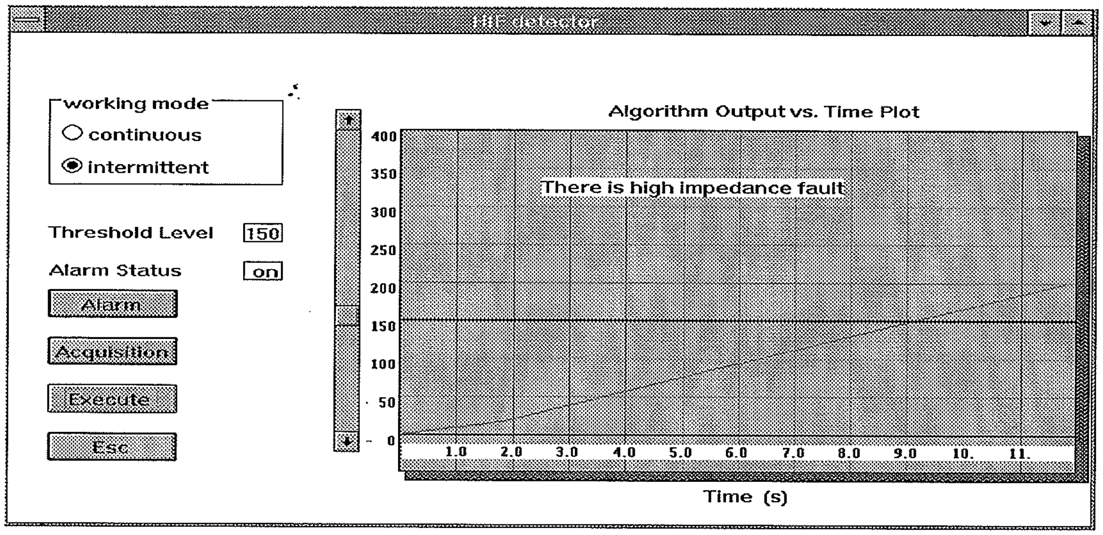

HIF detector

working mode
○ continuous
● intermittent

Threshold Level 150

Alarm Status on

Fig. 42 Current Waveform of Fault on Grassy Wet Soil–2


HIF detector

working mode
○ continuous
● intermittent

Threshold Level 150

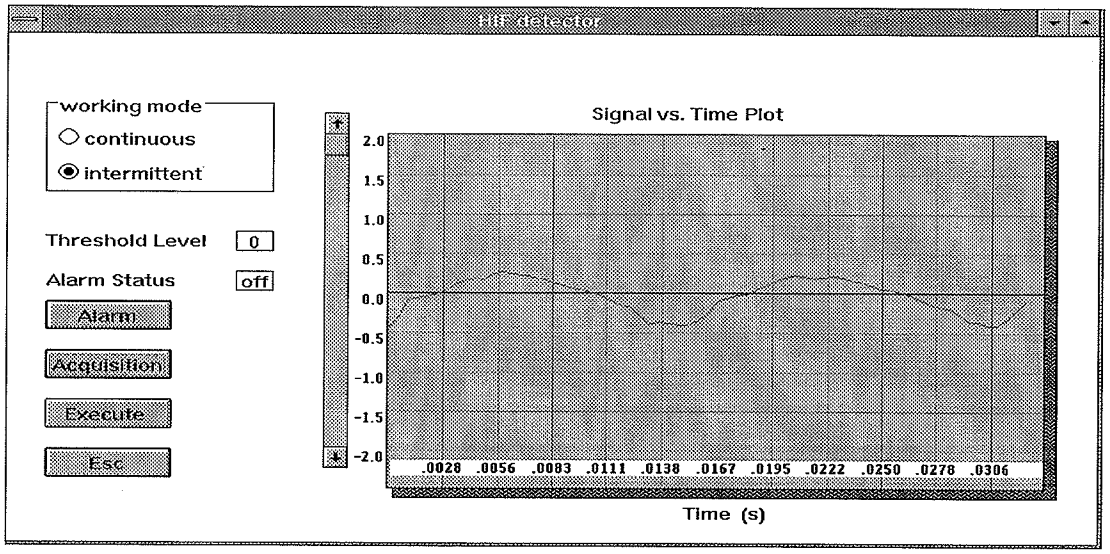
Alarm Status on

There is high impedance fault

Fig. 43 Windows Output of Fault on Grassy Wet Soil–2

5.6 Windows Outputs of Faults on Grassy Dry Soil

Finally, Figures 44 to 47 show the performances of the Windows program when faults happen on grassy dry soil. Obviously, the results are satisfactory.

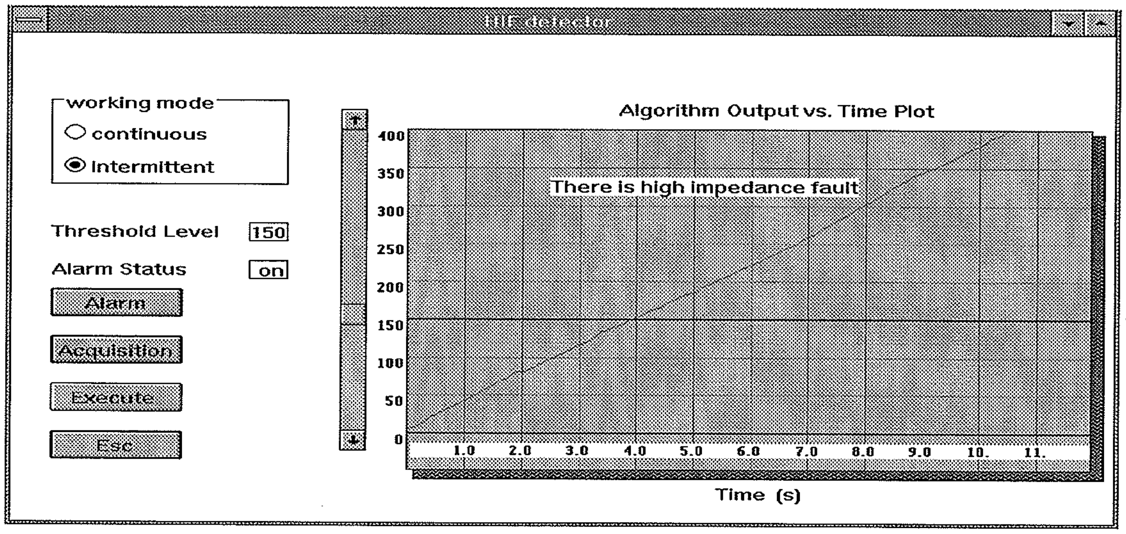


working mode: continuous [✓ intermittent]

Threshold Level 0

Alarm Status off

Fig. 44 Current Waveform of Fault on Grassy Dry Soil–1



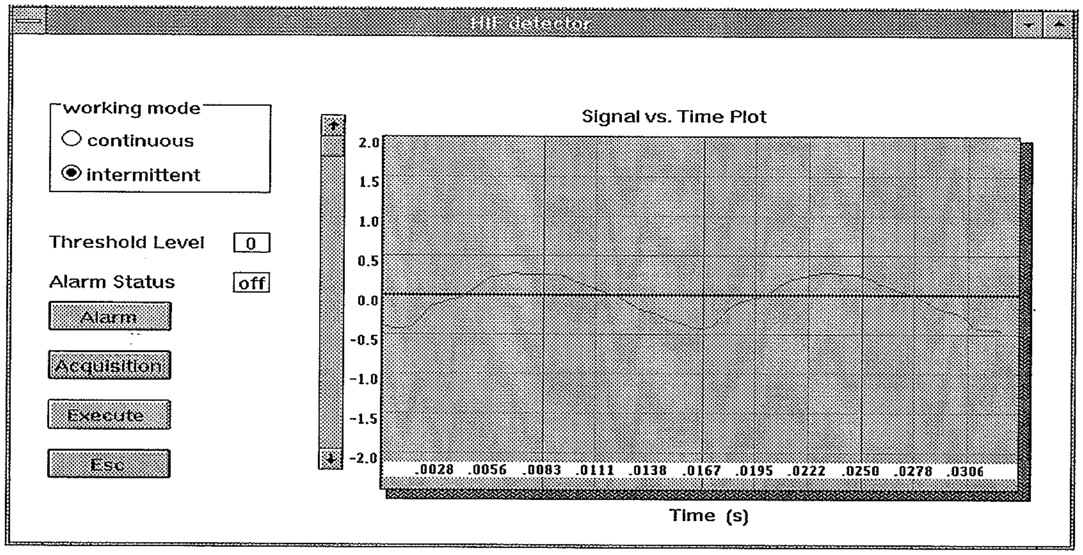
working mode: continuous [✓ intermittent]

Threshold Level 150

Alarm Status on

There is high impedance fault

Fig. 45 Windows Output of Fault on Grassy Dry Soil–1

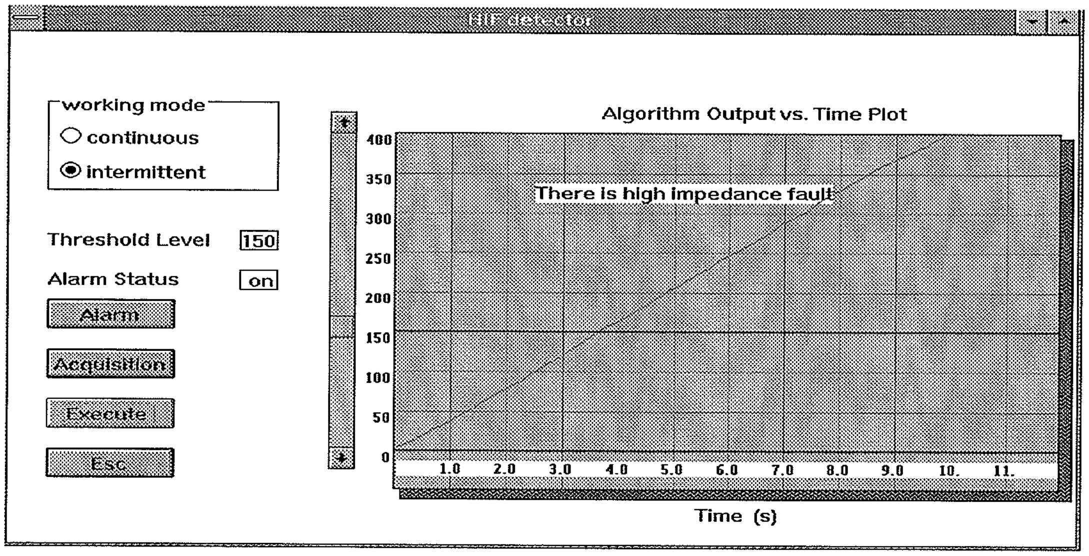


working mode
○ continuous
◉ intermittent

Threshold Level 0

Alarm Status off

Fig. 46 Current Waveform of Fault on Grassy Dry Soil–2



working mode
○ continuous
◉ intermittent

Threshold Level 150

Alarm Status on

There is high impedance fault

Fig. 47 Windows Output of Fault on Grassy Dry Soil–2

CHAPTER 6 CONCLUSIONS AND FUTURE WORK

6.1. Evaluation of the Circuit, Algorithm and Program

A HIF simulation circuit was set up in the laboratory at the University of Manitoba. The fault current proved to be a credible fault current source for simulating HIF in a power distribution system since the fault current waveforms showed largely resistive and nonlinear V–I characteristics.

The algorithm proposed for detecting HIF includes three parts: Flicker, Asymmetry and Quarter–Cycle Asymmetry as indications of fault current waveforms. This algorithm performed well in identifying the characteristics of the HIF. It guaranteed ***dependability*** (ability to trip when it should) when detecting high impedance faults. In addition, for loads similar to high impedance faults (HIFLL), it also gave quite satisfactory results when checking ***security*** *(* ability to not trip when it should not).

In terms of the circuit and the algorithm, a DOS based program was written using C++ language to realize the algorithm. All the results showed good performance on both HIF and HIFLL, meaning the program matched the algorithm. Due to having too many manual operations needed during the data acquisition procedure and the lack of automatic control ability, the program needed to be improved.

6.2. Advantages of Using Windows Program

To overcome the disadvantages already mentioned, the program based on DOS was developed into a Windows program.

The Windows program, the data acquisition board, and the software together make up the **High Impedance Fault Detector**. Besides ***dependability*** and ***security***, the **Detector** has the following advantages over the program which was developed based on DOS :

1. It can run intermittently and continuously. It is very convenient for customers to use.

2. In the Intermittent working mode, it can send 15000 data points to the Windows program which makes the final decision and sends a signal out to represent either trip or no trip.

3. In the Continuous working mode, the detector can run continuously and automatically. Every 20 seconds, it will repeat the whole procedure and send either a five–volt or zero–volt signal out according to a fault or no fault situation. It will continuously run forever until terminated by the operator.

With the development of computer technology, the detector could be improved further totally depending on the customers' requirements and how deeply the customers want to investigate the area in detecting HIF.

6.3. Future Work

6.3.1. Security Check for Fluorescent Light Loads

It is known, a fluorescent light load should be considered as a normal load. The HIF detector should definitely not trip this kind of load . As explained by Sultan [1], a fluorescent load has a nonlinear V– I characteristic. It will be necessary to check the detector to guarantee security for this load in future.

6.3.2. Field Test

For various reasons, the detector has not been tested in the field. To work practically as expected, it should be first installed onto a power distribution line system to field test.

BIBLIOGRAPHY

[1] Ahmad Fathi Sultan, " High Impedance Arcing Faults Detection Using An Artificial Neural Network ", Thesis for Doctor of Philosophy, Department of Electrical and Computer Engineering, the University of Manitoba, Winnipeg, Manitoba ,Canada, February, 1992.

[2] G. W. Swift, "Power Systems Protection Based on Computer Relays", Graduate Course Notes, Department of Electrical and Computer Engineering , the University of Manitoba, Winnipeg, Manitoba ,Canada,1992

[3] "Detection of Downed Conductors on Utility Distribution Systems ". IEEE Tutorial Course No. 90EH0310–3–PWR, IEEE/PES 1990 Summer Meeting, July 15–19,1990.

[4] B.M. Aucoin, B.D. Russell, C.L. Benner. "High Impedance Fault Detection for Industrial Power Systems". IEEE Industrial Application Society Conference, San Diego, October,1989.

[5] J. Carr. "Detection of High Impedance Faults on Multi–Grounded Primary Distribution System". IEEE Transactions on Power Apparatus and Systems, Vol. PAS–100, No. 4, April 1981, pp. 2008–2016.

[6] B.M. Aucoin, B.D. Russell. "Distribution High Impedance Fault Detection Utilizing High Frequency Current Components ". IEEE Transactions on Power Apparatus

and Systems, Vol.PAS–101, No.6, June 1982, pp. 1596–1606.

[7] B.M. Aucoin, J.Zeigler, B.D. Russell. " Feeder Protection and Monitoring System, Part I: " Design, Implementation and Testing". IEEE Transactions on PAS, Vol. PAS–104,No. 4, April 1985, pp. 873–880.

[8] B.M. Aucoin, J.Zeigler, B.D. Russell. " Feeder Protection and Monitoring System, Part II: "Staged Fault Test Demonstration". IEEE Transactions on PAS, Vol.PAS–104,No. 6, June 1985, pp. 1456–1462.

[9] B.M. Aucoin, B.D. Russell. "Detection of Distribution High Impedance Faults Using Burt Noise Signals Near 60 Hz". IEEE Transactions on Power Delivery, Vol. PWRD–2, No. 2, April 1987, pp. 342–348.

[10] D. I Jeerings, J.R. Linders. " Ground Resistance Revised ". IEEE Transactions on Power Delivery, Vol. PWRD–4, No. 2, April 1989, pp. 949–956.

[11] D. I Jeerings, J.R. Linders. " Unique Aspects of Distribution System Harmonics Due to High Impedance Ground Faults ". IEEE Transactions on Power Delivery, Vol. PWRD–5, No. 2, April 1990, pp. 1086–1094.

[12] D. I Jeerings, J.R. Linders. " Discussion: IEEE Tutorial Course; Detection of Downed Conductors on Utility Distribution Systems ". IEEE/PES Winter Meeting, Atlanta,GA, February 8, 1990.

[13] A. F. Sultan, G. W. Swift. Discussion: " High Impedance Fault Detection Utilizing Incremental Variance of Normalized Even Order Harmonic Power ". IEEE Transactions on Power Delivery, Vol. PWRD–6, No. 2, April 1991. pp. 564.

[14] A. F. Sultan, G. W. Swift. " Security Testing of High Impedance Fault Detectors ".

IEEE/WESCANEX May 29 & 30 1991, Regina, Saskatchewan, CANADA.

[15] A. F. Sultan, G. W. Swift, D. J. Fedirchuk. " Detection of High Impedance Arcing Faults Using a Multi–Layer Perception". IEEE/PES Winter Meeting, New York, N. Y., Jan. 1992.

[16] "AT–MIO–16 User Manual " , NATIONAL INSTRUMENTS, October 1993 Edition, Part Number 320476–01.

[17] "NI–DAQ Software Reference Manual for DOS", NATIONAL INSTRUMENTS, October 1993 Edition, Part Number 320498–01.

[18] "NI–DAQ Function Reference Manual for DOS", NATIONAL INSTRUMENTS, October 1993 Edition, Part Number 320499–01.

[19] "Measure for Lotus 1–2–3 , Data Acquisition Module Reference", NATIONAL INSTRUMENTS, August 1989 Edition, Part Number 320195–01.

[20] "EMTDC User's Manual", Version 3, Manitoba HVDC Research Center, Winnipeg, Manitoba Canada,1988.

[21] "C PROGRAMMING", Steven Holzner with The Peter Norton Computing Group, 1991.

[22] "C++ PROGRAMMING", Steven Holzner with The Peter Norton Computing Group, 1992.

[23] "DEVELOPING WINDOWS APPLICATIONS WITH BORLAND C++ 3.1, Second Edition". James W. McCord, 11711 North College, Carmel, Indiana 46032 U.S.A., 1992.

[24] "Windows 3.1 Programmer's Reference". James W. McCord, 11711 North College,

Carmel, Indiana 46032 U.S.A., 1992.

APPENDIX A

NI—DAQ AT–MIO–16 Board and Its Specifications

This section describes the AT–MIO–16; lists the contents of the AT–MIO–16 kit; describes the optional software and equipment; and explains how to unpack the AT–MIO–16.

1. About the AT–MIO–16

The AT–MIO–16 is a high–performance, software–configurable 12–bit DAQ board for laboratory, test and measurement, and data acquisition and control applications. The board performs high–accuracy measurements with high–speed settling to 12 bits, noise as low as 0,1 LSBrms, and a typical DNL of +/–0,5 LSB. Because of its FIFOs and dual –channel DMA, the AT–MIO–16 can achieve high performance. even when used in environments that may have long interrupt latencies such as Windows.

A common problem with DAQ boards is that you can not easily synchronize several measurement functions to a common trigger or timing event. The AT–MIO–16 has the Real Time System Integration (RTSI) bus to solve this problem. The RTSI bus consists of our custom RTSI bus interface chip and a ribbon cable to route timing and trigger signals between several functions on one or DAQ boards in your PC.

The AT–MIO–16 can interface to the Signal Conditioning Extensions for Instrumentation (SCXI) system so that you can acquire over 3000 analog signals from thermocouples, RTDs, strain gauges, voltage sources, and current sources. You can also acquire or generate digital signals for communication and control. SCXI is the instrumentation front–end for plug–in DAQ boards.

2. *What the Kit Should Contain*

Two versions of the AT–MIO–16 are available –one version for each of two gain ranges. The AT_MIO–16L (L stands for low–level signals). The AT–MIO–16H (H stands for high–level signals) has software–programmable gain settings of 1, 2, 4, and 8 for high–level analog input signals. The AT–MIO—16 (L/H) –9 contains an ADC with a 9 micro second conversion time. The AT–MIO–16 (L/H) –9 is capable of data acquisition rates of up to 100 kHz.

Each version of the AT–MIO–16 board has a different part number and kit part number, listed as follows.

Kit Name	**KIt part Number**	**Kit Component**	**Board Part Number**
AT–MIO–16L–9	77625–01	AT–MIO–16L–9 Board	180705–01
AT–MIO–16H–9	77625–11	AT–MIO–16H–9 Board	180705–11

The board part number is printed on your board along the top edge on the component side. You can identify which version of the AT–MIO–16 board you have by looking up the part number in the preceding table.

In addition to the board, each version of the AT–MIO–16 kit contains the following components.

Kit Component	**Part Number**
AT–MIO–16 User Manual	320476–01
NI–DAQ software For PC components, with manuals	776250–01
NI–DAQ Software User Manual for PC Compatibles	320498–01
NI–DAQ Function Reference Manual for PC Compatibles	320499–01

Detailed specifications of the At–MIO–16 are listed in Specifications.

3. *Software programming Choices*

There are four options to choose from when programming your National Instruments Plug–in data acquisition board and SXCI hardware.

4. *LabVIEW and LabWindows Applications Software*

LabVIEW and LabWindows are innovative program development software package for data acquisition and control applications. LabVIEW uses graphical programming, whereas LabWindows enhances traditional programming languages. Both packages include extensive libraries for data acquisition, instrument control, data analysis, and graphical data presentation.

LabVIEW currently runs on three different platforms–AT/MC/EISA computers running Microsoft Windows, the Macintosh platform, and the Sun SPARC station platform. LabVIEW features interactive graphics, a state–of–the–art user interface, and a powerful graphical programming language. The LabVIEW Data Acquisition VI Library, a series of VIs for using LabVIEW with National Instruments boards, is included with LabVIEW. The

LabVIEW Data Acquisition VI Libraries are functionally equivalent to the NI–DAQ software.

LabWindows has two versions–LabWindows for DOS is for use on PCs running DOS, and LabWindows/CVI is for use on PCs running Windows and Sun SPARC stations. LabWindows /CVI features interactive graphics, a state–of–the–art user interface, and uses the ANSI standard C programming language. The LabWindows Data Acquisition Library, a series of functions for using LabWindows for DOS and LabWindows with National Instruments Boards, is included with LabWindows for DOS and LabWindows /CVI. The LabWindows Data Acquisition libraries are functionally equivalent to the NI–DAQ software.

Using LabWindows or LabWindows software will greatly diminish the development time for your data acquisition and control application. Part numbers for these software products are as follows:

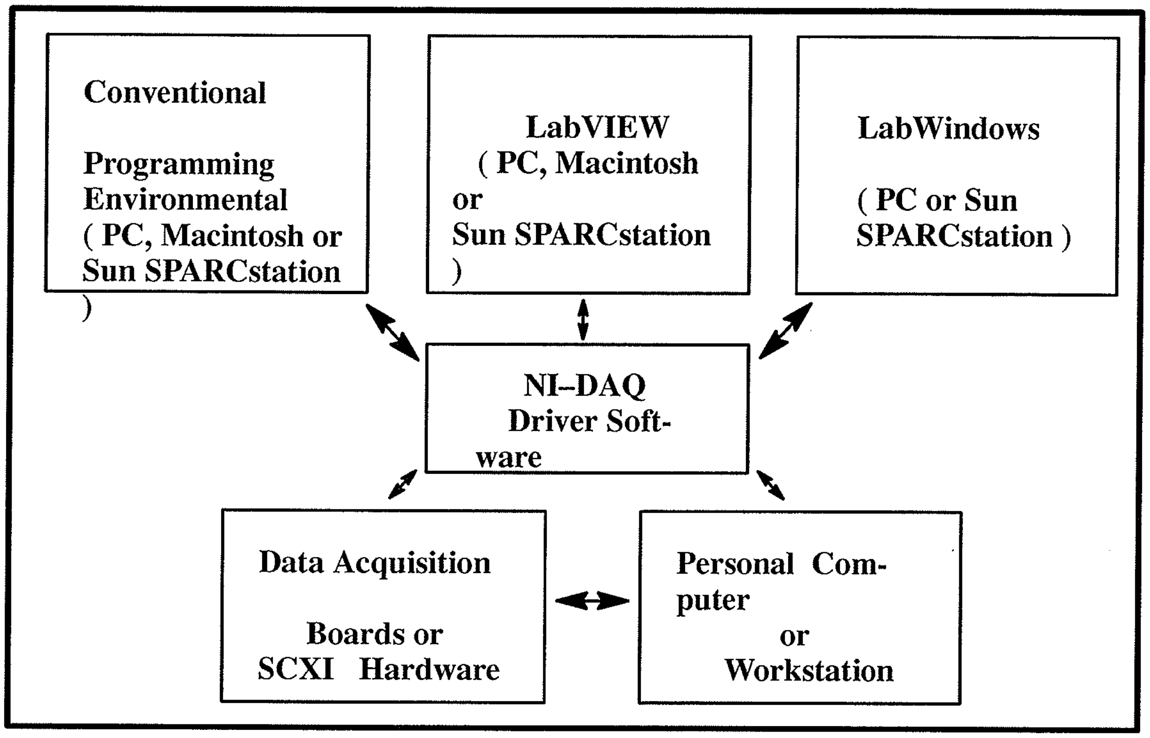
Software	Part Number
LabVIEW for Windows	776670–01
LabVIEW for Macintosh	776141–01
LabVIEW for Sun	776680–01
LabWindows for DOS	776475–01
LabWindows/CVI for Windows	776800–01
LabWindows/CVI For Sun	776820–031

5. *NI–DAQ Driver Software*

The NI–DAQ Driver software has an extensive library of functions that you can call from your application programming environment. These functions include routines for analog input (A/D conversion), buffered data acquisition (high–speed A/D conversion), analog output (D/A conversion), waveform generation, digital I/O, counter/timer operations, SCXI, RTSI, selfcalibration, messaging, and acquiring data to extended memory.

The NI–DAQ also internally addresses many of the complex issues between the computer and the plug–in board such as programming interrupts and DMA controllers. NI–DAQ maintains a consistent software interface among its different versions so that you can change platforms with minimal modifications to your code. The following block diagram illustrates the relationship between NI–DAQ and LabVIEW and LabWindows. You can see that the data acquisition parts of LabVIEW and LabWindows are functionally equivalent to the NI–DAQ software.

The National Instruments PC, AT, and MC Series data acquisition boards are packaged with NI–DAQ software for PC compatibles. NI–DAQ software for PC compatibles comes with language interfaces for Professional BASIC, Turbo Pascal, Turbo C, Turbo C++, Borland C++, and Microsoft C for DOS; And Visual Basic, Turbo Pascal, Microsoft C with SDK, and Borland C++ for Windows. You can use your AT–MIO–16, together with other PC, AT, and MC Series data acquisition Boards and SCXI hardware, with NI–DAQ software for PC compatibles.


Conventional
Programming Environmental (PC, Macintosh or Sun SPARCstation)
LabVIEW (PC, Macintosh or Sun SPARCstation)
LabWindows (PC or Sun SPARCstation)
NI–DAQ Driver Soft- ware
Data Acquisition Boards or SCXI Hardware
Personal Com- puter or Workstation

The National Instruments NB Series data Acquisition Boards are packaged with NI–DAQ software for Macintosh. NI–DAQ software for Macintosh comers with language interfaces for MPWC, THINK C, Pascal, and Microsoft QuickBASIC. Any Language that uses Device Manager Toolbox calls can access NI–DAQ software for Macintosh. You can use NB Series data acquisition Boards and SCXI hardware with NI–DAQ software for Macintosh.

The National Instruments SB Series data acquisition Boards are packaged with NI–DAQ software for Sun, which comes with a language interface for ANSIC.

6. Register–Level Programming

The final option for programming any National Instruments data acquisition hardware is to write register–level software. Writing register–level programming software can be very time consuming and inefficient, and is not recommended for most users. The only users who should consider writing register–level software should meet at least one of the following criteria:

* National Instruments does not support your operating system or programming language.

* You are an experienced register–level programmer who is more comfortable writing your own register–level software.

Even if you are an experienced register–level programmer, consider using NI–DAQ, LAbVIEW, or LabWindows to program your National Instruments data acquisition hardware. Using the NI–DAQ, LabVIEW, or LabWindows software is easier than, is as flexible as, and can save weeks of development time over register–level programming.

The AT–MIO–16 User Manual contains complete instructions for programming your data acquisition board with NI_DAQ, LabVIEW, or LabWindows. If you are using NI–DAQ, LabVIEW, or LabWindows to control your board, you should not need the register–level programmer manual. The AT–MIO–16 Register–Level Programmer Manual contains programming details, such as register maps, bit descriptions, and register programming hints that you will need only for register–level programming. Some hardware user manuals include register map description s and register programming hints.

7. Unpacking

Your AT–MIO–16 boards is shipped in an antistatic package to prevent electrostatic damage to the board. Electrostatic discharge can damage several components on the board. To avoid such damage in handling the board, take the following precautions:

* Touch the antistatic package to a metal part of your computer chassis before

removing the board from the package.

* Remove the board from the package and inspect the board for loose components or any other sign of damage. Notify National Instruments if the board appears damaged in any way. Do not install a damaged board into your computer.

8. *AT–MIO–16 Specifications*

This part lists the specifications for the AT–MIO–16. These specifications are typical at 25 degree C unless otherwise noted.

Analog Input

Input Characteristics

Number of channels	16 single–ended or 8 differential, jumper–selectable
Type of ADC	Sampling, successive approximation
Resolution	12 bits, 1 in 4096
Max sampling rate	100 KS/s
Input signal ranges	
AT–MIO–16H and AT–MIO–16DH	
AT–MIO–16L and AT–MIO–16DL	
Input coupling	DC
Max Working voltage (signal + common mode)	Each input should remain within 12 V of AIGND

Board Gain **(Software Selectable)**	**Board Range (Jumper Selectable)**		
	+/– 10 V	**+/– 5 V**	**0 to 10 V**
1 2 4 8	+/– 10 V +/– 5 V +/– 2.5 V +/– 1.25 V	+/– 5 V +/– 2.5 V +/– 1.25 V +/– 0.63 V	0 to 10 V 0 to 5 V 0 to 2.5 V 0 to 1.25 V

Board Gain **(Software Selectable)**	**Board Range (Jumper Selectable)**		
	+/– 10 V	**+/– 5 V**	**0 to 10 V**
1 10 100 500	+/– 10 V +/– 1 V +/– 0.1 V +/– 0.02 V	+/– 5 V +/– 0.5 V +/– 0.05 V +/– 0.01 V	0 to 10 V 0 to 1 V 0 to 0.01 V 0 to 0.02 V

Overvoltage protection	+/– 35 V powered on, +/– 20 V powered off
Inputs protected	ACH < 0..15 >
FIFO buffer size	16 samples
Data transfers	DMA, interrupts, programmed I/O
DMA modes	Demand

Transfer Characteristics

Relative accuracy	+/–0.9 LSB typical, +/–1.5 LSB max
DNL	+/–0.50 LSB typical, +/–0.95 LSB max
No missing codes	12 bits, guaranteed
Offset error	
Pregain error after calibration	+/–2.44 Micro V (–L board)
Pregain error before calibration	+/–153 Micro V (–H board)
Postgain error after calibration	+/–1.22 mV max
Postgain error before calibration	+/–85 V max
Gain error (relative to calibration reference)	
After calibration	0.0244% of reading (244 ppm) max
Before calibration	0.85% of reading (8500 ppm) max
Gain =! 1 with gain error adjusted to 0	
at gain = 1	0.02% of reading (200 ppm) max

Amplifier Characteristics

Input impedance	1 G Ohm in parallel with 50 pF
Input bias current	+/–25 nA
Input offset current	+/–15 nA
CMRR	

Dynamic Characteristics

Bandwith

Small signal (–3 dB)

Settling time to full–scale step

Gain	**CMRR DC to 100Hz**
1	**75 dB**
10	**95 dB**
100	**105 dB**

650 kHz @ gain =1

Gain	**Accuracy**	
	+/– 0.024% (+/–1 LSB)	**+/– 0.012% (+/– 0.0LSB)**
<= 10	10 Micro s	10 Micro s
100	14 Micro s	14 Micro s
500	47 Micro s	50 Micro s

System noise (including quantization error)

Gain	**20 V Range**	**10 V Range**
<= 10	0.10 LSB rms	0.20 LSB rms
100	0.15 LSB rms	0.20 LSB rms
500	0.30 LSB rms	0.40 LSB rms

Slew rate	5.0 V/micro second

Stability

Recommended warm–up time	15 min
Offset temperature coefficient	

Pregain	6 micro V/C degree
Postgain	160 micro V/C degree
Onboard calibration reference	
Level	2.5V +/– 10 mV
Temperature coefficient	10 ppm/C degree max
Long–term stability	20 ppm/1000 hr

Analog Output

Output Characteristics

Number of channels	2 voltage
Resolution	12 bits, 1 in 4096
Max update rate	250 KS/s
Type of DAC	Double–buffered, mulitiplying
Data transfers	Interrupts, programmed I/O

Transfer Characteristics

Relative accuracy (INL)	
Bipolar range	+/–0.25 LSB typical, +/–0.5 LSB max
Unipolar range	+/–0.50 LSB typical, +/–1.0 LSB max
DNL	+/–0.2 LSB typical, +/–1.0 LSB max
Monotonicity	12 bits guaranteed
Offset error	
After calibration	488 micro V max

Before calibration	+/– 64 mV max
Gain error (relative to internal reference)	
After calibration	+/–0.017% of reading (170 ppm) max
Before calibration	+/–0.77% of reading (7700 ppm) max

Voltage Output

Ranges	+/–10V, 0–10V, jumper selectable
Output coupling	DC
Output impedance	<=0.2 Ohm
Current drive	+/–2 mA max
Protection	Short–circuit protection
Power–on state	Undetermined
External reference input	
Range	+/–10V
Overvoltage protection	+/–25V powered on
Input impedance	11 KOhm

Dynamic Characteristics

Settling time to 0.024% FSR	4 micro second for a 20 V step
Slew rate	30V/micro S
Noise	1 mV rms, DC to 1 MHz

Digital I/O

Number of channels	8 I/O
Compatibility	TTL

Digital logic Levels

Level	Min	Max
Input low voltage	0 V	0.8 V
Input high voltage	2 V	6 V
Input low current (Vin = 0.4 V)		–20 micro A
Input high current (Vin = 2.7 V)		20 micro A
Output low voltage (I_{out} = 24 A)		0.5 V
Output high voltage (I_{out} = –2.6 A)	2.4 V	

Power on state	Configured as input
Data transfers	Programmed I/O

Timing I/O

Number of channels	3 counters/timers, 1 frequency scalers
Resolution	
counter/timers	16 bits
Frequency scalers	4 bits
Compatibility	TTL, pulled high with 4.7 Ohm resistors
Base clocks available	1MHz, 100kHz, 10 kHz, 1kHz, 100Hz
Base clock accuracy	+/–0.01%
Max source frequency	6.897 MHz
Min source pulse duration	70 ns

Min date pulse duration	145 ns
Data transfers	Programmed I/O

Triggers

Digital Trigger

Compatibility	TTL
Response	Falling edge
Pulse width	50 ns min

RTSI

Triggers	7
Bus Interface	Slave

Power requirement

+5 VDC (+/–5%)	1.6 A

Physical

Dimensions	13.3 by 3.9 in. (33.782 by 9.906 cm)
I/O connector	50–pin male ribbon connector
Form factor	AT

Environment

Operating temperature	0 to 70 C degree
Storage temperature	–55 to 150 C degree
Relative humidity	5% to 90% noncondensing

Appendix B.

Data Acquisition and Files Translation

1 Data Acquisition

This section is an overview of the data acquisition procedure using the National Instruments AT–MIO–16 board and the Lotus 1–2–3 data acquisition software package. For detailed descriptions and procedures on using the menu items in the Data Acquisition Module, consult Chapter 4 of 'Data Acquisition Module Command Reference' [17] . The procedure for using the software package includes the following steps:

1) To Load the Data Acquisition Module

Before using the Data Acquisition Module, make sure 1–2–3 is loaded with the data acquisition driver added to the current driver set. Follow the steps listed in ' Installing and Starting the Data Acquisition Module '[18] . Press [ALT]– F8 to display the Data Acquisition main menu.

2) To Set up an Experiment

Begin setting up the experiment, by selecting ID Settings. Use the ID Settings menus to specify where the data comes from(the input/output I/O channels), where to place or read data in the 1–2–3 worksheet, and how to convert the data. Then, go to the Stage–Settings menus to indicate the conditions under which data is to be collected or sent. It includes the acquisition rate and how much data to acquire or send.

3) To Enter Data I/O Information

An ID identifies the source of data I/O. To create an ID, select ID Settings followed by ID. Enter an ID name. Enter the type of the ID(analog– in, digital – in, binary – in, counter, analog – out, digital – out, binary – out), the board, and the channel to associate with. Use the Range menu item to indicate where in the 1–2–3 worksheet to place or read data from the ID. Use Formula to specify a conversion formula for the ID.

4) To Test Data Input

First, select Observe. On the observe screen, data values appear and change as you vary the input. You can compare the raw data values to a known input to determine a conversion formula for each ID. If you associate a formula with an ID, the converted values appear in the Observe screen.

5) To Set Stages

After you specify the active ID and the 1–2–3 worksheet range where to place or read data, switch to the Stage–Settings Sheet. There are three different stages you can set for a data acquisition session. Each stage has an individual set of data I/O, a sample rate, an amount and a trigger. In each of the stages you want to use, select the IDs from which to collect or send data. Then enter a sampling rate and the number of samples to acquire or send during each stage. The Data Acquisition Module collects or sends samples from the designated IDs at the specified rate until the correct number of samples is acquired or sent. Use the Trigger menu item to enter a trigger to start each stage.

6) To Start the Data Acquisition

Select Go from the Data Acquisition Module main menu to start the data acquisition.

The data is placed in the 1–2–3 worksheet. Once data is collected, leave the Data Acquisition Module by selecting Quit.

7) To Save Settings and Data

Save the settings by selecting Name from either the ID – Settings or Stage – Settings menus, choose a name for the worksheet you want to save the data file in and then it is ready to be used later. Now, the data is on the worksheet. The data format must next be considered.

2. *Translation of Worksheet Data File to DOS File*

Suppose everybody already understands the lotus 1–2–3 worksheet. There have been 15000 data items in the data acquisition worksheet as shown in Fig. 48, the worksheet data file should be translated into a suitable DOS file with which it is possible for us to use a C program for further analysis. It should be understood that the worksheet illustrated in Fig. 48 is a special worksheet for data acquisition which uses the 'National Instrument Software Package' ***Measure for lotus 1–2–3*** [19]. If the worksheet could be changed into a Lotus1–2–3 worksheet, it would be easily translated into a DOS file. It was found that the following steps will do the translation of the file.

1) Quit Data Acquisition worksheet and enter the Lotus 1–2–3 main menu.

2) Type Print , File. From the File list, select the name in which the data file was saved. Type Replace.

3) Type Range , Input Range A1 .. E3000 (the range for 15000 data items obtained).

4) Type Go, Quit, Quit, Finally, type Yes.

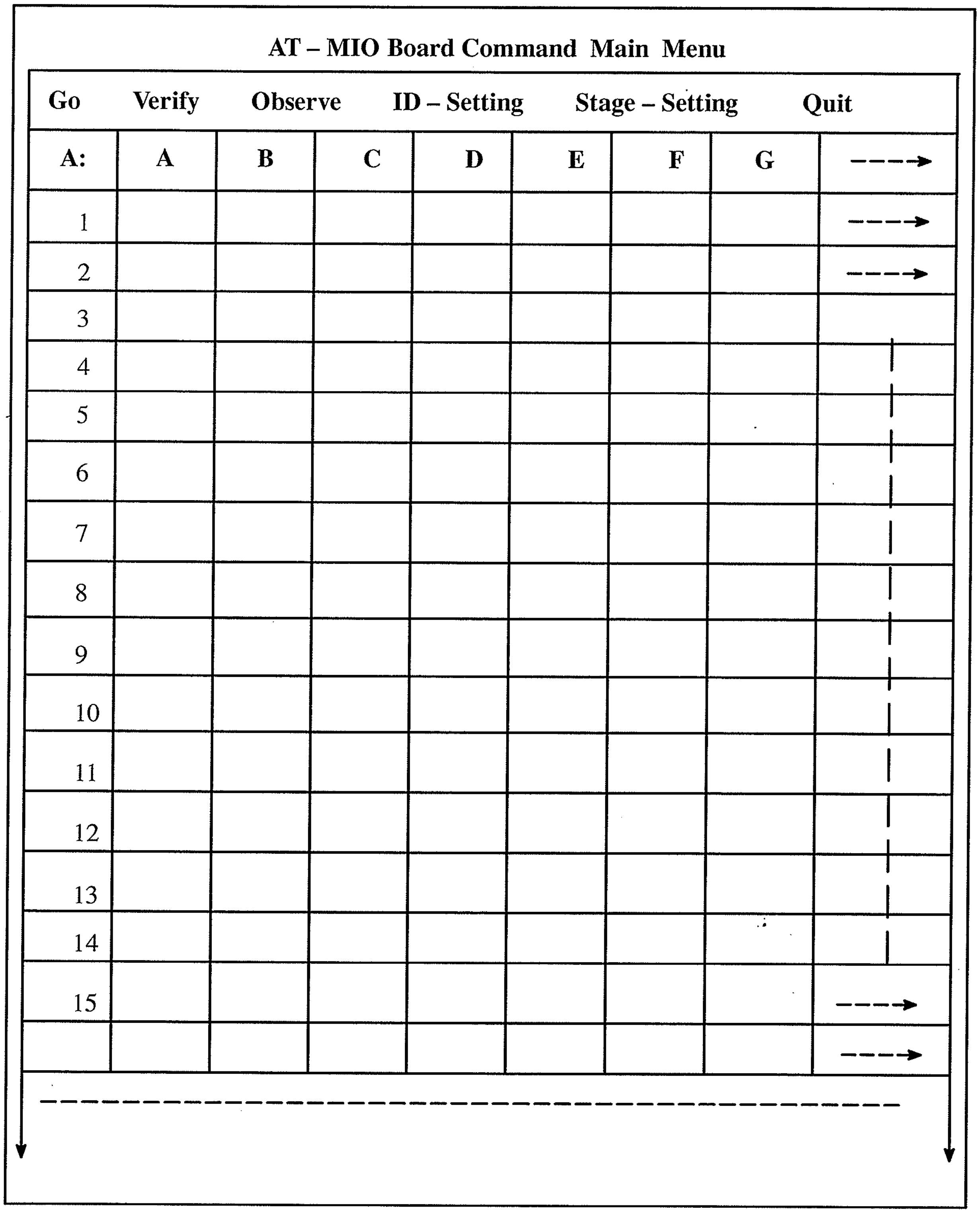


Fig. 48 Lotus 1–2–3 Data Acquisition Worksheet

3. *Translation of DOS File to EMTDC Multiplot File*

Now there is a DOS data file with 5 columns and 3000 rows, the same format as the worksheet. The file format should be changed into a 2 column and 15000 row format file suitable for the EMTDC [20] multiplot software to print. A program (see Appendix C) called TODOS was written to execute this step. Fig. 49 is the flowchart of the program.

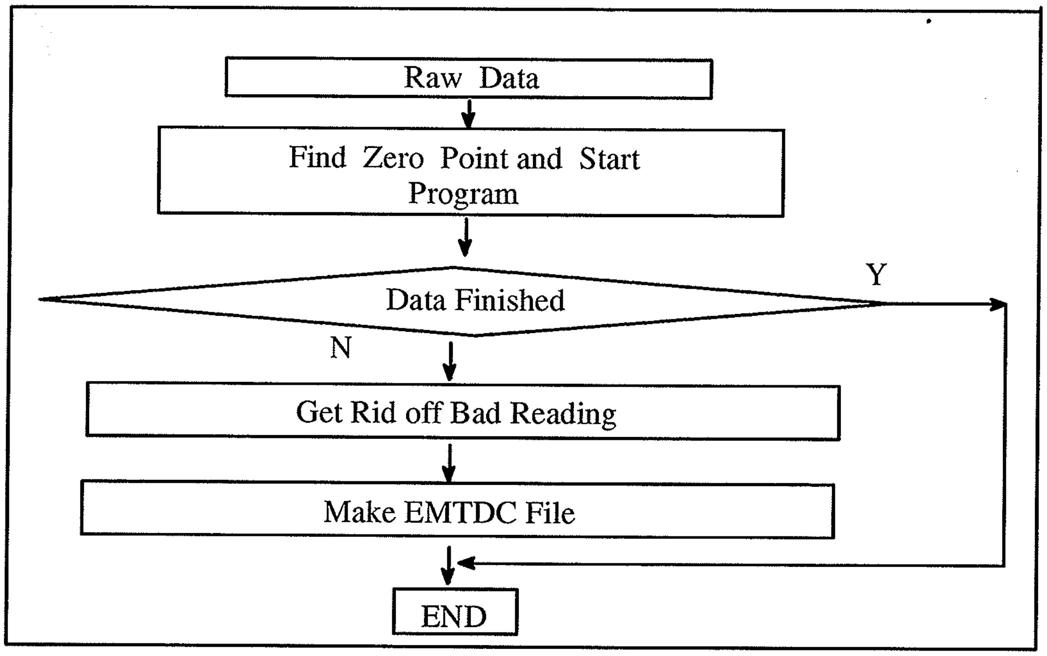


Fig. 49 Flowchart of Program from DOS to EMTDC

Finally, the format of the file is 2 columns, 15000 rows, which is easy for both EMTDC and C program to use. The software procedures which have been finished so far are illustrated in Fig. 50.

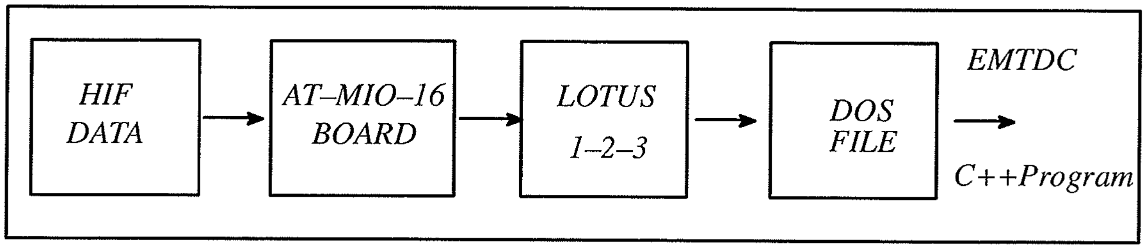


Fig. 50 Data Acquisition Software Blockdiagram

Appendix C.

Listings of Source Program

1 A Windows Program for Detecting High Impedance Fault

2 A C++ Program for Detecting High Impedance Fault

3 A C++ Program of Files Translation

A Windows Program for Detecting High Impedance Faults

Developed by DEHUA ZHENG, Jan, 1995

```
//**************************************************************************
// INCLUDE FILES
//**************************************************************************
#include <windows.h>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "wdaq_bc.h"
#include "daqhimpf.h"

long FAR PASCAL _export WndProc(HWND hWnd, UINT iMessage, UINT wParam, LONG lParam);

/**************************************************************************
// GLOBAL VARIABLES
//**************************************************************************

unsigned long    numSamples;
unsigned long    numResults;       /* number of samples stored in buffer */
HANDLE           hVoltBuffer = 0;  /* handle to a buffer of memory */
HANDLE           hResultBuffer=0;
short            currentmode=0;
short            color;
short            aquire_down=0;
short            alarm_on=0;//

//**************************************************************************
// CLASS DEFINITIONS
//**************************************************************************
------------------------- Main Program Class -------------------------

class Main
{
  public:
          static HANDLE hInstance;
          static HANDLE hPrevInstance;
           static int     nCmdShow;
           static int     MessageLoop();
};
```

```
// static field definitions

HANDLE        Main::hInstance = 0;
HANDLE        Main::hPrevInstance = 0;
int      Main::nCmdShow = 0;int Main::MessageLoop()
{
         MSG    msg;
         while(GetMessage(&msg,NULL,0,0))
         {
                  TranslateMessage(&msg);
                  DispatchMessage(&msg);
         }
         return msg.wParam;
}
//*****************************************************************************
//—————————————————— Base Window Class ——————————————————
//*****************************************************************************

class Window
{
         protected:
                  HWND  hWnd;
         public:
                  HWND        GetHandle(void) {return hWnd;}
                  BOOL        Show(int nCmdShow) { return ShowWindow(hWnd, nCmdShow); }
                  void        Update(void) {UpdateWindow(hWnd);}
                  virtual long WndProc( UINT iMessage, UINT wParam,LONG lParam) = 0;
};

//*****************************************************************************
//————————————————— Derived Graph Window Class ——————————————————
//*****************************************************************************

class GraphWindow : public Window
{
         public:
                  GraphWindow(HWND hWnd);
         static void Register()
         {
                  WNDCLASS wndclass;  // Structure used to register Windows class.
                  wndclass.style            = NULL;
                  wndclass.lpfnWndProc   = ::WndProc;
                  wndclass.cbClsExtra      = 0;
                  wndclass.cbWndExtra    = sizeof(GraphWindow *);
                  wndclass.hInstance       = Main::hInstance;
```

```
                    wndclass.hIcon          = LoadIcon(Main::hInstance, IDI_APPLICATION);
                    wndclass.hCursor        = LoadCursor(NULL, IDC_ARROW);
                    wndclass.hbrBackground = GetStockObject(LTGRAY_BRUSH);
                    wndclass.lpszMenuName  = NULL;
                    wndclass.lpszClassName  = "GraphWndClass";
 if (! RegisterClass(&wndclass))
                        exit(FALSE);
        }          void Paint();
        long WndProc( UINT iMessage, UINT wParam,LONG lParam);
};

// class method definitions

GraphWindow::GraphWindow(HWND hParentWnd)
{

     if (!(hWnd = CreateWindow("GraphWndClass",NULL,
                              WS_CHILD | WS_VISIBLE | WS_BORDER,300,100,520,360,
                              hParentWnd,PB_GRAPHWIN,Main::hInstance,(LPSTR)this)))

                exit(FALSE);
        Show(Main::nCmdShow);
        Update();
}

void GraphWindow::Paint()
{
        HDC             hDC;        // display context for graph window
        PAINTSTRUCT  ps;          // paint structure
        HPEN            hNewPen,hOldPen;
        HCURSOR       hNewCur,hOldCur;
        HBRUSH        hBKBrush,hOldBrush;
        HFONT           hNewFont,hOldFont;
        long int        i,j;          // loop variable
        int      ix;          // loop variable
        double          xscale;       // scale for x axis
        float  huge       *ipVoltBuffer; // huge pointer to sample buffer
        hDC = BeginPaint(hWnd,&ps);
        hNewCur = LoadCursor(NULL,IDC_WAIT);   // change cursor to hour glass
        hOldCur = SetCursor(hNewCur);   SetMapMode(hDC,MM_ANISOTROPIC);
        SetViewportOrg(hDC,0,360);
        SetViewportExt(hDC,520,360);
```

// define logical coordinate system

```
        SetWindowExt(hDC,XHI–XLO,YHI–YLO);
        SetWindowOrg(hDC,0,1280);
```

// Draw graph

```
        hNewPen = CreatePen(PS_SOLID,1,RGB(128,128,128));    // draw hash marks
        hOldPen = SelectObject(hDC,hNewPen);
        for (i=XLO; i<=XHI; i+=(XHI–XLO)/XNUMDIM)
        {
                MoveTo(hDC,i,YHI);          // vertical hash
                LineTo(hDC,i,YLO);
        }
        for (i=YLO; i<=YHI; i+=(YHI–YLO)/YNUMDIM)
        {
                MoveTo(hDC,XLO,i);          // horizontal hash
                LineTo(hDC,XHI,i);
        }
        SelectObject(hDC,hOldPen);
        DeleteObject(hNewPen);
        if(aquire_down==0)
        {
                hNewFont = CreateFont(5,0,0,0,FW_BOLD,FALSE,FALSE,FALSE,ANSI_CHARSET,
                OUT_DEFAULT_PRECIS,CLIP_DEFAULT_PRECIiS,DEFAULT_QUALITY,
                VARIABLE_PITCH | FF_ROMAN,"Tms Rmn");
                hOldFont = SelectObject(hDC,hNewFont);
                hBKBrush=CreateSolidBrush(RGB(128,128,128));
                hNewPen = CreatePen(PS_SOLID,1,RGB(200,200,200));        // draw axes
                hOldPen = SelectObject(hDC,hNewPen);
                hOldBrush=SelectObject(hDC,hBKBrush);
                SetTextAlign(hDC,TA_LEFT);
                TextOut(hDC,2,1160,"        ",9);
                SetTextAlign(hDC,TA_CENTER);
                TextOut(hDC,40,1160,"1.0",3);
                TextOut(hDC,60,1160,"       ",7);
                TextOut(hDC,80,1160,"2.0",3);
                TextOut(hDC,100,1160,"       ",7);
                TextOut(hDC,120,1160,"3.0",3);
                TextOut(hDC,140,1160,"       ",7);
                TextOut(hDC,160,1160,"4.0",3);
                TextOut(hDC,180,1160,"       ",7);
                TextOut(hDC,200,1160,"5.0",3);
                TextOut(hDC,220,1160,"       ",7);
                TextOut(hDC,240,1160,"6.0",3);
                TextOut(hDC,260,1160,"       ",7);
                TextOut(hDC,280,1160,"7.0",3);
                TextOut(hDC,300,1160,"       ",7);
                TextOut(hDC,320,1160,"8.0",3);
```

```
TextOut(hDC,340,1160,"       ",7);
TextOut(hDC,360,1160,"9.0",3);
TextOut(hDC,380,1160,"       ",7);
TextOut(hDC,400,1160,"10.0",3);
TextOut(hDC,420,1160,"       ",7);
TextOut(hDC,440,1160,"11.0",3);
SetTextAlign(hDC,TA_RIGHT);
TextOut(hDC,479,1160,"        ",8);
SetTextAlign(hDC,TA_LEFT);
SelectObject(hDC,hOldPen);
DeleteObject(hNewPen);
SelectObject(hDC,hOldBrush);
DeleteObject(hBKBrush);
SelectObject(hDC,hOldFont);
DeleteObject(hNewFont);
hNewPen = CreatePen(PS_SOLID,1,RGB(0,255,0));     // draw axes
hOldPen = SelectObject(hDC,hNewPen);
MoveTo(hDC,XLO,1120);           // x–axis
LineTo(hDC,XHI,1120);
SelectObject(hDC,hOldPen);
DeleteObject(hNewPen);
hNewPen = CreatePen(PS_SOLID,1,RGB(0,0,255));
```

// Draw axes

```
hOldPen = SelectObject(hDC,hNewPen);
MoveTo(hDC,XLO,(int)((float)color*3.2–160));          // threshold
LineTo(hDC,XHI,(int)((float)color*3.2–160));
SelectObject(hDC,hOldPen);
DeleteObject(hNewPen);
alarm_on=0;
AO_VWrite(1,1,0.0);
//himp_fault=1;
RECT rect;
{
        rect.left=100; rect.top=240;
        rect.right=210; rect.bottom=280;
}
InvalidateRect(GetParent(hWnd),&rect,FALSE);
UpdateWindow(GetParent(hWnd));
```

// Plot buffer if there are samples

```
numResults=117.0;
int himp_fault=0;
if (ipVoltBuffer = (float huge *)GlobalLock(hVoltBuffer)) // lock buffer
{
        hNewPen = CreatePen(PS_SOLID,1,RGB(255,0,0));
```

```
                hOldPen = SelectObject(hDC,hNewPen);
                MoveTo(hDC,0,(int)(1120+ipVoltBuffer[0] * 3));    //was YSCALE
                xscale = (double)numResults / (XHI–XLO);
                i = 0;
                ix = 0;
                while (i < (numResults–1))
                {
                    if( (((int)((1280.0+ipVoltBuffer[i]*3.0)/3.2))<color)
                        && himp_fault==0 )
                    {
                        alarm_on=1;
                        TextOut(hDC,100,40,"There is high impedance fault",29);
                        himp_fault=1;
                        RECT rect;
                        {
                            rect.left=100;
                        rect.top=240;
                            rect.right=210;
                        rect.bottom=280;
                        }
                        InvalidateRect(GetParent(hWnd),&rect,FALSE);
                        UpdateWindow(GetParent(hWnd));
                    }
                    LineTo(hDC,ix,(int)(1120+ipVoltBuffer[i] *3)); //was YSCALE
                    for(j=0;j<90000; j++)
                    {
                        j=j;
                        j=j;
                        j=j;
                    }
                    if (xscale > 1.0)     // if more samples than X–coord
                    {
                        ++ix;
                        i = ix * xscale;
                    }
                else                // if less samples than X–coord
                    {
                        ++i;
                        ix = i * 1 / xscale;
                    }
                }
                SelectObject(hDC,hOldPen);
                DeleteObject(hNewPen);
                GlobalUnlock(hVoltBuffer);        // unlock buffer
            }
    }
```

```
else
{
        hNewFont = CreateFont(5,0,0,0,FW_BOLD,FALSE,FALSE,FALSE,ANSI_CHARSET,
        OUT_DEFAULT_PRECIS,CLIP_DEFAULT_PRECIS,DEFAULT_QUALITY,
                VARIABLE_PITCH | FF_ROMAN,"Tms Rmn");
        hOldFont = SelectObject(hDC,hNewFont);
        hBKBrush=CreateSolidBrush(RGB(128,128,128));
        hNewPen = CreatePen(PS_SOLID,1,RGB(200,200,200));
```

// Draw axes

```
        hOldPen = SelectObject(hDC,hNewPen);
        hOldBrush=SelectObject(hDC,hBKBrush);
        SetTextAlign(hDC,TA_LEFT);
        TextOut(hDC,2,1160,"        ",8);
        SetTextAlign(hDC,TA_CENTER);
        TextOut(hDC,40,1160,".0028",5);
        TextOut(hDC,60,1160,"   ",3);
        TextOut(hDC,80,1160,".0056",5);
        TextOut(hDC,100,1160,"   ",3);
        TextOut(hDC,120,1160,".0083",5);
        TextOut(hDC,140,1160,"   ",3);
        TextOut(hDC,160,1160,".0111",5);
        TextOut(hDC,180,1160,"   ",3);
        TextOut(hDC,200,1160,".0138",5);
        TextOut(hDC,220,1160,"   ",3);
        TextOut(hDC,240,1160,".0167",5);
        TextOut(hDC,260,1160,"   ",3);
        TextOut(hDC,280,1160,".0195",5);
        TextOut(hDC,300,1160,"   ",3);
        TextOut(hDC,320,1160,".0222",5);
        TextOut(hDC,340,1160,"   ",3);
        TextOut(hDC,360,1160,".0250",5);
        TextOut(hDC,380,1160,"   ",3);
        TextOut(hDC,400,1160,".0278",5);
        TextOut(hDC,420,1160,"   ",3);
        TextOut(hDC,440,1160,".0306",5);
        SetTextAlign(hDC,TA_RIGHT);
        TextOut(hDC,479,1160,"       ",7);
        SetTextAlign(hDC,TA_LEFT);
        SelectObject(hDC,hOldPen);
        DeleteObject(hNewPen);
        SelectObject(hDC,hOldBrush);
        DeleteObject(hBKBrush);
        SelectObject(hDC,hOldFont);
        DeleteObject(hNewFont);        hNewPen = CreatePen(PS_SOLID,1,RGB(0,255,0));
```

```
// Draw axes
                hOldPen = SelectObject(hDC,hNewPen);
                MoveTo(hDC,XLO,480);            // x–axis
                LineTo(hDC,XHI,480);
                SelectObject(hDC,hOldPen);
                DeleteObject(hNewPen);  // Plot buffer if there are samples

        numResults=68.0;
                if (ipVoltBuffer = (float huge *)GlobalLock(hVoltBuffer)) // lock buffer
                {
                        hNewPen = CreatePen(PS_SOLID,1,RGB(255,0,0));
                        hOldPen = SelectObject(hDC,hNewPen);
                        MoveTo(hDC,0,(int)(480+ipVoltBuffer[0] * 300));    //was YSCALE
                        xscale = (double)numResults / (XHI–XLO);
                        i = 0;
                        ix = 0;
                        while (i < (numResults–1))
                        {
                             LineTo(hDC,ix,(int)(480+ipVoltBuffer[i] *300)); //was YSCALE
                             for(j=0;j<90000; j++)
                             {
                                j=j;
                                j=j;
                                j=j;
                             }
                             if (xscale > 1.0)      // if more samples than X–coord
                             {
                                ++ix;
                                i = ix * xscale;
                             }
                        else                // if less samples than X–coord
                             {
                                ++i;
                                ix = i * 1 / xscale;
                             }
                        }
                        SelectObject(hDC,hOldPen);
                        DeleteObject(hNewPen);
                        GlobalUnlock(hVoltBuffer);          // unlock buffer
                }
                aquire_down=0;
 }
 SetCursor(hOldCur);          // change cursor back to previous value
 EndPaint(hWnd,&ps);  return;
}long GraphWindow::WndProc( UINT iMessage, UINT wParam,LONG lParam)
{
        switch (iMessage)
        {
                case WM_PAINT:
                        Paint();
                        break;
                default:
                        return(DefWindowProc(hWnd,iMessage,wParam,lParam));
        }
        return TRUE;
}
//**********************************************************************************
//———————————————— Derived Main Window Class ————————————————————————
//**********************************************************************************

class MainWindow : public Window
{
        private:
                static char szCaption[29];
                HWND            hEbBoard,       /* board number edit box */
                                hEbChannel,      /* channel number edit box */
                                hEbGain,        /* voltage gain edit box */
                                hEbCount,       /* number of samples edit box */
                                hEbRate,        /* sampling rate edit box */
                                hEbAlarm,
                                hEbErrCode,      /* error code output box */
                                hQuitButton,     /* quit application button */
                                hWriteButton,    /* output button*/
                                hAquireButton,
                                hGraphWnd;       /* child window handle for graph */
 public:
                        MainWindow();
                static void Register()
                        {
                            WNDCLASS wndclass;
                            wndclass.style         = CS_HREDRAW | CS_VREDRAW;
                            wndclass.lpfnWndProc   = ::WndProc;
                            wndclass.cbClsExtra    = 0;
                            wndclass.cbWndExtra    = sizeof(MainWindow *);
                            wndclass.hInstance     = Main::hInstance;
                            wndclass.hIcon         = LoadIcon(Main::hInstance, "daqroy");
                            wndclass.hCursor       = LoadCursor(NULL, IDC_ARROW);
                            wndclass.hbrBackground = GetStockObject(WHITE_BRUSH);
                            wndclass.lpszMenuName  = NULL;
                            wndclass.lpszClassName = "MainWndClass";
                if (! RegisterClass(&wndclass))
                              exit(FALSE);
                        }              void MakeClient(HWND hClientWnd)
                        {
                            hGraphWnd = hClientWnd;
                        }              void Paint();
                        void Execute_DAQ_Op();
            void Execute_AO_VWrite();

            long WndProc( UINT iMessage, UINT wParam,LONG lParam);
};

char MainWindow::szCaption[] = "high impedance fault";
HWND hTemp;
MainWindow::MainWindow()
{
 if (!(hWnd = CreateWindow("MainWndClass",szCaption,
                WS_OVERLAPPEDWINDOW,CW_USEDEFAULT,CW_USEDEFAULT,870,550,
                NULL,NULL,Main::hInstance,(LPSTR)this)))
            exit (FALSE);

 if (!(hEbGain = CreateWindow("Button","intermittant",WS_CHILD | WS_VISIBLE |
                BS_AUTORADIOBUTTON,40,125,125,30,hWnd,PB_INTER,Main::hInstance,NULL)))
            exit (FALSE);

if (!(hEbChannel = CreateWindow("Button","continuous",WS_CHILD | WS_VISIBLE |
                BS_AUTORADIOBUTTON,40,90,125,30,hWnd,PB_CONTI,Main::hInstance,NULL)))
            exit (FALSE);
                CheckRadioButton(hWnd,PB_INTER,PB_CONTI,PB_INTER);

if (!(hEbBoard =CreateWindow("Button","working mode",WS_CHILD | WS_VISIBLE |
                BS_GROUPBOX,30,60,180,100,hWnd,-1,Main::hInstance,NULL)))
            exit (FALSE);

if (!(hWriteButton = CreateWindow("Button","alarm",BS_DEFPUSHBUTTON |
                WS_CHILD | WS_VISIBLE,30,270,100,30,hWnd,PB_WRITE,Main::hInstance,NULL)))
            exit (FALSE);

if (!(hAquireButton = CreateWindow("Button","Acquisition",BS_DEFPUSHBUTTON |
                  WS_CHILD     |     WS_VISIBLE,30,320,100,30,hWnd,PB_AQUIRE,Main::hIns-
tance,NULL)))
            exit (FALSE);

if (!(hTemp = CreateWindow("Button","Execute ",BS_PUSHBUTTON |
                WS_CHILD | WS_VISIBLE,30,370,100,30,hWnd,PB_EXEC,Main::hInstance,NULL)))
            exit (FALSE);

if (!(hEbAlarm = CreateWindow("static","off",SS_CENTER | WS_CHILD | WS_VISIBLE
  |WS_BORDER,180,240,30,20,hWnd,-1,Main::hInstance,NULL)))
            exit (FALSE);
if (!(hQuitButton = CreateWindow("Button","Quit",BS_DEFPUSHBUTTON |
                WS_CHILD | WS_VISIBLE,30,420,100,30,hWnd,PB_QUIT,Main::hInstance,NULL)))
            exit (FALSE);
if (!(hEbCount = CreateWindow("scrollbar",NULL,WS_CHILD | WS_VISIBLE |
                SBS_VERT,250,80,20,360,hWnd,99,Main::hInstance,NULL)))
            exit (FALSE);

if (!(hEbRate = CreateWindow("static","0",SS_CENTER|WS_CHILD|WS_VISIBLE
                          |WS_BORDER,180,200,30,20,hWnd,-1,Main::hInstance,NULL)))
            exit (FALSE);
  SetScrollRange(hEbCount,SB_CTL,0,400,FALSE);
   SetScrollPos(hEbCount,SB_CTL,0,FALSE);
    numSamples = 0;   Show(Main::nCmdShow);
    Update();
}

void MainWindow::Paint()
{
        HDC             hDC;        /* handle to the display context */
        PAINTSTRUCT  ps;         /* paint structure for HDC */
        HBRUSH          hOldBrush,hNewBrush;
        HFONT           hNewFont,hOldFont;
        hDC = BeginPaint(hWnd,&ps);
        TextOut(hDC,30,200,"Threshold Level",15);
        TextOut(hDC,30,240,"Alarm Status",12);
        if(aquire_down==0)
        {
                TextOut(hDC,535,475,"Time  (s)",9);
                TextOut(hDC,470,70,"Result vs. Time Plot",20);
                //TextOut(hDC,535,475,"Time (ms)",9);
                //TextOut(hDC,535,475,"Time (s) ",9);
        }
        else
        {
                TextOut(hDC,535,495,"Time (ms)",9);
                TextOut(hDC,470,70,"Signal vs. Time Plot",20);
        }
        hNewFont = CreateFont(5,0,0,0,FW_BOLD,FALSE,FALSE,FALSE,ANSI_CHARSET,
                OUT_DEFAULT_PRECIS,CLIP_DEFAULT_PRECIS,DEFAULT_QUALITY,
                        VARIABLE_PITCH | FF_ROMAN,"Tms Rmn");
        hOldFont = SelectObject(hDC,hNewFont);
        SetTextAlign(hDC,TA_RIGHT);
        if(aquire_down==0)
        {
                TextOut(hDC,299,100, " 400",4);
                TextOut(hDC,299,140, " 350",4);
                TextOut(hDC,299,180, " 300",4);
                TextOut(hDC,299,220, " 250",4);
                TextOut(hDC,299,260, " 200",4);
                TextOut(hDC,299,300, " 150",4);
                TextOut(hDC,299,340, " 100",4);
                TextOut(hDC,299,380, "  50",4);
                TextOut(hDC,299,420, "   0",4);
        }
        else
        {
                TextOut(hDC,299,100, " 2.0 ",4);
                TextOut(hDC,299,140, " 1.5",4);
                TextOut(hDC,299,180, " 1.0",4);
                TextOut(hDC,299,220, " 0.5",4);
                TextOut(hDC,299,260, "0   ",4);
                TextOut(hDC,299,300, "–0.5",4);
                TextOut(hDC,299,340, "–1.0",4);
                TextOut(hDC,299,380, "–1.5",4);
                TextOut(hDC,299,420, "–2.0 ",4);
        }
        SelectObject(hDC,hOldFont);
        DeleteObject(hNewFont);  /* shade the graph window */
        hNewBrush = CreateSolidBrush(RGB(63,63,63));
        hOldBrush = SelectObject(hDC,hNewBrush);
        Rectangle(hDC,305,105,830,470);
        SelectObject(hDC,hOldBrush);
        DeleteObject(hNewBrush);
        EndPaint(hWnd,&ps);
        return;
}

void MainWindow::Execute_DAQ_Op()
{
        HCURSOR         hNewCur,hOldCur;
        int     brd=1,
                        ch=2,           // channel number
                        gain=1,         // voltage gain
                        err,
                        output,temp,i,j,k,sp,flag,num_sec,offset,ii;        // return error code
        unsigned long   count;          // number of samples to capture
        double          rate,sum,sum1,sum2;         // sampling rate
        char            sz[MAXSTRINGLENGTH];  // temporary string variable
        HANDLE          hSampleBuffer;      // handle to sample buffer
        HANDLE          hWorkBuffer;
        HANDLE          hMesuBuffer;
        double          rmsp1[4],rmsp2[4],s1[16];
        double          delt=1.0/(60.0*32);
        int huge        *ipSampleBuffer;
        int huge        *ipResultBuffer;     // huge pointer to sample buffer
        double huge     *ipMesuBuffer;
        float huge            *ipVoltBuffer;        // huge pointer to voltage buffer
        float huge            *ipWorkBuffer;
        BOOL            bSuccessful;        // whether DAQ functions were successful
hNewCur = LoadCursor(NULL,IDC_WAIT);
        hOldCur = SetCursor(hNewCur);#ifdef
        GetWindowText(hEbBoard,sz,MAXSTRINGLENGTH); // read input
        brd = atoi(sz);
        GetWindowText(hEbChannel,sz,MAXSTRINGLENGTH);
        ch = atoi(sz);
        GetWindowText(hEbGain,sz,MAXSTRINGLENGTH);
        gain = atoi(sz);
        GetWindowText(hEbCount,sz,MAXSTRINGLENGTH);
        count = (unsigned long)atol(sz);
        GetWindowText(hEbRate,sz,MAXSTRINGLENGTH);
#endif  rate  = atof(sz);
        count=15000; // was 64;
        rate = 1920.0;
        GlobalFree(hVoltBuffer);      // free previous block of memory
        hVoltBuffer = 0;
```

// Allocate and lock two buffers of memory to hold sample values,

```
        bSuccessful = FALSE;
        hSampleBuffer = (HANDLE)GlobalAlloc(GMEM_MOVEABLE,(DWORD)sizeof(int)*count);
        hMesuBuffer  = (HANDLE)GlobalAlloc(GMEM_MOVEABLE,(DWORD)sizeof(double)*count);
        if (hSampleBuffer && hMesuBuffer)
        {
                ipSampleBuffer = (int huge *)GlobalLock(hSampleBuffer);
                ipMesuBuffer  = (double huge *)GlobalLock(hMesuBuffer);
                if (ipSampleBuffer && ipMesuBuffer)
                { /* call NI-DAQ function */
                        err = DAQ_Op(brd,ch,gain,ipSampleBuffer,count,rate);
                        if (!err)
                        {
                            err = DAQ_VScale(brd,ch,gain,1,0,count,ipSampleBuffer,ipMesuBuffer);
                            if (!err)
                                bSuccessful = TRUE;
                        }
                }
                GlobalUnlock(hMesuBuffer);    // unlock voltage sample buffer
```

```
            GlobalUnlock(hSampleBuffer);   // unlock binary sample buffer
        }
        GlobalFree(hSampleBuffer);        // deallocate binary sample buffer
        hSampleBuffer = 0;    hVoltBuffer            =            (HANDLE)GlobalAlloc(GMEM_MOVE-
ABLE,(DWORD)sizeof(float)*count);
        hResultBuffer = (HANDLE)GlobalAlloc(GMEM_MOVEABLE,(DWORD)sizeof(int)*118);
        hWorkBuffer   = (HANDLE)GlobalAlloc(GMEM_MOVEABLE,(DWORD)sizeof(float)*256);
        ipMesuBuffer  = (double huge *)GlobalLock(hMesuBuffer);
        ipVoltBuffer  = (float huge *)GlobalLock(hVoltBuffer);
        for(i=0; i<count; i++)
        {
            ipVoltBuffer[i]=(float)ipMesuBuffer[i];
        }
        GlobalUnlock(hVoltBuffer);
        GlobalFree(hMesuBuffer);
        output=0;
        if (bSuccessful&&(aquire_down==0))
        {
            numSamples = count;               // DAQ_Op successful
            //start of algorithms
            ipVoltBuffer  = (float huge *)GlobalLock(hVoltBuffer);
            ipResultBuffer=(int huge *)GlobalLock(hResultBuffer);
            i=0;
            while(ipVoltBuffer[i]*ipVoltBuffer[i+1]>0)
            {
              i++;
            }
            sp=i;          if(ipVoltBuffer[sp]>0.0)
                flag=1;
            else flag=0;       ipWorkBuffer = (float huge *)GlobalLock(hWorkBuffer);
            num_sec=(count-sp)/128;
            offset=0;   //zero crossing point offset
            int countb=0;
            int countc=0;
            int pt_sinwv=0;
            int slp_inc=0;
            int pt_slpwv=0;
            for( ii=0; ii<num_sec-1; ii++)
            {
                if(ipVoltBuffer[sp+offset+ii*32*4]*ipVoltBuffer[sp+offset+1+ii*32*4]<0)
                    offset+=1;
                for(i=0; i<32*4; i++)
                {
                    ipWorkBuffer[i]=ipVoltBuffer[i+sp+offset+ii*32*

                int pt_fk=0;
int pt_sym=0;
int pt_comput=0;
int pt_zero=0;
float Im1_o=0.0;
float Im1_n=0.0;
//float Im2_o=0.0;
//float Im2_n=0.0;
float min1=0.0;
float min2=0.0;
int min1_n,min2_n;
//for (j=0;j<4;j++)
//{
     int zeros=0;
     min1=fabs(ipWorkBuffer[0]);
     for(i=1; i<15; i++)
     {
        Im1_o=ipWorkBuffer[i];
        Im1_n=fabs(Im1_o);
        if((min1-Im1_n)>0.0001)
        {
        min1=Im1_n;
        min1_n=i;
        }
     }
     min2=fabs(ipWorkBuffer[19]);
     for (i=19;i<30;i++)
     {
        Im1_o=ipWorkBuffer[i];
        Im1_n=fabs(Im1_o);
        if((min2-Im1_n)>0.0001)
        {
         min2=Im1_n;
         min2_n=i;
        }
     }
     if((min2_n<=28)&&(min2_n>=24)
       &&(min1_n>=8)&&(min1_n<=12)  )
       //&&(min1<0.015)&&(min2<0.015))
     {
        pt_comput=1;
        countb++;
     }
     else
     {
        countc++;
     }
```

```
if(countb>25)
{
        pt_comput=1;
        countc=0;
}
if(countc>25)
{
        pt_comput=0;
        countb=0;
}
if ( (pt_comput>0)&&(output>5) )
            output-=5;
for(j=0; j<4; j++)
{
        sum1=0.0;
        sum2=0.0;
        for(i=0; i<16; i++)
        {
             sum1+=ipWorkBuffer[i+j*32]*ipWorkBuffer[i+j*32];
             sum2+=ipWorkBuffer[i+16+j*32]*ipWorkBuffer[i+16+j*32];
        }
        rmsp1[j]=sqrt(fabs(sum1))/16.0;
        rmsp2[j]=sqrt(fabs(sum2))/16.0;
}
for(i=0; i<4; i++)
{
        if( rmsp1[i]<0.001 ) {pt_zero=1;}
        else
        pt_zero=0;
}
```

//***

// Check flickering of the rms value at either positive or negative waveform side

//*

```
float rms_min=0.0;
rms_min=rmsp1[0];
for(i=1; i<4; i++)
{
        if(rmsp1[i]<rms_min)
        {
             rms_min=rmsp1[i];
        }
}
if (   /*(((rmsp1[0]-rmsp1[1])*(rmsp1[2]-rmsp1[1]))>0.0)&&*/
                (fabs(rmsp1[1]-rmsp1[0])>0.1*rms_min)
```

```
                                    || (fabs(rmsp1[2]–rmsp1[1])>0.1*rms_min)
                                    ||  (fabs(rmsp1[3]–rmsp1[2])>0.1*rms_min)        )
                                  //||( /*(((rmsp2[0]–rmsp2[1])*(rmsp2[2]–rmsp2[1]))>0.0)&& */
                                  //  (fabs(rmsp2[0]–rmsp2[1])>0.1*rmsp2[1])
                                  //  &&(fabs(rmsp2[2]–rmsp2[1])>0.1*rmsp2[1]) ) )
                         pt_fk=2;

//************************************************************
// Check for Asymmetry
//************************************************************
if(   (fabs(fabs(rmsp1[0])–fabs(rmsp2[0]))>0.1*rmsp1[0])
                                  && (fabs(fabs(rmsp1[1])–fabs(rmsp2[1]))>0.1*rmsp1[0])  )
                                  pt_sym=1;

                              for(j=0;j<8; j++)
                              {
                                  int counter=0;
                                  for(i=0; i<15; i++)
                                  {
                                      s1[i]=ipWorkBuffer[i+j*16+1]–ipWorkBuffer[i+j*16];
                                      if ( ((flag==1)&&(fabs(s1[i])<0.28*rmsp1[1])&&(s1[i]*s1[i–1]>0))
                                      ||((flag==0)&&(fabs(s1[i])<0.28*rmsp2[1])&&s1[i])&&(s1[i]*s1[i–1]>
0) )
                                      counter+=1;
                                  }
                                  if ((counter>=3))    // 3 could be changed
                                      pt_slp=1;//output–=3;
                         if(flag==1) flag=0;
                                  if(flag==0) flag=1;
                              }
                              slp_inc++;

//**********************************************************************
//Check for quater cycle symmetry, useful for fault on wet soil.
//**********************************************************************
          int qpt_sym=0;
                              int count_qpt=0;

                              {
                                  for(j=0;j<8;j++)
                                  {
                                      sum=0.0;
                                      for(i=0;i<7;i++)
                                      {
                                            sum+=fabs(fabs(ipWorkBuffer[i+j*16]–ipWorkBufer[15–i+j*16])
                              –fabs(ipWorkBuffer[i+1+j*16]–ipWokBfer[15–i–1+j*16]) );
```

```
                    }
                    if((sum>2*rmsp2[1])&&(pt_comput==0))
                         count_qpt+=1;
                    else if((sum>rmsp2[1])&&(pt_comput==0))
                         count_qpt=count_qpt;
                    else if ((sum<rmsp2[1])&&(output>5)&&(pt_comput==0))
                         count_qpt=count_qpt;
                    else
                         count_qpt=0;
               }
               if(count_qpt>4)
                    qpt_sym=1;
               else
                    qpt_sym=0;
          }
          if(pt_sym==0&&pt_fk==0&&(slp_inc<20))
          {
               qpt_sym=0;
               pt_sinwv++;
          }
          if((pt_slp==1)&&(slp_inc<20))
          pt_slpwv++;// Set the detection threshold and Output Detection
temp=pt_slp+(int)(qpt_sym)+pt_sym+pt_fk;
          if((pt_sinwv>15)&&(pt_slpwv<3)&&(pt_fk==0)) temp=0;
          if(pt_zero==1) temp=0;
          if(temp==0)
          {
               if(output>0)
                    output-=0; // was 1;
          }
          else
          {
               output+=temp;
          }
          if (output<0)
          output=0;
          ipResultBuffer[ii]=output;
     }
     for (ii=0;ii<num_sec; ii++)
     {
          ipVoltBuffer[ii]=-ipResultBuffer[ii];
     }
     GlobalUnlock(hResultBuffer);
     GlobalUnlock(hVoltBuffer);
     InvalidateRect(hGraphWnd,NULL,TRUE);  // repaint graph window
}
```

```
        else if (bSuccessful&&(aquire_down==1))
        {
                InvalidateRect(hGraphWnd,NULL,TRUE);
        }
        else
        {
                numSamples = 0;            // DAQ_Op unsuccessful
                GlobalFree(hVoltBuffer);    // deallocate voltage sample buffer
                hVoltBuffer = 0;
        }       sprintf(sz,"%d",err);               // display output
        SetWindowText(hEbErrCode,sz);#ifdef
        hVoltBuffer=(HANDLE)GlobalAlloc(GMEM_MOVEABLE,(DWORD)szeof(float)*118);
                ipVoltBuffer  = (float huge *)GlobalLock(hVoltBuffer);
                num_sec=117;
                ipVoltBuffer[0]=0;
                for (ii=1;ii<num_sec; ii++)
                {
                        ipVoltBuffer[ii]=(float)–ii;
                }
                GlobalUnlock(hVoltBuffer);     // deallocate voltage sample buffer
                InvalidateRect(hGraphWnd,NULL,TRUE);  // repaint graph window
```

// Change the cursor back to the previous value

```
        SetCursor(hOldCur);
        if(currentmode==1)
        {
                SendMessage(hTemp,BM_SETSTATE,1,0L);
                SendMessage(hTemp,BM_SETSTATE,0,0L);
        }
}void MainWindow::Execute_AO_VWrite()
{
  HCURSOR hNewCur,           // new (hourglass) cursor
                  hOldCur;          // old (arrow) cursor
  int    brd=1,             // board number
                  ch=1,              // channel number
                  err;             // return error code
  double  volt;             // voltage
  char   sz[MAXSTRINGLENGTH];  // temporary string variable
if (alarm_on==0)
        {
                volt=5.0;
                alarm_on=1;
        }
        else
        {
                volt=0.0;
                alarm_on=0;
        }// change the cursor to the hourglass
  hNewCur = LoadCursor(NULL,IDC_WAIT);
  hOldCur = SetCursor(hNewCur);  // call DLL function
  err = AO_VWrite(brd,ch,volt);
  sprintf(sz,"%d",err);
  SetCursor(hOldCur);
}
```

// MainWindow::WndProc(UINT, UINT,LONG)
// — specifies actions based on incoming messages

```
'
long MainWindow::WndProc( UINT iMessage, UINT wParam,LONG lParam)
{
        char szbuffer[10];
        short i;
        switch (iMessage)
        {
                case WM_CLOSE:          // executed upon closing of application
                        GlobalFree(hVoltBuffer);  // free any allocated buffer
                        hVoltBuffer = 0;
                        return(DefWindowProc(hWnd,iMessage,wParam,lParam));
                case WM_COMMAND:
                        switch(wParam)
                        {
                             case PB_EXEC:        // call DAQ_Op() DLL function call
                             RECT rect;
                             {
                                rect.left=300; rect.top=10;
                                rect.right=820; rect.bottom=99;
                             }
                             InvalidateRect((hWnd),&rect,FALSE);  // repaint graph window
                             {
                                rect.left=269; rect.top=99;
                                rect.right=301; rect.bottom=460;
                             }
                             InvalidateRect((hWnd),&rect,FALSE);  // repaint graph window
                                if(currentmode==0)
                                {
                                        Execute_DAQ_Op();
                                }
                                else
                                {
                                        GetAsyncKeyState(27);
                                        while(!(GetAsyncKeyState(27)&1))
                        {
        Execute_DAQ_Op();
                            rect.left=300; rect.top=99;
                            rect.right=820; rect.bottom=431;
                            InvalidateRect((hWnd),&rect,FALSE);
                            UpdateWindow(hWnd);
                        }
                    }
                  break;
              case PB_AQUIRE:
                {
                    aquire_down=1;
                    Execute_DAQ_Op();
                }
                  {
                    rect.left=300; rect.top=10;
                    rect.right=820; rect.bottom=99;
                  }
                  InvalidateRect((hWnd),&rect,FALSE);   // repaint graph window
                  {
                    rect.left=269; rect.top=99;
                    rect.right=301; rect.bottom=440;
                  }
                  InvalidateRect((hWnd),&rect,FALSE);   // repaint graph window
break;
              case PB_WRITE:
                  Execute_AO_VWrite();
                  {
                    rect.left=100; rect.top=230;
                    rect.right=250; rect.bottom=290;
                  }
                  InvalidateRect((hWnd),&rect,FALSE);  // repaint graph window
                  break;          case PB_QUIT:        // quit the application
                  PostQuitMessage(0);
                  break;
              case PB_INTER:
                  currentmode=0;
                  break;
              case PB_CONTI:
                  currentmode=1;
                  break;
              default:
                return(DefWindowProc(hWnd,iMessage,wParam,lParam));
              }
        break;
case WM_CREATE:
```

```
             break;
   case WM_VSCROLL:
            switch(wParam)
            {
                    case SB_LINEDOWN:
                         if (color<400)
                         color++;
                         else
                           color=400;
                         break;
                    case SB_LINEUP:
                         if(color>0)
                             color—;
                         else
                             color=0;
                         break;
                    case SB_THUMBPOSITION:
                    case SB_THUMBTRACK:
                         color=LOWORD(lParam);
                         break;
                    default:
                         break;
            }
            SetScrollPos(hEbCount,SB_CTL,color,TRUE);
            SetWindowText(hEbRate,itoa(400–color,szbuffer,10));
            RECT rect;
            //if(direction==0)
            {
                    rect.left=300; rect.top=99;//+(int)(4.0*(float)color/5.0);
                    rect.right=820; rect.bottom=431;//+(int)(4.0*(float)color/5.0);
            }
            InvalidateRect((hWnd),&rect,FALSE);   // repaint graph window
            break;
  case WM_PAINT:
            Paint();
                         if(alarm_on==1)
                             {
                             AO_VWrite(1,1,5.0);
                             wsprintf((LPSTR)szbuffer," on");
                             }
                         else
                             wsprintf((LPSTR)szbuffer,"off");
                         SetWindowText(hEbAlarm,(LPSTR)szbuffer);
            break;
  case WM_DESTROY:
            PostQuitMessage(0);
```

```
                    break;
            default:
                    return(DefWindowProc(hWnd,iMessage,wParam,lParam));
            }
            return TRUE;
}

//**************************************************************************************
//      WINDOWS     PROCEDURES
**************************************************************************************

# if defined(__SMALL__) || defined (__MEDIUM__)
// data pointers are near pointers
inline Window *GetPointer ( HWND hWnd )
  {
          return (Window *)GetWindowWord(hWnd, 0);
  }
  inline void SetPointer(HWND hWnd,Window *pWindow)
  {
          SetWindowWord(hWnd,0,(WORD)pWindow);
  }
#elif defined(__LARGE__) || defined(__COMPACT__)
  // else pointers are far
  inline Window *GetPointer(HWND hWnd)
  {
          return (Window *)GetWindowLong(hWnd, 0);
  }
  inline void SetPointer(HWND hWnd,Window *pWindow)
  {
          SetWindowLong(hWnd,0,(LONG)pWindow);
  }
#else
 #error Choose another memory model!
#endif// WndProc(HWND, UINT, UINT,LONG)

//  — is the callback function for Windows. First to trap incoming
//    Windows messages to application. Delegates most messages to
//    Window class WndProc defined earlier

long FAR PASCAL _export WndProc(HWND hWnd, UINT iMessage, UINT wParam,LONG lParam)
{
  Window *pWindow = GetPointer(hWnd);
          if (pWindow == 0)
          {
          if (iMessage == WM_CREATE)
                  {
```

```
                LPCREATESTRUCT lpcs;          lpcs = (LPCREATESTRUCT)lParam;
                pWindow = (Window *)lpcs–>lpCreateParams;
                SetPointer(hWnd,pWindow);
                return(pWindow–>WndProc(iMessage,wParam,lParam));
                }
            else

                return(DefWindowProc(hWnd,iMessage,wParam,lParam));
            }
  else
            return(pWindow–>WndProc(iMessage,wParam,lParam));
}


//**************************************************************
// WinMain(HANDLE,HANDLE,LPSTR,int)
// — Instantiates the application and initiates the message loop
//**************************************************************

int PASCAL WinMain(HANDLE hInstance,HANDLE hPrevInstance,LPSTR lpszCmdLine,
                              int nCmdShow)
{
  Main::hInstance = hInstance;
  Main::hPrevInstance = hPrevInstance;
  Main::nCmdShow = nCmdShow;   if (!Main::hPrevInstance)
            {
            MainWindow::Register();
            GraphWindow::Register();
            }  MainWindow MainWnd;                  // instantiate main window
GraphWindow  GraphWnd(MainWnd.GetHandle());    // instantiate child window    MainWnd.MakeCli-
ent(GraphWnd.GetHandle());   // make graph window a client   return Main::MessageLoop();
}
```

A C++ porgram for detecting High Impetdance Fault in power systems

Developed by *DEHUA ZHENG,* May, 1994

Algorithm:

1; Flicker

Measure and Compare the consective 3 cycle current RMS values to see if the deviations change sign in either poistive or negtive side

2; Aysmmtry

Observe the current wavwform at both sides in each cycle for two continuous cycles to see if there is Asymmetry

3; Quarter-Cycle-Aysmmetry

Investigate the two parts of a half cycle waveform to see if there is half cycle Asymmtry.

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <alloc.h>
#include <conio.h>
#include <graphics.h>
#include <dos.h>

struct point
{
        int x;
        int y;
};

main( )
{
        FILE        *fp;
        float  huge *I;
        float  *Iw;
        float       ftr,sum,sum1,sum2,t,p;
        float       rmsp1[4],rmsp2[4],s1[16];
        float  Im1_o,Im1_n,Im2_o,Im2_n,delt;
        int             threshold,pt_fk,qpt_sym,pt_sym,pt_slp,pt_comput;
        int             temp,output,first_time,count,num_sec,sp,k,kk,i,ii,j;
        int     *ot;
        int     offset,xmax, ymax;
        int     gmode, gdriver;
        int     flag,size;
        point   st,ed;

//Initializition of values

        pt_fk=0;
        pt_sym=0;
        pt_quat=0;
        size=15000;
        first_time=0;
  delt=0.000521;
        k=0;
```

// Now start reading from data file in which the first column
// is time , second column is high impedance fault current

```
        output=0;
        I=(float huge *) farcalloc(size,sizeof(float));
        ot=(int *) calloc(128,sizeof(int));
```

// Here is finding zero point , from which we start to do S.SHOT

```
        fp=fopen("himp.dat","r");
        i=0;
        t=-1.0;
        while ( t<0)
        {
                fscanf(fp," %f  %f", &t, &p);
                i++;
        }
        sp=i;   // sp represents starting point
        fclose(fp);
        fp=fopen("himp.dat","r");
        for(i=0; i<size; i++)
        {
                fscanf(fp," %f  %f", &t, &I[i]);
        }

        if(I[sp]>0.0) flag=1;
        else  flag=0;

// should feed the next 32x3 samples into an working array

        Iw = (float *) calloc(128, sizeof(float));
        num_sec=(size-sp)/128;
```

```
//start graphics

                printf("please input threshold\n");
                scanf("%d",&threshold);  // can be changed

// now the grahpics portion

                gdriver = VGA;
                gmode=VGAHI;

  /* initialize graphics and local variables */

                initgraph(&gdriver, &gmode, "c:\\borlandc\\bgi");

  /* read result of initialization */

                setcolor(getmaxcolor());
                xmax = getmaxx();
                ymax = getmaxy();

  //draw a frame

                setcolor(LIGHTGREEN);
                setlinestyle(0,1,3);
                line( 1,1,xmax-1,1);
                line(1,1,1,ymax-1);
                line(xmax-1,1,xmax-1,ymax-1);
                line(1,ymax-1,xmax-1,ymax-1);

  // draw coordinate

                setcolor(LIGHTMAGENTA);
                line(3, (ymax-5), xmax, (ymax-5));
                line(10,1,10,ymax);

                //draw threshold
                ftr=0.5;
                setcolor(LIGHTRED);
                line((xmax-5),-threshold*ftr+(ymax-17),10,-threshold*ftr+(ymax-17));
                setcolor(WHITE);
                moveto(20,-threshold*3/5+(ymax-2)-30);
                outtext("Fault Trip Level");

                //draw the change in the output

                setlinestyle(3,1,3);

                setcolor(YELLOW);

                st.x=10;
                st.y=ymax-17;

                //output must be replaced by ot[ii] if we have more points later
                /* move the C.P. to the center of the screen */
```

```
        offset=0;
        for( ii=0; ii<num_sec-1; ii++)

        {

                if(  I[sp+offset+ii*32*4]
                        *I[sp+offset+1+ii*32*4]<0)
                offset+=1;
                for(i=0; i<32*4; i++)
                {
                        Iw[i]=I[i+sp+offset+ii*32*4];
                }
```

//
Check for flickering of the rms value at either positive or negative
// waveform side
//***

```
                pt_fk=0;
                pt_sym=0;
                pt_quat=0;

                for(j=0; j<4; j++)
                {
                        sum1=0.0;
                        sum2=0.0;
                        for(i=0; i<16; i++)
                        {
                                sum1+=Iw[i+j*32]*Iw[i+j*32];
                                sum2+=Iw[i+16+j*32]*Iw[i+16+j*32];
                        }
                        rmsp1[j]=sqrt(sum1)/16.0;
                        rmsp2[j]=sqrt(sum2)/16.0;

                }

                if ( ( (((rmsp1[0]-rmsp1[1])*(rmsp1[2]-rmsp1[1]))>0.0)
                        &&(fabs(rmsp1[0]-rmsp1[1])>0.1*rmsp1[1])
                        &&(fabs(rmsp1[2]-rmsp1[1])>0.1*rmsp1[1]) )
                  ||( (((rmsp2[0]-rmsp2[1])*(rmsp2[2]-rmsp2[1]))>0.0)
                        &&(fabs(rmsp2[0]-rmsp2[1])>0.1*rmsp2[1])
                        &&(fabs(rmsp2[2]-rmsp2[1])>0.1*rmsp2[1]) ) )

                        pt_fk=1;
```

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

// Check for Asymmetry at each cycle for two continouous cycles
//***********************************************************

                    if((fabs(fabs(rmsp1[0])–fabs(rmsp2[0]))>0.1*rmsp1[0])&&
                     (fabs(fabs(rmsp1[1])–fabs(rmsp2[1]))>0.1*rmsp1[0]))
                      pt_sym=1;

                    for(j=0;j<8; j++)
                    {
                              count=0;
                              for(i=0; i<15; i++)
                              {
                                        s1[i]=Iw[i+j*16+1]–Iw[i+j*16];
                    if ( ((flag==1)&&(fabs(s1[i])<0.25*rmsp1[1])&&(s1[i]*s1[i–1]>0))
                              ||        ((flag==0)&&(fabs(s1[i])<0.25*rmsp2[1])&&s1[i])
                                        &&(s1[i]*s1[i–1]>0) )
                              count+=1;
                              }
                     //       printf("\n %d",count);

                              if ((count>=3)&&(output>3))   // 3 could be changed
                              output–=3;

                              if(flag==1) flag=0;
                              if(flag==0) flag=1;
                              break;
                    }

                     Im1_o=0.0;
                     Im1_n=0.0;
                     Im2_o=0.0;
                     Im2_n=0.0;

               for (j=0;j<8;j++)

               {
                              Im1_o=Im1_n;
                              Im2_o=Im2_n;
                              for (i=0;i<15;i++)
                                        { if(fabs(Iw[i+1+j*16])>=fabs(Iw[i+j*16]))
                                                  {

                              k=i+1+j*16;
                                                  }
                                        }
                               // printf("\n %d %f\n",k,Iw[k]);
if ((((Iw[k]–Iw[k–1])/delt)>100.0)
            &&(((Iw[k]–Iw[k+1])/delt)>100.0))

           Im1_n=Iw[k];
                  // printf("\n %d %d %f %f\n",k,pt_comput,Iw[k],Im1_n);

                  for (i=0;i<14;i++)

                  { if((fabs(Iw[i+1+j*16])>=fabs(Iw[i+j*16]))
                          &&((Iw[i+1+j*16])!=Im1_n)
                          &&((Iw[i+1+j*16])!=Iw[k–1])
                          &&((Iw[i+1+j*16])!=Iw[k–2])
                          &&((Iw[i+1+j*16])!=Iw[k+1])
                          &&((Iw[i+1+j*16])!=Iw[k+2]))

                                 {

           kk=i+1+j*16;
           //     printf("\n %d %f\n",k,Iw[k]);
                                 }
                          }

           if ((((Iw[kk]–Iw[kk–1])/delt)>100.0)
            &&(((Iw[kk]–Iw[kk+1])/delt)>100.0))
                  {
                          pt_comput+=1;
                  }

           Im2_n=Iw[kk];
                  //printf("\n %d %d %f %f\n",kk,pt_comput,Iw[kk],Im2_n);

                  if((Im1_o==0.0)||(Im2_o==0.0))
                          continue;

            if((fabs(fabs(Im1_n)–fabs(Im1_o))<=0.1*fabs(Im1_n))
                  &&(fabs(fabs(Im2_n)–fabs(Im2_o))<=0.1*fabs(Im2_n)))
                          {
                          pt_comput+=1;
                 // printf("\n %d %f %f\n",pt_comput, Im1_o,Im1_n);
                                 }
                          if ((pt_comput>=0)&&(output>5))
                            output–=5;
                          }
//                printf("\n %d\n",pt_comput);
           //     printf("\n %g %g %g %g\n", kk,&Iw[kk],Im1_n,Im2_n);
```

//**

Check for quater cycle asymmetry, useful for fault on wet soil.

//**
***

```
                qpt_sym=0;

                for(j=0;j<8;j++)
                {
                        sum=0.0;
                        for(i=0;i<7;i++)
                        {
                                sum+=fabs(  fabs(Iw[i+j*16]–Iw[15–i+j*16])
                                                                        –fabs(Iw[i
+1+j*16]–Iw[15–i–1+j*16]) );
                                                                //              printf("\n
%g %g \n",sum,rmsp2[1]);
                        }
                        if((sum>2*rmsp2[1])&&(count<3)&&(pt_comput<0))
                                output+=5;
                        else if ((sum<rmsp2[1])&&(count<3)&&(output>5))
                                output–=1;
                        else if((sum>rmsp2[1])&&(count<3)&&(pt_comput<=0))
                                output+=1.5;
                        else
                                output+=0;

                }
```

// Set the detection threshold and Output Filtering

```
                temp=pt_slp+(int)(0.5*(float)qpt_sym)+pt_sym+pt_fk;
                //printf("%d\n",temp);
                if(temp==0)
                {
                        if(output>0)
                                output–=1;
                }
                else
                        output+=temp;

                ot[ii]=output;

                ed.x=10+5*ii;
                ed.y=ymax–17–ot[ii]*ftr;
                line(st.x,st.y,ed.x,ed.y);
                delay(200);
                st.x=ed.x
```

```
                    st.y=ed.y;
                    if((output>threshold)&&(first_time==0))
                    {
                              sound(400);
                              delay(4000);
                              nosound();
                              first_time=1;
                    }
          }
          free(Iw);
          farfree(I);

// get a final output result before taking action –

          setcolor(WHITE);
          settextstyle(1,0,2);
          moveto(150,43);
          if (output>=threshold)
          {
                    outtext("There Is A High Impedance Fault");
          }
          else
                    outtext("There Is No High Impedance Fault");
 /* clean up */
          getch();
          closegraph();
          return 0;
}
```

A C++ program which translates Louts 1-2-3 print file to DOS file suitable for EMTDC multiplot or other softwares to use.

Developed by DEHUA ZHENG, April, 1994

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <alloc.h>

int main()
{
        int   i,j,size,num_row,zero_time;
        float  t,interval;
        float  b,a0,a1;
        float  huge *I;
        FILE   *fp, *fp1;

        //printf("The name of inputfile from lotus 123 \n");
        //gets(fname);

/*  printf("Input the number of rows \n");

// to accept a 5 cols and 3000 rows data file form Lotus 1-2-3 print file
        scanf("%d",&num_row);

        printf("\tInput number of cols \n");
        scanf("%d",&num_col);

        printf("\tInput the interval between two samples\n");
        scanf("%f",&interval); */

        num_row=3000; // needs to be changed to 3000
        size=15000;
        interval=0.000521;
        fp=fopen("himp.prn","r");
        I=(float huge *) farcalloc( size, sizeof(float));

        i=0;
        fscanf(fp," %f %f %f %f %f\n",&a0,&b,&b,&b,&b);
        a1=a0;
        while(a1*a0>0.0)
        {
        fscanf(fp," %f %f %f %f %f\n",&a1,&b,&b,&b,&b);
```

```
                i++;
        }
        zero_time=i—;
        fclose(fp);

        fp=fopen("himp.prn","r");
        fp1=fopen("himp.dat","w");

        t=0.0–zero_time*interval;
        for (i=0; i<num_row; i++)
        {

        fscanf(fp," %f %f %f %f %f\n",&I[i],&I[i+3000],&I[i+6000],
                                        &I[i+9000],&I[i+12000]);
        }
        fclose(fp);

        for(j=0;j<size;j++)
        {
                t+=interval;
                fprintf(fp1,"%12.8f %12.8f\n",t,I[j]);
        }

        fclose(fp1);
        farfree(I);
        return 0;
}
```