# A NEW DIGITAL RELAY FOR DISPERSED STORAGE AND GENERATION (DSG) UNITS

By

Dewayne Randolph Brown

Dissertation submitted to the Faculty of Virginia Polytechnic Institute and State University in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Electrical Engineering

Approved By

February 1997

Blacksburg, Virginia

# A NEW DIGITAL RELAY FOR DISPERSED STORAGE AND GENERATION (DSG) UNITS

BY

Dewayne Randolph Brown

Arun G. Phadke,Chairman

ELECTRICAL ENGINEERING

(ABSTRACT)

A dispersed storage and generation system consists of both a source of energy for conversion to electricity and a means to interconnect this unit to the electric utility system. Most dispersed generators connected to the utility distribution system are less than 5 MW. A new emerging problem arises when a utility circuit breaker is opened while a capacitor-bank bus and load bus is energized very near a dispersed storage and generation unit. A new resonant- detecting relay will be developed that has the ability to trip if dangerously high off-nominal frequency voltages are produced at the capacitor-bank bus or load bus during this condition (self-excitation). Self-excitation may lead to severe overvoltages, and may induce undesirable transients in neighoring circuits (low power and control circuits being particularly vulnerable) on the connected network. It can create harmonics that can fail surge arrestors or cause transformer saturation. The new relay will be able to detect this condition at the inter-tie and trip the cogeneration facility. In order to make this multi-function relay cost effective, it includes protection functions for traditional relay functions such as directional inverse time-delay overcurrent, directional instantaneous overcurrent, islanding, loss-of- excitation, differential, overfrequency, underfrequency, overvoltage, undervoltage, negative-sequence current, check-synchronism, and directional power. Also, the new relay will have metering capability.

Keywords: Protection, Relays, Shunt Capacitor-Banks

# DEDICATION

I dedicated this book to my mother who still is my best friend. I thank you mama for listening to my sorrows and giving me advice when I needed it. I thank you for having confidence in me and believing that I would make it even when the odds were against me.

# ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. Arun G. Phadke, for all his support and guidance during my research. I would also like to thank my other committee members, Dr. Liu, Dr Greenberg, Dr. Broadwater and Dr. De La Ree for their help while I pursued my graduate degree. I would also like to acknowledge The Office of Naval Research for their financial support. I want to thank Derrek Dunn and John Wicks for being my friends at Virginia Tech. and providing me inspirational advice. Last but not least I would like to thank god who stuck by me through it all.

# Contents

# List of Figures

# Chapter 1

# INTRODUCTION

## 1.1 PROBLEM STATEMENT

As the number of dispersed storage and generation units connected to the power system increases, the need to develop more multifunction relays at the utility-industrial tie that will detect new emerging problems will become increasingly evident. Figure 1.1 shows a network configuration that contains a dispersed storage and generation (DSG) unit. Multifunction relays exist that can provide protection for overfrequency, underfrequency, overvoltage, undervoltage, check-synchronism, negative-sequence, differential, and islanding functions at the inter-tie. A new emerging problem arises when a utility circuit breaker is opened while a capacitor-bank and load is energized very near a dispersed storage and generation unit. The capacitor-bank may lead to severe off-nominal high frequency voltages due to self-excitation of the DSG. In some instances, capacitor-banks may self-excite and create harmonics that will damage surge arrestors and prevent relays from operating properly. This may occur upon a trip of a breaker that causes islanding. The objective of this research is to design a relay that detects these conditions and will trip the DSG unit. In order to make this multifunction relay cost effective, it includes all the traditional relaying functions and metering capability.

## 1.2 REVIEW OF DISPERSED STORAGE AND GENERATION

A dispersed storage and generation (DSG) system consists of both a source of energy for conversion to electricity and a means to interconnect this unit to the electric utility system. Most dispersed generators connected to the utility distribution system are less than 20 MW. Installations of this size form the majority of DSG interconnections in the United States. Larger DSG sytems are normally connected to utility transmission and subtransmission circuits and with capacities up to several hundred megawatts.

Small power producers, cogenerators, and fossil-fuel-powered generators connected to electric distribution systems make up the DSG category. The term “small power producer” is

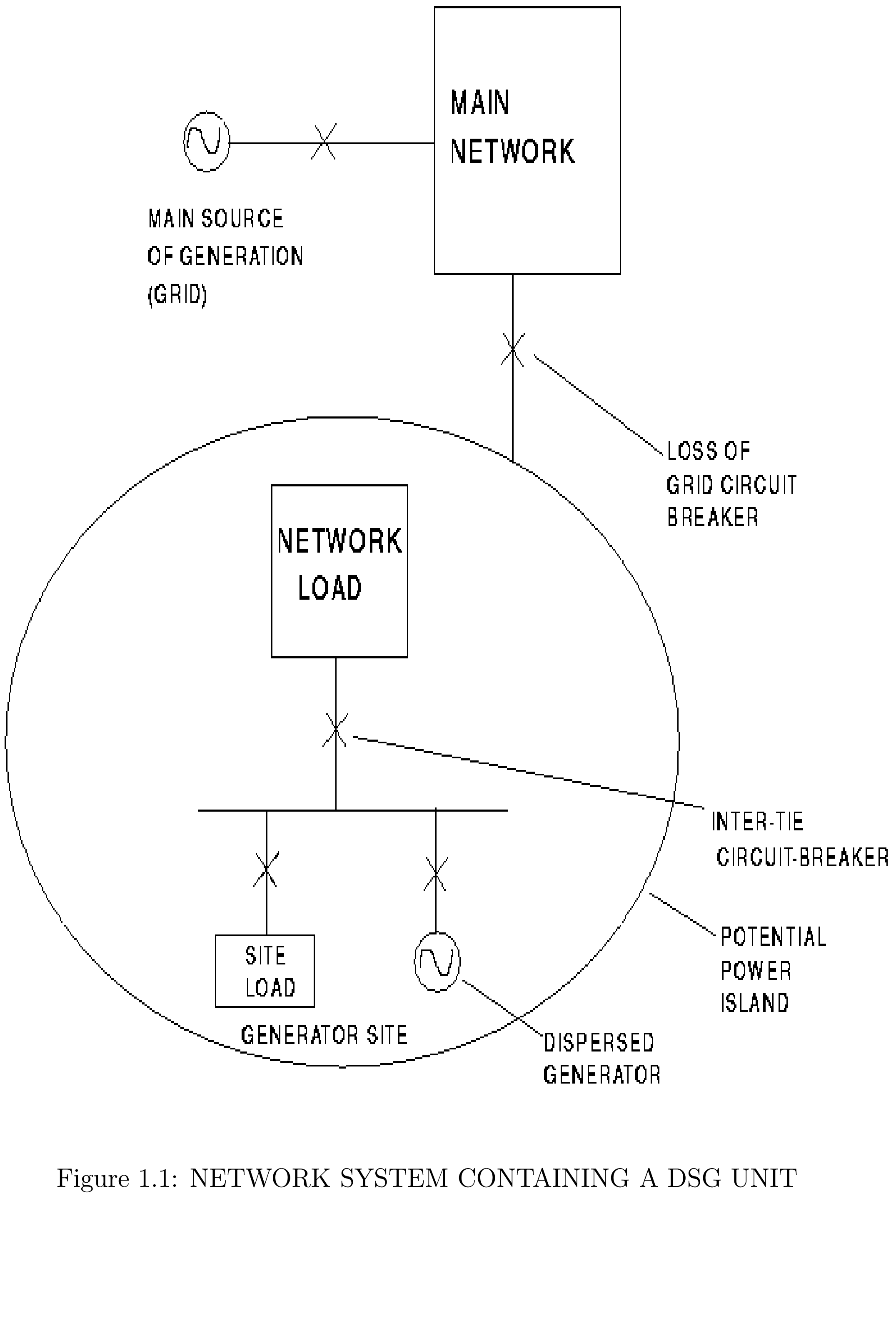

Figure 1.1: NETWORK SYSTEM CONTAINING A DSG UNIT

used by the Federal Energy Regulatory Commission (FERC) to describe power generation from renewable resources.[1] Available resources include hydro, wind, solar, biomass, and urban waste. Cogeneration refers to the simultaneous production of several forms of energy. Most often, cogeneration denotes electric power and process steam.

The Public Utility Regulatory Policies Act (PURPA) of 1978, incorporates a section on cogeneration which provides that the FERC shall prescribe rules requiring electric utilities to offer to sell power to, or buy power from qualifying cogenerators. Under this idea, an industrial cogeneration installation would be set up, for instance, to meet the entire plant process heat requirement as well as to generate as much electric power as process heat and fuel considerations allow. If the generated power exceeds the industrial demand, the excess power flows to the electric utility which pays for these kilowatts according to a negotiated power contract. In the past practices, recognized power flowed only into the industry (radial system).

In recent years, the increasing use of customer owned generators (cogenerators), independent power producers (IPPs) and feeder-to-feeder switching has resulted in the distribution system protection taking on more of the characteristics of transmission protection.[2] Cogenerators, and IPPs, establish energy sources within the distribution system that result in fault current that differ in magnitude and direction from the fault currents produced by the utility sources. The free-flowing nature of cogeneration requires that the relays at the distribution stations recognize the potential variations in short circuit current in the same way as the relays in a networked or looped system.

Electric energy generated by cogenerators and small power producers may be directly connected to a utility system, or indirectly connected through a device called a static power converter (SPC). Directly connected synchronous generators must run at a synchronous shaft speed so that the power output is electrically in sychronism with the utility system. Directly connected induction generators are asynchronous (not in synchronism). Indirect connection through a static power converter allows the electric energy source to operate independently of the utility voltage and frequency.

## 1.3 RESONANT OVERVOLTAGES

Shunt capacitors, connected between line and neutral are commonplace on power systems. Capacitors can correct lagging power factor conditions and can provide voltage support for the system. In some applications they are switched in and out quite frequently as the system load changes and the system voltage fluctuates. Overvoltages are never present in normal operations, because the source impedance of a normal power grid connection is too low and the source voltage is too well regulated to allow resonant conditions to occur. During faults, resonance can occur between the capacitor bank and the inductance of the dispersed generator to produce high voltages on the unfaulted phases.

The use of shunt capacitors for power factor compensation may result in resonance or near resonance.[3] The worst case of overvoltage due to resonance occurred when shunt capacitors were connected to the feeder at no load without a surge arrestor.[4] When load was cogitated, the resonant problem was damped. Overvoltage could be several times normal voltage if the generator becomes isolated in the faulted section of a three-phase feeder with capacitor bank and load less than or equal to the capacity of the dispersed generator.[4]

Surge arrestors are the principal defense against extreme overvoltages. There is some concern that standard distribution class arrestors may not be adequate to withstand the fault discharge energy, because it could be several cycles until the generator breaker opens. Surge arrestors will continue to fire every half cycle or at least every cycle as long as the DSG remained on the system during resonant conditions. If the arrestor does fire, it will only clip the voltage for a half cycle. The voltage can recover to sparkover on the following cycle. In some instances surge arrestors can increase the overvoltage frequency by ensuring that the following peak would also reach surge arrestor sparkover, because the arrestor will act like one more switch activating the resonant circuit.

## 1.4 SIGNIFICANCE OF TRANSFORMER ISOLATION

To provide isolation and to reduce harmonic injection, some utilities require that DSG units be connected to the utility system through a transformer. This transformer will be helpful in the case of a three-phase generator, where isolation of the zero-sequence network of the DSG from the zero-sequence network of the utility is necessary in order to properly protect the DSG from damage due to utility ground faults. The delta\wye connection is the most common arrangement, but delta\zig-zag and ungrounded-wye\delta\grounded-wye connections are also employed. A delta-connected winding on the utility side permits the DSG ground fault protection to be designed without restrictions based on the necessity of coordinating with the utility. There was not a transformer connection that completely eliminated the overvoltage conditions which could exist upon separation of a DSG unit with a small portion of the feeder system.[5]

A grounded-wye winding on the utility side significantly reduces the overvoltages produced during an island condition, particularly when the aggregate DSG capability in the island exceeds the load. This has the disadvantage of desensitizing the utility ground relays, since some ground fault current will be diverted from the substation breakers. A resistor or reactor may be connected in series with the wye ground connection in order to limit the flow of current during faults or unbalanced conditions. The actual value chosen for the resistor or reactor will be determined by the sensitivity of available protective relays and usually in practice it is assumed that the medium voltage system neutral is grounded under all circumstances.

A grounded-wye on the DSG side may cause dangerous overvoltages when a fault occurs between an energized phase conductor and ground, particularly if the substation transformer is the only zero-sequence current source on the circuit. Once the fault is detected and cleared by the opening the station breaker, the circuit is no longer effectively grounded.

## 1.5 TYPES OF DSG ENERGY SOURCES

The operation of DSG power processors can be categorized into line-independent and line-dependent devices, depending on whether they require an external source of excitation. Synchronous generators and self-commutated inverters are line-independent devices that can operate on stand-alone basis. Induction generators and line-commutated inverters are line-dependent devices that need an external source of reactive power. It is possible for shunt capacitors on a distribution feeder to supply enough reactive power to self-excite line-dependent DSG devices, even when the utility supply is not connected.

Synchronous generators are alternating current machines in which the rotational speed of normal operation is constant and in synchronism with the frequency of the electric utility system to which it is connected. Synchronous generators can be run either stand alone or interconnected with the utility. The advantages of synchronous generators are their ability to provide power during utility outages and flexibilty in permitting the DSG owner to control the power factor at his facility by adjusting the dc field current. The disadvantages of synchronous generators are their requirement for more complex control than induction generators, both to synchronize them and to control their field excitation. Synchronous generators also requires special protective equipment to isolate them from the utility under fault conditions.

Induction generators are asynchronous machines that requires an external source to provide the magnetizing current necessary to establish the magnetic field across the air gap between the generator rotor and stator. Without an external source, an induction generator cannot supply electric power but must operate in parallel with a utility, a synchronous machine, or a capacitor that can supply the reactive requirements of the induction generator. The advantages of induction generators are their need for only a very basic control system, since their operation is relatively simple. Induction generators do not require special procedures to synchronize them with the electric utility. They will normally cease to operate when a utility outage occurs. One of the disadvantages of induction generators is their ability to draw reactive power from the utility system and this may adversely affect the voltage regulation on the circuit to which they are connected. Another disadvantage is the potentially damaging inrush currents and associated torques that can result when connecting some types of induction generators to the utility at speeds significantly below synchronous speed.

In certain instances, an induction generator may continue to generate electric power after the utility source is removed. This condition is called self-excitation. It can occur whenever there is sufficient capacitance in parallel with the induction generator to provide the necessary excitation.

The Wind Energy Conversion Systems (WECS) electric generators are usually one of four types: induction, synchronous, variable frequency, or direct current. Induction generators are commonly used in small (less than 100 KW) to intermediate capacity (between 100 KW to 1 MW) capacity WECS. Synchronous generators are used mostly in sizes from small upward to large capacity (greater than 1 MW) WECS. In the case of variable frequency and direct current, the WECS needs a static power converter (SPC) to connect and regulate electric power into a form acceptable for interconnection to an electric distribution system.

## 1.6 PROTECTION OF THE DSG\UTILITY INTERFACE

There are four principles that should be considered when protecting the DSG\utility interface, regardless of the type of DSG. These are dependability, security, selectivity, and speed.

In order to meet dependability, the protective devices should have a very high probability of clearing faults that occur in the power system. In order to meet security, the protective devices should have a low probability of interrupting the circuit unnecessarily. In order to meet the selectivity, the protective devices should isolate only the faulted area of the power system. The protective devices should operate as rapidly as possible and be consistent with coordination requirement in order to minimize damage. A reliable relaying system must be dependable and secure. Most protection systems are designed for high dependability. As a relaying system becomes dependable, its tendency to become less secure increases. A majority of relay mis-operations are unwanted trips caused by insecure relay operations. This happens because of a bias toward making relays more dependable at the expense of some degree of security.

### 1.6.1 INVERSE TIME-DELAY OVERCURRENT

The principal application of overcurrent relays is on radial systems where they provide both the phase and ground protection. Overcurrents relays are also used in industrial systems, and on subtransmission lines, which cannot justify more expensive protection such as distance or pilot relays. Overcurrent protection involves the inclusion of a suitable device in each phase, since the objective is to detect faults which may affect only one or two phases. Where relays are used, they will usually be energized through current transformers.

A basic supplement of time-delay overcurrent relays would be two phase relays and one ground relay. This arrangement will protect the equipment for all combinations of phase and ground faults. This arrangement uses the minimum of relays. Complete backup protection can be obtained if a third phase relay is added.

The first step in applying inverse time-delay overcurrent relays is to choose the pickup setting of the relay, so that it will operate for all short circuits in its own line section. This setting should provide backup for the next line section or adjoining equipment. The pickup of a relay is defined as the minimum value of the operating current, voltage or other input quantity reached by progressive increases of the operating parameter, that will cause the relay to reach its completely operated state when started from the reset condition.

The time-delay setting of the relay is an independent variable that is obtained in a variety of ways, depending on the design of the relay. For an electromechanical relay, a time dial is provided that positions the moving contact relative to a fixed contact. The dial is marked from a setting of one-half to ten, fastest to slowest operating times respectively. The speed of the disc is determined by the operating current, and the time is determined by the distance the moving contact has to travel before it reaches the fixed contact. Other time-delay mechanisms include clock movements, bellows, or diaphragms, or electrical circuits using R-L-C timing circuits. In the electromechanical design, this adjustment is continuous, whereas in solid-state or digital relays this setting may be continuous or discrete.

For phase relays, it is necessary to establish a reasonable maximum load current and calculate the minimum fault current for the system configuration. The relay should then be set somewhere between twice maximum load current and one third of the minimum fault current value.

A ground relay must "see" all phase-to-ground faults within its zone of protection, and under conditions which give a minimum fault current. In calculating ground current ($3I_0$), it is the zero-sequence current that is of interest. There is no concern for load current, but normal phase and load unbalance and CT errors must be considered and the relays set above these values. The ground relay must be set between twice the "normal" ground current and one third of the minimum fault value. In the absence of any other information, the "normal" ground current may be taken to be 10% of the load current.

### 1.6.2 INSTANTANEOUS OVERCURRENT

In order to properly apply instantaneous overcurrent relays, there must be a significant reduction of short circuit current as the fault is moved away from the relay toward the far end of the line. The relay must be set not to overreach the bus at the remote end of the line and there still must be enough of a difference in the fault current between the near and far end faults to allow a setting for the near end fault.

The simple inverse time overcurrent relays cannot be used without backup from instantaneous relays because the closer the fault is to the source, the greater the fault current magnitude, and the longer the tripping time. The instantaneous relay must not see beyond its own line section, so the range of values for which it must operate are very much higher than even emergency loads. Load is not usually a consideration for the instantaneous relay setting.

### 1.6.3 UNDERVOLTAGE\OVERVOLTAGE

Low voltage prevents motors from reaching rated speed on starting or causes them to lose speed and draw heavy overloads. While the overload relays will eventually detect this condition in many installations, the low voltage may jeopardize production or affect electronic or digital controls. Motors should be disconnected quickly in this case. A time-delay undervoltage relay can detect these low voltages. These relays should be set at 0.9 per unit.[6]

The voltage at the terminals of a generator is a function of the excitation and speed. Overvoltage may result in thermal damage to cores due to excessive high flux in the magnetic circuits. Excessive flux saturates the core steel and flows into the adjacent structures causing high eddy current losses in the core and adjacent conductors. Severe overexcitation can cause rapid damage and equipment failure.

The unit of measure for excitation is defined as per unit voltage divided by per unit frequency (V\Hz). Overvoltage exists whenever the per unit voltage per Hz exceeds the design limit. Overvoltage exists at 105% of rated voltage and per unit frequency or per unit voltage and 95% frequency. Transformers are designed to withstand 110% of rated voltage at no-load and 105% at rated load with 80% power factor.[7]

For overvoltage protection, at least one overvoltage element per phase should be used to detect a condition that might damage other loads and equipment on the feeder. If only one relay is used it should be set at 1.1 per-unit.[6] If two relays are used per phase, the high speed unit should be set anywhere from 1.3 to 1.5 per unit and the slow speed unit at 1.1 per unit.[6]

If the system conditions are such that the islanded system is prone to ferroresonance, the voltage waveform at that time will be expected to be very rich in harmonics. At such time, it is possible that the peak voltage of the sinusoidal wave will be dangerously high even though the rms value of the same voltage remains in an acceptable range. The instantaneous overcurrent relay will be set at 1.3 to 1.5 per unit with a short or minimum time delay, especially at the higher value.

### 1.6.4 NEGATIVE-SEQUENCE

Unsymmetrical faults may produce more severe heating in machines than symmetrical faults or balanced three-phase operation. The negative-sequence currents which flow during these unbalanced faults induce rotor currents which tend to flow on the surface of the rotor forging and in the non-magnetic rotor wedges and retaining rings.[2] The $I^2R$ loss quickly raises the temperature. If the fault continues, the metal will melt, damaging the rotor.

For generators, the cause of system unbalance is very often the failure of the protection or equipment external to the machine. The typical conditions that can give rise to the unbalanced generator currents are: accidental single-phase tripping of the generator due to open leads or buswork, unbalanced generator step-up transformer, unbalanced system fault conditions and a failure of the relays or breakers, planned single-phase tripping without rapid reclosing. When such an unbalance occurs, it is not uncommon to apply negative-sequence relays, on the generator to warn the operator of this abnormal situation and allow corrective action to be taken before removing the machine from service. For negative-sequence protection, at least one relay should be used to compare the magnitude of the currents flowing in the three phases.

### 1.6.5 LOSS-OF-EXCITATION

Rotor faults may be either to ground or between turns and may be caused by the severe mechanical and thermal stresses acting upon the winding insulation. The field system is normally connected to ground so that a single-ground fault does not give rise to any fault current. However, a second single-ground fault would short circuit part of the field winding and thereby produce an asymmetrical field system, and unbalanced forces on the rotor. Such forces will cause excess pressure on bearing and shaft distortions, if not quickly removed. Under the general heading of rotor faults can be included loss-of-excitation. This may be caused by an open circuit in the main field or a failure elsewhere in the excitation system.

Loss-of-excitation results in a generator, losing synchronism and running above synchronous speed, since the power input to the machine is constant. The generator will operate as an induction generator under this condition, producing its main flux from a wattless stator current drawn from the power system to which it was still connected. Excitation under these conditions requires components of reactive current which may well exceed the rating of the generator and so overload the stator winding. The slip frequency currents induced in the damper windings of the rotor would cause abnormal heating of the rotor. Operation as an induction generator does not, therefore, damage the set immediately, but, as the higher ratings of modern machines are obtained by advanced cooling techniques rather than increased frame size, the short thermal time constants require the machine to be deloaded and tripped in a matter of seconds.[8] This condition should not be allowed to persist indefinitely and corrective action either to restore the field, or to off-load and shut down the machine should be done.

When a generator loses synchronism, the quantity which changes most is its impedance as measured at the stator terminals. Loss of field will cause the terminal voltage of the generator to begin to fall, while the current begins to increase. The apparent impedance of the machine will therefore be seen to decrease and its power factor changes. An offset impedance measuring relay will detect the change of impedance from the normal load value and provide protection against asynchronous operation resulting from the loss-of-excitation.

### 1.6.6 DIRECTIONAL GROUND FAULT

Stator fault involve the main current carrying conductors must be cleared quickly from the power system. They may be faults to ground, between phases or between turns of a phase, singly or in combination. The greatest danger from stator faults is the possibility of damage to the laminations of the stator core and stator windings due to the heat generated at the point of fault. If the damage so caused is other than trivial, the stator will be destroyed, the damaged laminations and windings replaced and the stator rebuilt, all of which is a lengthy and costly process. When a low value grounding resistance is used to limit ground-fault current to 200 to 300 amperes, or when the generator is grounded through a high resistance (limiting the maximum stator ground-fault current to approximately 10 amperes), the neutral ends of the stator windings are unprotected against ground faults (typically the bottom 5-10% of the windings) and, although ground faults near the neutral ends are unlikely to be caused by electrically stressing, faults due to mechanical stressing cannot be ruled out and 100% winding protection is required.

The stator ground fault relay will monitor the amplitude current in the generator neutral connection as drawn by the total system to ground capacitance (i.e. that of the generator stator). The effect of a ground fault on the windings would reduce the system impedance and so increase the level of injected current and trip the set if the deviation from the datum level exceeded a predetermined value. Ground-fault currents on rural networks may be of very low magnitude owing to long line lengths, the use of neutral grounding resistors, and different grounding conditions. Such circumstances arise when an overhead conductor breaks and falls on ground of high resistivity or a hedge or haystack.[9]

The residual voltage of the power system can be used to polarize directional relays. The residual voltage is the vector of the three phase voltages, and may be derived from an open-delta connected residual windings of three single-phase transformers. The residual voltage is zero for balanced phase voltages, but during a ground fault on a system with a solidly grounded neutral, it is equal to the voltage depression of the faulted phase, and is in this way related to the fault current. When the system neutral is grounded through a resistance, the fault current is less and does not cause as much collapse of the phase-to-neutral voltage. The line at the point of fault is brought to ground potential and the neutral point is raised above ground potential by the voltage drop on the grounding resistance due to the fault current flowing in it. This displacement of the neutral potential is added to the voltage on the two phases. This will cause the phase voltages to increase. The residual voltage is the vector sum of all the phase voltages and may be quite high, approaching three times the normal phase voltage. The residual voltage is the vector sum of all phase voltages and may be quite high, approaching three times the normal phase voltage. When the voltage transformer is not provided with a winding for open-delta connection, a residual voltage can be extracted by a wye-open delta connected group of auxiliary voltage transformers.

If the residual voltage is too low, or if voltage transformers are not available, or not suitable to supply residual voltage, the neutral current of a power transformer or of a grounding transformer can be used to polarize directional relays. The directional ground fault relay must have a low-impedance 'current' winding in place of the more usual 'voltage' winding. The neutral current of a wye-delta transformer will always flow from the ground into the system; residual current flows toward the fault and may pass through a group of line current transformer secondaries. The relay can therefore discriminate between faults in the alternate directions.

### 1.6.7 CHECK-SYNCHRONISM

There can exist the potentially disatrous scenario where the main source of supply could be reconnected to the power island out-of-synchronism with the dispersed storage and generation unit. Although the best practice is to avoid reclosing, it is very difficult for utilities to disable all reclosing close to the dispersed storage and generation unit. Should out-of-synchronism reclosure occur, there is a high probability that the dispersed storage and generation unit will be damaged.

Check-synchronism relaying is required for the intertie breakers to ensure that the generator is safely connected in parallel with the utility supply. These relays compare the relative phase angle difference between the voltage on both sides of an open circuit breaker to determine if the two voltages are in synchronism (that is, of the same phase relationships).[1] The check-synchronism relays will execute circuit breaker closing. When the utility circuit is deenergized, or the DSG is not synchronized with the utility, the relay will block closing.

### 1.6.8 DIFFERENTIAL

Differential comparison is a sensitive and effective method that can detect very small magnitude of fault currents. The protection scheme requires currents from the extremities of a zone of protection. This restricts the application of differential relays to power apparatus, such as transformers, generators, motors, buses, capacitors, reactors, etc.

Currents and voltages in the power system have a sinusoidal waveform of the fundamental power system frequency. During normal operations, generators may produce third order harmonic voltages and currents (deviations from a pure sinusoid). During abnormal conditions, energization of transformers create odd higher-order harmonics (fifth,seventh,ninth,etc.)

that are associated with transformer saturation. These abnormal conditions can be detected by sensing the harmonic content through filters in electromechanical or solid-state relays, or by calculation in digital relays. By selecting a proper weighting coefficient for the harmonic component, it is possible to prevent the differential relay from tripping the transformer during its energization, or tripping the generator during normal operations. A substantial tripping current may be produced due to the magnetizing inrush current.

### 1.6.9 FREQUENCY

Islanding is the abnormal condition where a small section of the utility's load network remains connected to the dispersed storage and generation unit with no main source of utility power (main power station, or a transmission supply point). Islanding can be caused by a switching operation to clear a fault, load shedding, maintenance outages, or equipment failure. However, for the islanding to persist, the load on the isolated section of the feeder must closely match the power output of the dispersed generator connected to the feeder. Continued energization of the isolated section of feeder by the dispersed generator may cause damage to the DSG device and distribution line equipment if the voltages on the isolated section of feeder are out-of-phase with the utility reference voltage at the instant of reclosing.

Islanding requires that the DSG device be capable of self-excitation. If a capacitor bank is insufficient to maintain excitation, then the voltage would gradually decrease as the DSG generator begins to accelerate, causing the generator to self-excite for several cycles. If a capacitor bank overexcites the DSG generator, then the voltage would gradually increase as the generator begins to decelerate, causing self-excitation of the generator. This suggests, that the generator's reactive power requirement could be matched well enough to sustain islanding for several seconds.

The principal objective for an islanding protection technique is to detect the condition where the DSG unit is left connected to a portion of the utility's load network, but disconnected from the main source of utility power following a switching operation. When an islanding condition is detected, the protection should trip the inter-tie breaker between the generator's site and the utility. Once this has been done the presence of the DSG unit will not impede the orderly restoration of the utility supply to the rest of the network. Since the inter-tie breaker is used to connect two active systems, once the network supply has been established, the DSG unit can be reconnected to the utility.

Reduced frequency results in reduced current; therefore operation at reduced frequency should be at reduced KVA.[2] Operating precautions should be taken to stay within the short time thermal rating of the generator rotor and stator.[10] Underfrequency is a system condition that affects the turbine more than the generator, because the turbine develops mechanical resonant stresses as a result of deviations from synchronous speed.

System load shedding is considered the primary turbine underfrequency protection because appropriate load shedding will cause system frequency to return to normal before the turbine trouble-free limit is reached. The amount of load shed varies with coordinating regions and individual utilities but varies from 25% to 75%.

For underfrequency protection, at least one underfrequency element per phase should be used to detect those cases where the self-excitation voltage of the dispersed generator is between 0.9 and 1.1 per unit, and the feeder load causes a decrease in frequency. This relay would normally be set at 59.5 Hz.[7]

For overfrequency protection, at least one frequency element per phase should be used to detect those cases where the self-excitation voltage of the dispersed generator is between 0.9 and 1.1 per unit, and the feeder load causes an increase in frequency. The relay should be set at 60.5 Hz, and would operate on the resulting increase in frequency.[7]

### 1.6.10 DIRECTIONAL POWER

Directional power relays are often used to alarm or trip the DSG when power flows into it. This is useful, of course, only for cogenerators where the electrical loads are normally much larger than the generated power. In this case, power into the dispersed generator indicates an abnormal condition, such as the cogenerator having been islanded with the utility load.

Directional power relays can be used to trip turbine generators to prevent motoring of the generator. Generator motoring would occur if, for example, the turbine is tripped but the generator breaker does not. Generator motoring would cause damage to the prime mover such as turbine overheating in steam turbines and cavitation due to low flow in hydro turbines. These relays are sometimes used as backup to other relays in this situation, such as a steam turbine or exhaust hood in a gas turbine.

Directional power relays operate under conditions of balanced load and relatively high power factor. The voltage and current phasors are nearly in phase with each other and of constant magnitude. These connections were chosen so that maximum torque in the relay occurs under power factor load. The relay will then pickup for power flowing in one direction and will reset for opposite direction of power flow.

### 1.6.11 SELF-EXCITATION

Self-excitation is a transient condition in which dangerously high off-nominal frequency voltages are produced at the energized capacitor-bank or load when the utility breaker is opened. The transient voltage phenomena occurring during energization of shunt capacitors are due to prestrike and temporary interruption in the switching device of inrush currents. The initial energization of the capacitor bank circuit will nearly always take place as a prestrike across the contacts of the switching device prior to actual contact engagement.[5]

A prestrike is the tendency for the interconnect gap to breakdown and establish current before the contacts physically touch, because of the voltage stress between them.[5] Self-excitation can occur in both isolated bank switching as well as bank-to-bank switching.

Upon energizing an isolated capacitor bank (a single capacitor without other energized capacitor banks on the bus), moderate inrush transients and severe voltage transients can be generated. Depending on relative magnitude and distribution of lumped and inherent L and C elements in the circuit, the overvoltage transients may reach substantial levels in case of momentary interruption of the initial inrush current followed by a subsequent prestrike.

When energizing a capacitor bank with other energized capacitor banks on the bus (bank-to-bank switching), current transients can be more severe; however, the voltage transients are soothed to some degree by the support given the bus voltage by the energized capacitor banks. During a bank-to-bank switching operation, the first rapid transient brings about an exchange of charge between the capacitors being switched. The capacitors are brought to a common voltage since losses damp out the transients in practical installations. The common voltage is different from the supply voltage, so a second transient follows during which the two capacitor banks are restored to supply potential. The initial inrush transient and the restoring transient can usually be treated separately, since the first is usually completed before the second has really begun.

# Chapter 2

# SELF-EXCITATION

## 2.1 SWITCHING OF CAPACITOR BANKS

At the instant of energization, a fully discharged capacitor bank looks like a short circuit and, therefore, causes a collapse of bus voltage. The capacitor bank charges then to the system voltage in an oscillatory manner (with a possible overshoot). The frequency of these off-nominal oscillations depends on the effective system inductance and the capacitance of the bank (400 Hz-1000 Hz). Peak voltages of 2.0 per unit can be theoretically expected for grounded banks, and 2.5 per unit for ungrounded banks. Lower voltage will, however, result in practice due to system damping. If the switching device is slow, and contains a good interrupting medium, it is possible for the device to prestrike, extinguish the arc at one of the high frequency current zeros, and strike again with the capacitor charged in opposite polarity to the system with even higher resultant overvoltages.

## 2.2 METHODS FOR CONTROLLING OVERVOLTAGES

There are three common methods for reducing transient phenomena on capacitor bank switching. The surge impedance of the capacitor can be increased by adding a series reactor. Resistance can be added to damp the oscillation. Another method is to use synchronous switching which means closing the switching device at a selected point on the cycle.

When pre-insertion impedances are used to control inrush currents, there are two transient periods. The first occurs when the capacitor bank is initially energized through the pre-insertion impedance. The second occurs when the pre-insertion impedance is removed from the circuit. The voltage transients associated with the initial energization of a capacitor bank through a pre-insertion impedance are much greater than the voltage transients associted with the shorting out of the pre-insertion impedance. The first voltage transient is driven by the full system voltage, while the second transient voltage is driven only by the voltage drop across the pre-insertion impedance, typically on the order of 10% to 40%

of full system voltage.

### 2.2.1 UNCONTROLLED ENERGIZATION

At the instant of capacitor-bank energization, the bus voltage abruptly falls to zero, since the capacitor bank instantaneously appears as a very low impedance. This abrupt change of voltage injects a step-voltage wave into the transmission lines connected to the capacitor-bank bus. After the initial drop to zero voltage, the phase voltage recover in a transient oscillatory fashion. The frequency of this transient is due primarily to the surge impedance of the transmission lines connected to the capacitor-bank bus. Some additional damping may come from system load connected at the capacitor-bank bus. Because this transient is under-damped, the transient voltage overshoots the source voltage.

### 2.2.2 PRE-INSERTION RESISTOR METHOD

Forty-ohm pre-insertion resistors have been in service many years on high-voltage switches used to switch shunt capacitor banks. The forty-ohm resistor value was chosen to provide optimum control of inrush currents in back-to-back switching. A lower resistance will cause an increase in the inrush current during the first transient period. An increase in resistance will increase the inrush current during the second transient period. Forty ohms is an optimal value, yielding approximately the same inrush current for both periods.

Should the emphasis for the use of a pre-insertion resistor be placed solely on the control of overvoltages, disregarding inrush currents, a higher resistance value would be more appropriate.

When a 40-ohm pre-insertion resistor is inserted during the energization of the capacitor bank, the bus voltage at the capacitor bank does not collapse to zero. The extent to which the bus voltage collapses depends upon the ratio of the resistance of the pre-insertion resistor to the resultant surge impedance of the transmission lines connected to the capacitor-bank bus. The reduction in the collapse of bus voltage manifests itself as a reduction in the step-voltage wave (approximately 63% of the magnitude for a phase-to-ground voltage compared to the uncontrolled energized case) injected into the system. The pre-insertion resistor results in the capacitor-bank transient oscillation being nearly critically damped, so that very little overswing of the capacitor-bank voltage occurs. During the transient period, there are small discontinuities in the bus voltage at the capacitor-bank. These discontinuities occur because traveling waves returning from the remote bus will see the 40-ohm pre-insertion resistor rather than the very low surge impedance of the capacitor bank.

There are three advantages of pre-insertion resistors. They yield the lowest phase-to-ground voltages at the capacitor-bank. They also yield moderate phase-to-ground and phase-to-phase voltages at remote, radially fed stations. The overvoltages are not significantly affected by the system load, because the load does not signnificantly add to the damping obtained from the resistor.

There are two disadvantages of pre-insertion resistors. The resistors are not effective in reducing the very high rate of change of voltage associated with the energization of a capacitor bank. Additionally, pre-insertion resistors may have thermal-capability limitations. Typically, pre-insertion resistors absorb large amounts of energy with each energization of a capacitor bank. Depending upon the type of switching device utilized, the energy capability of the resistor can limit the size of the capacitor bank which can be switched, as well as the frequency of switching operations.

## 2.3 PRE-INSERTION INDUCTOR METHOD

Pre-insertion inductors are favorable over the fixed inductors. Fixed inductors generally must be designed to carry normal load current, to have a system BIL rating, and to withstand system short-time currents. As a result, the fixed inductors are physically large, relatively expensive, and may require costly mounting structures. Furthermore, the fixed inductors losses add to the cost of utilizing the inductors. Because the pre-insertion inductor is only inserted for a few cycles, the normal current, short-time current, and full BIL ratings are not required.

An inductor of 10 milihenries was chosen for the pre-insertion inductor, such that it would yield approximately the same initial inrush current as that for a 40-ohm pre-insertion resistor. Because the impedance of the pre-insertion inductor is significantly lower at supply system frequency, the inrush currents experienced during the second transient period will be significantly lower than the initial inrush currents. There is a possibility that the pre-insertion inductor could be further optimized for both inrush current control and overvoltage control. Further reductions in inrush currents, as well as overvoltages, could be obtained by a pre-insertion inductor with a higher inductance. Higher inductances used to optimize inrush current and voltage control may be economically feasible.

When a 10-mH pre-insertion inductor is inserted during the energization of the capacitor-bank, the extent to which voltage collapses at the capacitor-bank is reduced (approximately 54% of the magnitude for a phase-to-ground voltage and approximately 51% of the magnitude for a phase-to-phase voltage compared to the uncontrolled energization case). Because the inductor has a very high surge impedance relative to the surge impedance of the lines connected to the bus, there is no abrupt step change in bus voltage. The voltage initially decays exponentially as determined by an LR circuit comprised of the inductance of the pre-insertion inductor and the surge impedance of the lines connected to the capacitor-bank bus. Since the pre-insertion inductor has a relatively small resistance, the transient oscillation is not significantly damped as it is with the pre-insertion resistor. Also, transients generated when energizing a capacitor-bank through an inductor generally have moderately rising ramp voltages instead of fast-rising step voltages.

There is a significant benefit to the fact that the overvoltages produced by the use of a pre-insertion inductor are characterized by a ramp function rather than a step function as produced by an uncontrolled energization or energization through a pre-insertion resistor. Step-rising wave forms may cause damaging internal resonances in transformers.[11] For a steeply rising transient voltage wave, the voltage distribution across a transformer winding will be initially determined by stray capacitances rather than the inductance of the winding, creating stress concentrations in the first several turns of the winding.[12] Even without high peak overvoltages, rapid changes in voltage, with their associated stress concentrations, may be harmful to transformers. The lower rate of change of voltage produced by the pre-insertion inductor will allow the voltage to be distributed more evenly across the initial turns of the wingings of the transformers.

The main disadvantage of dealing with a fixed inductor or a pre-insertion inductor is that the peak overvoltages are significantly affected by the system load during energization. This is primarily because the peak overvoltages in this case are associated with slowly rising ramp waveforms. The slow-changing transients can interact with loads which are located beyond the leakage impedances of transformers.

### 2.3.1 SYNCHRONIZED CLOSING METHOD

Synchronous energization of a capacitor-bank can be an extremely effective means of controlling overvoltages. To accomplish synchronous closing at or near a voltage zero, thereby avoiding high prestrike voltages, it is necessary to apply a switching device which maintains a dielectric strength sufficient to withstand system voltage until its contacts touch. Such ideal closing characteristics may be difficult to attain with present-day high-voltage switches and circuit breakers. If the switching device has a closing consistency within (plus or minus) 2 milliseconds, overvoltage magnitudes will be limited to acceptable values.[13]

Synchronous closing has the potential of being an ideal means of controlling overvoltages and inrush currents associated with energizing capacitor banks. With the constraints of practical switching devices, an ideal synchronized closing of capacitor-banks is unattainable. Performance of a controlled closing system, operating within an accuracy attainable with present-day technology, will allow a degree of control of overvoltage similar to that obtainable with a pre-insertion resistor, without the thermal disadvantage of the pre-insertion resistor.

There is a fundamental difference between grounded and ungrounded systems when applying controlled synchronized closing. In a grounded system, closing of each phase should occur at a phase-to-ground voltage zero, assuming an uncharged capacitor bank is being energized. In an ungrounded system, energization of the first phase can occur at random.

The second phase should be closed when the phase-to-phase voltage between the second phase and the first phase is zero, which occurs when both phase-to-ground voltages are of the same polarity and have a magnitude of one-half per unit. The third phase is then closed when its phase-to-ground voltage is zero. The optimum time for energizing a capacitor bank, therefore, is different for an ungrounded system than for a grounded system. The overvoltages which result from errors in closing, i.e., not closing at the ideal time, are also fundamentally different for grounded systems and ungrounded systems.

If the capacitor-bank neutral is grounded in a ground supply system, all three phases can act independently. Thus, simultaneous closing of two phases should be highly unlikely. With closely coupled phases on a grounded system, the transients experienced when one phase is energized may induce a second phase to prestrike, thereby creating a simultaneous closing of two phases.

If the capacitor bank or the supply system is ungrounded, a simultaneous energization of the first two phases will occur. Under this assumption that the first two phases close simultaneously, the overvoltages associated with the simultaneously closed phases are the same for a grounded and an ungrounded capacitor-bank.

Peak overvoltages are not significantly affected by the system load during energization through synchronous closing. The peak overvoltages in this case are associated with very steeply rising waveforms. The fast-changing transients cannot interact with loads which are located beyond the leakage impedances of transformers.

The main disadvantage of a controlled closing scheme is its complexity. Since the ideal time to close is different for each case of the three phases, the switching device must have three independent closing mechanisms or an accurate mechanical delay between the three poles. The scheme also relies on accurately monitoring voltages, both at the capacitor-bank bus and on the capacitor-bank. The reliability of controlled closing schemes may be less than for a pre-insertion impedance control means.

## 2.4 OTHER METHODS OF CONTROLLING OVER-VOLTAGES

It has been suggested that contact closing speed of the switching device could be an effective means for overvoltage control. It is, of course, possible to avoid interruption of the low frequency inrush current by giving the switch a contact speed such that the prestrike contact gap is closed within one loop of the low frequency inrush current. The higher contact speed, while curing one cause of overvoltages, may complicate the contact rebound problem and thus enhance one of the conditions that were given as possible causes of the energization surge. Moreover, even if the “cure” were acceptable, it could only be helpful without the limits of an upper value of low frequency. Among the thousands of capacitor installations in this country alone, there will be always some with a higher value than such a limiting low frequency. Consequently interruption of the prestrike current and voltage escalation by subsequent prestriking could still take place. The increased contact speed of the switching device is no cure at all for the interruption of the high frequency inrush current. This method is not totally acceptable.

Choice of contact material can increase the dielectric strength and reduce the prestrike gap so as to avoid a current zero prior to physical contact engagement. This method is not a totally acceptable solution.

## 2.5 SELF-EXCITATION METHOD

### 2.5.1 DISCRETE FOURIER TRANSFORM (DFT)

The DFT is a basic operation used in many different signal processing applications to transform an ordered sequence of data samples, usually from the time domain into the frequency domain, so that spectral information about the sequence can be known.[14] The DFT is an ordered sequence of complex numbers, each number consisting of a real part and an imaginary part. Suppose that a real data sequence consisting of N real samples of a signal x(t), given by:

$$[x_k] = [x_0, x_1, x_2, ...x_{N-1}] \tag{2.1}$$

In this notation, k is usually a time index and ranges from 0 to N-1. When $[x_k]$ is a real data sequence, the computed DFT of $[x_k]$ consists of $(\frac{N}{2}+1)$ complex samples (N is even for simplicity), given by:

$$[X_m] = DFT[x_k] = [X_0, X_1, X_2, ...X_{\frac{N}{2}}] \tag{2.2}$$

$[X_m]$ is the DFT of $[x_k]$ and the index m designates the frequency of each component $[X_m]$. Since the DFT is complex, each $[X_m]$ can be separated in polar form as $X_m = |X_m|\mathrm{e}^{j\Theta_m}$. In this notation, $|X_m|$ is the amplitude of $X_m$, and a plot of $|X_m|$ versus the frequency index m is called the amplitude spectrum of $[x_k]$.

### 2.5.2 FAST FOURIER TRANSFORM (FFT)

The Fast Fourier Transform (FFT) is an algorithm for computing the DFT, and its output are precisely the same set of complex values expressed in (2.2).[15] The FFT algorithm eliminates most of the separated complex products in the DFT, so the execution time is much shorter. The ratio of computing time is approximately:

$$\frac{\text{FFT Computing Time}}{\text{DFT Computing Time}} = \frac{\log_2 \times \mathrm{N}}{2N} \tag{2.3}$$

Using the FFT, one can also do the computation "in place" so that $[X_m]$ in (2.2) replaces $[x_k]$ in (2.1), with only a limited amount of auxiliary storage needed for work space. However, the FFT algorithm is more complicated than the DFT and becomes lengthy when N, the number of data samples, is not a power of two. DFT routines requires twice as much storage as the FFT routines, because the FFT routines all replace data values with the transform values, whereas the DFT routines do not.

### 2.5.3 FORWARD TRANSFORM AND FREQUENCY INDEX

The forward transform is the DFT (or FFT), which transforms $[x_k]$ in (2.1) into $[X_m]$ in (2.2). The relationship implemented by the forward transform between $[x_k]$ and $[X_m]$ can be expressed as:

$$X_m = \sum_{k=0}^{N-1} x_k \times \mathrm{e}^{\frac{-j(2\pi \times k)}{N}} \tag{2.4}$$

where m = 0,1,...,$\frac{N}{2}$ and N is the even number of data samples.

In this formula for $[X_m]$, the exponential function, $\mathrm{e}^{\frac{-j(2\pi \times k)}{N}}$, is a complex sinusoid and is periodic. If $\mathrm{e}^{\frac{-j(2\pi \times k)}{N}}$, is thought of as a function of k, the time index, then its period is seen to be $\frac{N}{m}$; that is, when k goes through a range of $\frac{N}{m}$, $\mathrm{e}^{\frac{-j(2\pi\times k)}{N}}$ goes through one cycle. When the real and imaginary parts of (2.4) is separated this is easier to see.

$$X_m = \sum_{k=0}^{N-1} x_k \cos(\frac{2\pi \mathrm{m} \times \mathrm{k}}{N}) - j\sum_{k=0}^{N-1} x_k \sin(\frac{2\pi \mathrm{m} \times \mathrm{k}}{N}) \tag{2.5}$$

Thus, each part (real or imaginary) of each DFT component $[X_m]$ is a correlation (summed part) of the data sequence $[x_k]$ with a cosine or sine sequence having a period of $\frac{N}{m}$ data sequences.

## 2.5.4 THEORY OF SELF-EXCITATION

In the self-excitation method, a real sequence consisting of N real sample voltages, $[x_0, x_1, x_2, ...x_{N-1}]$ will be transformed into a complex function of frequency voltages. The digital relay will determine if the off-nominal frequency voltages violate its criteria. The details for the digital relay operation of these functions will be discussed in Section 3.12.

In Brigham's method, x(t) is a complex sequence of $\frac{N}{2}$ points: [15]

$$[x_0, x_1, x_2, ...x_{N-1}] = [v_0, v_1, v_2, ...v_{\frac{N}{2}-1}] \tag{2.6}$$

In the complex sequence $[V_k]$, the real part of $v_0$ is $x_0$. The imaginary part of $v_0$ is $x_1$. The real part of $v_1$ is $x_2$, and so on. The SPFFTR algorithm (see Appendix) is then used to take the $\frac{N}{2}$-point transform of $[v_k]$, replacing $[x_k]$ with $[V_m]$.

$$[V_m] = [V_0, V_1, V_2, ...V_{\frac{N}{2}-1}] \tag{2.7}$$

The desired transform $[X_m]$ can then be obtained from $[V_m]$ in accordance with the following formulas, which are expressions of Brigham's formulas:

$$\mathrm{X_m} = (\frac{(1 - \mathrm{U_m})}{2} \times \mathrm{V_m}) + (\frac{(1 + \mathrm{U_m})}{2} \times \mathrm{V^*_{\frac{N}{2}-m}}) \tag{2.8}$$

$$\mathrm{X^*_{\frac{N}{2}-m}} = (\frac{(1 + \mathrm{U_m})}{2} \times \mathrm{V_m}) + (\frac{(1 - \mathrm{U_m})}{2} \times \mathrm{V^*_{\frac{N}{2}-m}}) \tag{2.9}$$

The formulas of equation (2.8) and equation (2.9) are needed in SPFFTR (see Appendix) because $[X_m]$ replaces $[V_m]$ during the computation. Thus $X_0$, and $X_{\frac{N}{2}-1}$ replace $V_1$, and $V_{\frac{N}{2}-1}$, and so on. Also, since $[V_m]$ is a $\frac{N}{2}$-point transform, $V_{\frac{N}{2}}$, which is not given in equation (2.7) must equal $V_0$ in accordance with equation (2.5). Thus, SPFFTR accomplishes the transformation of $[x_k]$ using equations (2.8) and (2.9).

## 2.6 EVALUATION OF THE CONTROL METHODS

To varying degrees of efficiency, all control methods will limit peak switching-surge voltages, both at the local capacitor-bank station and at remote stations. Without a control means, the phase-to-phase switching overvoltage could be as high as 6.5 per unit.[16] Adding resistance to the capacitor-bank bus will reduce this overvoltage to 4.0 per unit. Adding inductance to the capacitor-bank bus will reduce this overvoltage to 3.3 per-unit. Synchronized closing of the switching contacts can theoretically reduce this overvoltage to 1.7 per unit.[16] The self-excitation relay will monitor the amplitude spectrum of the off-nominal frequency voltages. If their per unit values are larger than 1.7 per unit, then it will trip the cogeneration facility.

# Chapter 3

# DSG RELAY FUNCTIONS

## 3.1 OVERALL FUNCTION DIAGRAM FOR THE DSG RELAY

The overall function diagram for the multifunctional digital relay is shown in Figure 3.1. Ia, Ib, Ic, In, Va, Vb, and Vc are sample data of a phase A neutral current. The surge filter will prevent equipment damage and loss of data from power surges. There are two scenarios that will exist after the application of the surge filters. The traditional low frequency functions are executed on the left hand side. The new high frequency function is executed on the right hand side. On the left hand side, the anti-aliasing low-pass filters will remove unwanted harmonic content above the cut-off frequency [1] to accommodate a sampling rate of 12 times per cycle. Digital anti-aliasing filters will reduce the bandwidth of the input signals. The Analog-to-Digital converter will convert analog voltages and currents to their digital representation. The sampling clock provides pulses at a sampling frequency. Sampling frequencies in modern digital relays vary between 8-32 times the fundamental power system frequency. A recursive phasor computation will be done in order to determine the voltage and current phasors. A start-up block will be used to prevent the relay from mis-operating, during its first two cycles. The low frequency relays will accept the phasors as inputs, process them digitally, and make a decision resulting in an output signal. The relay will provide a meter reading for each input. The metering output will consist of total and phase apparent power, real power, and reactive power flowing in the lines. The voltages and currents will be measured as well. On the right hand side, the relaying program will use a high sampling-frequency on all phase voltages. The high-frequency sampler will produce the desired frequency response for estimating high frequency transients. The sampling A/D converter and clock will convert the analog current and voltage signals to their digital form. A start-up block will be used again to prevent relay tripping during the first two cycles. The high frequency relay will monitor the amplitude spectrum of the high frequency voltages produced at the capacitor and load.

[1]The cut-off frequency is equal to one-half the sampling rate.

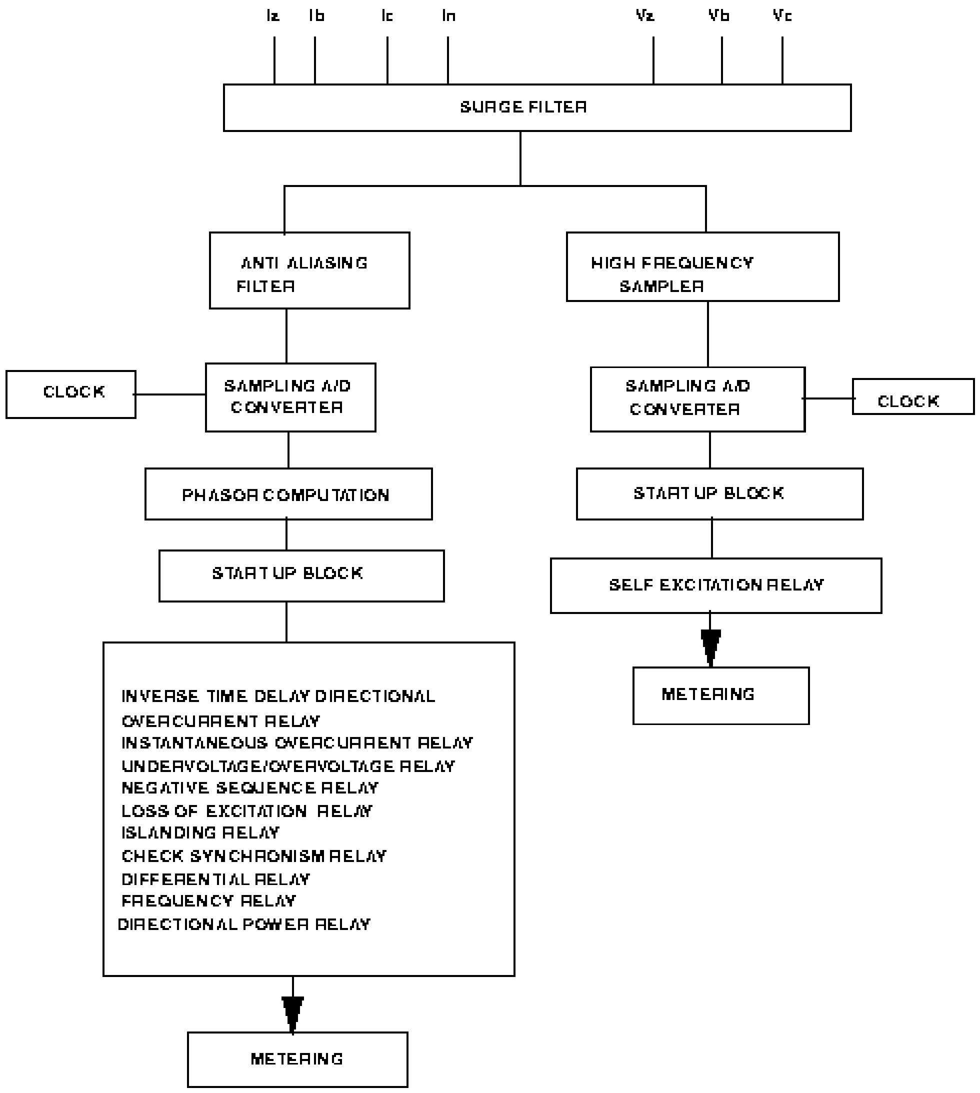

Figure 3.1: OVERALL FLOW DIAGRAM FOR THE DSG RELAY

### 3.1.1 HIGH-FREQUENCY SAMPLER

### 3.1.2 ANTI-ALIASING

An ideal anti-aliasing filter characteristic with a cut-off frequency $f_c$ is shown in Figure 3.2. A practical filter can only approximate the square shape, as shown by the dotted line in Figure 3.2. The primary purpose of digital filtering is to alter the spectral information contained in an input signal $x_k$, thus producing an enhanced output signal $y_k$.

$$y_k = \sum_{n=0}^{L} b_n x_{k-n} \tag{3.1}$$

In (3.1), the $[b_n]$ coefficients are generated by the SPFIRL algorithm (see Appendix). The low-pass FIR filter design procedure will include the window function w(n):

$$b_n = h(n) = h_d(n)w(n) \tag{3.2}$$

where $0 \leq n \leq L$

$$h_d(n) = \frac{\sin(\Omega_c(n - \frac{L}{2}))}{\pi(n - \frac{L}{2})} \tag{3.3}$$

The $b_n$ coefficient are used to implement the difference equation in (3.1), $h_d(n)$ is the ideal impulse-response sequence in (3.3), and $w(n)$ is any window function. The magnitude response of the resulting filter will depend upon the shape of the window w(n).

The low-pass FIR filter designed via the rectangular window is shown in Figure 3.3. The approximate realizable characteristic is shown by the dotted line. The cut-off frequency $f_c$ was 360 Hz. The sample interval (T) is equal to 0.001 sec. The normalized cut-off frequency must be between 0.0 and 0.5 Hertz − sec. This value is computed as FCN = $\mathrm{f_c T}$. The normalized cut-off frequency (FCN) is equal to 0.36 Hz. The length of the filter (L) is equal to 12.

### 3.1.3 ANALOG-TO-DIGITAL CONVERTERS

The Analog to Digital Converter (ADC) converts an analog voltage or current level to its digital represention. The most important feature is its word length expressed in bits. This will affect the ability of ADC to represent the analog signal with a sufficiently detailed digital representation.

Consider an ADC with 12 bit word length, which is the most common word length commercially available today.[17] Using a two's complement notation, the binary number (0111 1111 1111 97ff in hexadecimal notation) represented by a 12 bit ADC, while 1000 0000 0000 (800 in hexadecimal notation) represents the smallest (negative) number. In decimal notation, hexadecimal 7FF is equal to $(2^{11} - 1) = 2047$, and hexadecimal 800 is equal to

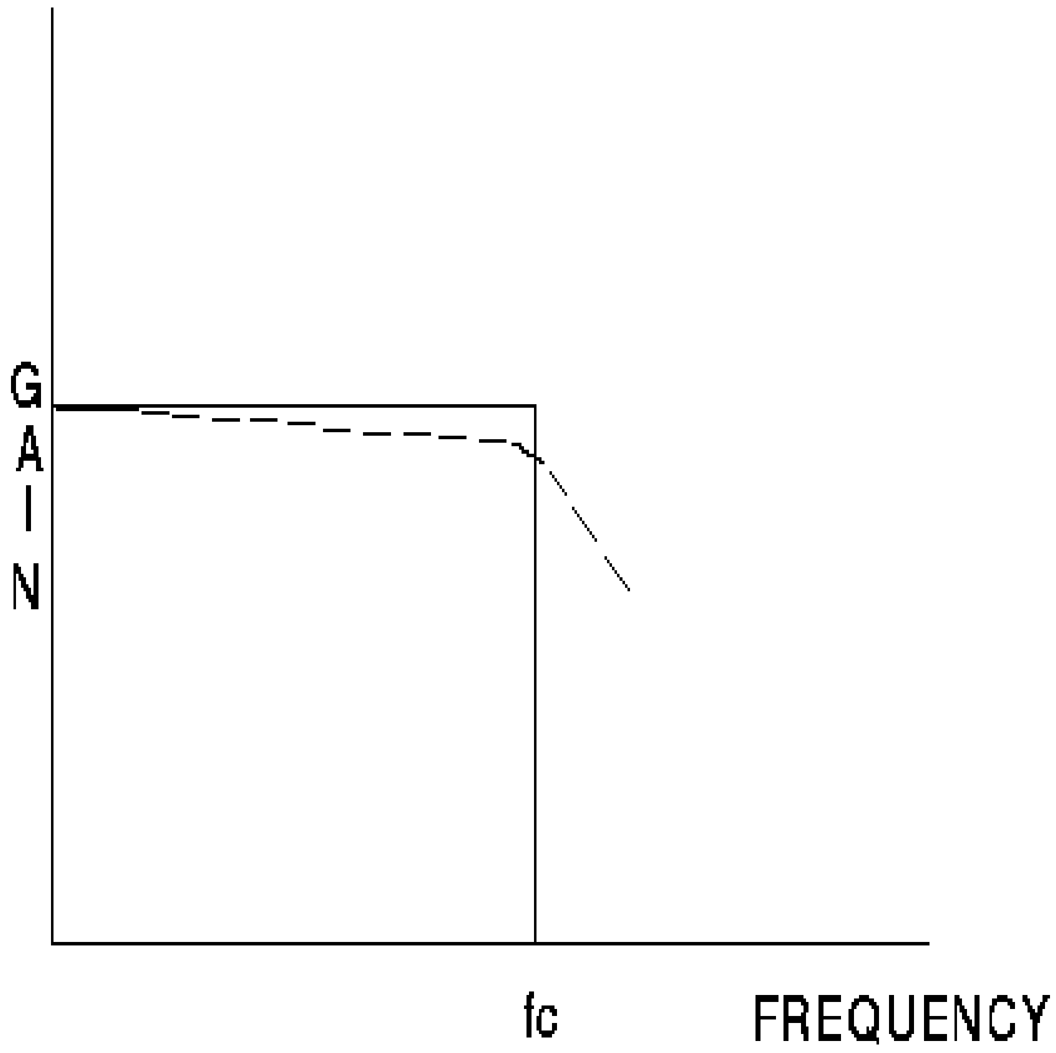

Figure 3.2: IDEAL ANTI-ALIASING FILTER

$-2^{11} = -2048$. If the analog input signal ranges between $\pm 10$ volts, then each bit of the 12 bit ADC word represents $\frac{10}{2048}$ volts or 4.833 millivolts. Thus any input voltage can be converted into its corresponding two's complement or an equivalent decimal.

## 3.2 PHASOR COMPUTATION

The flow diagram of the phasor computation is shown in Figure 3.4. First, the voltage and current waveforms were sampled. Second, the real and imaginary components of the phasors were calculated at the k-th instant as follows:

$$\mathrm{Var[k]} = \frac{2}{\mathrm{N}} \times \sum_{\mathrm{k}=0}^{11} \mathrm{v1y} \times \cos(\mathrm{k} \times \frac{\pi}{6}) \tag{3.4}$$

$$\mathrm{Vbr[k]} = \frac{2}{\mathrm{N}} \times \sum_{\mathrm{k}=0}^{11} \mathrm{v2y} \times \cos(\mathrm{k} \times \frac{\pi}{6}) \tag{3.5}$$

$$\mathrm{Vcr[k]} = \frac{2}{\mathrm{N}} \times \sum_{\mathrm{k}=0}^{11} \mathrm{v3y} \times \cos(\mathrm{k} \times \frac{\pi}{6}) \tag{3.6}$$

$$\mathrm{Iar[k]} = \frac{2}{\mathrm{N}} \times \sum_{\mathrm{k}=0}^{11} \mathrm{i1y} \times \cos(\mathrm{k} \times \frac{\pi}{6}) \tag{3.7}$$

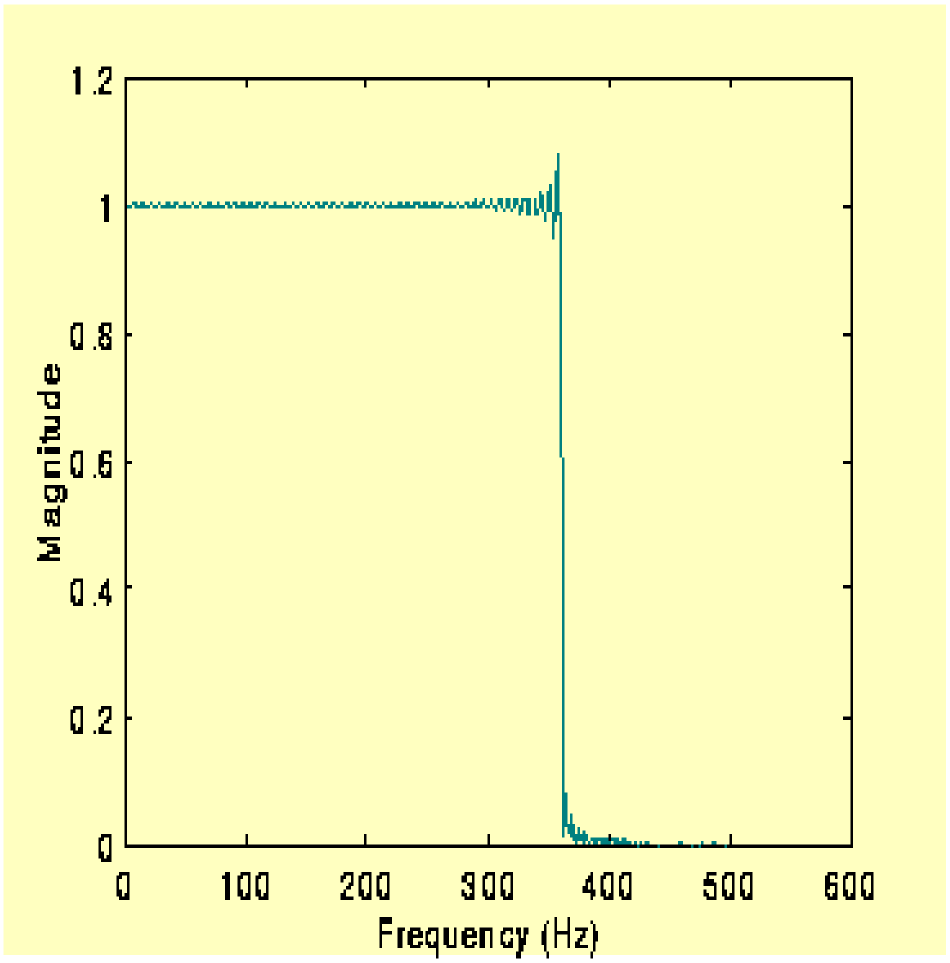

Figure 3.3: LOW-PASS RECTANGULAR FIR FILTER

$$\mathrm{Ibr[k]} = \frac{2}{\mathrm{N}} \times \sum_{\mathrm{k}=0}^{11} \mathrm{i2y} \times \cos(\mathrm{k} \times \frac{\pi}{6}) \tag{3.8}$$

$$\mathrm{Icr[k]} = \frac{2}{\mathrm{N}} \times \sum_{\mathrm{k}=0}^{11} \mathrm{i3y} \times \cos(\mathrm{k} \times \frac{\pi}{6}) \tag{3.9}$$

$$\mathrm{Vai[k]} = \frac{2}{\mathrm{N}} \times \sum_{\mathrm{k}=0}^{11} \mathrm{v1y} \times \sin(\mathrm{k} \times \frac{\pi}{6}) \tag{3.10}$$

$$\mathrm{Vbi[k]} = \frac{2}{\mathrm{N}} \times \sum_{\mathrm{k}=0}^{11} \mathrm{v2y} \times \sin(\mathrm{k} \times \frac{\pi}{6}) \tag{3.11}$$

$$\mathrm{Vci[k]} = \frac{2}{\mathrm{N}} \times \sum_{\mathrm{k}=0}^{11} \mathrm{v3y} \times \sin(\mathrm{k} \times \frac{\pi}{6}) \tag{3.12}$$

$$\mathrm{Iai[k]} = \frac{2}{\mathrm{N}} \times \sum_{\mathrm{k}=0}^{11} \mathrm{i1y} \times \sin(\mathrm{k} \times \frac{\pi}{6}) \tag{3.13}$$

$$\mathrm{Ibi[k]} = \frac{2}{\mathrm{N}} \times \sum_{\mathrm{k}=0}^{11} \mathrm{i2y} \times \sin(\mathrm{k} \times \frac{\pi}{6}) \tag{3.14}$$

$$\mathrm{Ici[k]} = \frac{2}{\mathrm{N}} \times \sum_{\mathrm{k}=0}^{11} \mathrm{i3y} \times \sin(\mathrm{k} \times \frac{\pi}{6}) \tag{3.15}$$

v1y, v2y, and v3y were the filtered phase voltages for phases A, B, and C respectively. i1y, i2y, and i3y were the filtered phase currents for phases A, B, and C respectively.

Third, the new phasor components were calculated as:

$$\mathrm{Var\_new} = \mathrm{Var\_old} + \frac{2}{\mathrm{N}} \times (\mathrm{v1y[k]} - \mathrm{v1y[k-N]}) \times \cos(\mathrm{k} \times \frac{\pi}{6}) \tag{3.16}$$

$$\mathrm{Vbr\_new} = \mathrm{Vbr\_old} + \frac{2}{\mathrm{N}} \times (\mathrm{v2y[k]} - \mathrm{v2y[k-N]}) \times \cos(\mathrm{k} \times \frac{\pi}{6}) \tag{3.17}$$

$$\mathrm{Vcr\_new} = \mathrm{Vcr\_old} + \frac{2}{\mathrm{N}} \times (\mathrm{v3y[k]} - \mathrm{v3y[k-N]}) \times \cos(\mathrm{k} \times \frac{\pi}{6}) \tag{3.18}$$

$$\mathrm{Iar\_new} = \mathrm{Iar\_old} + \frac{2}{\mathrm{N}} \times (\mathrm{i1y[k]} - \mathrm{i1y[k-N]}) \times \cos(\mathrm{k} \times \frac{\pi}{6}) \tag{3.19}$$

$$\mathrm{Ibr\_new} = \mathrm{Ibr\_old} + \frac{2}{\mathrm{N}} \times (\mathrm{i2y[k]} - \mathrm{i2y[k-N]}) \times \cos(\mathrm{k} \times \frac{\pi}{6}) \tag{3.20}$$

$$\mathrm{Icr\_new} = \mathrm{Icr\_old} + \frac{2}{\mathrm{N}} \times (\mathrm{i3y[k]} - \mathrm{i3y[k-N]}) \times \cos(\mathrm{k} \times \frac{\pi}{6}) \tag{3.21}$$

$$\mathrm{Vai\_new} = \mathrm{Vai\_old} + \frac{2}{\mathrm{N}} \times (\mathrm{v1y[k]} - \mathrm{v1y[k-N]}) \times \sin(\mathrm{k} \times \frac{\pi}{6}) \tag{3.22}$$

$$\mathrm{Vbi\_new} = \mathrm{Vbi\_old} + \frac{2}{\mathrm{N}} \times (\mathrm{v2y[k]} - \mathrm{v2y[k-N]}) \times \sin(\mathrm{k} \times \frac{\pi}{6}) \tag{3.23}$$

$$\mathrm{Vci\_new} = \mathrm{Vci\_old} + \frac{2}{\mathrm{N}} \times (\mathrm{v3y[k]} - \mathrm{v3y[k-N]}) \times \sin(\mathrm{k} \times \frac{\pi}{6}) \tag{3.24}$$

$$\mathrm{Iai\_new} = \mathrm{Iai\_old} + \frac{2}{\mathrm{N}} \times (\mathrm{i1y[k]} - \mathrm{i1y[k-N]}) \times \sin(\mathrm{k} \times \frac{\pi}{6}) \tag{3.25}$$

$$\mathrm{Ibi\_new} = \mathrm{Ibi\_old} + \frac{2}{\mathrm{N}} \times (\mathrm{i2y[k]} - \mathrm{i2y[k-N]}) \times \sin(\mathrm{k} \times \frac{\pi}{6}) \tag{3.26}$$

$$\mathrm{Ici\_new} = \mathrm{Ici\_old} + \frac{2}{\mathrm{N}} \times (\mathrm{i3y[k]} - \mathrm{i3y[k-N]}) \times \sin(\mathrm{k} \times \frac{\pi}{6}) \tag{3.27}$$

Since the process was recursive, Var_old, Vai_old, Vbr_old, Vbi_old, Vcr_old, Vci_old were the previous voltage phasor components. Iar_old, Iai_old, Ibr_old, Ibi_old, Icr_old, Ici_old were the previous current phasor components. N was the number of samples in one cycle of the fundamental frequency 60 Hertz. k was the sample number.

Fourth, the magnitudes of the new phasor components were calculated.

$$|\mathrm{V_a}| = \sqrt{\mathrm{Var\_new}^2 + \mathrm{Vai\_new}^2} \tag{3.28}$$

$$|\mathrm{V_b}| = \sqrt{\mathrm{Vbr\_new}^2 + \mathrm{Vbi\_new}^2} \tag{3.29}$$

$$|\mathrm{V_c}| = \sqrt{\mathrm{Vcr\_new}^2 + \mathrm{Vci\_new}^2} \tag{3.30}$$

$$|\mathrm{I_a}| = \sqrt{\mathrm{Iar\_new}^2 + \mathrm{Iai\_new}^2} \tag{3.31}$$

$$|\mathrm{I_b}| = \sqrt{\mathrm{Ibr\_new}^2 + \mathrm{Ibi\_new}^2} \tag{3.32}$$

$$|\mathrm{I_c}| = \sqrt{\mathrm{Icr\_new}^2 + \mathrm{Ici\_new}^2} \tag{3.33}$$

Last, the phases of the new phasor components were calculated.

$$< \mathrm{V_a} = \frac{180}{\pi} \times \mathrm{atan}(\frac{\mathrm{Vai\_new}}{\mathrm{Var\_new}}) \tag{3.34}$$

$$< \mathrm{V_b} = \frac{180}{\pi} \times \mathrm{atan}(\frac{\mathrm{Vbi\_new}}{\mathrm{Vbr\_new}}) \tag{3.35}$$

$$< \mathrm{V_c} = \frac{180}{\pi} \times \mathrm{atan}(\frac{\mathrm{Vci\_new}}{\mathrm{Vcr\_new}}) \tag{3.36}$$

$$< \mathrm{I_a} = \frac{180}{\pi} \times \mathrm{atan}(\frac{\mathrm{Iai\_new}}{\mathrm{Iar\_new}}) \tag{3.37}$$

$$< \mathrm{I_b} = \frac{180}{\pi} \times \mathrm{atan}(\frac{\mathrm{Ibi\_new}}{\mathrm{Ibr\_new}}) \tag{3.38}$$

$$< \mathrm{I_c} = \frac{180}{\pi} \times \mathrm{atan}(\frac{\mathrm{Ici\_new}}{\mathrm{Icr\_new}}) \tag{3.39}$$

## 3.3 CURRENT RELAY FUNCTION

The flow diagram of the inverse time-delay and instantaneous overcurrent relay function is shown in Figure 3.5. First, the current magnitudes were calculated as:

$$\mathrm{Ia\_f} = |\mathrm{Ia\_max}| \tag{3.40}$$

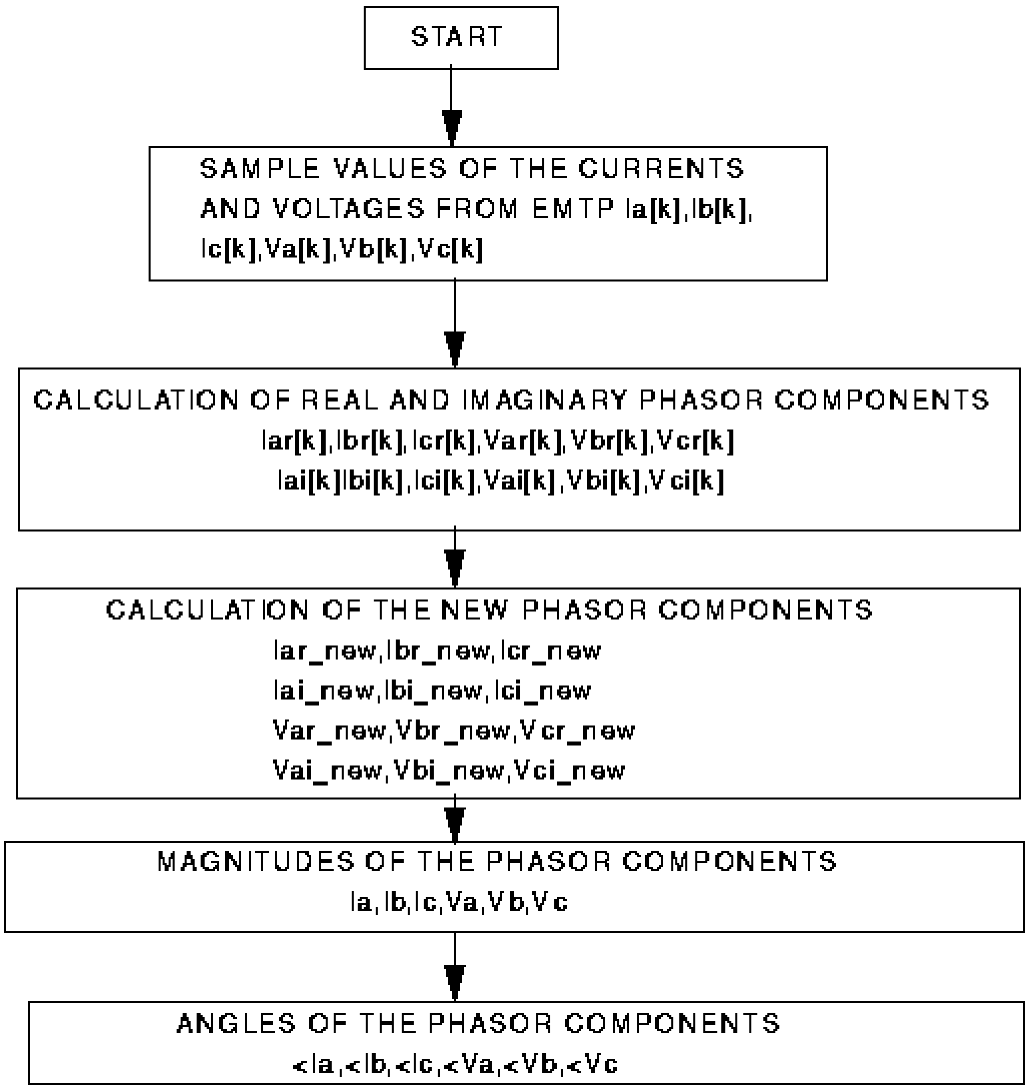
START
SAMPLE VALUES OF THE CURRENTS AND VOLTAGES FROM EMTP Ia[k],Ib[k], Ic[k],Va[k],Vb[k],Vc[k]
CALCULATION OF REAL AND IMAGINARY PHASOR COMPONENTS
Iar[k],Ibr[k],Icr[k],Var[k],Vbr[k],Vcr[k]
Iai[k]Ibi[k],Ici[k],Vai[k],Vbi[k],Vci[k]
CALCULATION OF THE NEW PHASOR COMPONENTS
Iar_new,Ibr_new,Icr_new
Iai_new,Ibi_new,Ici_new
Var_new,Vbr_new,Vcr_new
Vai_new,Vbi_new,Vci_new
MAGNITUDES OF THE PHASOR COMPONENTS
Ia,Ib,Ic,Va,Vb,Vc
ANGLES OF THE PHASOR COMPONENTS
<Ia,<Ib,<Ic,<Va,<Vb,<Vc

Figure 3.4: PHASOR COMPUTATION FLOW DIAGRAM

$$\text{Ib\_f} = |\text{Ib\_max}| \tag{3.41}$$

$$\text{Ic\_f} = |\text{Ic\_max}| \tag{3.42}$$

Ia_max, Ib_max, and Ic_max were the complex sample current data. Ia_max consisted of the real component (Iar_new) and the imaginary component (Iai_new). Ib_max consisted of the real component (Ibr_new) and the imaginary component (Ibi_new). Ic_max consisted of the real component (Icr_new) and the imaginary component (Ici_new).

Va_max, Vb_max, and Vc_max were the complex sample voltage data. Va_max consisted of the real component (Var_new) and the imaginary component (Vai_new). Vb_max consisted of the real component (Vbr_new) and the imaginary component (Vbi_new). Vc_max consisted of the real component (Vcr_new) and the imaginary component (Vci_new).

Second, the current phase angles were computed.

$$< \text{Theta\_Ia} = \frac{180}{\pi} \times \text{atan}\frac{\text{imag(Ia\_max)}}{\text{real(Ia\_max)}} \tag{3.43}$$

$$< \text{Theta\_Ib} = \frac{180}{\pi} \times \text{atan}\frac{\text{imag(Ib\_max)}}{\text{real(Ib\_max)}} \tag{3.44}$$

$$< \text{Theta\_Ic} = \frac{180}{\pi} \times \text{atan} \frac{\text{imag(Ic\_max)}}{\text{real(Ic\_max)}} \tag{3.45}$$

Third, the current transformer ratio (CT) was calculated.

$$\text{CT} = \frac{\text{i}_\text{L}}{5} \tag{3.46}$$

Fourth, the minimum fault current (I_min) and the maximum fault current (I_max) were calculated. I_min was the lowest magnitude value among Ia_f, Ib_f and Ic_f. I_max was the highest magnitude value among Ia_f, Ib_f and Ic_f. The equation for ground current (I_gnd) was:

$$\text{I\_gnd} = 0.1 \times \text{i}_\text{L} \tag{3.47}$$

Fifth, the phase relay pickup setting was computed.

$$\text{IPICKP} = \frac{(\frac{\text{i\_L}}{\text{CT}} \times 2) + \frac{\frac{\text{I\_min}}{3}}{\text{CT}}}{\text{CT}} \tag{3.48}$$

Sixth, the ground relay pickup setting was computed.

$$\text{IPICKG} = \frac{(0.1 \times \text{i\_L}) \times 2}{\text{CT}} \tag{3.49}$$

Seventh, the instantaneous overcurrent phase relay setting was calculated.

$$\text{IPEAKP} = 1.35 \times \text{IPICKP} \tag{3.50}$$

Eighth, the instantaneous overcurrent ground relay setting was computed.

$$\text{IPEAKG} = 1.35 \times \text{IPICKG} \tag{3.51}$$

Ninth, the per unit current magnitudes were calculated.

$$\text{Ia\_pu} = \frac{\text{Ia\_f}}{\text{I\_BASE}} \tag{3.52}$$

$$\text{Ib\_pu} = \frac{\text{Ib\_f}}{\text{I\_BASE}} \tag{3.53}$$

$$\text{Ic\_pu} = \frac{\text{Ic\_f}}{\text{I\_BASE}} \tag{3.54}$$

$I\_BASE$ was the base current. $V\_BASE$ was the base voltage. $S\_BASE$ was the base power. The base current was calculated as follows:

$$\mathrm{I\_BASE} = \frac{\mathrm{S\_BASE}}{\sqrt{3} \times \mathrm{V\_BASE}} \tag{3.55}$$

The inverse time and instantaneous overcurrent relay function were directional. The relay will run its instantaneous overcurrent, and inverse time overcurrent subroutines, if any of the following three equations were true.

$$-85.0 < \mathrm{Theta\_Ia} < -15.0 \tag{3.56}$$

$$-85.0 < \mathrm{Theta\_Ib} < -15.0 \tag{3.57}$$

$$-85.0 < \mathrm{Theta\_Ic} < -15.0 \tag{3.58}$$

The relay will trip for instantaneous overcurrent protection if:

$$\mathrm{Ia\_pu} > \mathrm{I\_peak} \tag{3.59}$$

$$\text{Ib\_pu} > \text{I\_peak} \tag{3.60}$$

$$\text{Ic\_pu} > \text{I\_peak} \tag{3.61}$$

The relay will trip for overcurrent protection if any of the three following equations were true:

$$\text{Ia\_pu} > \text{I\_high} \tag{3.62}$$

$$\text{Ib\_pu} > \text{I\_high} \tag{3.63}$$

$$\text{Ic\_pu} > \text{I\_high} \tag{3.64}$$

I_peak, I_high, and I_low were user-defined values. I_peak was the trip setting for the instantaneous overcurrent relay. I_high was the trip setting for the overcurrent relay.

The relay operating time for the phase and ground relay were represented by $T_p$, and $T_g$ respectively. $T_{p1}$ was sum of the first three terms of $T_p$. $T_{p2}$ was sum of the fourth and fifth term of $T_p$. $T_{p3}$ was the sum of the last three terms of $T_p$. $T_{g1}$ was the sum of the first three terms of $\mathrm{T_g}$. $\mathrm{T_{g2}}$ was the sum of the fourth and fifth term of $\mathrm{T_g}$. $\mathrm{T_{g3}}$ was the sum of the last three terms of $\mathrm{T_g}$.

$$\mathrm{T_p = T_{p1} + T_{p2} + T_{p3}} \tag{3.65}$$

$$\mathrm{T_{p1} = C_1 + C_2 \times (TDS) + C_3 \times \frac{(TDS)}{(I\_multp - 1)}} \tag{3.66}$$

$$\mathrm{T_{p2} = C_4 \times \frac{(TDS)^2}{(I\_multp - 1)} + C_5 \times \frac{(TDS)^2}{(I\_multp - 1)^2}} \tag{3.67}$$

$$\mathrm{T_{p3} = C_6 \times \frac{(TDS)}{(I\_multp - 1)^3} + C_7 \times \frac{(TDS)^2}{(I\_multp - 1)^4}} \tag{3.68}$$

$$\mathrm{T_g = T_{g1} + T_{g2} + T_{g3}} \tag{3.69}$$

$$\mathrm{T_{g1} = C_1 + C_2 \times (TDS) + C_3 \times \frac{(TDS)}{(I\_multg - 1)}} \tag{3.70}$$

$$\mathrm{T_{g2} = C_4 \times \frac{(TDS)^2}{(I\_multg - 1)} + C_5 \times \frac{(TDS)^2}{(I\_multg - 1)^2}} \tag{3.71}$$

$$T_{g3} = C_6 \times \frac{(TDS)}{(I\_multg - 1)^3} + C_7 \times \frac{(TDS)^2}{(I\_multg - 1)^4} \tag{3.72}$$

$C_1$, $C_2$, $C_3$, $C_4$, $C_5$, $C_6$,and $C_7$ were the constants obtained from the regression analysis for the CO-9 Westinghouse relay. They were calculated by using the least error squares technique. $C_1$ = 0.0344, $C_2$ = 0.0807, $C_3$ = 1.9500, $C_4$ = 0.0577, $C_5$ = -0.0679, $C_6$ = -0.7000, and $C_7$ = 0.0199[2][18]. I_multp was the multiple of the pickup tap setting for the phase relay. I_multg was the multiple of the pickup tap setting for the ground relay. TDS was the user-defined time dial setting.

$$I\_multp = |\frac{I\_max}{I\_setp}| \tag{3.73}$$

$$I\_multg = |\frac{I\_gnd}{I\_setp}| \tag{3.74}$$

I_setp was the pickup current for the phase relay. I_setg was the pickup current for the ground relay.

If the instantaneous overcurrent relay tripped for phase protection:

$$I\_setp = (CT \times Ipeakp) \tag{3.75}$$

[2]These coefficients of a seven term model, were obtained for the CO-9 Westinghouse Relay.

If the instantaneous overcurrent relay tripped for ground protection:

$$\text{I\_setg} = (\text{CT} \times \text{Ipeakg}) \tag{3.76}$$

If the inverse time overcurrent relay tripped for phase protection:

$$\text{I\_setp} = (\text{CT} \times \text{Ipickp}) \tag{3.77}$$

If the inverse time overcurrent relay tripped for ground protection:

$$\text{I\_setg} = (\text{CT} \times \text{Ipickg}) \tag{3.78}$$

## 3.4 ISLANDING RELAY FUNCTION

The flow diagram of the islanding relay is shown in Figure 3.6. First, the complex three phase power (Pg) was calculated.

$$\text{Pg} = (|\text{V}_\text{a}| \times |\text{I}_\text{a}|) + (|\text{V}_\text{b}| \times |\text{I}_\text{b}|) + (|\text{V}_\text{c}| \times |\text{I}_\text{c}|) \tag{3.79}$$

Second, the power while the DSG unit was connected to the utility (Pg_pre) was calculated.

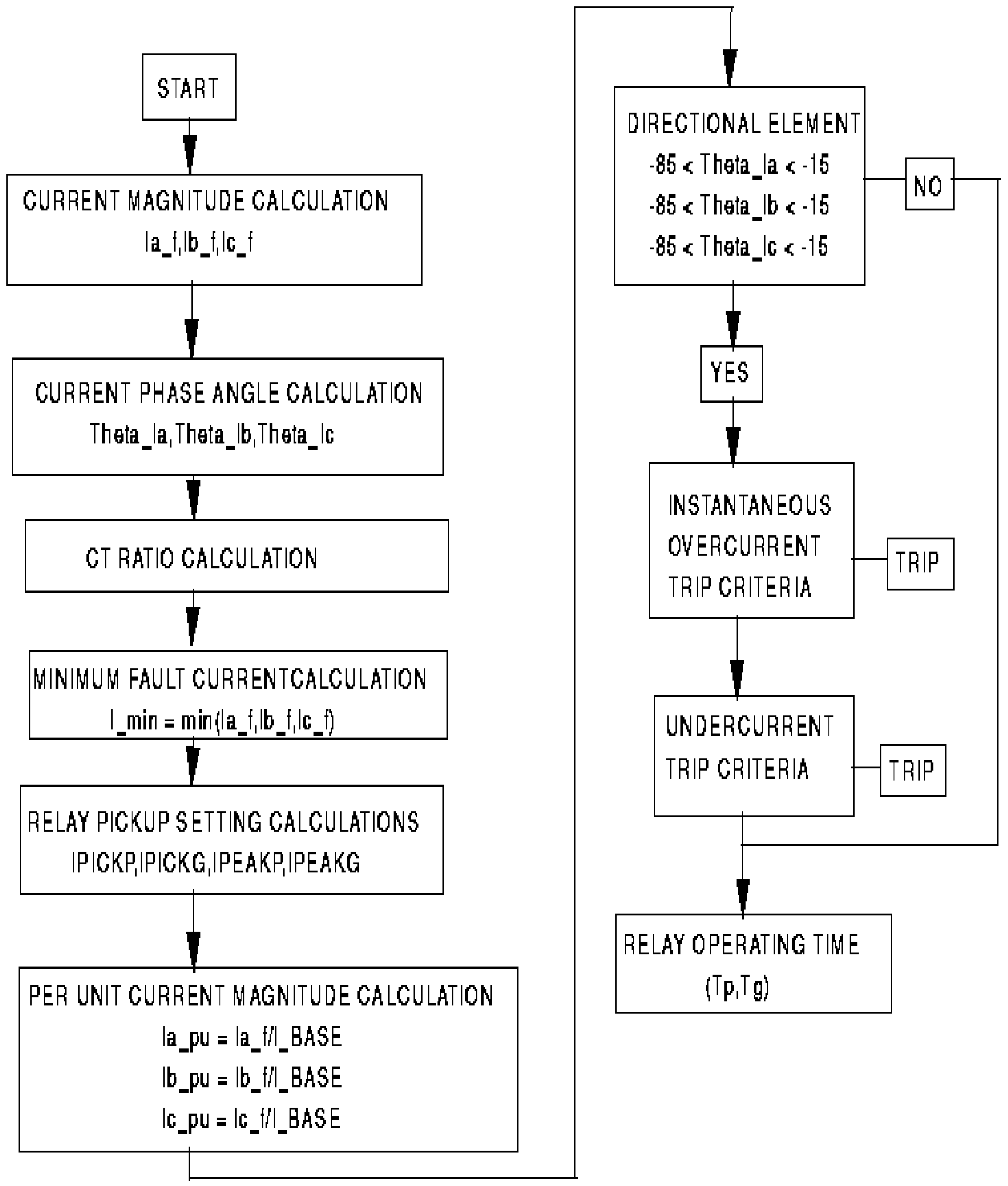
START
CURRENT MAGNITUDE CALCULATION
Ia_f,Ib_f,Ic_f
CURRENT PHASE ANGLE CALCULATION
Theta_Ia,Theta_Ib,Theta_Ic
CT RATIO CALCULATION
MINIMUM FAULT CURRENTCALCULATION
I_min = min(Ia_f,Ib_f,Ic_f)
RELAY PICKUP SETTING CALCULATIONS
IPICKP,IPICKG,IPEAKP,IPEAKG
PER UNIT CURRENT MAGNITUDE CALCULATION
Ia_pu = Ia_f/I_BASE
Ib_pu = Ib_f/I_BASE
Ic_pu = Ic_f/I_BASE
DIRECTIONAL ELEMENT
-85 < Theta_Ia < -15
-85 < Theta_Ib < -15
-85 < Theta_Ic < -15
NO
YES
INSTANTANEOUS OVERCURRENT TRIP CRITERIA
TRIP
UNDERCURRENT TRIP CRITERIA
TRIP
RELAY OPERATING TIME
(Tp,Tg)

Figure 3.5: CURRENT RELAY FLOW DIAGRAM

$$\text{Pg\_pre} = \text{Pg}[\text{k} - \text{N}] \tag{3.80}$$

Third, the power after the DSG unit was disconnected from the utility (Pg_post) was calculated.

$$\text{Pg\_post} = \text{Pg}[\text{k}] \tag{3.81}$$

The islanding relay will trip if:

$$\text{isl} = \frac{|\text{Pg\_post} - \text{Pg\_pre}|}{\text{S\_BASE}} > \text{ks} \tag{3.82}$$

isl was the per unit change in power. ks was a user-defined trip setting.

## 3.5 LOSS-OF-EXCITATION RELAY FUNCTION

The flow diagram for the loss-of-excitation relay is shown in Figure 3.7. First, the complex sample impedance data (Za_max,Zb_max,Zc_max) were calculated. Second, the impedance magnitudes (Za,Zb,Zc) were calculated.

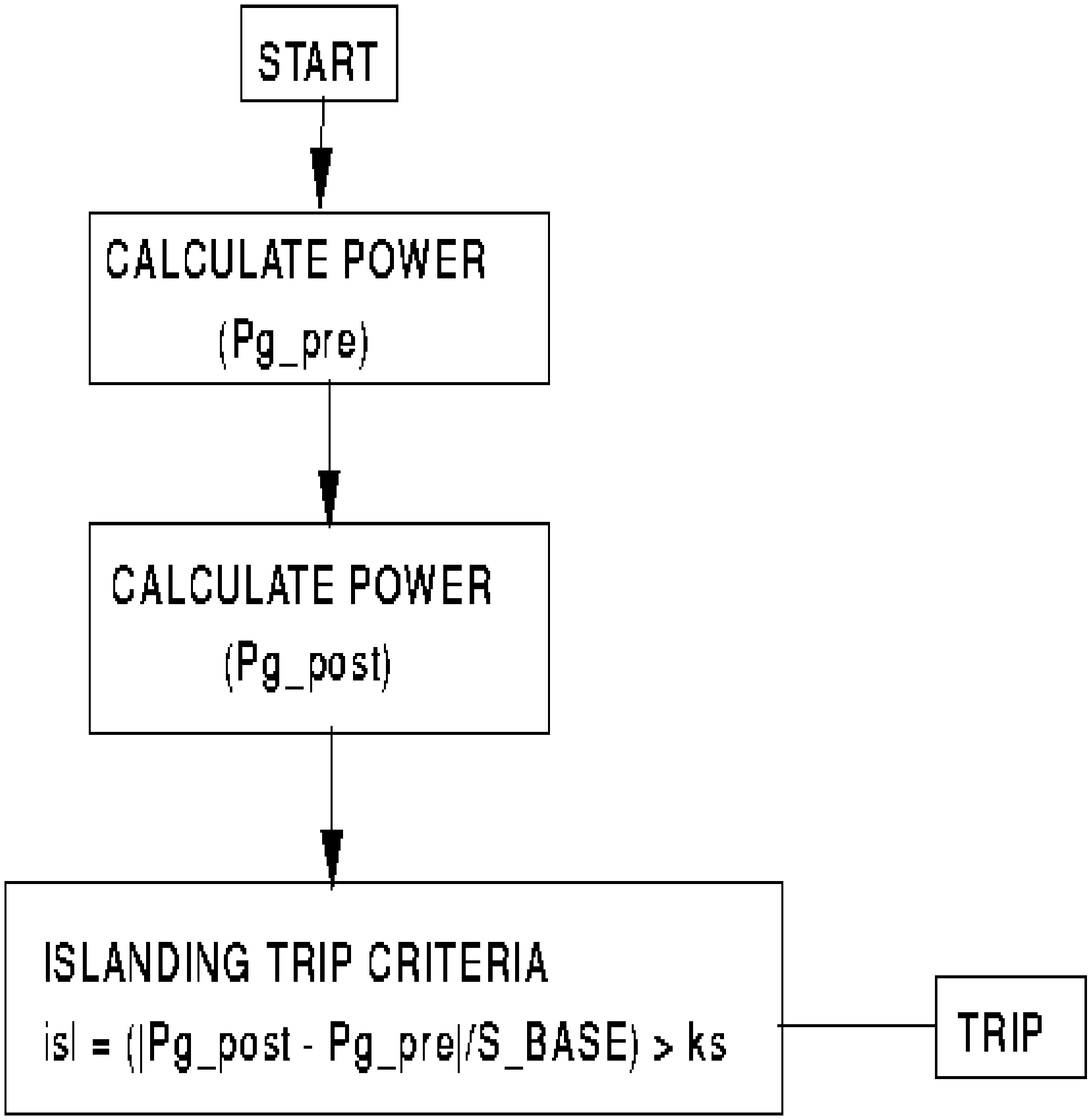

Figure 3.6: ISLANDING RELAY FLOW DIAGRAM

$$\text{Za\_max} = \frac{\text{Va\_max}}{\text{Ia\_max}} \tag{3.83}$$

$$\text{Zb\_max} = \frac{\text{Vb\_max}}{\text{Ib\_max}} \tag{3.84}$$

$$\text{Zc\_max} = \frac{\text{Vc\_max}}{\text{Ic\_max}} \tag{3.85}$$

$$\text{Za} = |\text{Za\_max}| \tag{3.86}$$

$$\text{Zb} = |\text{Zb\_max}| \tag{3.87}$$

$$\text{Zc} = |\text{Zc\_max}| \tag{3.88}$$

Third, the apparent impedance was calculated.

$$\text{Zapp} = \frac{0.5 \times \text{X\_d}}{\text{Z\_BASE}} \tag{3.89}$$

Z_BASE was the base impedance. $X_d$ was the user-defined machine impedance value [3].

The base impedance was calculated using the following equation:

$$\text{Z\_BASE} = \frac{\text{V\_BASE}}{\text{I\_BASE}} \tag{3.90}$$

The loss-of-excitation relay will trip if any of the phase impedances were smaller than the appparent impedance.

$$\text{Za} < \text{Zapp} \tag{3.91}$$

$$\text{Zb} < \text{Zapp} \tag{3.92}$$

$$\text{Zc} < \text{Zapp} \tag{3.93}$$

[3] $X_d$ can be the synchronous, transient, or sub-transient reactance of the machine.

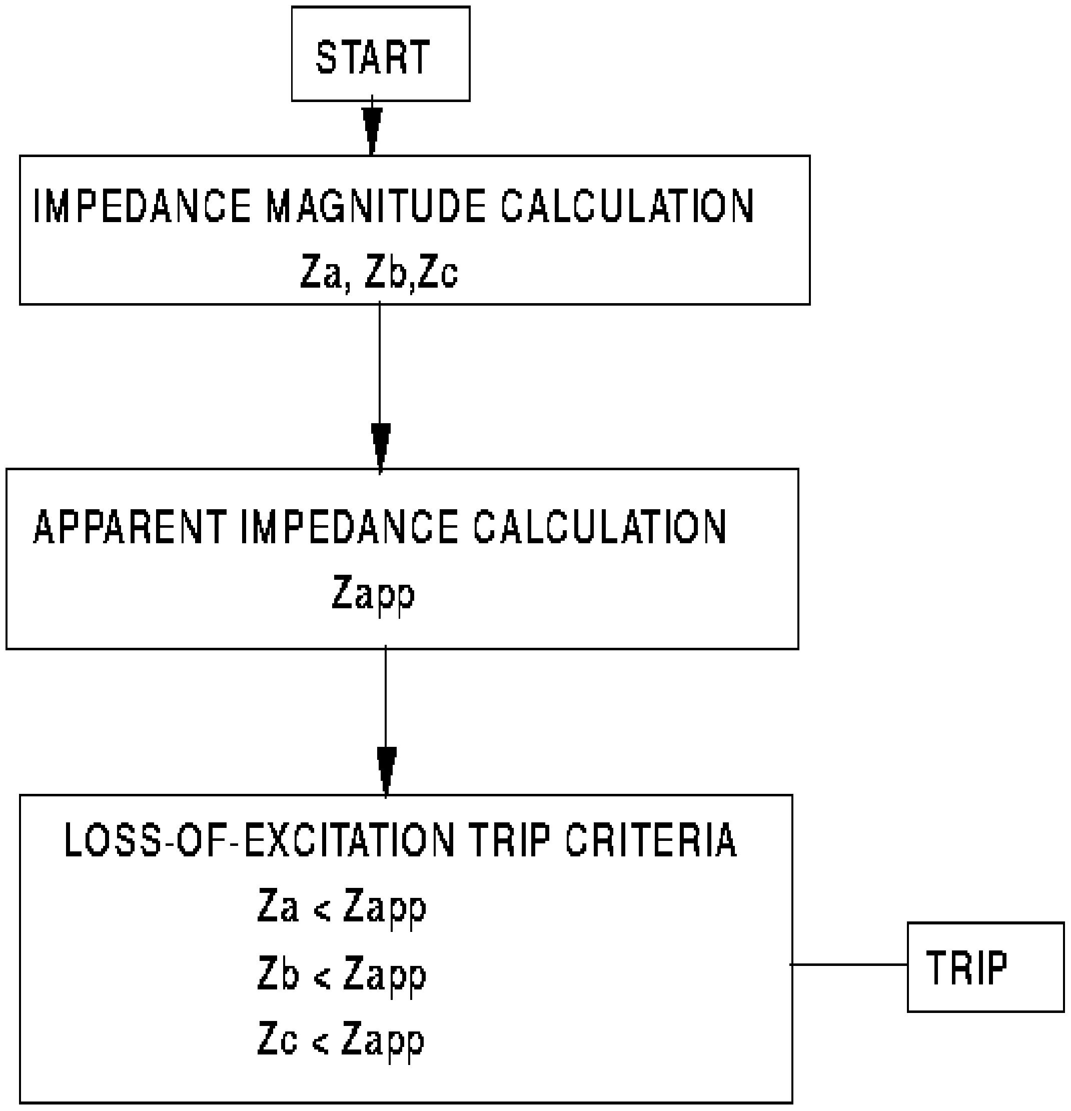
START
IMPEDANCE MAGNITUDE CALCULATION
Za, Zb,Zc
APPARENT IMPEDANCE CALCULATION
Zapp
LOSS-OF-EXCITATION TRIP CRITERIA
Za < Zapp
Zb < Zapp
Zc < Zapp
TRIP

Figure 3.7: LOSS-OF-EXCITATION RELAY FLOW DIAGRAM

## 3.6 DIFFERENTIAL RELAY FUNCTION

The flow diagram of the differential relay function is shown in Figure 3.8. The complex sample current data Ia, Ib, and Ic were calculated first.

$$\mathrm{Ia} = \mathrm{Ia\_max} \tag{3.94}$$

$$\mathrm{Ib} = \mathrm{Ib\_max} \tag{3.95}$$

$$\mathrm{Ic} = \mathrm{Ic\_max} \tag{3.96}$$

Second, the current magnitudes ( Ia_pre,Ib_pre,Ic_pre) for the dispersed generator were computed.

$$\mathrm{Ia\_pre} = \mathrm{Ia}[\mathrm{k} - \mathrm{N}] \tag{3.97}$$

$$\mathrm{Ib\_pre} = \mathrm{Ib}[\mathrm{k} - \mathrm{N}] \tag{3.98}$$

$$\text{Ic\_pre} = \text{Ic}[\text{k} - \text{N}] \tag{3.99}$$

Ia_pre was the current flowing in phase A with the dispersed generator connected to the utility. Ib_pre was the current flowing in phase B with the dispersed generator connected to the utility. Ic_pre was the current flowing in phase C with the dispersed generator connected to the utility.

Third, the current magnitudes ( Ia_post,Ib_post,Ic_post) for the dispersed generator were computed.

$$\text{Ia\_post} = \text{Ia}[\text{k}] \tag{3.100}$$

$$\text{Ib\_post} = \text{Ib}[\text{k}] \tag{3.101}$$

$$\text{Ic\_post} = \text{Ic}[\text{k}] \tag{3.102}$$

Ia_post was the current flowing in phase A with the dispersed generator disconnected from the utility. Ib_post was the current flowing in phase B with the dispersed generator disconnected from the utility. Ic_post was the current flowing in phase C with the dispersed generator disconnected from the utility.

Fourth, the per unit change in current for the three phases were calculated.

$$\mathrm{S\_1} = \frac{|\mathrm{Ia\_post} - \mathrm{Ia\_pre}|}{\mathrm{I\_BASE}} \tag{3.103}$$

$$\mathrm{S\_2} = \frac{|\mathrm{Ib\_post} - \mathrm{Ib\_pre}|}{\mathrm{I\_BASE}} \tag{3.104}$$

$$\mathrm{S\_3} = \frac{|\mathrm{Ic\_post} - \mathrm{Ic\_pre}|}{\mathrm{I\_BASE}} \tag{3.105}$$

The differential relay will trip if any of the three conditions were met:

$$\mathrm{S\_1} > \mathrm{S} \tag{3.106}$$

$$\mathrm{S\_2} > \mathrm{S} \tag{3.107}$$

$$\mathrm{S\_3} > \mathrm{S} \tag{3.108}$$

S was called the sensitivity slope and was a user-defined value.

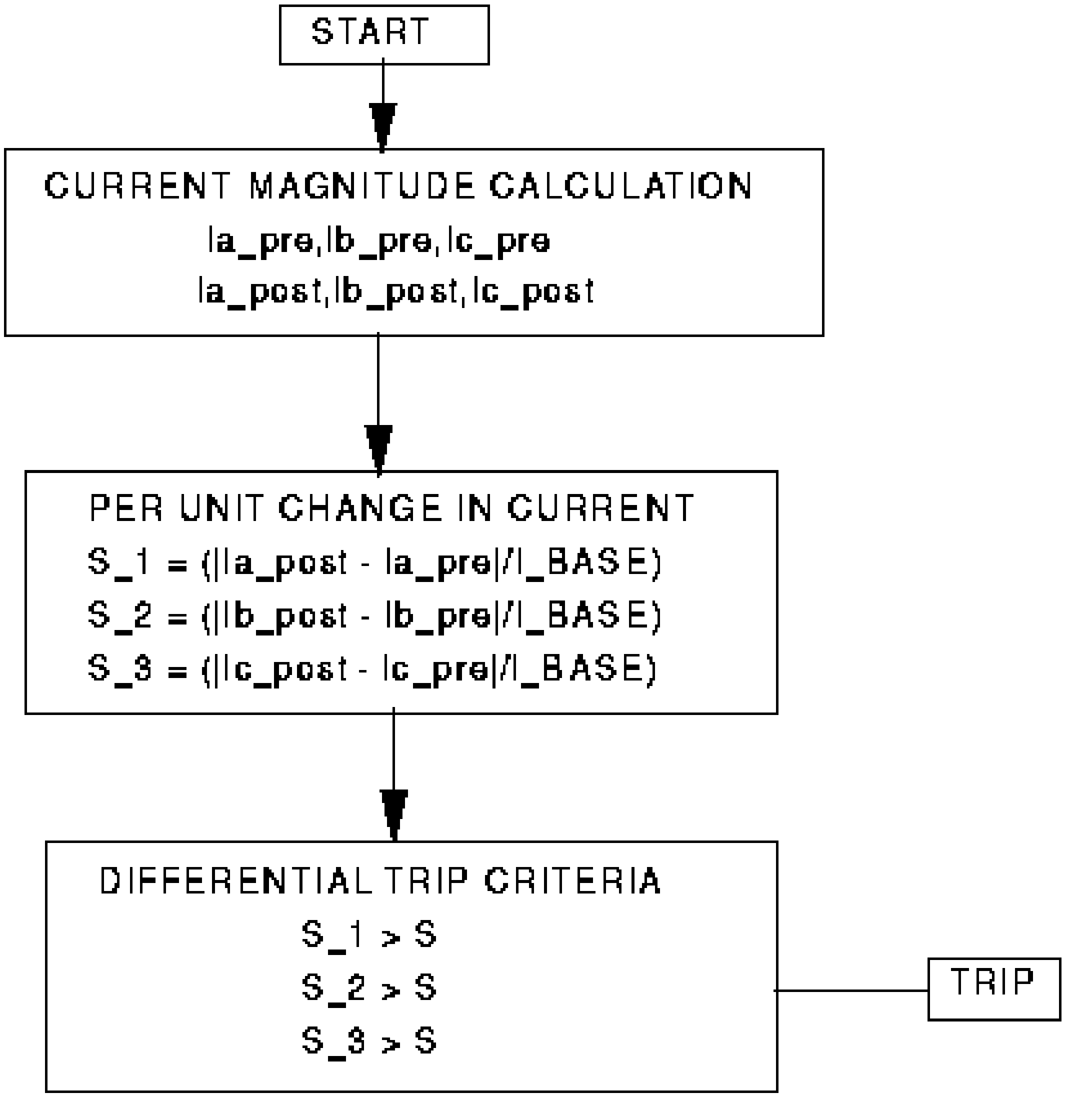
START
CURRENT MAGNITUDE CALCULATION
Ia_pre,Ib_pre,Ic_pre
Ia_post,Ib_post,Ic_post
PER UNIT CHANGE IN CURRENT
S_1 = (|Ia_post - Ia_pre|/I_BASE)
S_2 = (|Ib_post - Ib_pre|/I_BASE)
S_3 = (|Ic_post - Ic_pre|/I_BASE)
DIFFERENTIAL TRIP CRITERIA
S_1 > S
S_2 > S
S_3 > S
TRIP

Figure 3.8: DIFFERENTIAL RELAY FLOW DIAGRAM

## 3.7 FREQUENCY RELAY FUNCTION

The flow diagram of the frequency relay function is shown in Figure 3.9. First, the positive sequence voltage phasor at the kth instant was calculated as:

$$\mathrm{Va\_1} = \frac{1}{3}(\mathrm{Va\_max} + \mathrm{a} \times \mathrm{Vb\_max} + \mathrm{a}^2 \times \mathrm{Vc\_max}) \tag{3.109}$$

Second, the positive sequence voltage phasor at the (kth-N) instant was calculated as:

$$\mathrm{Va1\_del} = \frac{1}{3}(\mathrm{Va\_del} + \mathrm{a} \times \mathrm{Vb\_del} + \mathrm{a}^2 \times \mathrm{Vc\_del}) \tag{3.110}$$

Va_del, Vb_del, and Vc_del were the complex sample voltage data at the (kth-N) instant from EMTP.

Third, the positive sequence voltage phase angle at the kth instant was calculated as:

$$\mathrm{theta} = \frac{180}{\pi} \times \mathrm{atan}\frac{\mathrm{imag(Va\_1)}}{\mathrm{real(Va\_1)}} \tag{3.111}$$

Fourth, the positive sequence voltage phase angle at the (kth-N) instant was calculated as:

$$\mathrm{theta\_del} = \frac{180}{\pi} \times \mathrm{atan}\frac{\mathrm{imag(Va1\_del)}}{\mathrm{real(Va1\_del)}} \tag{3.112}$$

Fifth, the incremental frequency (freq_del) was computed.

$$\text{freq\_del} = \frac{1}{360} \times (\text{theta} - \text{theta\_del}) \times \text{freq\_nom} \tag{3.113}$$

freq_nom was the nominal frequency value of 60 Hz.

Sixth, the frequency value (freq_new) at the next kth instant was computed by using the following equation:

$$\text{freq\_new} = \text{freq\_nom} + \text{freq\_del} \tag{3.114}$$

The overfrequency relay will trip if:

$$\text{freq\_new} > \text{high\_f} \tag{3.115}$$

The underfrequency relay will trip if:

$$\text{freq\_new} < \text{low\_f} \tag{3.116}$$

high_f and low_f were user-defined values. high_f was the trip setting for the overfrequency relay. low_f was the trip setting for the underfrequency relay.

As the difference between theta and theta_del becomes larger, the frequency estimates will increase in noise. Several phasors could be used in order to increase accuracy, by using a least squared polynominal to compute the frequency estimates.[19] This procedure was not used in this work, because only two phasors were used.

## 3.8 VOLTAGE RELAY FUNCTION

The flow diagram of the voltage relay function is shown in Figure 3.10. First, the voltage magnitudes (Va,Vb,Vc) were calculated exactly the same as for the loss-of-excitation relay function. Second, the per unit rms voltage magnitudes (Vapu,Vbpu,Vcpu) were calculated.

$$\mathrm{Vapu} = \frac{\mathrm{Va}}{\mathrm{V\_BASE}} \tag{3.117}$$

s

$$\mathrm{Vbpu} = \frac{\mathrm{Vb}}{\mathrm{V\_BASE}} \tag{3.118}$$

$$\mathrm{Vcpu} = \frac{\mathrm{Vc}}{\mathrm{V\_BASE}} \tag{3.119}$$

Third, the per unit peak voltage magnitudes (Vinsta,Vinstb,Vinstc) were calculated.

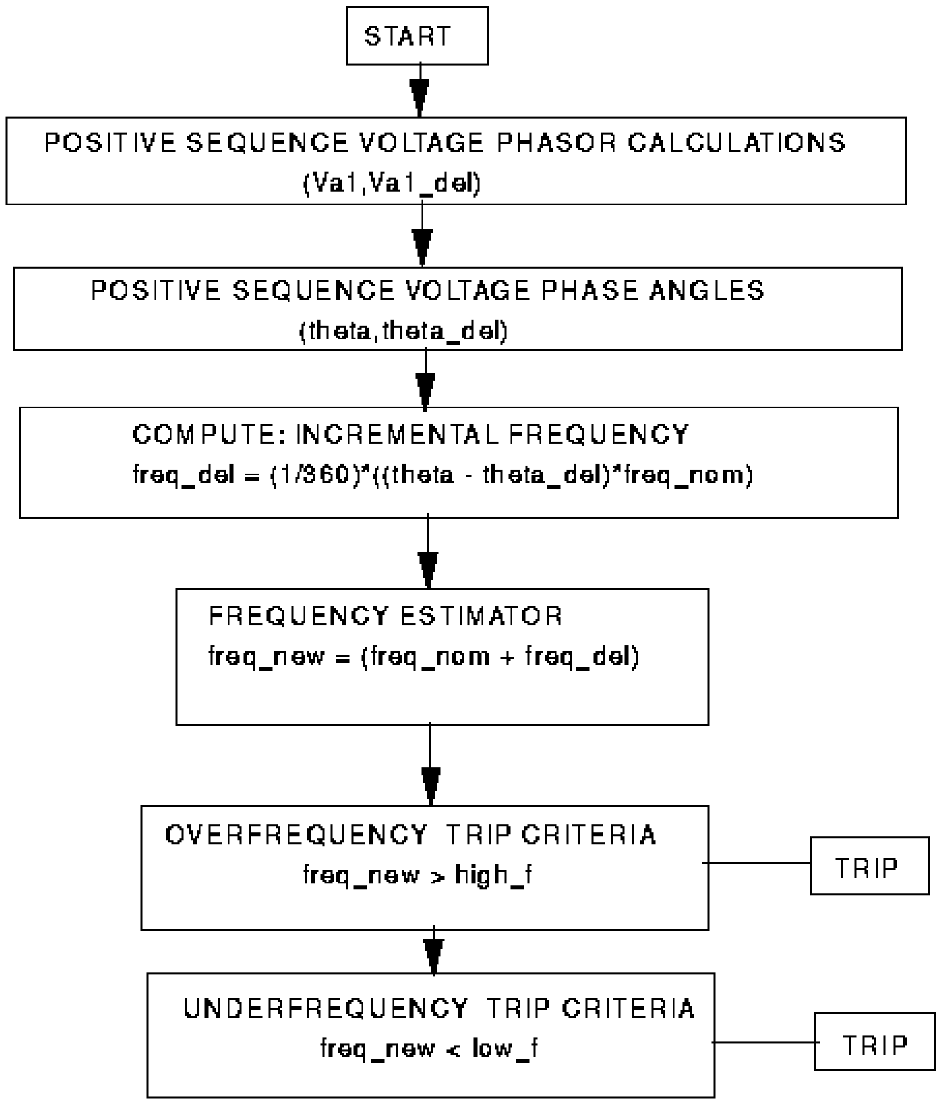
START
POSITIVE SEQUENCE VOLTAGE PHASOR CALCULATIONS
(Va1,Va1_del)
POSITIVE SEQUENCE VOLTAGE PHASE ANGLES
(theta,theta_del)
COMPUTE: INCREMENTAL FREQUENCY
freq_del = (1/360)*((theta - theta_del)*freq_nom)
FREQUENCY ESTIMATOR
freq_new = (freq_nom + freq_del)
OVERFREQUENCY TRIP CRITERIA
freq_new > high_f
TRIP
UNDERFREQUENCY TRIP CRITERIA
freq_new < low_f
TRIP

Figure 3.9: FREQUENCY RELAY FLOW DIAGRAM

$$\text{Vinsta} = \frac{\sqrt{2} \times \text{Va}}{\text{V\_BASE}} \tag{3.120}$$

$$\text{Vinstb} = \frac{\sqrt{2} \times \text{Vb}}{\text{V\_BASE}} \tag{3.121}$$

$$\text{Vinstc} = \frac{\sqrt{2} \times \text{Vc}}{\text{V\_BASE}} \tag{3.122}$$

The voltage relay will trip for peak overvoltage protection if:

$$\text{Vinsta} \geq \text{V\_peak} \tag{3.123}$$

$$\text{Vinstb} \geq \text{V\_peak} \tag{3.124}$$

$$\text{Vinstc} \geq \text{V\_peak} \tag{3.125}$$

The voltage relay will trip for rms overvoltage protection if:

$$\text{Vapu} \geq \text{V\_high} \tag{3.126}$$

$$\text{Vbpu} \geq \text{V\_high} \tag{3.127}$$

$$\text{Vcpu} \geq \text{V\_high} \tag{3.128}$$

The voltage relay will trip for undervoltage protection if:

$$\text{Vapu} \leq \text{V\_low} \tag{3.129}$$

$$\text{Vbpu} \leq \text{V\_low} \tag{3.130}$$

$$\text{Vcpu} \leq \text{V\_low} \tag{3.131}$$

V_peak,V_high and V_low were user-defined values. V_peak was the trip setting for the peak overvoltage relay. V_high was the trip setting for the rms overvoltage relay. V_low was the trip setting for the undervoltage relay.

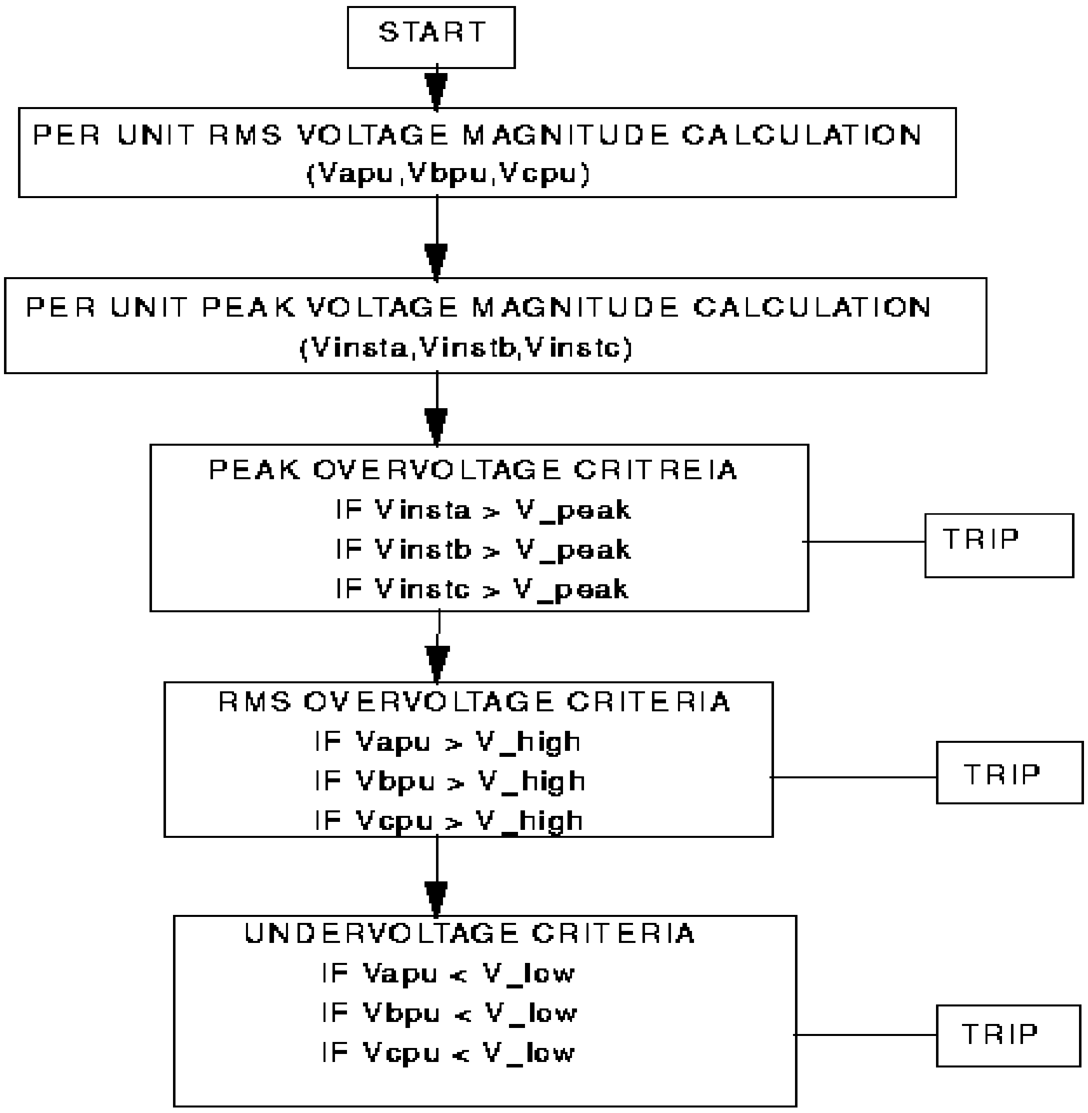

Figure 3.10: VOLTAGE RELAY FLOW DIAGRAM

## 3.9 NEGATIVE-SEQUENCE CURRENT RELAY FUNCTION

The flow diagram of the negative-sequence current relay function is shown in Figure 3.11. First, the negative-sequence current (Ia_2) was calculated as:

$$\mathrm{Ia\_2} = \frac{1}{3}(\mathrm{Ia\_max} + \mathrm{a}^2 \times \mathrm{Ib\_max} + \mathrm{a} \times \mathrm{Ic\_max}) \tag{3.132}$$

The symbol a was the complex operator (-0.5+j0.866).

Second, the per unit negative-sequence current Ia2_pu was calculated as:

$$\mathrm{Ia2\_pu} = |\frac{\mathrm{Ia}_2}{\mathrm{I\_BASE}}| \tag{3.133}$$

Third, the time dial setting was calculated.

$$\mathrm{Tdial} = (\mathrm{Ia2\_pu})^2 \times \mathrm{k} \tag{3.134}$$

$$\mathrm{dial\_sum} = \sum_{\mathrm{i}=0}^{\mathrm{max}} \mathrm{Tdial} \tag{3.135}$$

Tdial, the time dial setting at the kth instant, was the per unit negative-sequence current squared multiplied by the sample number. dial_sum was the integrated sum of the time dial setting [4]. The negative-sequence current relay can trip if the per unit negative-sequence current exceeded a predetermined trip setting. The relay can also trip if the predetermined trip setting of dial_sum was exceeded.

The negative-sequence current relay will trip if any of the following equations were true:

$$\text{Ia2\_pu} > \text{I\_peak} \tag{3.136}$$

$$\text{dial\_sum} > \text{T\_peak} \tag{3.137}$$

I_peak and T_peak were user-defined values. I_peak was the trip setting for the per unit negative-sequence current. T_peak was the trip setting for the excessive heating element (dial_sum).

[4]This function was included, because it is a very efffective means to protect the dispersed generator from excessive heating resulting from current unbalance.

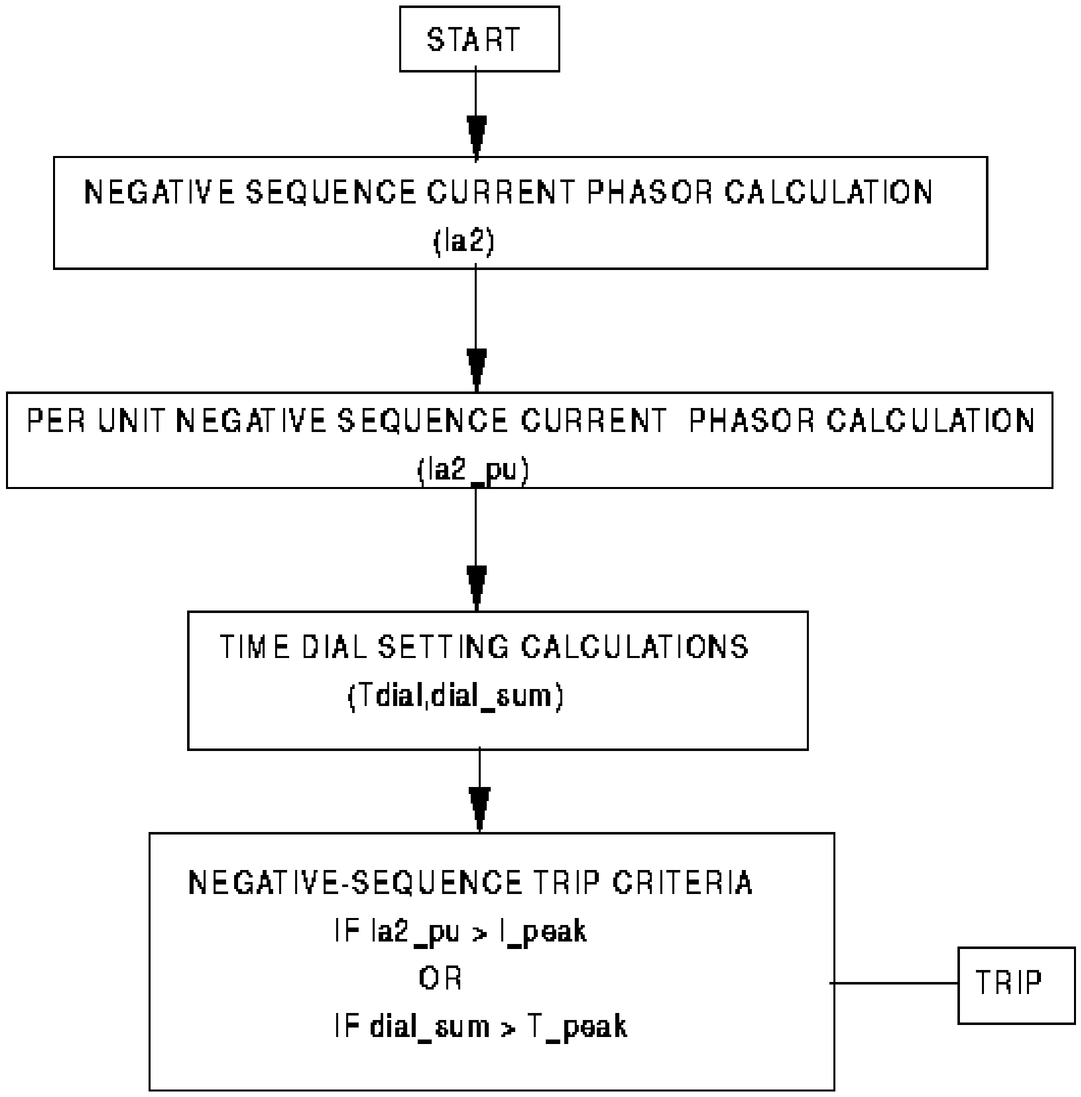
START
NEGATIVE SEQUENCE CURRENT PHASOR CALCULATION
(Ia2)
PER UNIT NEGATIVE SEQUENCE CURRENT PHASOR CALCULATION
(Ia2_pu)
TIME DIAL SETTING CALCULATIONS
(Tdial,dial_sum)
NEGATIVE-SEQUENCE TRIP CRITERIA
IF Ia2_pu > I_peak
OR
IF dial_sum > T_peak
TRIP

Figure 3.11: NEGATIVE-SEQUENCE RELAY FLOW DIAGRAM

## 3.10 CHECK-SYNCHRONISM RELAY FUNCTION

The flow diagram of the check-synchronism relay function is shown in figure 3.12. First, the voltage phase angles were calculated as:

$$\mathrm{Va\_angle} = \frac{180}{\pi} \times \mathrm{atan}\frac{\mathrm{imag(Va\_max)}}{\mathrm{real(Va\_max)}} \tag{3.138}$$

$$\mathrm{Vb\_angle} = \frac{180}{\pi} \times \mathrm{atan}\frac{\mathrm{imag(Vb\_max)}}{\mathrm{real(Vb\_max)}} \tag{3.139}$$

$$\mathrm{Vc\_angle} = \frac{180}{\pi} \times \mathrm{atan}\frac{\mathrm{imag(Vc\_max)}}{\mathrm{real(Vc\_max)}} \tag{3.140}$$

Second, the voltage phase angles (Va_angi,Vb_angi,Vc_angi) were computed.

$$\mathrm{Va\_angi} = \mathrm{Va\_angle[k - N]} \tag{3.141}$$

$$\mathrm{Vb\_angi} = \mathrm{Vb\_angle[k - N]} \tag{3.142}$$

$$\mathrm{Vc\_angi} = \mathrm{Vc\_angle[k - N]} \tag{3.143}$$

Va_angi was the utility's voltage angle for phase A. Vb_angi was the utility's voltage angle for phase B. Vc_angi was the utility's voltage angle for phase C.

Third, the voltage phase angles (Va_angf,Vb_angf,Vc_angf) were computed.

$$\text{Va\_angf} = \text{Va\_angle[k]} \tag{3.144}$$

$$\text{Vb\_angf} = \text{Vb\_angle[k]} \tag{3.145}$$

$$\text{Vc\_angf} = \text{Vc\_angle[k]} \tag{3.146}$$

Va_angf was the dispersed generator's voltage angle for phase A. Vb_angf was the dispersed generator's voltage angle for phase B. Vc_angf was the dispersed generator's voltage angle for phase C.

Fourth, the phase angle differences between the utility substation generator voltages and the dispersed generator voltages were calculated as:

$$\text{Theta\_1} = |\text{Va\_angf} - \text{Va\_angi}| \tag{3.147}$$

$$\text{Theta\_2} = |\text{Vb\_angf} - \text{Vb\_angi}| \tag{3.148}$$

$$\text{Theta\_3} = |\text{Vc\_angf} - \text{Vc\_angi}| \tag{3.149}$$

The check-synchronism relay will trip if any of the following equations were true:

$$\text{Theta\_1} > \text{crit\_ang} \tag{3.150}$$

$$\text{Theta\_2} > \text{crit\_ang} \tag{3.151}$$

$$\text{Theta\_3} > \text{crit\_ang} \tag{3.152}$$

crit_ang was a user-defined value.

## 3.11 DIRECTIONAL POWER RELAY FUNCTION

The flow diagram of the directional power relay function is shown in Figure 3.13. First, the complex sample power for each phase Pa_max, Pb_max, and Pc_max were calculated.

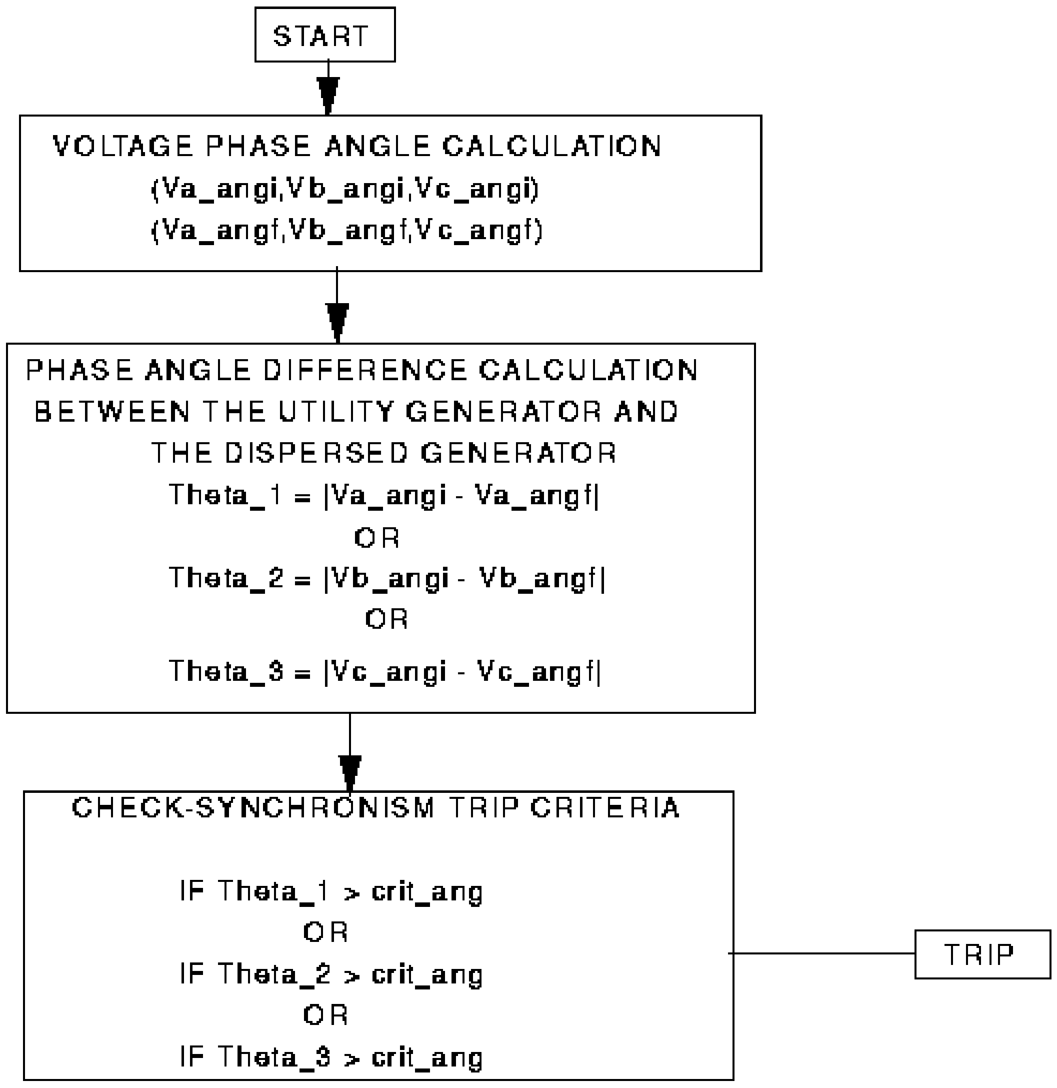
START
VOLTAGE PHASE ANGLE CALCULATION
(Va_angi,Vb_angi,Vc_angi)
(Va_angf,Vb_angf,Vc_angf)
PHASE ANGLE DIFFERENCE CALCULATION BETWEEN THE UTILITY GENERATOR AND THE DISPERSED GENERATOR
Theta_1 = |Va_angi - Va_angf|
OR
Theta_2 = |Vb_angi - Vb_angf|
OR
Theta_3 = |Vc_angi - Vc_angf|
CHECK-SYNCHRONISM TRIP CRITERIA
IF Theta_1 > crit_ang
OR
IF Theta_2 > crit_ang
OR
IF Theta_3 > crit_ang
TRIP

Figure 3.12: CHECK-SYNCHRONISM RELAY FLOW DIAGRAM

$$\text{Pa\_max} = \text{Va\_max} \times \text{Ia\_max} \tag{3.153}$$

$$\text{Pb\_max} = \text{Vb\_max} \times \text{Ib\_max} \tag{3.154}$$

$$\text{Pc\_max} = \text{Vc\_max} \times \text{Ic\_max} \tag{3.155}$$

Second, the real sample power for each phase (Par,Pbr,Pcr) were calculated.

$$\text{Par} = \cos(\text{Pa\_max}) \tag{3.156}$$

$$\text{Pbr} = \cos(\text{Pb\_max}) \tag{3.157}$$

$$\text{Pcr} = \cos(\text{Pc\_max}) \tag{3.158}$$

Third, the reactive sample power for each phase (Pai,Pbi,Pci) were calculated.

$$\text{Pai} = \sin(\text{Pa\_max}) \tag{3.159}$$

$$\text{Pbi} = \sin(\text{Pb\_max}) \tag{3.160}$$

$$\text{Pci} = \sin(\text{Pc\_max}) \tag{3.161}$$

Fourth, the total real power P_t was calculated as:

$$\text{P\_t} = \text{Par} + \text{Pbr} + \text{Pcr} \tag{3.162}$$

Fifth, the total reactive power Q_t was calculated as:

$$\text{Q\_t} = \text{Pai} + \text{Pbi} + \text{Pci} \tag{3.163}$$

Sixth, the power factor was calculated as:

$$\text{PF} = \frac{\text{P\_t}}{\sqrt{(\text{P\_t})^2 + (\text{Q\_t})^2}} \tag{3.164}$$

The directional power relay will trip for reverse power protection if:

$$\text{Par} < \text{crit\_pow} \tag{3.165}$$

$$\text{Pbr} < \text{crit\_pow} \tag{3.166}$$

$$\text{Pcr} < \text{crit\_pow} \tag{3.167}$$

crit_pow was a user-defined value.

## 3.12 SELF-EXCITATION RELAY FUNCTION

The flow diagram of the self-excitation relay function is shown in Figure 3.14. A Fast Fourier Transform (FFT) high frequency sampler was used to transform the sample voltages (Va,Vb,Vc) into their frequency components ($\text{Va}[\text{m}], \text{Vb}[\text{m}], \text{Vc}[\text{m}]$). Each of these components had real and imaginary parts. The variable m was the frequency index.

The amplitude of the frequency components were calculated as follows:

$$\text{Va\_f} = |\text{Va}(\text{m})| \tag{3.168}$$

$$\text{Vb\_f} = |\text{Vb}(\text{m})| \tag{3.169}$$

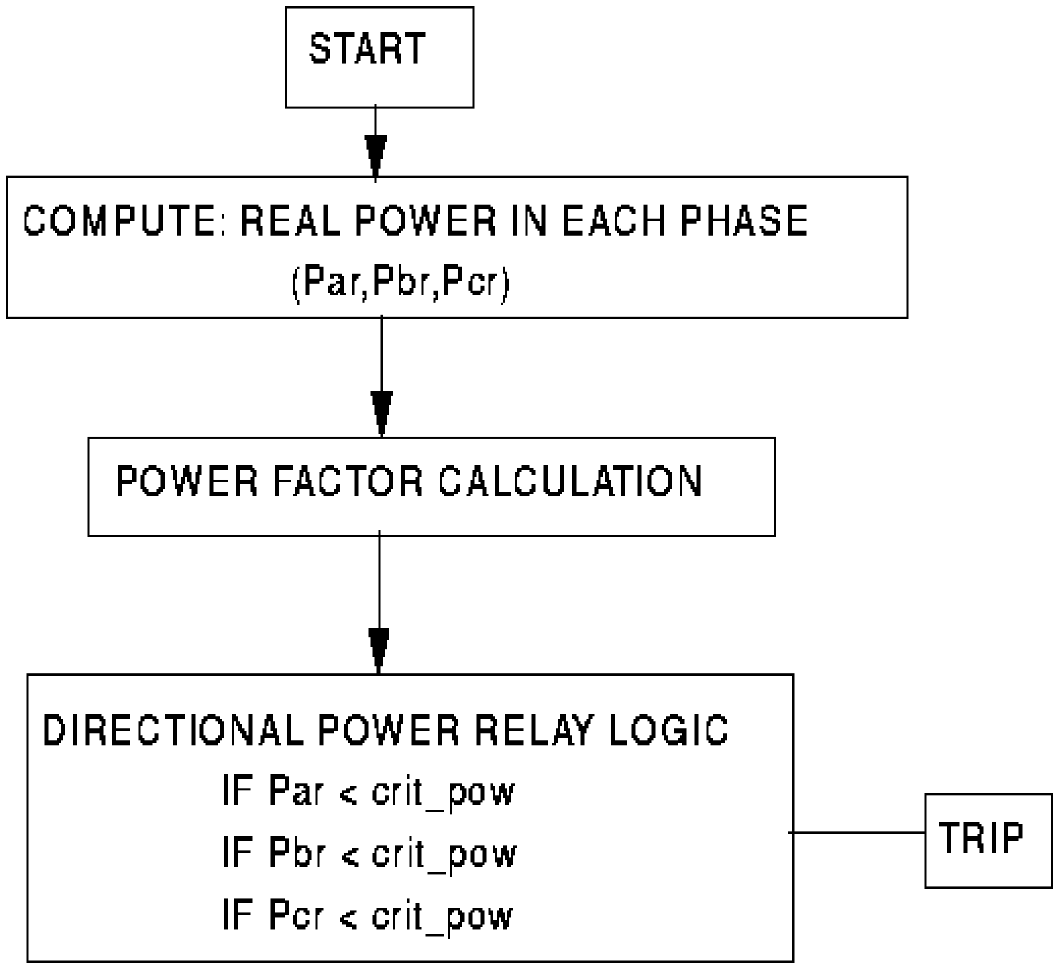

Figure 3.13: DIRECTIONAL POWER RELAY FLOW DIAGRAM

$$\text{Vc\_f} = |\text{Vc(m)}| \tag{3.170}$$

The per-unit amplitude of the frequency components were calculated as follows:

$$\text{Va\_pu} = \frac{|\text{Va\_f}|}{\text{V\_BASE}} \tag{3.171}$$

$$\text{Vb\_pu} = \frac{|\text{Vb\_f}|}{\text{V\_BASE}} \tag{3.172}$$

$$\text{Vc\_pu} = \frac{|\text{Vc\_f}|}{\text{V\_BASE}} \tag{3.173}$$

The self-excitation relay will trip if:

$$\text{Va\_pu} > \text{cap\_peak} \tag{3.174}$$

$$\text{Vb\_pu} > \text{cap\_peak} \tag{3.175}$$

$$\text{Vc\_pu} > \text{cap\_peak} \tag{3.176}$$

cap_peak was a user-defined value that represents the cutoff maximum value that the self-excitation relay could operate without tripping.

## 3.13 METERING

The flow diagram of the metering function is shown in Figure 3.15. First, the magnitude and angle of the phase voltages were calculated.

$$\text{Va} = |\text{Va\_max}| \tag{3.177}$$

$$\text{Vb} = |\text{Vb\_max}| \tag{3.178}$$

$$\text{Vc} = |\text{Vc\_max}| \tag{3.179}$$

$$\text{Va\_angle} = \frac{180}{\pi} \times \text{atan}\frac{\text{imag(Va\_max)}}{\text{real(Va\_max)}} \tag{3.180}$$

$$\text{Vb\_angle} = \frac{180}{\pi} \times \text{atan}\frac{\text{imag(Vb\_max)}}{\text{real(Vb\_max)}} \tag{3.181}$$

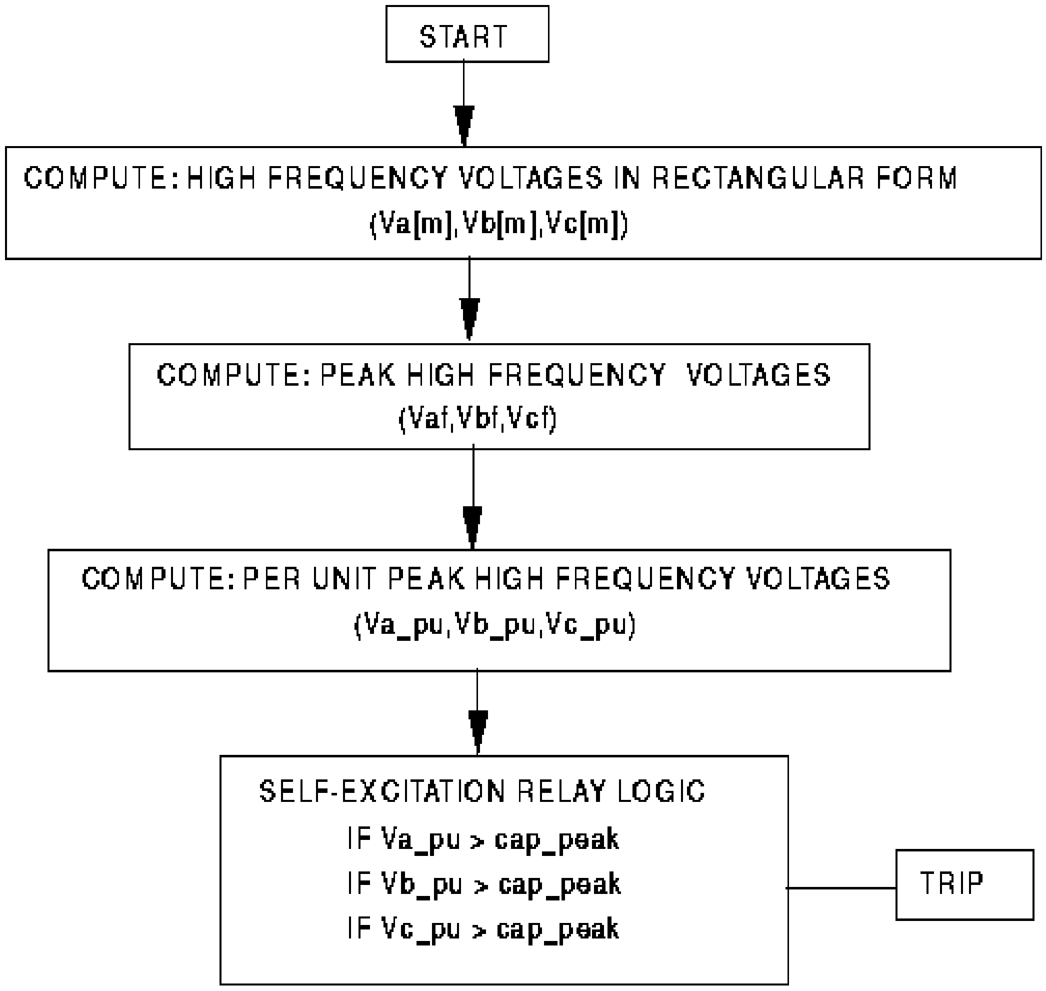
START
COMPUTE: HIGH FREQUENCY VOLTAGES IN RECTANGULAR FORM
(Va[m],Vb[m],Vc[m])
COMPUTE: PEAK HIGH FREQUENCY VOLTAGES
(Vaf,Vbf,Vcf)
COMPUTE: PER UNIT PEAK HIGH FREQUENCY VOLTAGES
(Va_pu,Vb_pu,Vc_pu)
SELF-EXCITATION RELAY LOGIC
IF Va_pu > cap_peak
IF Vb_pu > cap_peak
IF Vc_pu > cap_peak
TRIP

Figure 3.14: SELF-EXCITATION RELAY FLOW DIAGRAM

$$\text{Vc\_angle} = \frac{180}{\pi} \times \text{atan}\frac{\text{imag(Vc\_max)}}{\text{real(Vc\_max)}} \tag{3.182}$$

Second, the magnitude and angle of the phase currents were calculated.

$$\text{Ia} = |\text{Ia\_max}| \tag{3.183}$$

$$\text{Ib} = |\text{Ib\_max}| \tag{3.184}$$

$$\text{Ic} = |\text{Ic\_max}| \tag{3.185}$$

$$\text{Ia\_angle} = \frac{180}{\pi} \times \text{atan}\frac{\text{imag(Ia\_max)}}{\text{real(Ia\_max)}} \tag{3.186}$$

$$\text{Ib\_angle} = \frac{180}{\pi} \times \text{atan}\frac{\text{imag(Ib\_max)}}{\text{real(Ib\_max)}} \tag{3.187}$$

$$\text{Ic\_angle} = \frac{180}{\pi} \times \text{atan}\frac{\text{imag(Ic\_max)}}{\text{real(Ic\_max)}} \tag{3.188}$$

Third, the magnitude and angle of the single phase complex powers were calculated.

$$|\text{Sa}| = |\text{Va\_max} \times \text{Ia\_max}| \tag{3.189}$$

$$|\text{Sb}| = |\text{Vb\_max} \times \text{Ib\_max}| \tag{3.190}$$

$$|\text{Sc}| = |\text{Vc\_max} \times \text{Ic\_max}| \tag{3.191}$$

$$\text{Sa\_angle} = \frac{180}{\pi} \times \text{atan}\frac{\text{imag(Sa)}}{\text{real(Sa)}} \tag{3.192}$$

$$\text{Sb\_angle} = \frac{180}{\pi} \times \text{atan}\frac{\text{imag(Sb)}}{\text{real(Sb)}} \tag{3.193}$$

$$\text{Sc\_angle} = \frac{180}{\pi} \times \text{atan}\frac{\text{imag(Sc)}}{\text{real(Sc)}} \tag{3.194}$$

Fourth, the real and reactive power of each phase were calculated.

$$\text{Pa} = \text{Sa} \times \cos\theta \tag{3.195}$$

$$\mathrm{Pb} = \mathrm{Sb} \times \cos\theta \tag{3.196}$$

$$\mathrm{Pc} = \mathrm{Sc} \times \cos\theta \tag{3.197}$$

$$\mathrm{Qa} = \mathrm{Sa} \times \sin\theta \tag{3.198}$$

$$\mathrm{Qb} = \mathrm{Sb} \times \sin\theta \tag{3.199}$$

$$\mathrm{Qc} = \mathrm{Sc} \times \sin\theta \tag{3.200}$$

Fifth, the total apparent, real, and reactive power were calculated.

$$S\_t = Sa + Sb + Sc \tag{3.201}$$

$$P\_t = Pa + Pb + Pc \tag{3.202}$$

$$Q\_t = Qa + Qb + Qc \tag{3.203}$$

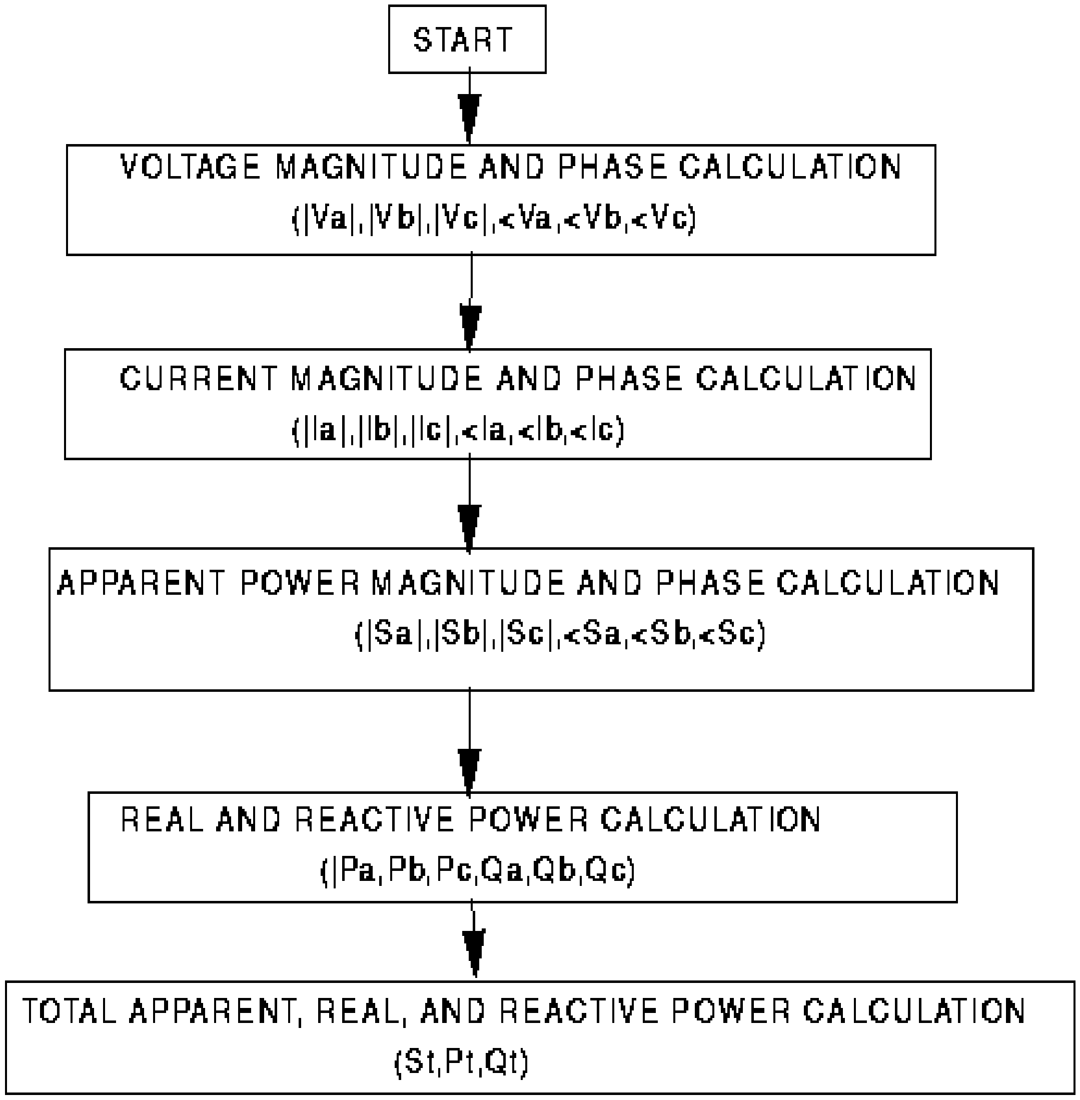
START
VOLTAGE MAGNITUDE AND PHASE CALCULATION
(|Va|,|Vb|,|Vc|,<Va,<Vb,<Vc)
CURRENT MAGNITUDE AND PHASE CALCULATION
(|Ia|,|Ib|,|Ic|,<Ia,<Ib,<Ic)
APPARENT POWER MAGNITUDE AND PHASE CALCULATION
(|Sa|,|Sb|,|Sc|,<Sa,<Sb,<Sc)
REAL AND REACTIVE POWER CALCULATION
(|Pa,Pb,Pc,Qa,Qb,Qc)
TOTAL APPARENT, REAL, AND REACTIVE POWER CALCULATION
(St,Pt,Qt)

Figure 3.15: FLOW DIAGRAM OF THE METERING RELAY FUNCTION

# Chapter 4

# COMPUTER SIMULATION RESULTS

## 4.1 COMPUTER SIMULATION OF A SAMPLE SYSTEM

A simplified one-line diagram of a DSG distribution system is shown in Figure 4.1. The utility substation voltage was 12.47 KV. The dispersed storage generator voltage was 4 KV. The isolation transformer was represented by xfmr. The relay was tested at the dispersed storage generator.

The system in Figure 4.1 was simulated on EMTP (Electro-Magnetic Transient Program).[20] After the utility circuit breaker was opened, the multifunction relay will monitor the amplitude spectrum of the high frequency voltages for the self-excitation relay using the high sampling rate. Also, the multifunction relay will monitor the voltages, currents, and the frequency at the cogeneration site.

## 4.2 INPUT DATA FOR THE LOW FREQUENCY RELAYS

This section contains graphs of the input data used by the low frequency relays to produce output results. A balanced three-phase load of 2 Ω per phase was used. The sample voltage Va is shown in Figure 4.2. The digitally filtered sample voltage Va is shown in Figure 4.3. Figure 4.4 shows the amplitude of the phasor voltage Va. The graphs for Vb, Vc, Ia, Ib, and Ic looked very similar in form. The sample impedance Za is shown in Figure 4.5. The digitally filtered sample impedance Za is shown in Figure 4.6. Figure 4.7 shows the amplitude of the phasor impedance Za. The graphs for Zb, and Zc looked very similar in form. The single phase sample power Pa is shown in Figure 4.8. The digitally filtered sample power Pa is shown in Figure 4.9. Figure 4.10 shows the amplitude of the phasor power Pa. The graphs for Pb, and Pc looked very similar in form. The sample three phase

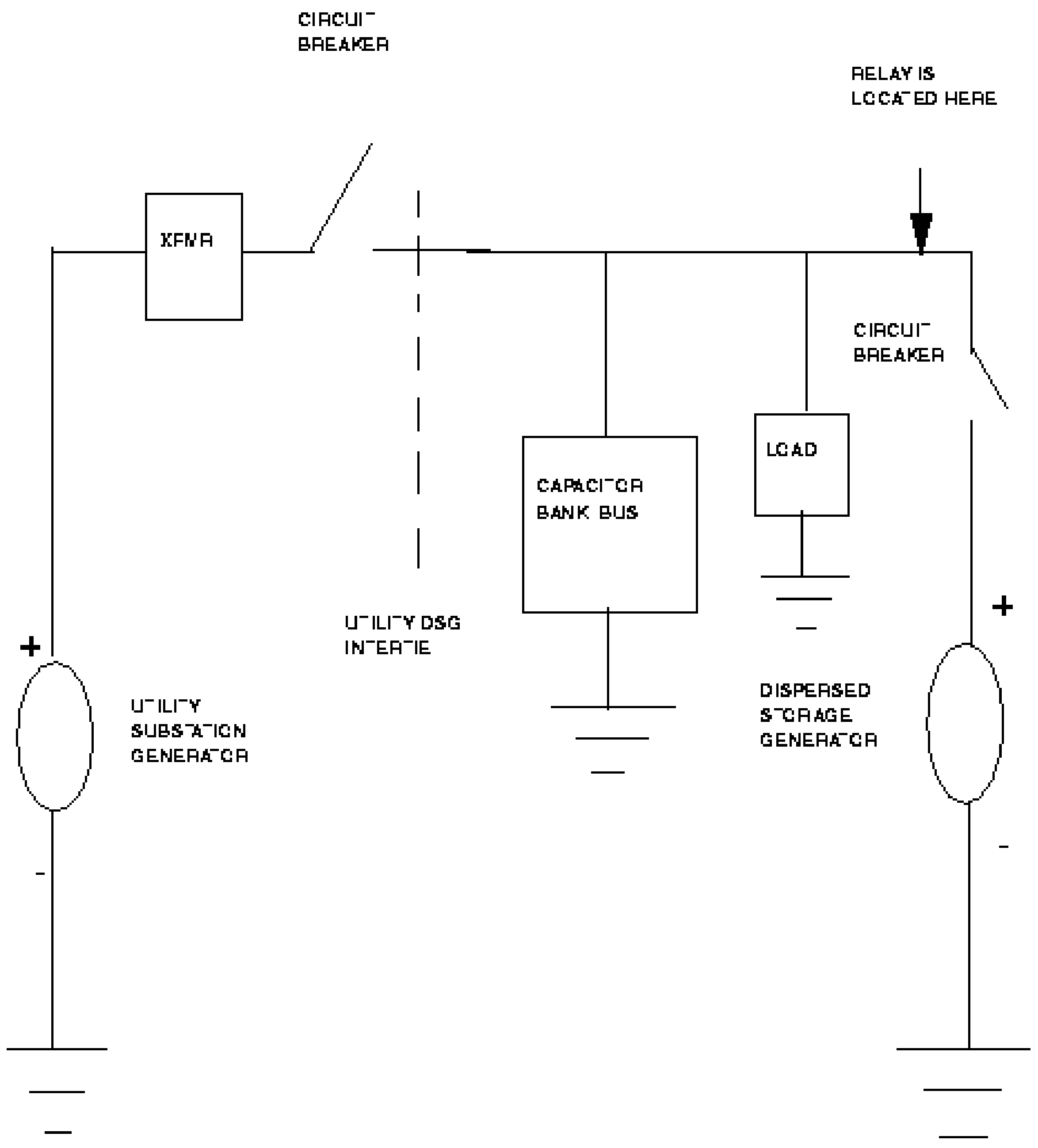

Figure 4.1: ONE-LINE DIAGRAM OF A DSG DISTRIBUTION SYSTEM

power Pg is shown in Figure 4.11. The digitally filtered sample three phase power Pg is shown in Figure 4.12. Figure 4.13 shows the amplitude of the three phase power Pg. All of these waveforms were graphed for six cycles. The symbol k represented the sample number. The nominal frequency was 60 Hz. The sampling frequency was 720 Hz. [1] The simulation time was 100 ms. The time step was $1400\mu s$. The total number of sample points was 72. The equations for the sample voltages were:

$$\mathrm{Va} = 100\sin(2\pi \times 60\mathrm{t}) \tag{4.1}$$

$$\mathrm{Vb} = 100\sin(2\pi \times 60\mathrm{t} - 120) \tag{4.2}$$

$$\mathrm{Vc} = 100\sin(2\pi \times 60\mathrm{t} + 120) \tag{4.3}$$

The equations for the sample currents were:

$$\mathrm{Ia} = 50\sin(2\pi \times 60\mathrm{t}) \tag{4.4}$$

[1]The cut-off frequency was 360 Hz (12 times the fundamental 60 Hz frequency).

$$\mathrm{Ib} = 50\sin(2\pi \times 60\mathrm{t} - 120) \tag{4.5}$$

$$\mathrm{Ic} = 100\sin(2\pi \times 60\mathrm{t} + 120) \tag{4.6}$$

The equations for the sample impedances were:

$$\mathrm{Za} = \frac{\mathrm{Va}}{\mathrm{Ia}} \tag{4.7}$$

$$\mathrm{Zb} = \frac{\mathrm{Vb}}{\mathrm{Ib}} \tag{4.8}$$

$$\mathrm{Zc} = \frac{\mathrm{Vc}}{\mathrm{Ic}} \tag{4.9}$$

The equations for the single phase sample powers were:

$$\mathrm{Pa} = \mathrm{Va} \times \mathrm{Ia} \tag{4.10}$$

$$\mathrm{Pb} = \mathrm{Vb} \times \mathrm{Ib} \tag{4.11}$$

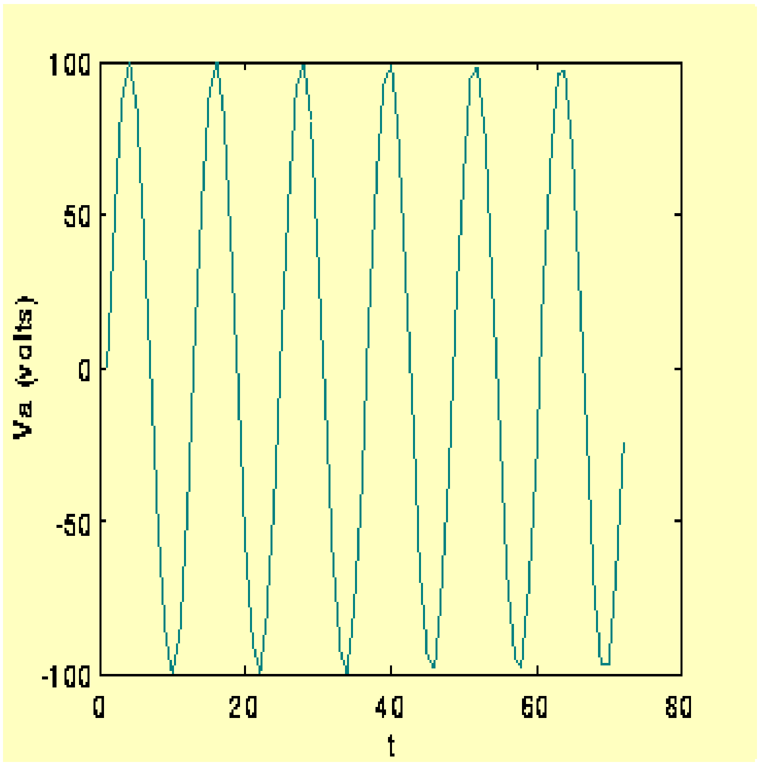

Figure 4.2: SAMPLE VOLTAGE Va

$$\text{Pc} = \text{Vc} \times \text{Ic} \tag{4.12}$$

The equation for the three phase sample power was:

$$\text{Pg} = (\text{Va} \times \text{Ia}) + (\text{Vb} \times \text{Ib}) + (\text{Vc} \times \text{Ic}) \tag{4.13}$$

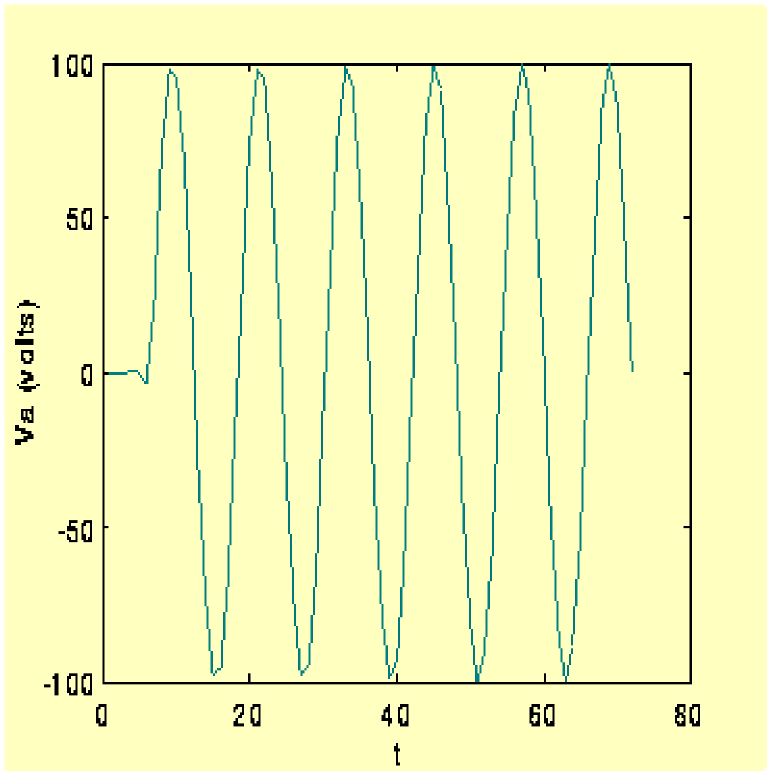

Figure 4.3: DIGITALLY FILTERED SAMPLE VOLTAGE Va

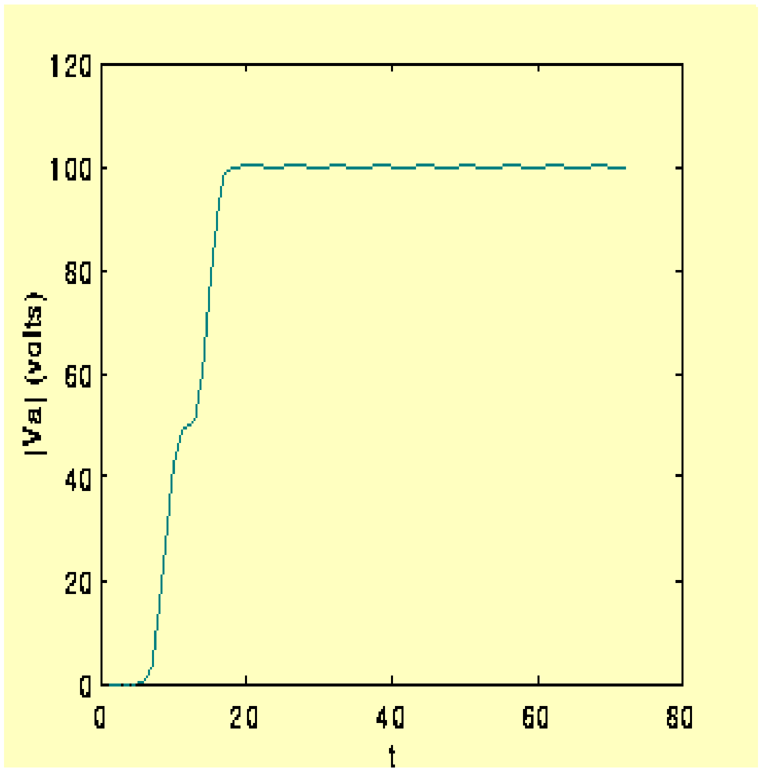

Figure 4.4: AMPLITUDE OF THE PHASOR VOLTAGE Va

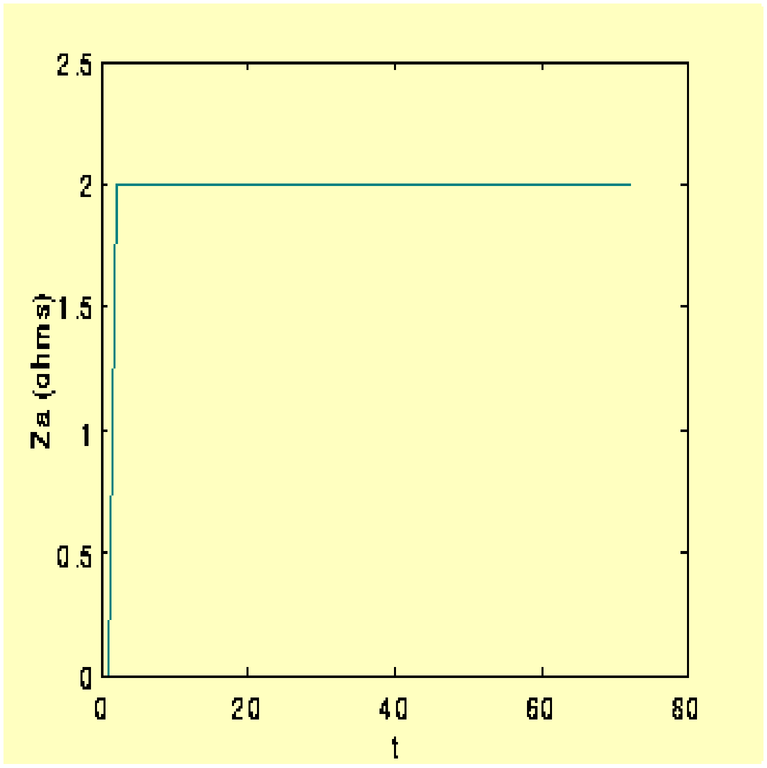

Figure 4.5: SAMPLE IMPEDANCE Za

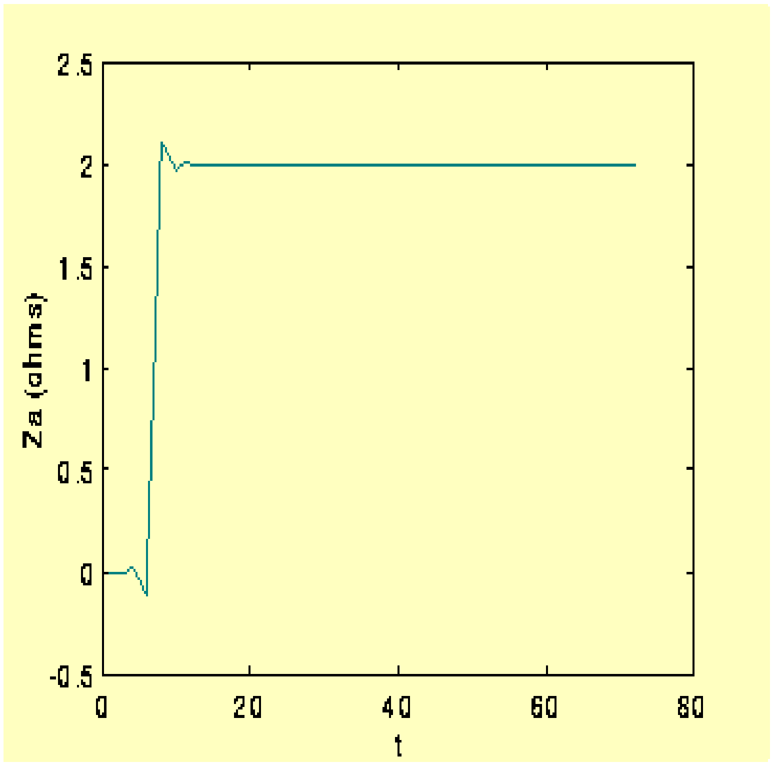

Figure 4.6: DIGITALLY FILTERED SAMPLE IMPEDANCE Za

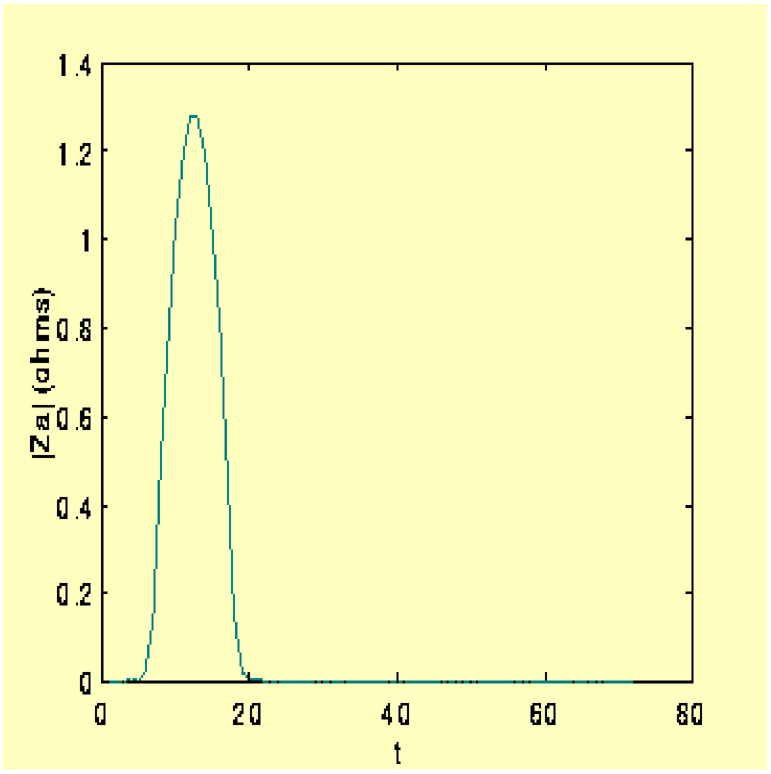

Figure 4.7: AMPLITUDE OF THE PHASOR IMPEDANCE Za

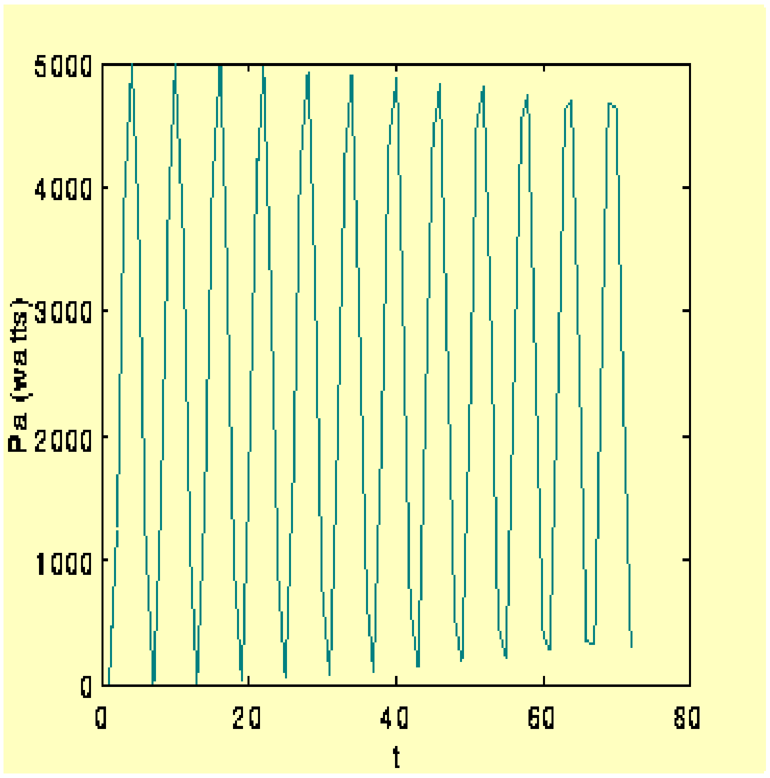

Figure 4.8: SINGLE PHASE SAMPLE POWER Pa

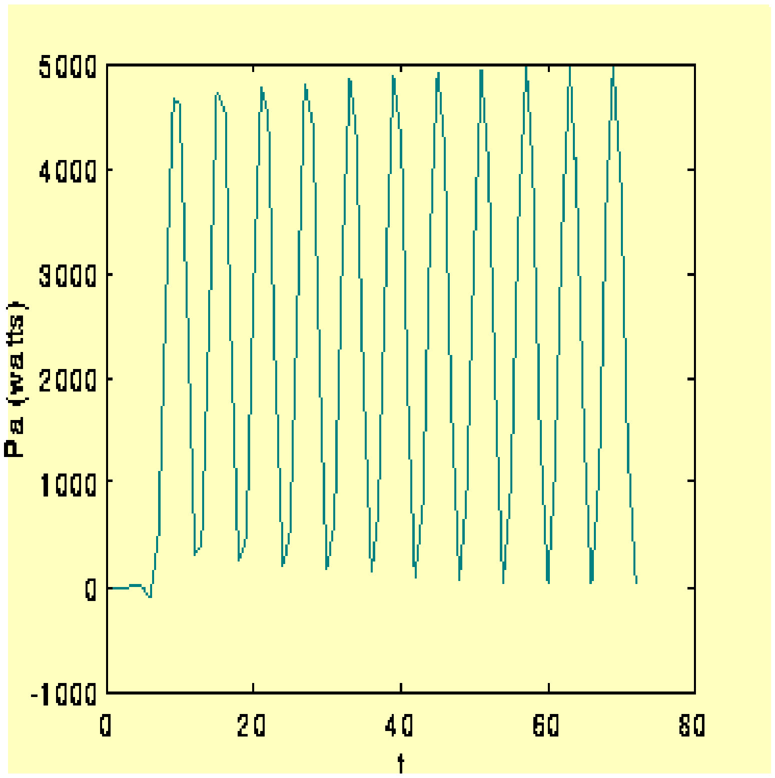

Figure 4.9: DIGITALLY FILTERED SAMPLE POWER Pa

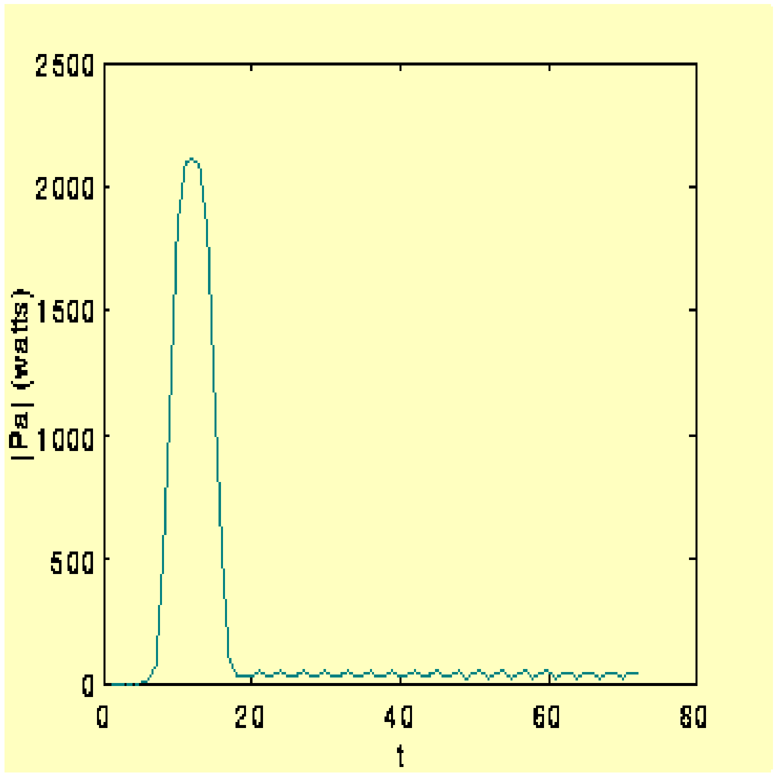

Figure 4.10: AMPLITUDE OF THE PHASOR POWER Pa

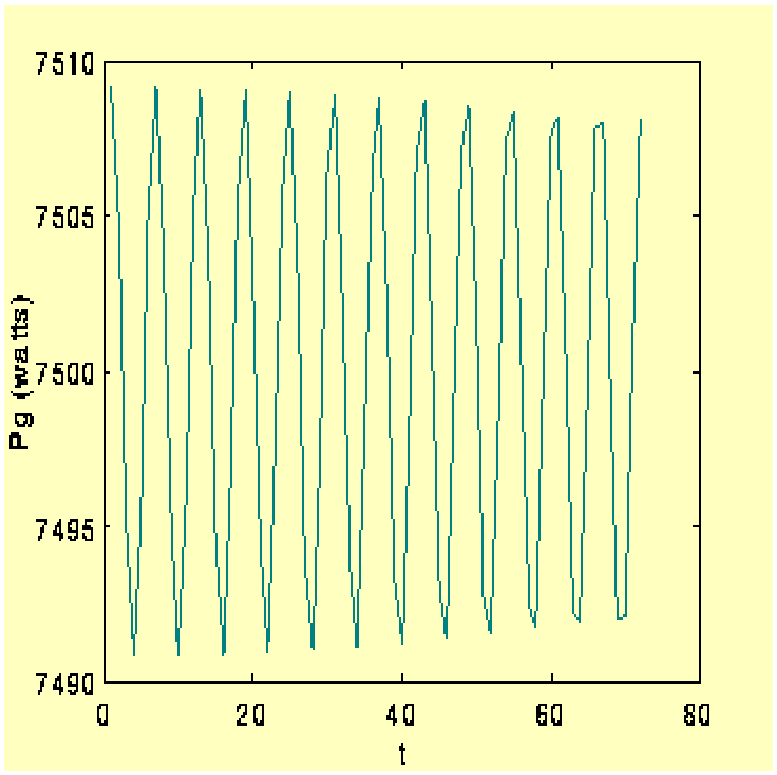

Figure 4.11: SAMPLE THREE PHASE POWER Pg

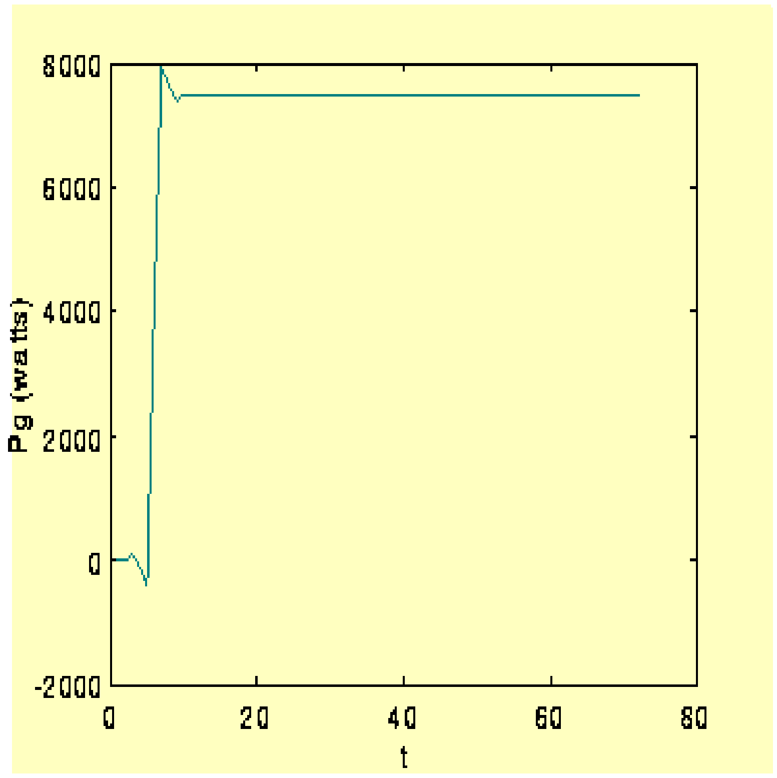

Figure 4.12: DIGITALLY FILTERED SAMPLE POWER Pg

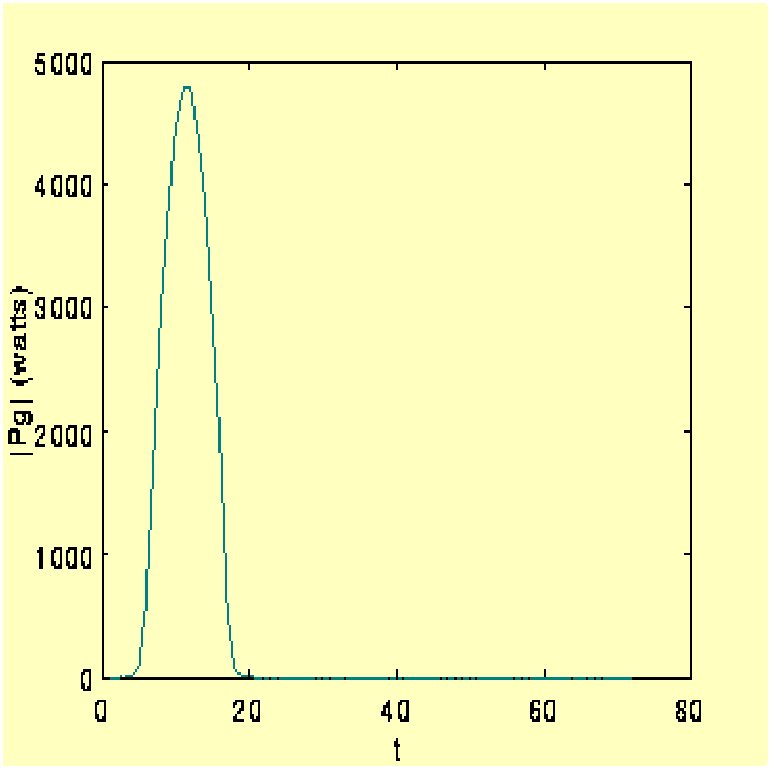

Figure 4.13: AMPLITUDE OF THE PHASOR POWER Pg

## 4.3 INPUT DATA FOR THE HIGH FREQUENCY RELAY

This section contains graphs of the input data used by the high frequency relay to produce output results. The actual high frequency voltage Va is shown in Figure 4.14. The sample high frequency voltage Va is shown in Figure 4.15. The amplitude spectrum of Voltage Va was plotted, in Figure 4.16. These waveforms were graphed for eight cycles. The amplitude spectrums for Vb, and Vc looked very similar. The symbol m represented the frequency index. The sampling frequency for the self-excitation relay (high-frequency function) was 3000 Hz.[2] The EMTP simulation time was $347\mu s$ for the high frequency function. The time step was $20\mu s$. The total number of sample points was 96.

## 4.4 SAMPLE SYSTEM OUTPUT FOR RELAY MODULES

In this section, a variety of operating conditions are applied to the individual relay modules that made up the multifunction relay. The phasors that were computed by the equations in Section 3.2, and graphed in Section 4.2 will be used by the relays to develop an output

[2]The cut-off frequency was 1000 Hz for the high-frequency sampler.

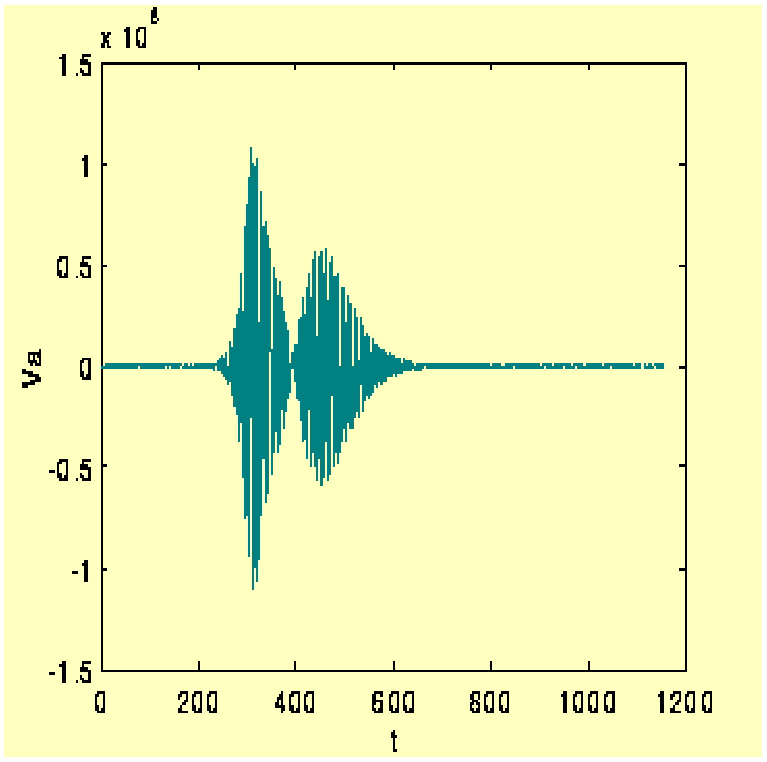

Figure 4.14: HIGH FREQUENCY VOLTAGE Va

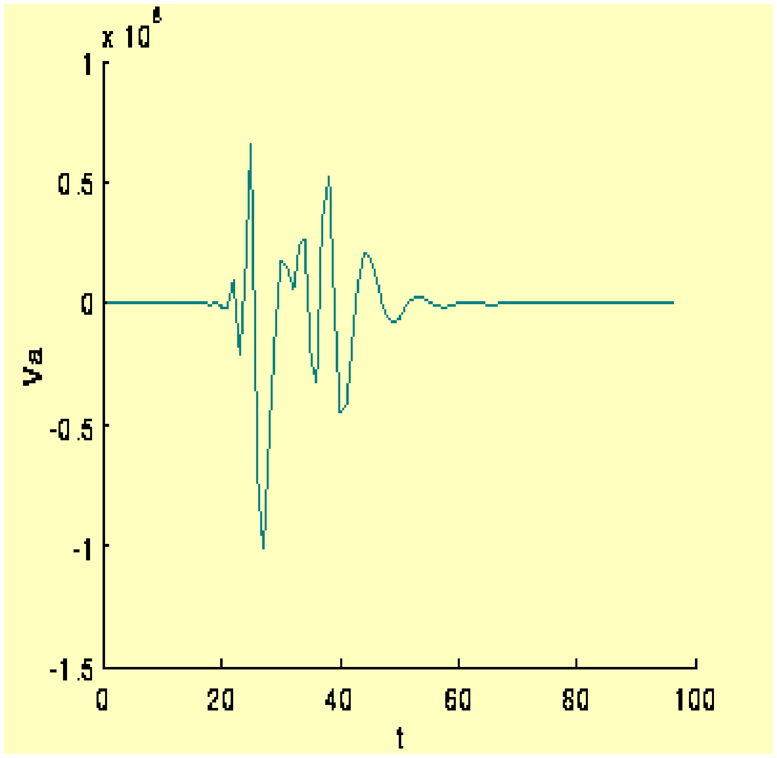

Figure 4.15: SAMPLE HIGH FREQUENCY VOLTAGE Va

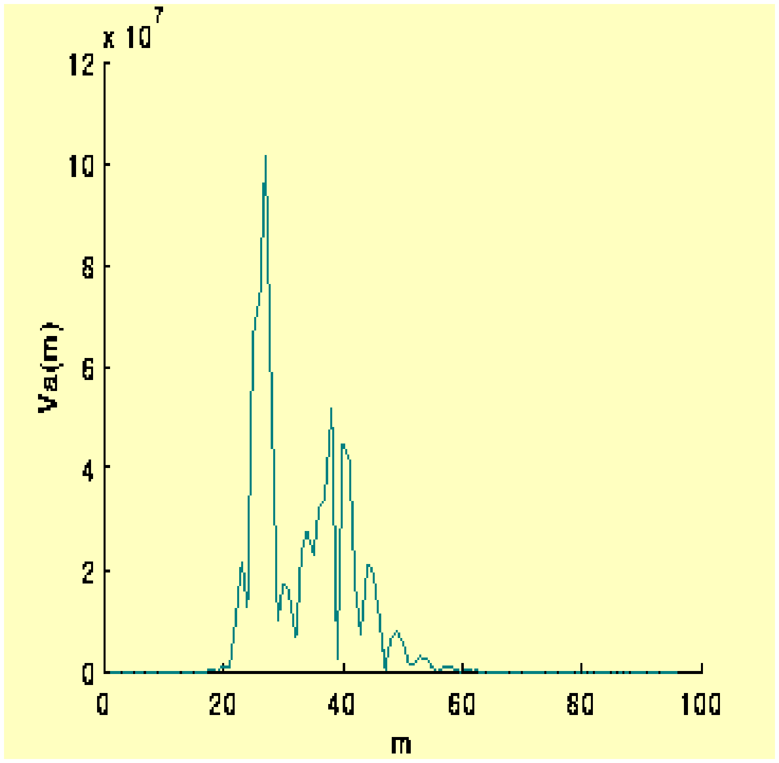

Figure 4.16: AMPLITUDE SPECTRUM OF HIGH FREQUENCY VOLTAGE Va

representing a system quantity. For each relay module, graphs of the critical operating parameters and the relay logic will be obtained. A block signal will be represented by a magnitude of zero on the relay output. A trip signal will be represented by a magnitude of one on the relay output.

### 4.4.1 CURRENT RELAY OUTPUT

In the simulation run for the current relay, the following parameters were fixed:

$$\text{i\_L} = 100 \text{ A} \tag{4.14}$$

$$\text{V\_BASE} = 300 \text{ V} \tag{4.15}$$

$$\text{S\_BASE} = 15 \text{ KW} \tag{4.16}$$

$$\text{I\_high} = 1.1 \text{ p.u.} \tag{4.17}$$

$$\text{I\_peak} = 1.5 \text{ p.u.} \tag{4.18}$$

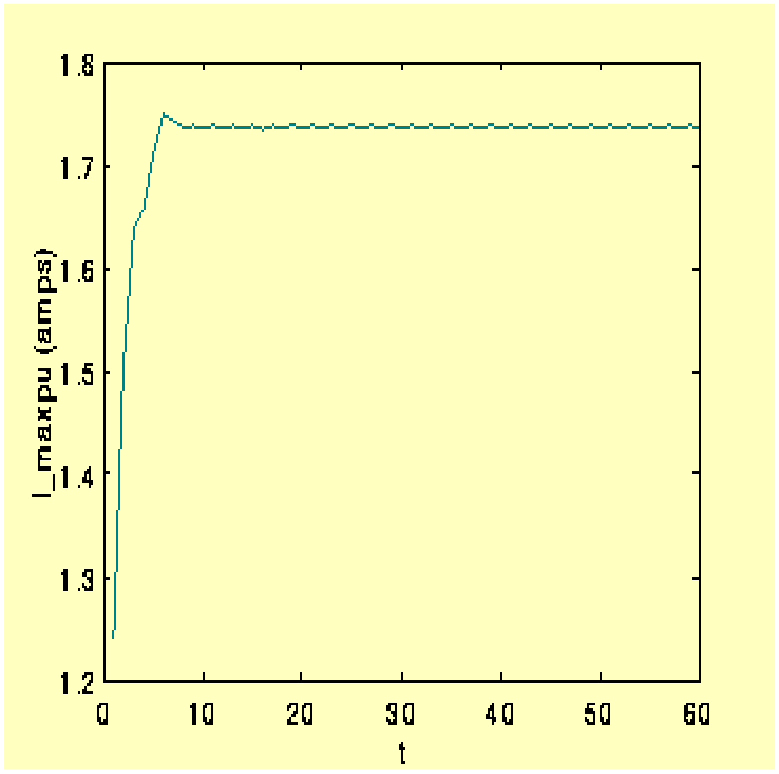

Figure 4.17: PER UNIT MAXIMUM FAULT CURRENT

$$\text{TDS} = \frac{1}{2} \tag{4.19}$$

The two variable parameters observed and plotted were: I_maxpu, and I_minpu. In Figure 4.17, the per unit values for the maximum fault current are plotted. In Figure 4.18, the per unit values for the minimum fault current are plotted.

The current relay did not trip at all. The current relay output status is graphed in Figure 4.19.

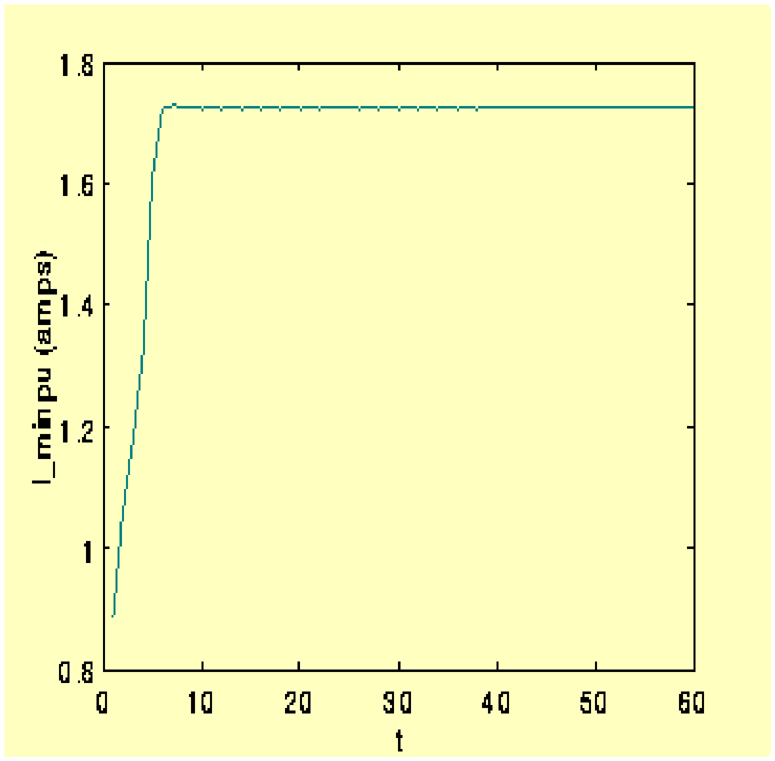

Figure 4.18: PER UNIT MINIMUM FAULT CURRENT

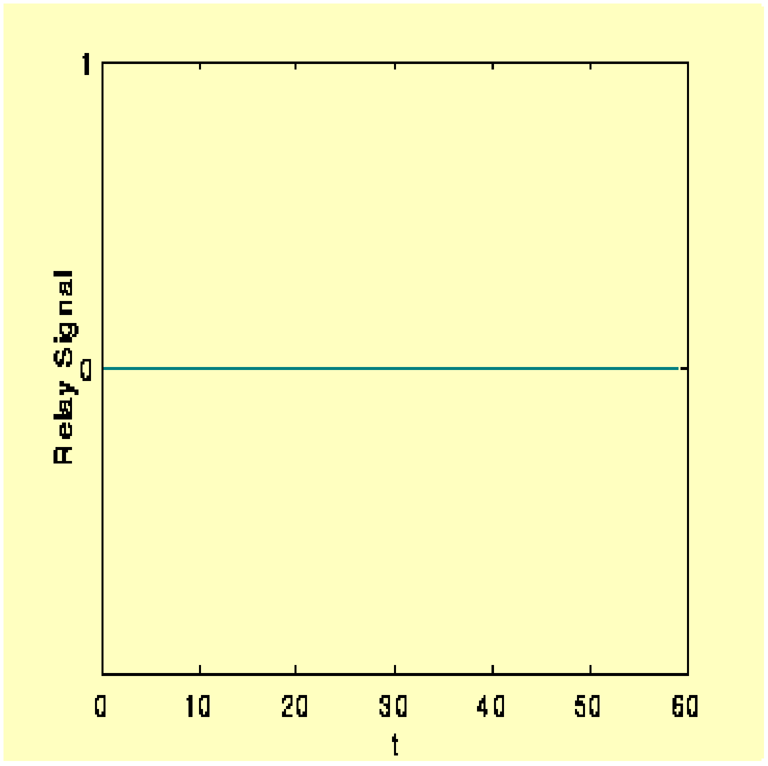

Figure 4.19: CURRENT RELAY OUTPUT STATUS

### 4.4.2 ISLANDING RELAY OUTPUT

In the simulation run for the islanding relay, the following parameters were fixed:

$$\text{S\_BASE} = 15 \text{ kW} \tag{4.20}$$

$$\text{ks} = 0.2 \text{ p.u.} \tag{4.21}$$

The DSG was disconnected from the utility at 0.15 sec. The variable parameter observed was per-unit change in power (isl). It is plotted in Figure 4.20. The islanding relay did not trip at all. The islanding relay output status is graphed in Figure 4.21.

### 4.4.3 LOSS-OF-EXCITATION RELAY OUTPUT

In the simulation run for the loss-of-excitation relay, the following parameters were fixed:

$$\text{imped} = 0.15 \text{ p.u.} \tag{4.22}$$

$$\text{V\_BASE} = 50 \text{ V} \tag{4.23}$$

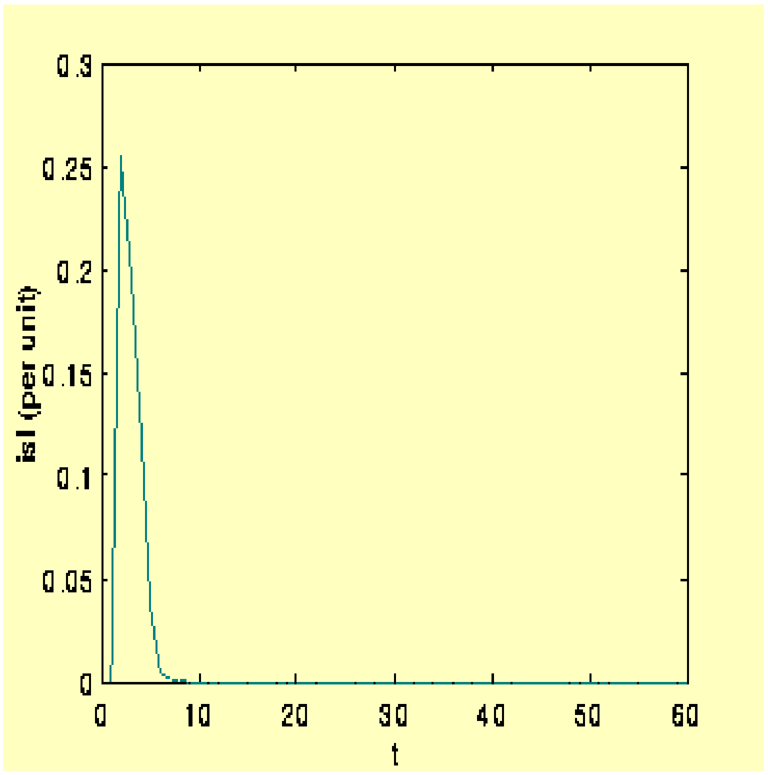

Figure 4.20: PER UNIT CHANGE IN POWER

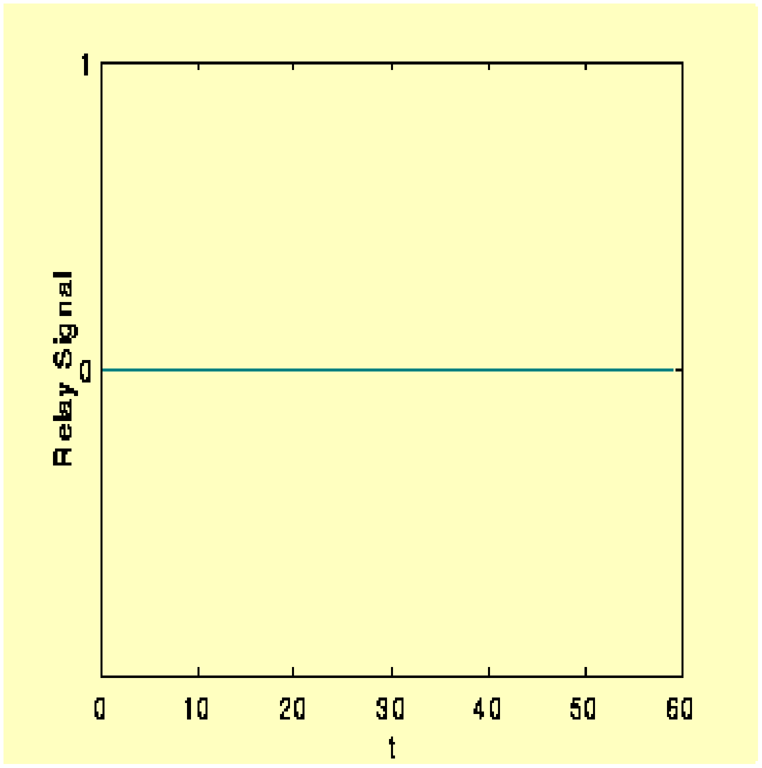

Figure 4.21: ISLANDING RELAY OUTPUT STATUS

$$\text{S\_BASE} = 5 \text{ KW} \tag{4.24}$$

The three variable parameters observed were: Z_a, Z_b, and Z_c. The variable parameter Z_a is plotted in Figure 4.22. The graphs for Z_b, and Z_c looked very similar in form. The loss-of-excitation relay tripped at the twenty-third sample point. Za was equal to 0.000004 Ω, which is less than Zapp. Zapp is equal to 0.064952 Ω for this particular operating condition. The loss-of-excitation relay tripped for the remaining portion of the simulation run. The loss-of-excitation relay output status was graphed in Figure 4.23.

### 4.4.4 DIFFERENTIAL RELAY OUTPUT

In the simulation run for the differential relay, the following parameters were fixed:

$$\text{V\_BASE} = 350.0 \text{ V} \tag{4.25}$$

$$\text{S\_BASE} = 74.5 \text{ KW} \tag{4.26}$$

$$\text{S} = 0.2 \text{ p.u.} \tag{4.27}$$

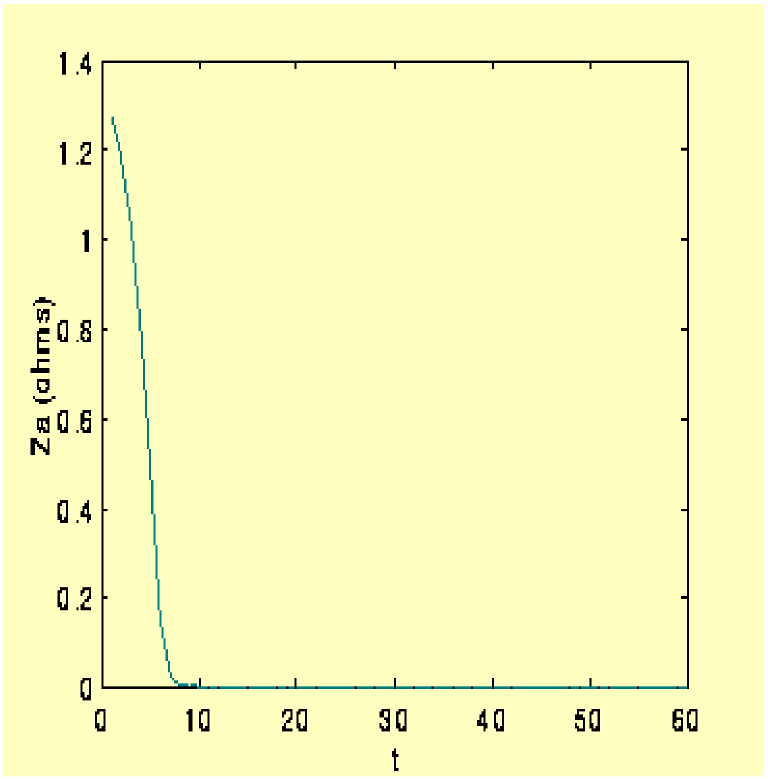

Figure 4.22: IMPEDANCE FOR PHASE A

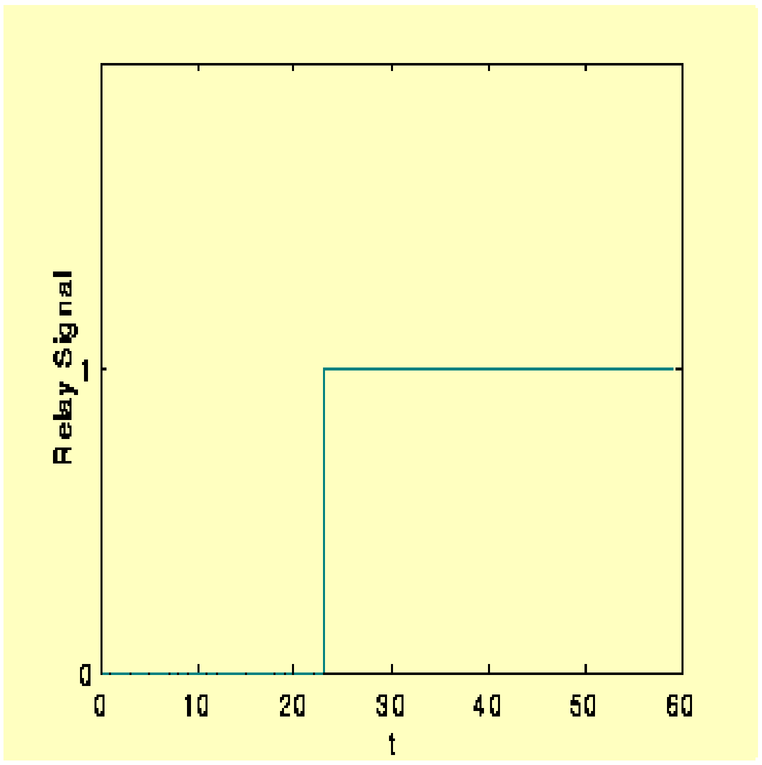

Figure 4.23: LOSS-OF-EXCITATION RELAY OUTPUT

The three variable parameters observed were: S_1, S_2, and S_3. The variable parameter S_1 is plotted in Figure 4.24. The graphs for S_2, and S_3 looked very similar in form. The differential relay tripped at the twenty-sixth sample point. S_1 was equal to 0.200117 per unit, which was greater than S. The differential relay resetted at the twenty-seventh sample point. The differential relay tripped again at the thirty-second sample point. S_1 was equal to 0.200100 per unit, which was greater than S. The differential relay resetted at the thirty-third sample point. The differential relay tripped at the thirty-eighth sample point. S_1 was equal to 0.200079 per unit, which was greater than S. The differential relay resetted at the thirty-ninth sample point. The differential relay tripped at the forty-fourth sample point. S_1 was equal to 0.2001171 per unit, which was greater than S. The differential relay resetted at the forty-fifth sample point. The differential relay tripped at the fiftieth sample point. S_1 was equal to 0.20027 per unit, which was greater than S. The differential relay resetted at the fifty-first sample point. The differential relay output is graphed in Figure 4.25.

### 4.4.5 FREQUENCY RELAY OUTPUT

In the first simulation run, the frequency was 65 Hz. The following parameters were fixed for the frequency relay:

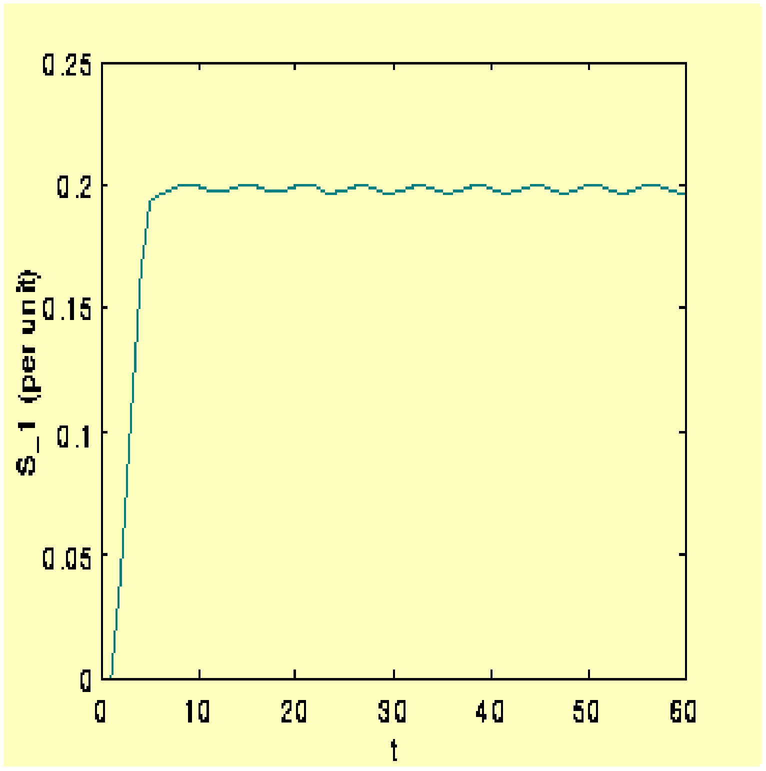

Figure 4.24: SENSITIVITY SLOPE FOR PHASE A

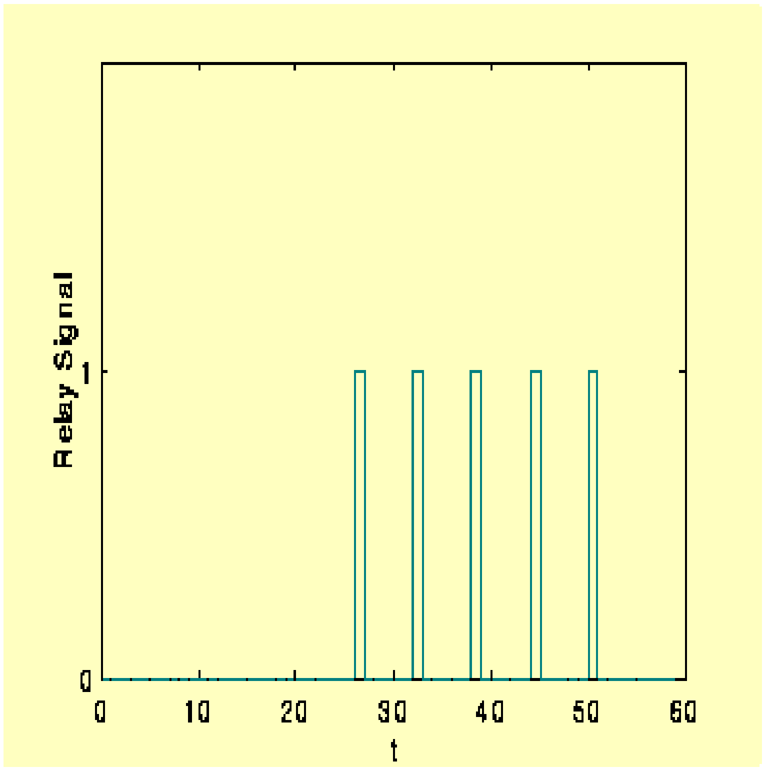

Figure 4.25: DIFFERENTIAL RELAY OUTPUT

$$\text{high\_f} = 62.0 \text{ Hz} \tag{4.28}$$

$$\text{low\_f} = 58.0 \text{ Hz} \tag{4.29}$$

The three variable parameters observed were: theta, theta_del, and freq_new. The variable parameter theta is plotted in Figure 4.26. The graph for theta_del looked very similar. The variable parameter freq_new is plotted in Figure 4.27. The frequency relay tripped at the twenty-third sample point. freq_new was equal to 65 Hz, which was greater than high_f. The frequency relay output is graphed in Figure 4.28.

In the second simulation run, the frequency was 55 Hz. The variable parameter theta is plotted in Figure 4.29. The variable parameter freq_new is plotted in Figure 4.30. The frequency relay tripped at the twenty-third sample point. freq_new was equal to 55 Hz, which was less than low_f. The frequency relay output is graphed in Figure 4.31.

In the third simulation run, the frequency was 60 Hz. The variable parameter theta is plotted in Figure 4.32. The variable parameter freq_new is plotted in Figure 4.33. The frequency relay did not trip at all. The frequency relay output is graphed in Figure 4.34.

In the fourth simulation run, the frequency was 60 Hz for the first three cycles. The frequency switched from 60 Hz to 65 Hz for the last three cycles. The variable parameter

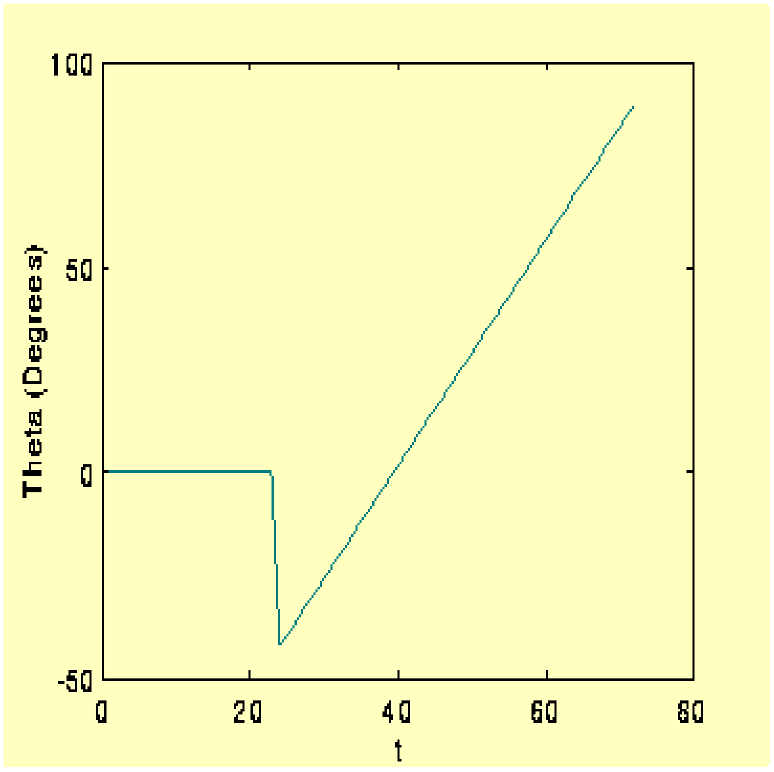

Figure 4.26: PHASE ANGLE AT THE KTH SAMPLE

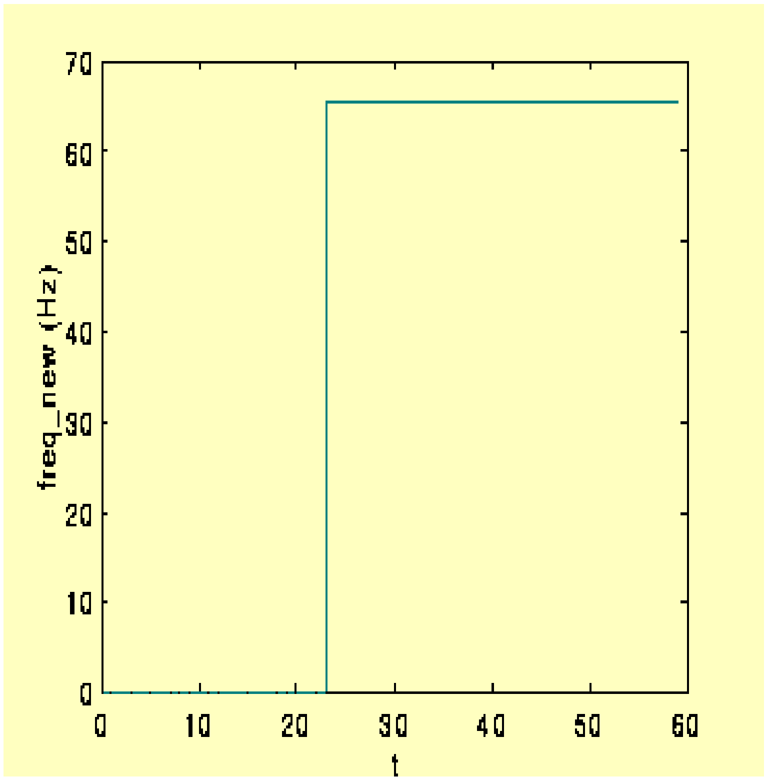

Figure 4.27: FREQUENCY ESTIMATOR

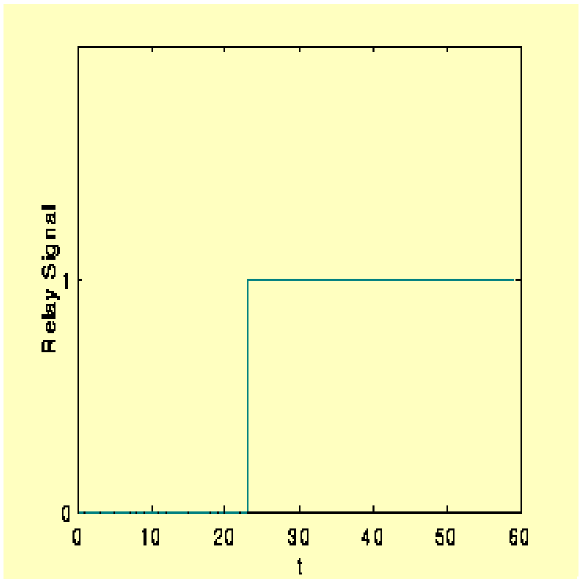

Figure 4.28: FREQUENCY RELAY OUTPUT FOR THE 65 Hz CASE

theta is plotted in Figure 4.35. The variable parameter freq_new is plotted in Figure 4.36. The frequency relay tripped at the forty-second sample point. freq_new was equal to 62.192307 Hz, which was greater than high_f. The frequency relay output is graphed in Figure 4.37.

### 4.4.6 VOLTAGE RELAY OUTPUT

In the simulation run for the voltage relay, the following parameters were fixed:

$$\text{V\_BASE} = 100 \text{ V} \tag{4.30}$$

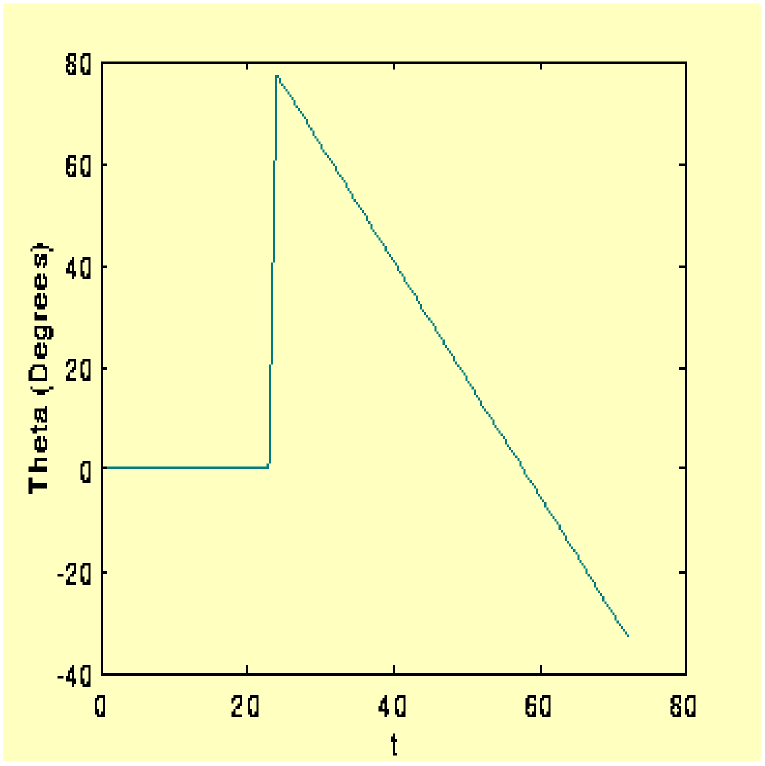

Figure 4.29: PHASE ANGLE AT THE KTH SAMPLE

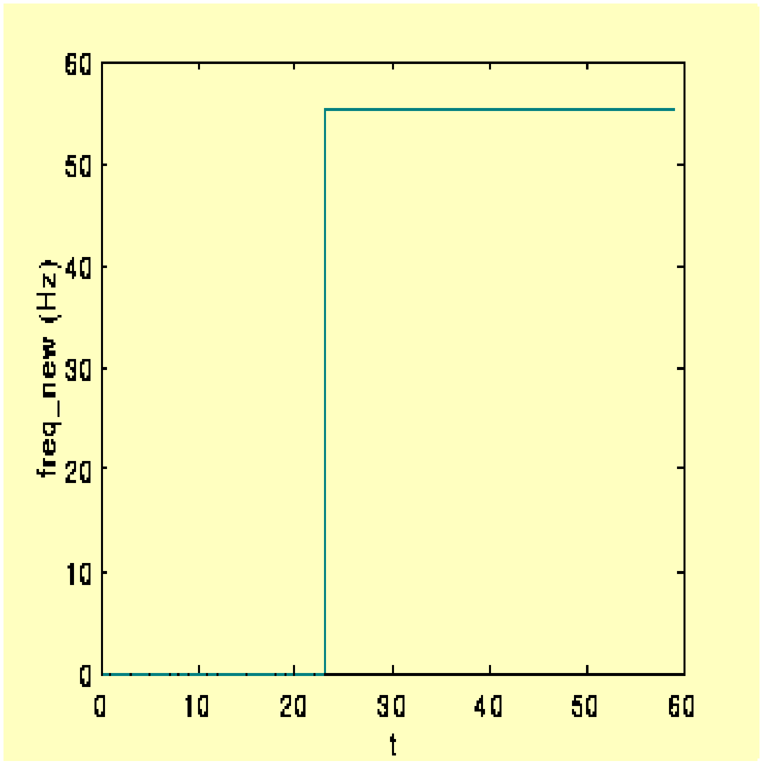

Figure 4.30: FREQUENCY ESTIMATOR

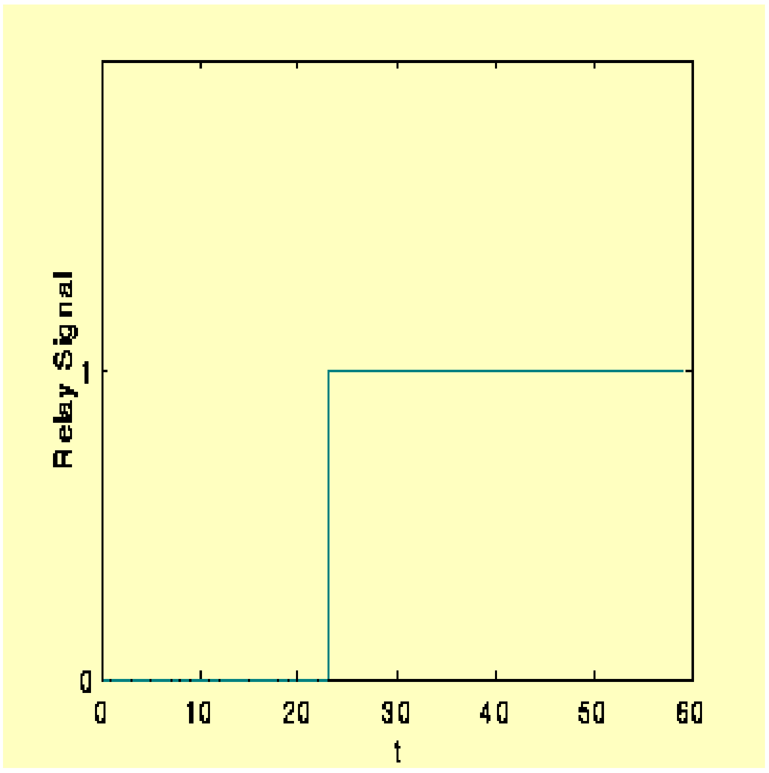

Figure 4.31: FREQUENCY RELAY OUTPUT FOR THE 55 Hz CASE

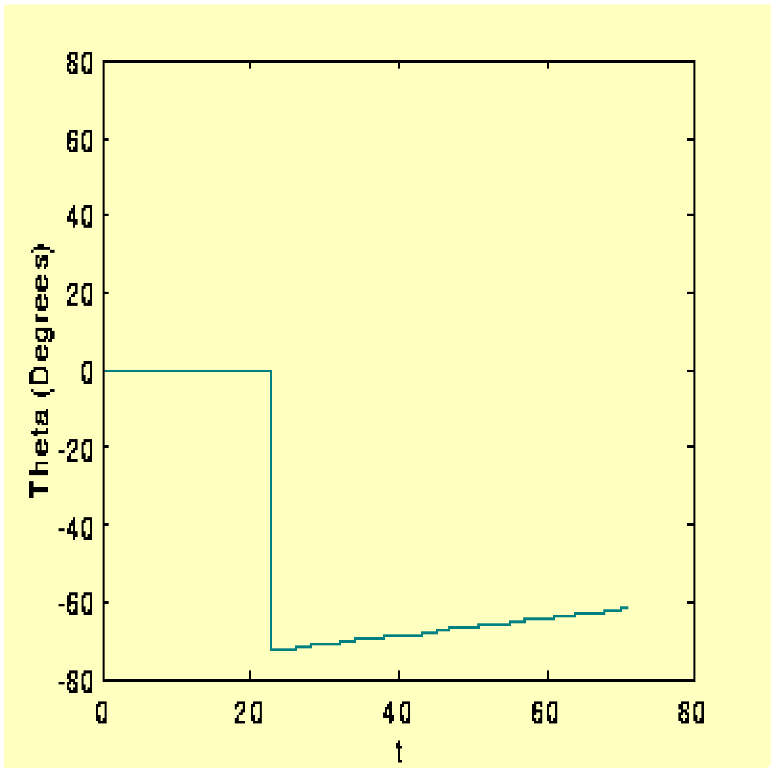

Figure 4.32: PHASE ANGLE AT THE KTH SAMPLE

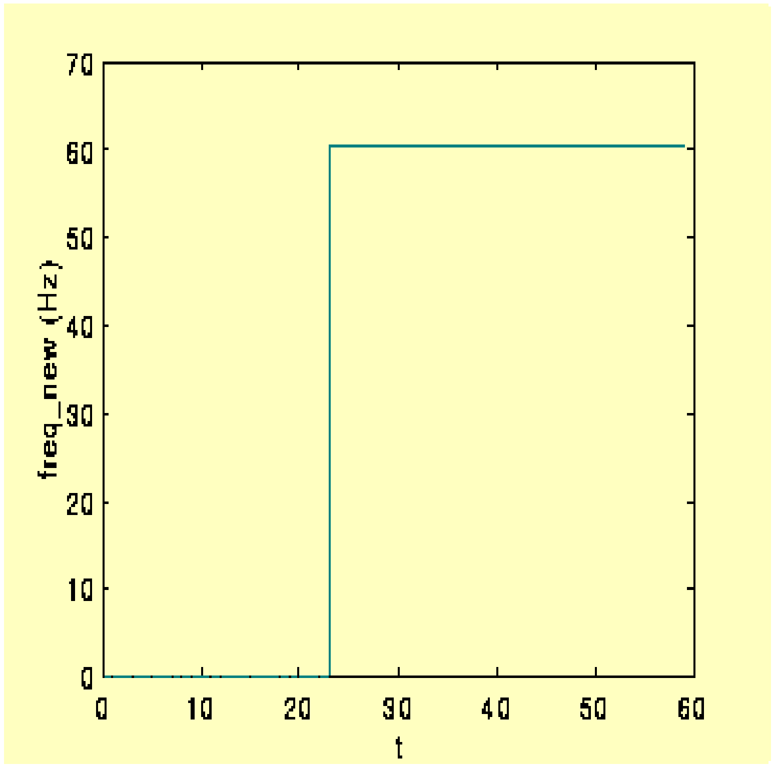

Figure 4.33: FREQUENCY ESTIMATOR

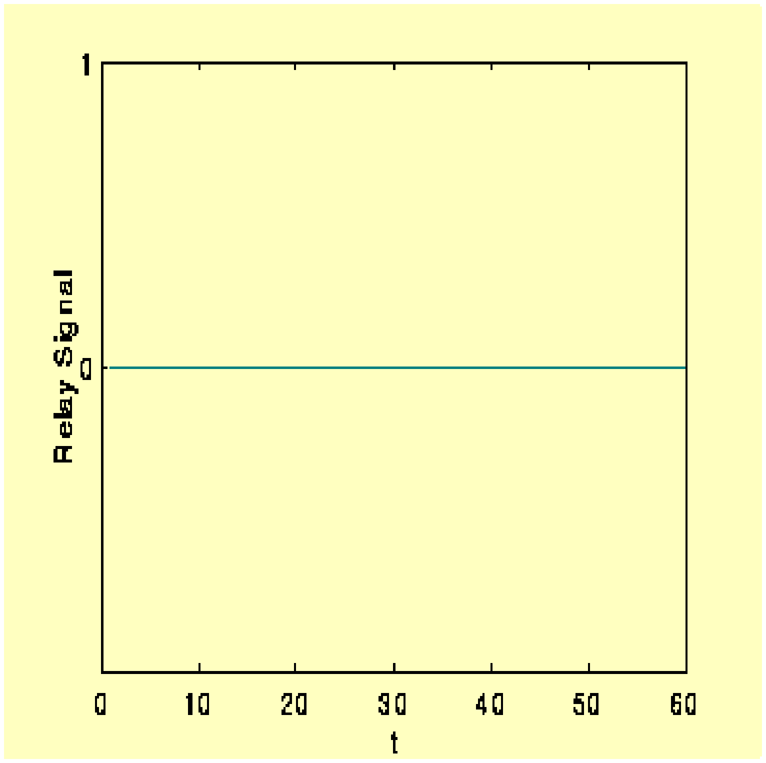

Figure 4.34: FREQUENCY RELAY OUTPUT FOR THE 60 Hz CASE

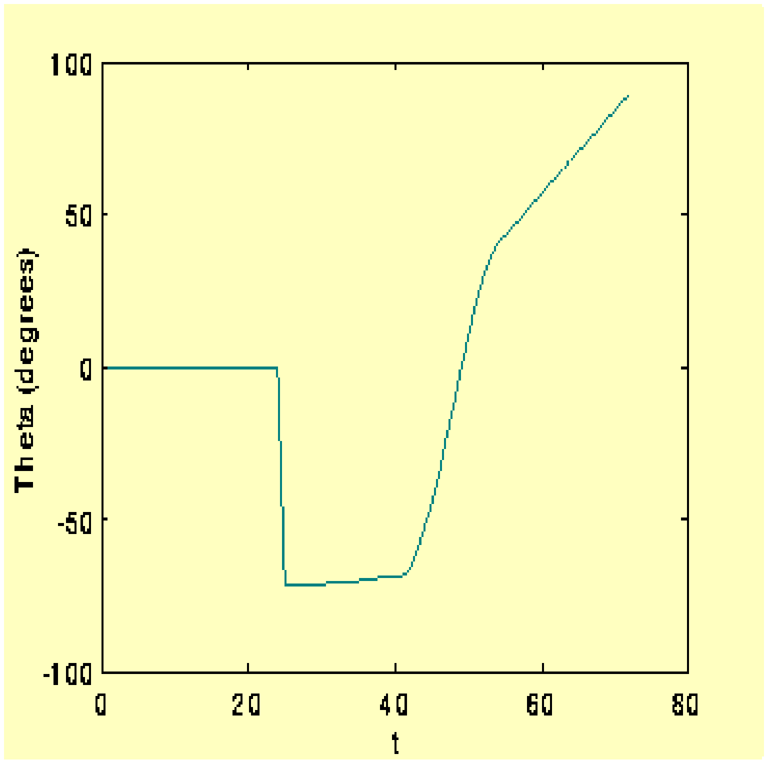

Figure 4.35: PHASE ANGLE AT THE KTH SAMPLE

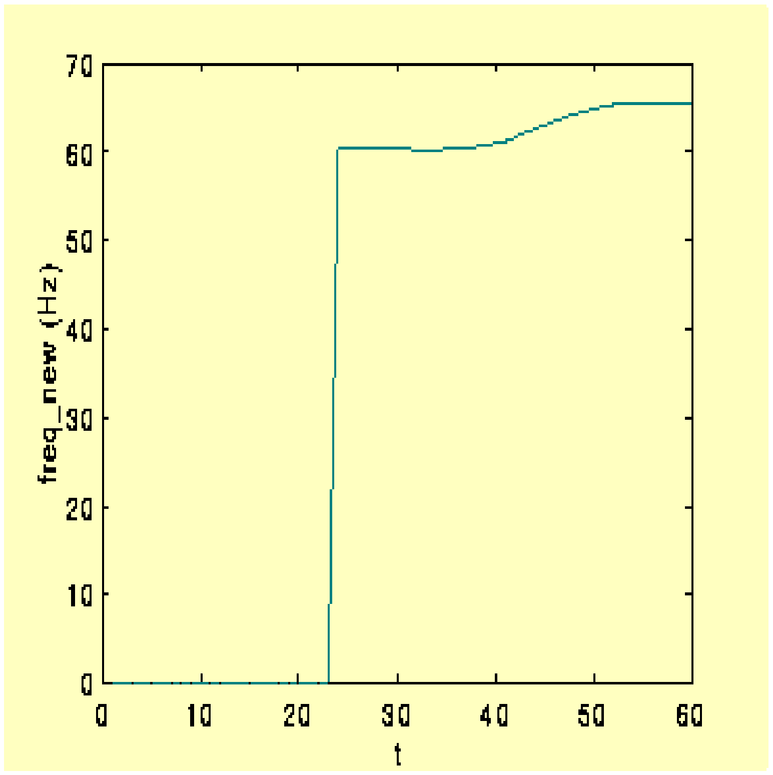

Figure 4.36: FREQUENCY ESTIMATOR

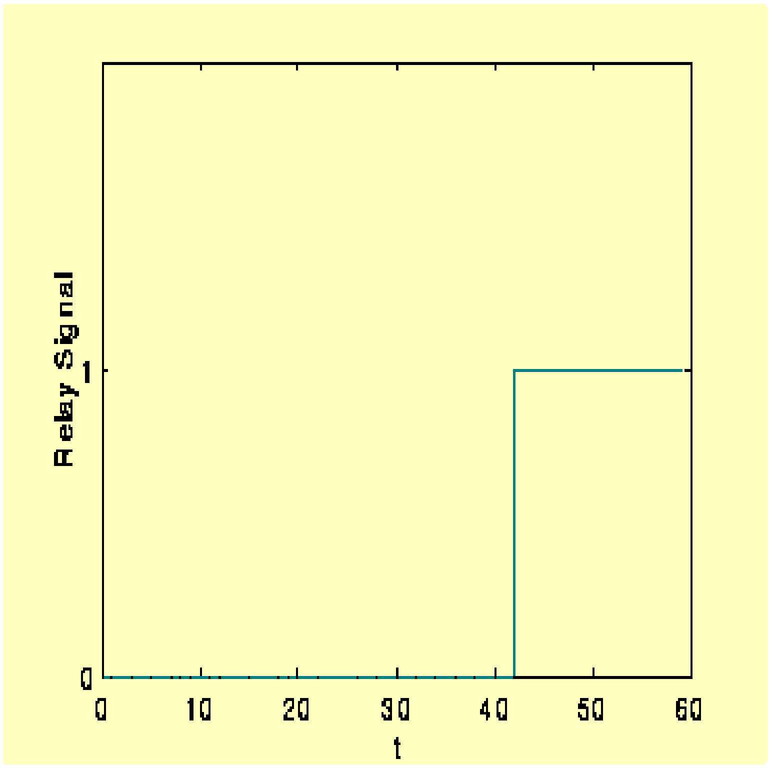

Figure 4.37: MIXED FREQUENCY RELAY OUTPUT CASE

$$\text{V\_peak} = 1.5 \text{ p.u.} \tag{4.31}$$

$$\text{V\_high} = 1.1 \text{ p.u.} \tag{4.32}$$

$$\text{V\_low} = 0.9 \text{ p.u.} \tag{4.33}$$

The six variable parameters observed were: Vapu, Vbpu, Vcpu, Vinsta, Vinstb, and Vinstc. The variable parameter Vapu is plotted in Figure 4.38. The graphs for Vbpu, and Vcpu looked very similar in form. The variable parameter Vinsta is plotted in Figure 4.39. The graphs for Vinstb, and Vinstc looked very similar. The voltage relay did not trip at all. The voltage relay output is graphed in Figure 4.40.

### 4.4.7 NEGATIVE-SEQUENCE RELAY OUTPUT

In the simulation run for the negative-sequence relay, the following parameters were fixed:

$$\text{V\_BASE} = 200.0 \text{ V} \tag{4.34}$$

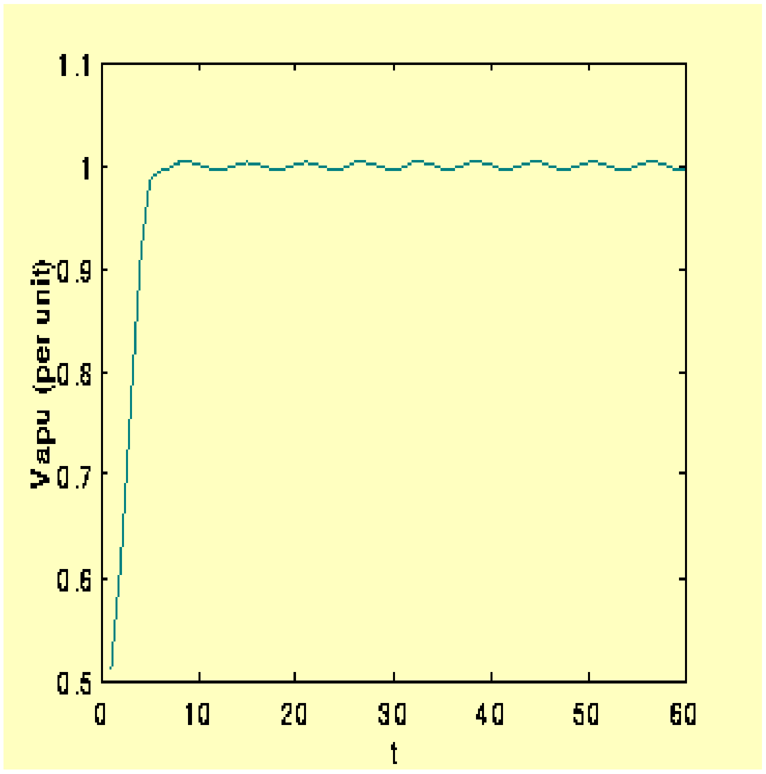

Figure 4.38: PER UNIT RMS VOLTAGE FOR PHASE A

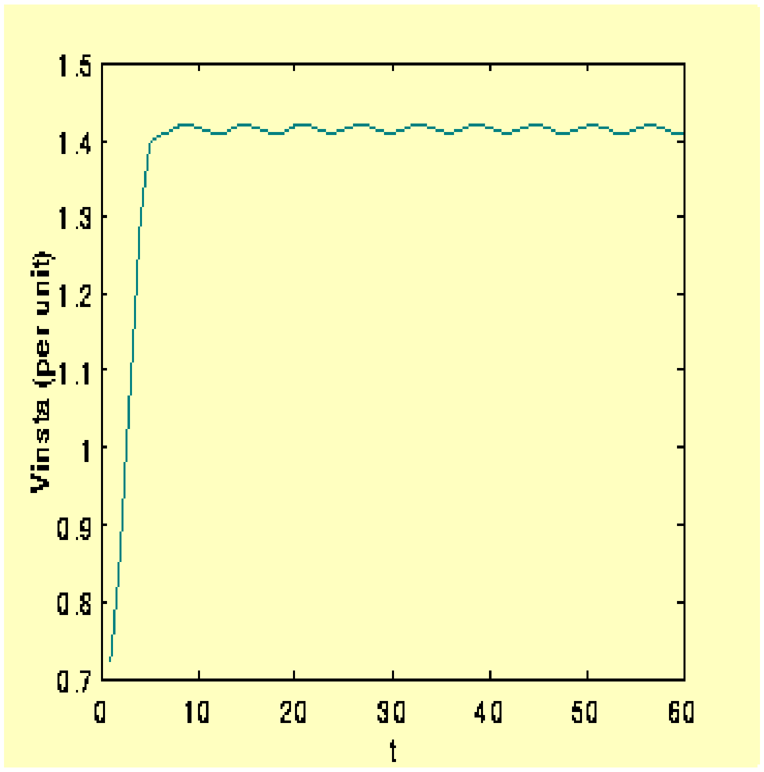

Figure 4.39: PER UNIT PEAK VOLTAGE FOR PHASE A

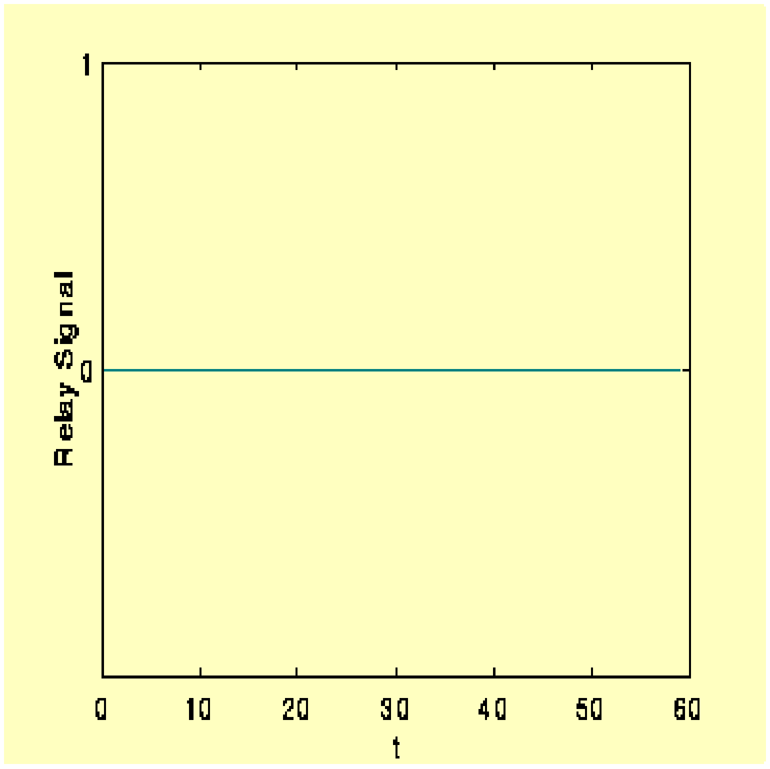

Figure 4.40: VOLTAGE RELAY OUTPUT

$$\text{S\_BASE} = 1 \text{ KW} \tag{4.35}$$

$$\text{I\_peak} = 1.1 \text{ p.u.} \tag{4.36}$$

$$\text{T\_peak} = 4.66 \text{ p.u.} \tag{4.37}$$

The two variable parameters observed were: Ia2_pu, and dial_sum. The variable parameter Ia2_pu is plotted in Figure 4.41. The variable parameter dial_sum is plotted in Figure 4.42. The negative-sequence relay tripped at the fifty-fifth sample point. dial_sum was equal to 4.661115 which is greater than T_peak. The negative-sequence relay output status is graphed in Figure 4.43.

### 4.4.8 CHECK-SYNCHRONISM RELAY OUTPUT

In the simulation run for the check-synchronism relay, the following parameter was fixed:

$$\text{crit\_ang} = 60.0 \tag{4.38}$$

Three variable parameters were observed: Theta_1, Theta_2, and Theta_3. The substation

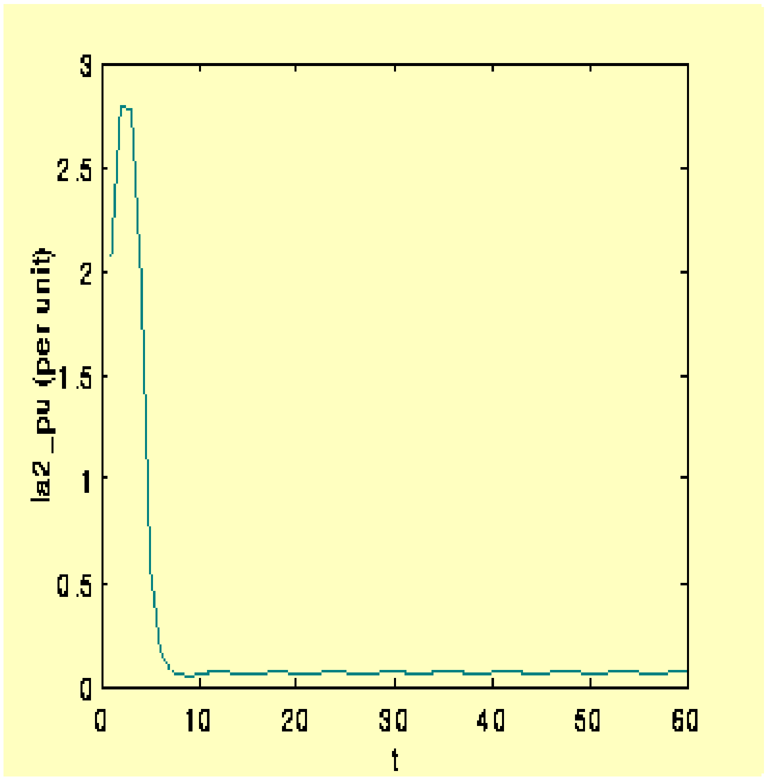

Figure 4.41: PER UNIT NEGATIVE SEQUENCE CURRENT

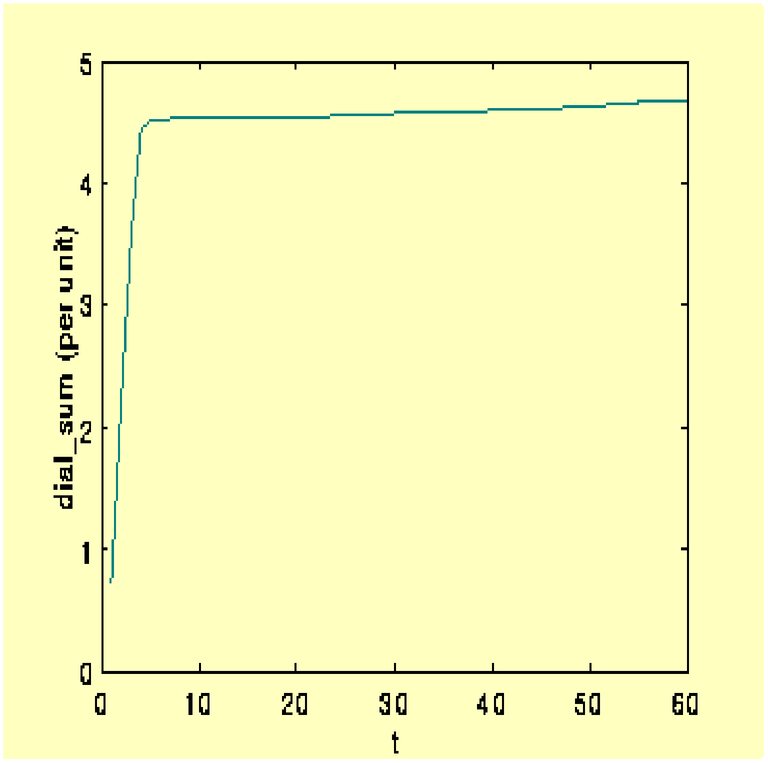

Figure 4.42: INTEGRATED SUM OF THE NEGATIVE SEQUENCE CURRENT

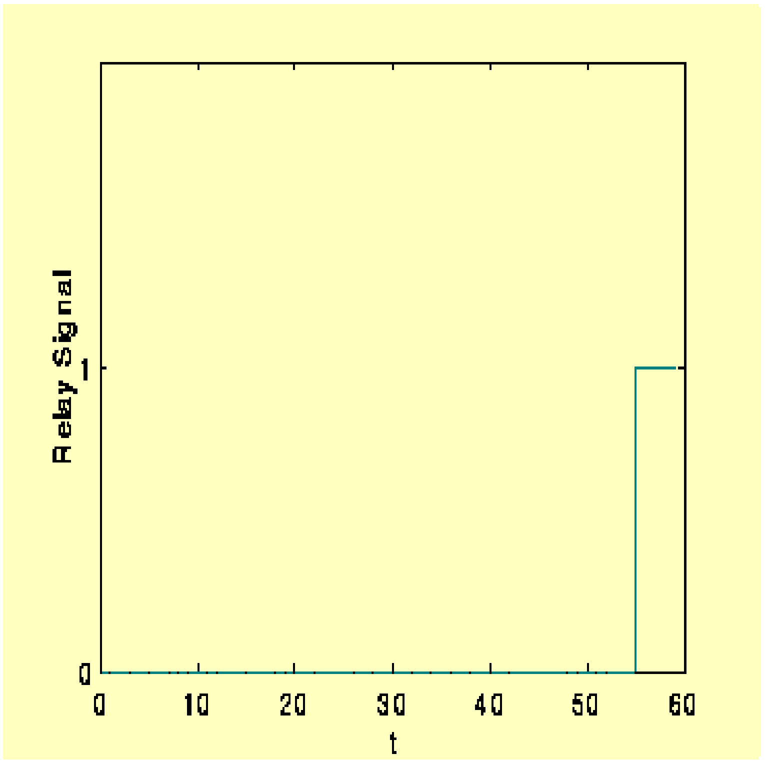

Figure 4.43: NEGATIVE-SEQUENCE RELAY OUTPUT

generator and the dispersed generator were synchronized at the first sample point. Theta_1 was 7 degrees, Theta_2 was 4 degrees, and Theta_3 was 2 degrees at this sample point. All the angles were less than 60 degrees.

### 4.4.9 DIRECTIONAL POWER RELAY OUTPUT

In the simulation run for the directional power relay, the following parameter was fixed:

$$\text{crit\_pow} = 0.0 \text{ W} \tag{4.39}$$

The three variable parameters observed were: P_a, P_b, and P_c. The directional power relay tripped at the twenty-third sample point. Par was equal to -47.035831 Watts, which was less than crit_pow. The directional power relay tripped for the remaining portion of the simulation run. The directional power relay output status is graphed in Figure 4.44.

### 4.4.10 SELF-EXCITATION RELAY OUTPUT

In the operating condition for the self-excitation relay, the following parameters were fixed:

$$\text{frequency} = 1000.0 \text{ V} \tag{4.40}$$

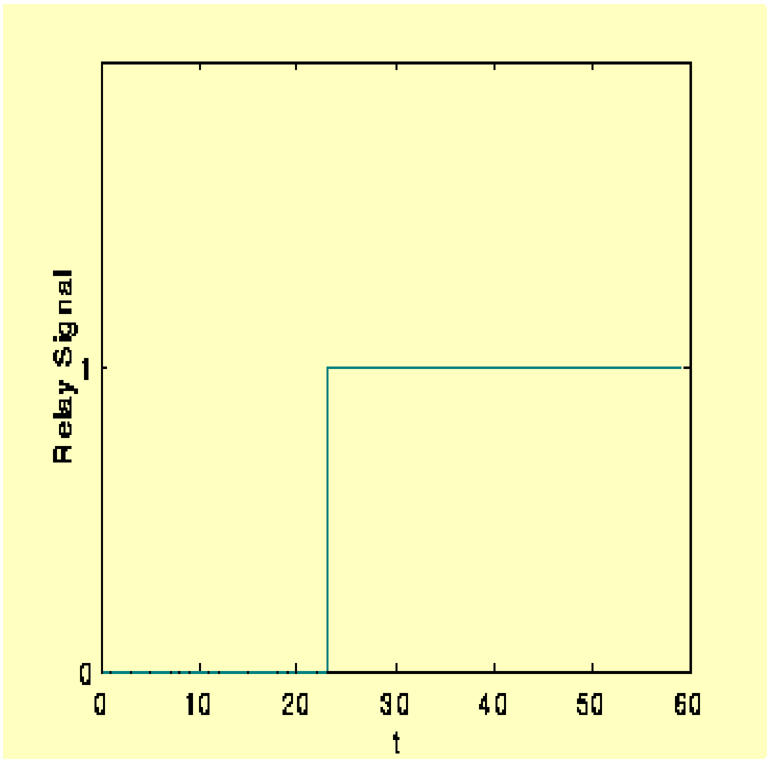

Figure 4.44: DIRECTIONAL POWER RELAY OUTPUT

$$\text{cap\_peak} = 5.0 \text{ p.u.} \tag{4.41}$$

$$\text{V\_BASE} = 345 \text{ kV} \tag{4.42}$$

The three variable parameters observed were: Va_pu, Vb_pu, and Vc_pu. The variable parameter Va_pu is plotted in Figure 4.45. The graphs for Vb_pu, and Vc_pu looked very similar in form. The self-excitation relay tripped at the twenty-third sample point. Va_pu was equal to 94.914200 per unit, which was greater than cap_peak. The relay naturally resetted at the forty-second sample point. The self-excitation relay output status is graphed in Figure 4.46.

## 4.5 SAMPLE SYSTEM OUTPUT FOR MULTIFUNCTION RELAY

Four cases were executed on EMTP for the multifunction relay. In the first case the three-phase load was unbalanced. The load in phase A was fixed at 10 kΩ. The load in phase B was 20 kΩ. [3] The load in phase C was 5 kΩ. [4] In the second case, a three-phase

[3]The load in phase B was increased by 50%.

[4]The load in phase C was decreased by 50%.

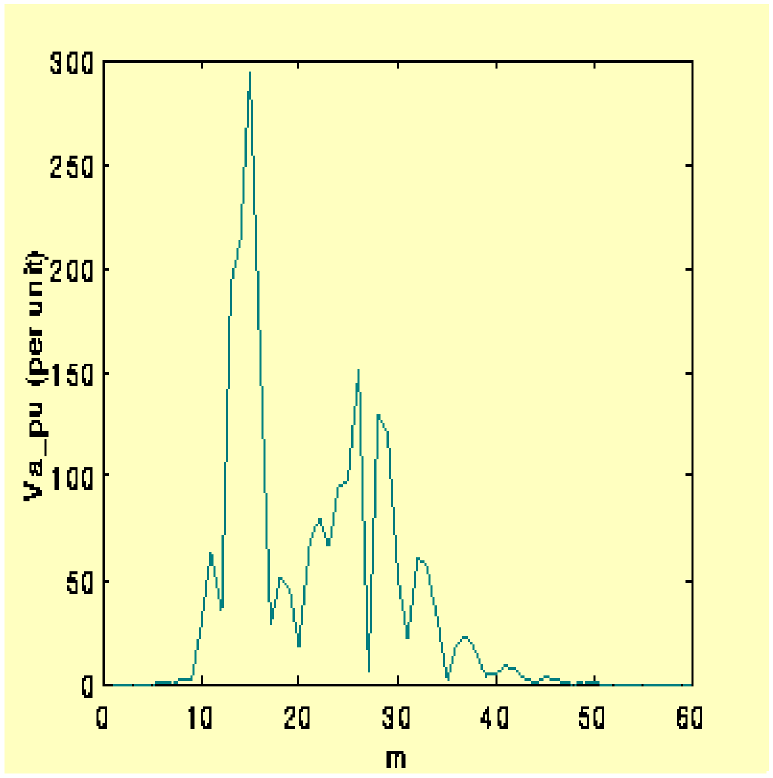

Figure 4.45: PER UNIT PEAK HIGH FREQUENCY VOLTAGE FOR PHASE A

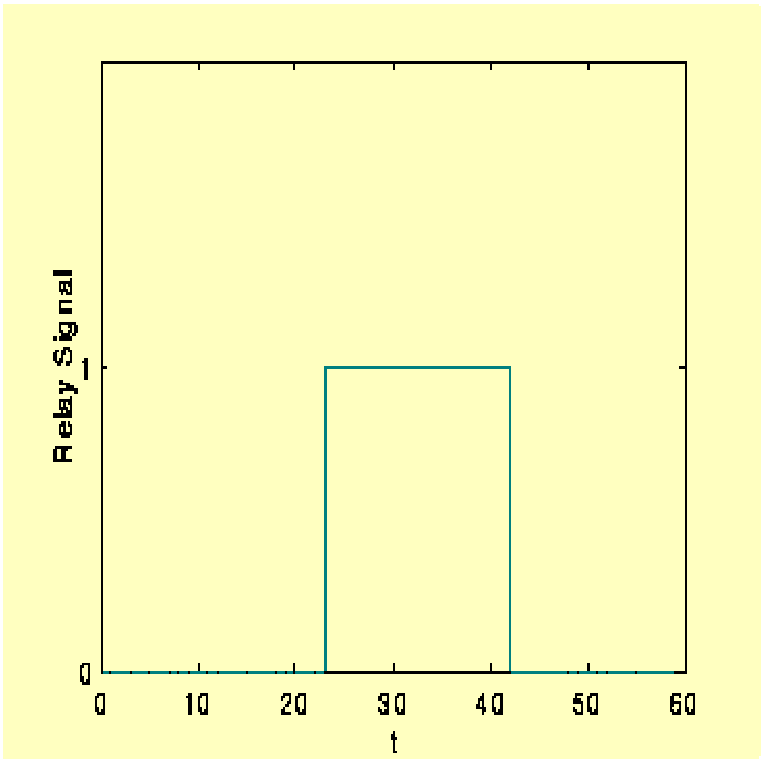

Figure 4.46: SELF-EXCITATION RELAY OUTPUT

fault ocurred at the terminals of the DSG. In the third case, a single-line-to-ground fault ocurred at the terminals of the DSG. In the fourth case, a line-to-line fault ocurred at the terminals of the DSG. For the low frequency relays, the nominal frequency was 60 Hz, and the sampling frequency was 720 Hz. For the high frequency relay, the nominal frequency was 1000 Hz, and the sampling frequency was 3000 Hz. The simulation time was 347 ms. The time step was $20\mu s$. The total number of sample points was 96.

### 4.5.1 UNBALANCED LOAD OUTPUT RESULTS

In the simulation run for the multifunction relay, the following parameters were fixed:

$$\text{i\_L} = 100 \text{ A} \tag{4.43}$$

$$\text{V\_BASE} = 2309.401 \text{ V} \tag{4.44}$$

$$\text{S\_BASE} = 15 \text{ MW} \tag{4.45}$$

$$\text{I\_high} = 1.1 \text{ p.u.} \tag{4.46}$$

$$\text{I\_peak} = 1.5 \text{ p.u.} \tag{4.47}$$

$$\text{TDS} = 0.5 \text{ p.u.} \tag{4.48}$$

$$\text{ks} = 0.9 \text{ p.u.} \tag{4.49}$$

$$\text{imped} = 0.74 \text{ p.u.} \tag{4.50}$$

$$\text{S} = 0.2 \text{ p.u.} \tag{4.51}$$

$$\text{high\_f} = 62.0 \text{ Hz} \tag{4.52}$$

$$\text{low\_f} = 58.0 \text{ Hz} \tag{4.53}$$

$$\text{freq\_nom} = 60.0 \text{ Hz} \tag{4.54}$$

$$\text{V\_peak} = 1.5 \text{ p.u.} \tag{4.55}$$

$$\text{V\_high} = 1.1 \text{ p.u.} \tag{4.56}$$

$$\text{V\_low} = 0.9 \text{ p.u.} \tag{4.57}$$

$$\text{I\_peak} = 1.1 \text{ p.u.} \tag{4.58}$$

$$\text{T\_peak} = 35.0 \text{ p.u.} \tag{4.59}$$

$$\text{crit\_ang} = 60.0 \text{ deg} \tag{4.60}$$

$$\text{crit\_pow} = 0.0 \text{ W} \tag{4.61}$$

$$\text{cap\_peak} = 1.7 \text{ p.u.} \tag{4.62}$$

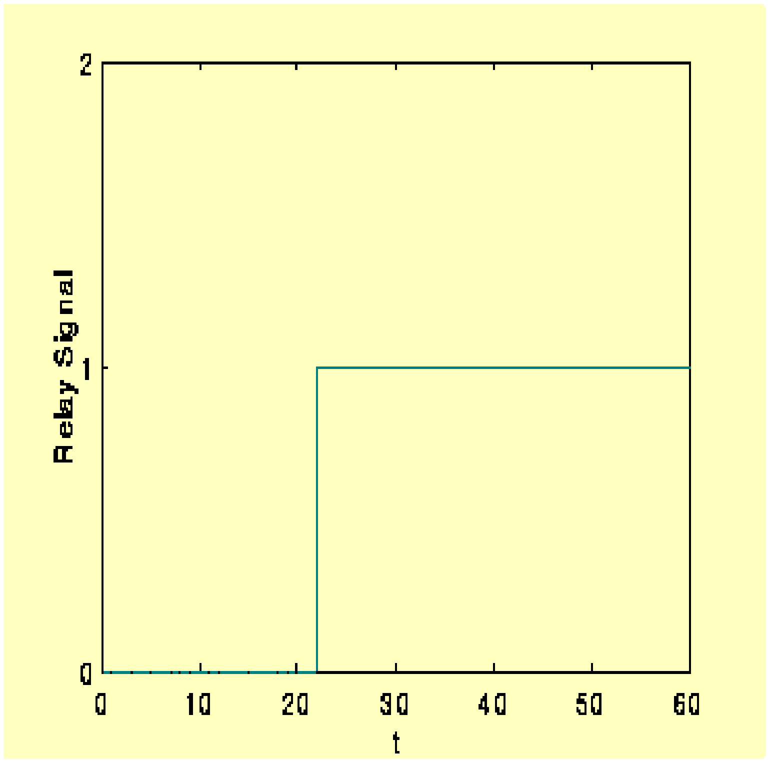

Figure 4.47: MULTIFUNCTION RELAY OUTPUT FOR UNBALANCED LOAD

The multifunction relay tripped at the twenty-third sample point. The frequency relay criteria was violated. freq_new was equal to 65.382904 Hz, which was greater than high_f. The relay continued to trip for the rest of the simulation, because freq_new remained above 62.0 Hz. The multifunction relay output status is graphed in Figure 4.47.

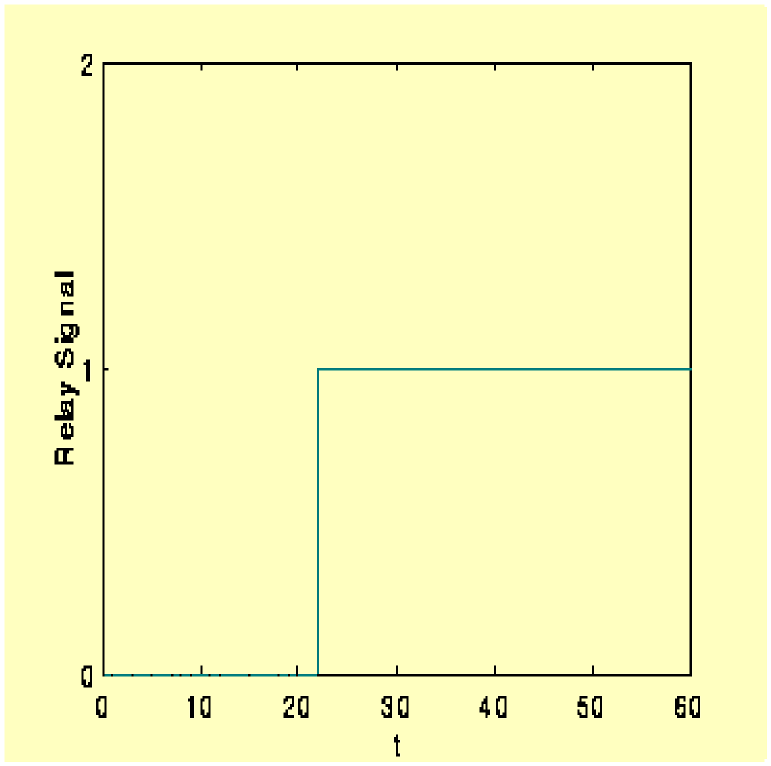

Figure 4.48: MULTIFUNCTION RELAY OUTPUT FOR THREE-PHASE FAULT

### 4.5.2 THREE-PHASE FAULT OUTPUT RESULTS

In the simulation run for the three-phase fault, the variable ks was changed to 0.1 per unit. The rest of the parameters were fixed at the same values used in the first case. The multifunction relay tripped at the twenty-third sample point. The islanding relay criteria was violated. isl was equal to 1.945400 per unit, which was greater than ks. The relay continued to trip for islanding protection for the rest of the simulation. The multifunction relay output status for the three-phase fault is graphed in Figure 4.48.

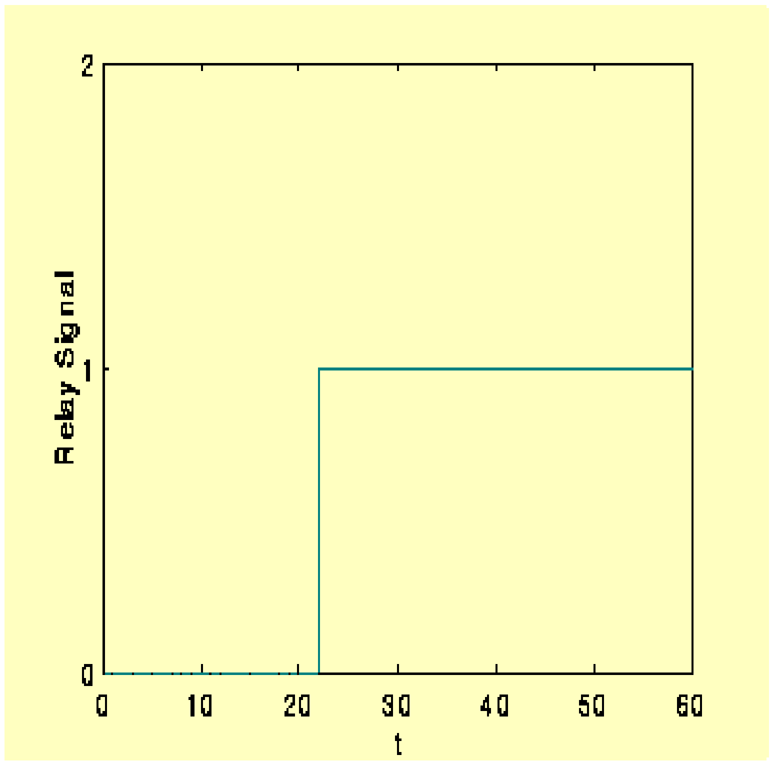

Figure 4.49: MULTIFUNCTION RELAY OUTPUT FOR SLG FAULT

### 4.5.3 LINE-TO-GROUND FAULT OUTPUT RESULTS

In the simulation run for the single line-to-ground (SLG) fault case, all the fixed parameters used in the unbalanced load case remained the same in this case. The multifunction relay tripped at the twenty-third sample point, because the loss-of-excitation relay criteria was violated. Zb was equal to 0.01659 Ω, which was less than Zapp. Zapp was equal to 0.227861 Ω for this particular operating condition. The relay continued to trip for the complete eight cycles. The multifunction relay output status for the SLG fault is graphed in Figure 4.49.

### 4.5.4 LINE-TO-LINE FAULT OUTPUT RESULTS

In the simulation run for the line-to-line fault case, all the fixed parameters remained the same as for the single line-to-ground fault case and the unbalanced load case.

The multifunction relay tripped at the twenty-third sample point, because its differential relay criteria was violated. S_2 was equal to 0.805229 per unit, which was greater than the trip setting variable S. The relay continued to trip for the rest of the simulation run, because its differential relay criteria was violated. The multifunction relay output status for the line-to-line fault case is graphed in Figure 4.50.

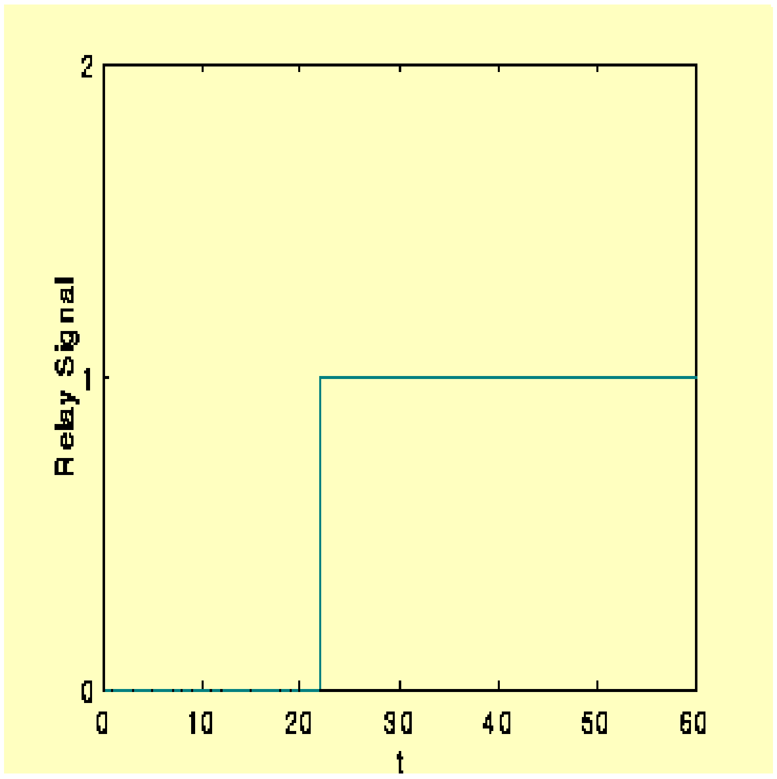

Figure 4.50: MULTIFUNCTION RELAY OUTPUT FOR LINE-TO-LINE FAULT

# CONCLUSION

A new multifunction digital relay will be developed for the DSG/utility interface. This relay will be able to provide relay requirements for the DSG/utility interface such as overcurrent, overvoltage, undervoltage, negative-sequence, loss-of-excitation, check-synchronism, differential, islanding, overfrequency, underfrequency, and self-excitation. This new relay will have metering capabilities. It will measure current, voltage, real power, and reactive power.

## ADVANTAGES OF THE SELF-EXCITATION METHOD

The Self-excitation method has several advantages over the other control methods. Under this method, the relay can reduce the very high rate-of-change of voltage associated with the energization of a capacitor-bank. It does not suffer from thermal-capability limitations compared to the pre-insertion resistor method. It does not cause damaging internal resonances in transformers. It does not suffer from system damping problems compared to the pre-insertion inductor method. The self-excitation method has a higher reliability than any of the other methods.

## RECOMMENDATIONS FOR FUTURE WORK

In order for digital relays to be the preferred protective system of the future, there are some very real problems that must be resolved. Among them, the effect of mixing of digital and analog relays within a common overall protection system and a lack of standardization of computer relay interfaces, so that different manufacturer's equipment can be integrated in one protection system. [2] Substantial periods of experience with actual commercial field installations is needed in order to know how a harsh environment (temperature,humidity,pollution,EMI) could effect relays located in an electric utility substation. Work is still needed in order for the higher level languages (Fortran,Pascal,C etc.) to replace much of the assembly language programming in computer relaying. The problem with assembly language is that it is not transportable between computers of different types. Some transportability between different computer models of the same family may exist, but even here it is generally desirable to develop new software in order to take advantage of differing capabilities among the different models.[2] The higher level languages are easy to transport between computers. The higher level languages tend to be insufficient for relaying applications, because of slow computer instruction time. Since computer instruction time is becoming faster, there is a higher probability that the higher level languages will completely replace the assembly language programming in computer relaying.

# Appendix A

# SPFTTR ALGORITHM

```
/* SPFFTR     11/12/85 */

/* FFT ROUTINE FOR REAL TIME SERIES (X) WITH N=2**K SAMPLES. */

/* COMPUTATION IS IN PLACE, OUTPUT REPLACES INPUT. */

/* INPUT:  REAL VECTOR X(0:N+1) WITH REAL DATA SEQUENCE IN FIRST N */

/*         ELEMENTS; ANYTHING IN LAST 2.  NOTE: X MAY BE DECLARED */

/*         REAL IN MAIN PROGRAM PROVIDED THIS ROUTINE IS COMPILED  */

/*         SEPARATELY ... COMPLEX OUTPUT REPLACES REAL INPUT HERE. */

/* OUTPUT: COMPLEX VECTOR XX(0:N/2), SUCH THAT X(0)=REAL(XX(0)),X(1)= */

/*         IMAG(XX(0)), X(2)=REAL(XX(1)), ..., X(N+1)=IMAG(XX(N/2). */
/* IMPORTANT:  N MUST BE AT LEAST 4 AND MUST BE A POWER OF 2. */



#ifndef KR

void r_cnjg(complex *r, complex *z)

#else

void r_cnjg(r, z)

complex *r, *z;

#endif

{

    r->r = z->r;

    r->i = -z->i;

}



#ifndef KR

void spfftr(complex *x, long *n)

#else

void spfftr(x, n)

long *n;
```

```
complex *x;

#endif

{

    /* Builtin functions */

//    void r_cnjg();

    //double temp_re,temp_im;

    /* Local variables */

   // void spfftc();



    long m, tmp_int;

    complex u, tmp, tmp_complex;

float tpn, tmp_float;

long neg_i1 = -1;



    tpn = (float) (2.0 * M_PI / (double) *n);



    tmp_int = *n / 2;

    spfftc(x, &tmp_int, &neg_i1);



    x[*n / 2].r = x[0].r;
    x[*n / 2].i = x[0].i;


    for (m = 0 ; m <= (*n / 4) ; ++m)

    {

u.r = (float) sin((double) m * tpn);

u.i = (float) cos((double) m * tpn);


r_cnjg(&tmp_complex, &x[*n / 2 - m]);

//   temp_re = real(&tmp_complex);

    //   temp_im = imag(&tmp_complex);


tmp.r = (((1.0 + u.r) * x[m].r - u.i * x[m].i)

+ (1.0 - u.r) * tmp_complex.r - -u.i * tmp_complex.i) / 2.0;


tmp.i = (((1.0 + u.r) * x[m].i + u.i * x[m].r)

+ (1.0 - u.r) * tmp_complex.i + -u.i * tmp_complex.r) / 2.0;


tmp_float = ((1.0 - u.r) * x[m].r - -u.i * x[m].i

    + (1.0 + u.r) * tmp_complex.r - u.i * tmp_complex.i) / 2.0;

x[m].i = ((1.0 - u.r) * x[m].i + -u.i * x[m].r
 + (1.0 + u.r) * tmp_complex.i + u.i * tmp_complex.r) / 2.0;

x[m].r = tmp_float;


r_cnjg(&x[*n / 2 - m], &tmp);

    }


    return;

} /* spfftr */
```

# Appendix B

# SPFTTC ALGORITHM

```
/* SPFFTC     02/20/87 */

/* FAST FOURIER TRANSFORM OF N=2**K COMPLEX DATA POINTS USING TIME */

/* DECOMPOSITION WITH INPUT BIT REVERSAL.  N MUST BE A POWER OF 2. */

/* X MUST BE SPECIFIED COMPLEX X(0:N-1)OR LARGER. */

/* INPUT IS N COMPLEX SAMPLES, X(0),X(1),...,X(N-1). */

/* COMPUTATION IS IN PLACE, OUTPUT REPLACES INPUT. */

/* ISIGN = -1 FOR FORWARD TRANSFORM, +1 FOR INVERSE. */

/* X(0) BECOMES THE ZERO TRANSFORM COMPONENT, X(1) THE FIRST, */

/* AND SO FORTH.  X(N-1) BECOMES THE LAST COMPONENT. */
```

```
#ifndef KR

void spfftc(complex *x, long *n, long *isign)

#else

void spfftc(x, n, isign)

long *n, *isign;

complex *x;

#endif

{

    /* Builtin functions */

    // void complex_exp();


    /* Local variables */

    long i, l, m, mr,tmp_int;

    complex t, tmp_complex, tmp;

    float pisign;


    pisign = (float) ((double) *isign * M_PI);


    mr = 0;
    for (m = 1 ; m < *n ; ++m)

    {

l = *n;

l /= 2;


while (mr + l >= *n)

{

    l /= 2;

}


mr = mr % l + l;


if (mr > m)

{

    t.r = x[m].r;

    t.i = x[m].i;

    x[m].r = x[mr].r;

    x[m].i = x[mr].i;

    x[mr].r = t.r;
    x[mr].i = t.i;
}
    }


    l = 1;


    while (l < *n)
    {
for (m = 0 ; m < l ; ++m)
{


    tmp_int = l * 2;


    for (i = m ; tmp_int < 0 ? i >= (*n - 1) : i < *n ;
 i += tmp_int)
    {
tmp.r = 0.0;
tmp.i = (float) m * pisign / (float) l;


complex_exp(&tmp_complex, &tmp);
t.r = x[i + l].r * tmp_complex.r - x[i + l].i * tmp_complex.i;

t.i = x[i + l].r * tmp_complex.i + x[i + l].i * tmp_complex.r;


x[i + l].r = x[i].r - t.r;

x[i + l].i = x[i].i - t.i;


x[i].r = x[i].r + t.r;

x[i].i = x[i].i + t.i;

    }

        }

        l *= 2;

    }


    return;

} /* spfftc */
```

# Appendix C

# SPFIRL ALGORITHM

The SPFIRL algorithm computed the digital filter coefficients for the FIR lowpass filter using windowed Fourier Series.

```
#include <math.h>

/* SPFIRL     10/15/90 */

/* FIR LOWPASS FILTER DESIGN USING WINDOWED FOURIER SERIES */

/* L=FILTER SIZE SUCH THAT F(Z) = B(0) + B(1)*Z**(-1) +...+ B(L)*Z**(-L) */

/* FCN=NORMALIZED CUT-OFF FREQUENCY IN HERTZ-SECONDS */

/* IWNDO=WINDOW USED TO TRUNCATE FOURIER SERIES */

/*         1-RECTANGULAR; 2-TAPERED RECTANGULAR; 3-TRIANGULAR */
/*         4-HANNING; 5-HAMMING; 6-BLACKMAN */

/* B(0:L)=DIGITAL FILTER COEFFICIENTS RETURNED */

/* IERROR=0      NO ERRORS DETECTED */

/*        1      INVALID FILTER LENGTH (L<=0) */

/*        2      INVALID WINDOW TYPE IWNDO */

/*        3      INVALID CUT-OFF FCN; <=0 OR >=0.5 */

#define M_PI    3.14159265358979323846

#ifndef KR

void spfirl(long *l, float *fcn, long *wndo, float *b, long *error)

#else

void spfirl(l, fcn, wndo, b, error)

long *l, *wndo, *error;

float *fcn, *b;

#endif

{

    /* Local variables */

//    double spwndo(long*, long*, long*);


    long i, lim, tmp_int;

    float wcn, dly;
    if (*l <= 0)

    {

*error = 1;

return;

    }

    if (*wndo < 1 || *wndo > 6)

    {

*error = 2;

return;

    }

    if (*fcn <= 0.0 || *fcn >= 0.5)

    {

*error = 3;

return;

    }


    for (i = 0 ; i <= *l ; ++i)

    {

b[i] = 0.0;
```

```
    }


    wcn = (float) (2.0 * M_PI * *fcn);

    dly = (float) *l / 2.0;

    lim = *l / 2;


    if (dly == (float) (*l / 2))

    {

--lim;

b[*l / 2] = (float) (wcn / M_PI);

    }


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

    {

tmp_int = 1+ *l;

b[i] = (float) (sin((double) (wcn *  ((float) i - dly)))

/ (M_PI * ((float) i - dly))

* spwndo(wndo, &tmp_int, &i));

b[*l - i] = b[i];

    }
    *error = 0;

    return;

} /* spfirl */
```

# Appendix D

# MULTIFUNCTIONAL RELAY PROGRAM

## D.1 THE MULTIFUNCTIONAL RELAY MAIN PROGRAM

The multifunctional relay program executed the modules for each of its relay functions and returned a output signal of block or trip. It also computed metering output at each sample point.

```
#include <stdio.h>
```

```
#include <math.h>

#include <complex.h>

#include <iostream.h>

#define    PI    3.14159

#define    M     96

#define    B     84

#define    N     12




void main()

{

int             i,k;

float          Iar_new,Iai_new,Ibr_new,Ibi_new,Icr_new,Ici_new;

float          Var_new,Vai_new,Vbr_new,Vbi_new,Vcr_new,Vci_new;

float          Zar_new,Zai_new,Zbr_new,Zbi_new,Zcr_new,Zci_new;

float          Par_new,Pai_new,Pbr_new,Pbi_new,Pcr_new,Pci_new;

float          Pgr_new,Pgi_new;

float          Vaf,Vbf,Vcf,Va1_ang,Va1_del,decision;

float          psv_ang[96],psv_del[96];

float          Va_freq[96],Vb_freq[96],Vc_freq[96],rel_sig;
float         V_BASE,S_BASE,I_BASE,Z_BASE;

complex   va_m[96],vb_m[96],vc_m[96];

complex   za_m[96],zb_m[96],zc_m[96];

complex   pa_m[96],pb_m[96],pc_m[96];

complex   ia_m[96],ib_m[96],ic_m[96],pg_m[96];

double     x_va,y_va,x_vb,y_vb,x_vc,y_vc;

double     za_x,za_y,zb_x,zb_y,zc_x,zc_y;

double     pa_x,pa_y,pb_x,pb_y,pc_x,pc_y;

double     x_ia,y_ia,x_ib,y_ib,x_ic,y_ic,pg_x,pg_y;

char        choice,answer;

FILE      *fp,*fp1,*fp2,*fp3,*fp4,*fp5,*fp6,*fp7;

FILE      *fp8,*fp9,*fp10,*fp11,*fp12,*fp13,*fp14;

FILE      *fp15,*fp16,*fp17,*rel_file,*file;




if ((fp = fopen("va.m","r")) == NULL)

printf("Data file va.m not found\n");
```

```
for (i=0;i< M;i++)

{

fscanf(fp,"%le,%le\n",&x_va,&y_va);

va_m[i]=complex(x_va,y_va);

}

fclose(fp);




if ((fp1 = fopen("vb.m","r")) == NULL)

printf("Data file vb.m not found\n");




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

{

fscanf(fp1,"%le,%le\n",&x_vb,&y_vb);

vb_m[i]=complex(x_vb,y_vb);

}

fclose(fp1);
```

```
if ((fp2 = fopen("vc.m","r")) == NULL)

printf("Data file vc.m not found\n");



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

{

fscanf(fp2,"%le,%le\n",&x_vc,&y_vc);

vc_m[i]=complex(x_vc,y_vc);

}

fclose(fp2);



if ((fp3 = fopen("ia.m","r")) == NULL)

printf("Data file ia.m not found\n");



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

{

fscanf(fp3,"%le,%le\n",&x_ia,&y_ia);

ia_m[i]=complex(x_ia,y_ia);
}

fclose(fp3);




if ((fp4 = fopen("ib.m","r")) == NULL)

printf("Data file ib.m not found\n");




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

{

fscanf(fp4,"%le,%le\n",&x_ib,&y_ib);

ib_m[i]=complex(x_ib,y_ib);

}

fclose(fp4);




if ((fp5 = fopen("ic.m","r")) == NULL)

printf("Data file ic.m not found\n");
```

```
for (i=0;i< M;i++)

{

fscanf(fp5,"%le,%le\n",&x_ic,&y_ic);

ic_m[i]=complex(x_ic,y_ic);

}

fclose(fp5);




if ((fp6 = fopen("va1_ang.m","r")) == NULL)

printf("Data file va1_ang.m not found\n");




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

{

fscanf(fp6,"%f\n",&psv_ang[i]);

}

fclose(fp6);




 if ((fp7 = fopen("va1_del.m","r")) == NULL)
```

```
printf("Data file va1_del.m not found\n");



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

{

fscanf(fp7,"%f\n",&psv_del[i]);

}

fclose(fp7);



if ((fp8 = fopen("Va_mag.m","r")) == NULL)

printf("Data file Va_mag.m not found\n");



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

{

fscanf(fp8,"%e\n",&Va_freq[i]);

}

fclose(fp8);
```

```
if ((fp9 = fopen("Vb_mag.m","r")) == NULL)

printf("Data file Vb_mag.m not found\n");



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

{

fscanf(fp9,"%e\n",&Vb_freq[i]);

}

fclose(fp9);



if ((fp10 = fopen("Vc_mag.m","r")) == NULL)

printf("Data file Vc_mag.m not found\n");



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

{

fscanf(fp10,"%e\n",&Vc_freq[i]);

}
```

```
fclose(fp10);



if ((fp11 = fopen("pg.m","r")) == NULL)

printf("Data file pg.m not found\n");



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

{

fscanf(fp11,"%le,%le\n",&pg_x,&pg_y);

pg_m[i]=complex(pg_x,pg_y);

}

fclose(fp11);



if ((fp12 = fopen("za.m","r")) == NULL)

printf("Data file za.m not found\n");



for (i=0;i< M;i++)
```

```
{

fscanf(fp12,"%le,%le\n",&za_x,&za_y);

za_m[i]=complex(za_x,za_y);

}

fclose(fp12);




if ((fp13 = fopen("zb.m","r")) == NULL)

printf("Data file zb.m not found\n");




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

{

fscanf(fp13,"%le,%le\n",&zb_x,&zb_y);

zb_m[i]=complex(zb_x,zb_y);

}

fclose(fp13);




if ((fp14 = fopen("zc.m","r")) == NULL)
```

```
printf("Data file zc.m not found\n");



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

{

fscanf(fp14,"%le,%le\n",&zc_x,&zc_y);

zc_m[i]=complex(zc_x,zc_y);

}

fclose(fp14);



if ((fp15 = fopen("pa.m","r")) == NULL)

printf("Data file pa.m not found\n");



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

{

fscanf(fp15,"%le,%le\n",&pa_x,&pa_y);

pa_m[i]=complex(pa_x,pa_y);

}
```

```
fclose(fp15);



if ((fp16 = fopen("pb.m","r")) == NULL)

printf("Data file pb.m not found\n");



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

{

fscanf(fp16,"%le,%le\n",&pb_x,&pb_y);

pb_m[i]=complex(pb_x,pb_y);

}

fclose(fp16);



if ((fp17 = fopen("pc.m","r")) == NULL)

printf("Data file pc.m not found\n");



for (i=0;i< M;i++)
```

```
{

fscanf(fp17,"%le,%le\n",&pc_x,&pc_y);

pc_m[i]=complex(pc_x,pc_y);

}

fclose(fp17);




for ( k = N; k < M; k++)

{

Var_new = real(va_m[k]);

Vai_new = imag(va_m[k]);

Vbr_new = real(vb_m[k]);

Vbi_new = imag(vb_m[k]);

Vcr_new = real(vc_m[k]);

Vci_new = imag(vc_m[k]);

Iar_new = real(ia_m[k]);

Iai_new = imag(ia_m[k]);

Ibr_new = real(ib_m[k]);

Ibi_new = imag(ib_m[k]);

Icr_new = real(ic_m[k]);
```

```
Ici_new = imag(ic_m[k]);

Par_new = real(pa_m[k]);

Pai_new = imag(pa_m[k]);

Pbr_new = real(pb_m[k]);

Pbi_new = imag(pb_m[k]);

Pcr_new = real(pc_m[k]);

Pci_new = imag(pc_m[k]);

Pgr_new = real(pg_m[k]);

Pgi_new = imag(pg_m[k]);

Zar_new = real(za_m[k]);

Zai_new = imag(za_m[k]);

Zbr_new = real(zb_m[k]);

Zbi_new = imag(zb_m[k]);

Zcr_new = real(zc_m[k]);

Zci_new = imag(zc_m[k]);

Va1_ang = psv_ang[k];

Va1_del = psv_del[k-N];

Vaf = Va_freq[k];

Vbf = Vb_freq[k];

Vcf = Vc_freq[k];
```

```
V_BASE = 2309.401;

S_BASE = 15.0E+6;


I_BASE = (S_BASE/((sqrt(3))*(V_BASE)));

Z_BASE = V_BASE/I_BASE;



char relay1;

char inverse(float, float, float, float, float, float, float, int);


relay1  = inverse(Iar_new,Iai_new,Ibr_new,Ibi_new,  Icr_new,Ici_new,I_BASE,k);



char relay2;

char island(float, float, float, int);


relay2 = island(Pgr_new,Pgi_new,S_BASE,k);
```

```
char relay3;

char loss(float, float, float, float, float, float, float, int);


relay3 = loss(Zar_new,Zai_new,Zbr_new,Zbi_new, Zcr_new,Zci_new,Z_BASE,k);




char relay4;

char diff(float, float, float, float, float,float, float, int);


relay4  = diff(Iar_new,Iai_new,Ibr_new,Ibi_new,Icr_new,Ici_new,I_BASE,k);




char relay5;

char freq(float, float, int);


relay5  = freq(Va1_ang, Va1_del, k);
```

```
char relay6;

char voltage(float, float, float, float,float, float, float, int);


relay6  = voltage(Var_new,Vai_new,Vbr_new,Vbi_new,Vcr_new,Vci_new,V_BASE,k);




char relay7;

char negative(float, float, float, float, float, float,float, int);


relay7  = negative(Iar_new,Iai_new,Ibr_new,Ibi_new,Icr_new,Ici_new,I_BASE, k);




char relay8;

char check(float, float, float, float, float, float,int);


relay8  = check(Var_new,Vai_new,Vbr_new,Vbi_new,Vcr_new,Vci_new,k);




char relay9;

char power(float, float, float, float, float, float, int);
```

```
relay9 = power(Par_new,Pai_new,Pbr_new,Pbi_new,Pcr_new,Pci_new,k);



char relay10;

char cap(float,float,float,float,int);


relay10 = cap(Vaf,Vbf,Vcf,V_BASE,k);




if (relay1== 't')

{

printf("The relay output signal is trip.\n");

printf("The undercurrent\n");

printf("relay criteria was violated.\n");

answer = 't';

}


else if (relay1== 'u')
```

```
{

printf("The relay output signal is trip.\n");

printf("The overcurrent\n");

printf("relay criteria was violated.\n");

answer = ’t’;

}


else if (relay1== ’z’)

{

printf("The relay output signal is trip.\n");

printf("The instantaneous overcurrent\n");

printf("relay criteria was violated.\n");

answer = ’z’;

}


else if (relay2 == ’t’)

{

printf("The relay output signal is trip.\n");

printf("The islanding\n");

printf("relay criteria was violated.\n");
answer = 't';

}


else if (relay3 == 't')

{

printf("The relay output signal is trip.\n");

printf("The loss-of-excitation\n");

printf("relay criteria was violated.\n");

answer = 't';

}


else if (relay4== 't')

{

printf("The relay output signal is trip.\n");

printf("The differential\n");

printf("relay criteria was violated.\n");

answer = 't';

}


else if (relay5== 't')
```

```
{

printf("The relay output signal is trip.\n");

printf("The overfrequency \n");

printf("relay criteria was violated.\n");

answer = 't';

}


else if (relay5== 'z')

{

printf("The relay output signal is trip.\n");

printf("The underfrequency \n");

printf("relay criteria was violated.\n");

answer = 'z';

}


else if (relay6== 't')

{

printf("The relay output signal is trip.\n");

printf("The undervoltage\n");

printf("relay criteria was violated.\n");
answer = 't';

}


else if (relay6== 'u')

{

printf("The relay output signal is trip.\n");

printf("The overvoltage\n");

printf("relay criteria was violated.\n");

answer = 't';

}


else if (relay7== 't')

{

printf("The relay output signal is trip.\n");

printf("The negative sequence\n");

printf("relay criteria was violated.\n");

answer = 't';

}


else if (relay8== 'b')
```

```
{

printf("The substation generator and the\n");

printf("dispersed generator are\n");

printf("synchronized at this sample point.\n");

answer = 't';

}


else if (relay9== 't')

{

printf("The relay output signal is trip.\n");

printf("The directional power\n");

printf("relay criteria was violated.\n");

answer = 't';

}


else if (relay10== 't')

{

printf("The relay output signal is trip.\n");

printf("The self-excitation\n");

printf("relay criteria was violated.\n");
answer = 't';

}


else

{

printf("The relay output signal is block.\n");

printf("None of the relay criteria was violated.\n");

}




char relay11;


char meter(float, float, float, float, float, float,

float, float, float, float, float, float,

float, float, float, float, float, float,int);


relay11 = meter(Iar_new,Iai_new,Ibr_new,Ibi_new,

Icr_new,Ici_new,Var_new,Vai_new, Vbr_new,

Vbi_new,Vcr_new,Vci_new, Par_new,Pai_new,

Pbr_new,Pbi_new,Pcr_new,Pci_new,k);
```

```
if (k < 24)

{

decision = 0.0;

}


else if (answer == 't')

{

decision = 1.0;

}


else if (answer == 'u')

{

decision = 1.0;

}


else if (answer == 'z')

{

decision = 1.0;
}


else

{

decision = 0.0;

}




rel_file  = fopen("fun_rel.m","a");

fprintf(rel_file, "%f\n",decision);

fclose(rel_file);





printf("Do you want the next sample[y/n]?");

scanf("%c",&choice);


cin >> choice;


if ((choice=='n') || (choice=='N'))
```

```
{
printf("C++ stop.");
break;
}


}


}


char inverse(float Iar, float Iai, float Ibr, float Ibi,float Icr, float Ici,
float I_base, int j)


{
int     i;
float   i_L,CT,Ipickp,Ipickg,I_setp,I_setg;
float   Ipeakp,Ipeakg;
float   I_multp,I_multg,Tp,Tg;
float   Tp_new,Tg_new,T_value[2],TDS;
float   C1,C2,C3,C4,C5,C6,C7;
float   Theta_Ia,Theta_Ib,Theta_Ic;
float   Ia_f,Ib_f,Ic_f,I_high,I_peak;

float   I_minpu,I_maxpu;

float   I_min,I_max;

float   Ipickg_t,I_gnd;

char    value,t,b,z,s,u,v,relay_type;

complex a(-0.5,0.866);

complex Ia_max(Iar,Iai),Ib_max(Ibr,Ibi),Ic_max(Icr,Ici);



i_L    = 100.0;

I_high = 1.1;

I_peak = 1.5;




Ia_f     = abs(Ia_max);

Ib_f     = abs(Ib_max);

Ic_f     = abs(Ic_max);
```

```
if ((real(Ia_max) == 0) && (real(Ib_max) == 0) && (real(Ic_max) == 0))

{

Theta_Ia = -(180.0/PI)*(PI/2.0);

Theta_Ib = -(180.0/PI)*(PI/2.0);

Theta_Ic = -(180.0/PI)*(PI/2.0);

}



else if ((real(Ia_max) == 0) || (real(Ib_max) == 0) || (real(Ic_max) == 0))

{

if ((real(Ia_max) == 0) && (real(Ib_max) == 0))

{

Theta_Ia = -(180.0/PI)*(PI/2.0);

Theta_Ib = -(180.0/PI)*(PI/2.0);

Theta_Ic =  (180.0/PI)*atan2(imag(Ic_max), real(Ic_max));

}


else if ((real(Ia_max) == 0) && (real(Ic_max) == 0))

{

Theta_Ia = -(180.0/PI)*(PI/2.0);
```

```
Theta_Ib =  (180.0/PI)*atan2(imag(Ib_max), real(Ib_max));

Theta_Ic = -(180.0/PI)*(PI/2.0);

}


else if ((real(Ib_max) == 0) && (real(Ic_max) == 0))

{

Theta_Ia =  (180.0/PI)*atan2(imag(Ia_max), real(Ia_max));

Theta_Ib = -(180.0/PI)*(PI/2.0);

Theta_Ic = -(180.0/PI)*(PI/2.0);

}


else if (real(Ia_max) == 0)

{

Theta_Ia = -(180.0/PI)*(PI/2.0);

Theta_Ib =  (180.0/PI)*atan2(imag(Ib_max), real(Ib_max));

Theta_Ic =  (180.0/PI)*atan2(imag(Ic_max), real(Ic_max));

}


else if (real(Ib_max) == 0)

{
```

```
Theta_Ia =  (180.0/PI)*atan2(imag(Ia_max), real(Ia_max));

Theta_Ib = -(180.0/PI)*(PI/2.0);

Theta_Ic =  (180.0/PI)*atan2(imag(Ic_max), real(Ic_max));

}


else if (real(Ic_max) == 0)

{

Theta_Ia =  (180.0/PI)*atan2(imag(Ia_max), real(Ia_max));

Theta_Ib =  (180.0/PI)*atan2(imag(Ib_max), real(Ib_max));

Theta_Ic = -(180.0/PI)*(PI/2.0);

}

}


else

{

Theta_Ia = (180.0/PI)*atan2(imag(Ia_max), real(Ia_max));

Theta_Ib = (180.0/PI)*atan2(imag(Ib_max), real(Ib_max));

Theta_Ic = (180.0/PI)*atan2(imag(Ic_max), real(Ic_max));

}
```

```
/*

The current ratio (CT)is based upon load.

A CT ratio will be selected that will give

5.0 amperes secondary current for the maximum

load.

*/




if ( i_L/5.0 <= 20.0)

{

CT = (20.0/1.0);

}


else if (i_L/5.0 <= 40.0)

{

CT = (40.0/1.0);

}


else if (i_L/5.0 <= 60.0)
```

```
{

CT = (60.0/1.0);

}


else if (i_L/5.0 <= 80.0)

{

CT = (80.0/1.0);

}


else if (i_L/5.0 <= 100.0)

{

CT = (100.0/1.0);

}


else if (i_L/5.0 <= 120.0)

{

CT = (120.0/1.0);

}

else if (i_L/5.0 <= 160.0)

{
```

```
CT = (160.0/1.0);

}


else if (i_L/5.0 <= 180.0)

{

CT = (180.0/1.0);

}


else if (i_L/5.0 <= 200.0)

{

CT = (200.0/1.0);

}


else if (i_L/5.0 <= 240.0)

{

CT = (240.0/1.0);

}


else

{
```

```
printf("A compatible standard current");

printf("transformer ratio was not found.");

}




if ((Ia_f > Ib_f) && (Ia_f > Ic_f))

{

I_max = Ia_f;

}


else if ((Ib_f > Ia_f) && (Ib_f > Ic_f))

{

I_max = Ib_f;

}


else

{

I_max = Ic_f;

}
```

```
if ((Ia_f < Ib_f) && (Ia_f < Ic_f))

{

I_min = Ia_f;

}


else if ((Ib_f < Ia_f) && (Ib_f < Ic_f))

{

I_min = Ib_f;

}


else

{

I_min = Ic_f;

}




/*

PHASE RELAY PICKUP SETTING

Ipickp = ((((i_L/CT)*2)+((I_min/3)/CT))/2)

*/
```

```
if (((((i_L/CT)*2.0)+((abs(I_min)/3.0)/CT))/2.0) <= 1.0)

{

Ipickp=1.0;

}


else if(((((i_L/CT)*2.0)+((abs(I_min)/3.0)/CT))/2.0)<=1.2)

{

Ipickp=1.2;

}


else if(((((i_L/CT)*2.0)+((abs(I_min)/3.0)/CT))/2.0)<=1.5)

{

Ipickp=1.5;

}


else if(((((i_L/CT)*2.0)+((abs(I_min)/3.0)/CT))/2.0)<=2.0)

{

Ipickp=2.0;

}
```

```
else if(((((i_L/CT)*2.0)+((abs(I_min)/3.0)/CT))/2.0)<=2.5)

{

Ipickp=2.5;

}


else if(((((i_L/CT)*2.0)+((abs(I_min)/3.0)/CT))/2.0)<=3.0)

{

Ipickp=3.0;

}


else if(((((i_L/CT)*2.0)+((abs(I_min)/3.0)/CT))/2.0)<=3.5)

{

Ipickp=3.5;

}


else if(((((i_L/CT)*2.0)+((abs(I_min)/3.0)/CT))/2.0)<=4.0)

{

Ipickp=4.0;

}
```

```
else if(((((i_L/CT)*2.0)+((abs(I_min)/3.0)/CT))/2.0)<=4.5)

{

Ipickp=4.5;

}


else if(((((i_L/CT)*2.0)+((abs(I_min)/3.0)/CT))/2.0)<=5.0)

{

Ipickp=5.0;

}


else if(((((i_L/CT)*2.0)+((abs(I_min)/3.0)/CT))/2.0)<=6.0)

{

Ipickp=6.0;

}


else if(((((i_L/CT)*2.0)+((abs(I_min)/3.0)/CT))/2.0)<=7.0)

{

Ipickp=7.0;

}
```

```
else if(((((i_L/CT)*2.0)+((abs(I_min)/3.0)/CT))/2.0)<=8.0)

{

Ipickp=8.0;

}


else if(((((i_L/CT)*2.0)+((abs(I_min)/3.0)/CT))/2.0)<=9.0)

{

Ipickp=9.0;

}


else if(((((i_L/CT)*2.0)+((abs(I_min)/3.0)/CT))/2.0)<=10.0)

{

Ipickp=10.0;

}


else if(((((i_L/CT)*2.0)+((abs(I_min)/3.0)/CT))/2.0)<=11.0)

{

Ipickp=11.0;

}
```

```
else if(((((i_L/CT)*2.0)+((abs(I_min)/3.0)/CT))/2.0)<=12.0)

{

Ipickp=12.0;

}


else if(((((i_L/CT)*2.0)+((abs(I_min)/3.0)/CT))/2.0)<=14.0)

{

Ipickp=14.0;

}


else if(((((i_L/CT)*2.0)+((abs(I_min)/3.0)/CT))/2.0)<=16.0)

{

Ipickp=16.0;

}


else if(((((i_L/CT)*2.0)+((abs(I_min)/3.0)/CT))/2.0)<=18.0)

{

Ipickp=18.0;

}
```

```
else if(((((i_L/CT)*2.0)+((abs(I_min)/3.0)/CT))/2.0)<=20.0)

{

Ipickp=20.0;

}


else if(((((i_L/CT)*2.0)+((abs(I_min)/3.0)/CT))/2.0)<=25.0)

{

Ipickp=25.0;

}


else if(((((i_L/CT)*2.0)+((abs(I_min)/3.0)/CT))/2.0)<=30.0)

{

Ipickp=30.0;

}


else if(((((i_L/CT)*2.0)+((abs(I_min)/3.0)/CT))/2.0)<=40.0)

{

Ipickp=40.0;

}
```

```
else

{

printf("The pickup current for the phase\n");

printf("relay is more than 40 amperes.\n");

Ipickp=40.0;

}




I_gnd    = 0.1*i_L;

Ipickg_t = ((((0.1)*(i_L))*2.0)/CT);




/*

GROUND RELAY PICKUP SETTING

Ipickg=((((0.1)*(i_L))*2)/CT);

*/
```

```
if (Ipickg_t<=1.2)

{

Ipickg=1.2;

}


else if (Ipickg_t<=1.5)

{

Ipickg=1.5;

}


else if (Ipickg_t<=2.0)

{

Ipickg=2.0;

}


else if (Ipickg_t<=3.0)

{

Ipickg=3.0;

}
```

```
else if (Ipickg_t<=4.0)

{

Ipickg=4.0;

}


else if (Ipickg_t<=5.0)

{

Ipickg=5.0;

}


else if (Ipickg_t<=6.0)

{

Ipickg=6.0;

}


else if (Ipickg_t<=7.0)

{

Ipickg=7.0;

}
```

```
else if (Ipickg_t<=8.0)

{

Ipickg=8.0;

}


else if (Ipickg_t<=9.0)

{

Ipickg=9.0;

}


else if (Ipickg_t<=10.0)

{

Ipickg=10.0;

}


else if (Ipickg_t<=11.0)

{

Ipickg=11.0;

}
```

```
else if (Ipickg_t<=12.0)

{

Ipickg=12.0;

}


else if (Ipickg_t<=14.0)

{

Ipickg=14.0;

}


else if (Ipickg_t<=16.0)

{

Ipickg=16.0;

}


else if (Ipickg_t<=18.0)

{

Ipickg=18.0;

}
```

```
else if (Ipickg_t<=20.0)

{

Ipickg=20.0;

}


else if (Ipickg_t<=40.0)

{

Ipickg=40.0;

}


else

{

printf("The pickup current for the ground\n");

printf("relay is more than 40 amperes.\n");

Ipickg=40.0;

}




Ipeakp = (1.35*Ipickp);

Ipeakg = (1.35*Ipickg);
```

```
I_minpu = I_min/I_base;

I_maxpu = I_max/I_base;




if ((Theta_Ia < -25.0) && (Theta_Ia > -85.0)

|| (Theta_Ib < -25.0) && (Theta_Ib > -85.0)

|| (Theta_Ic < -25.0) && (Theta_Ic > -85.0))


{

if (I_minpu > I_peak)

{

value='z';

I_setp=(CT*Ipeakp);

I_setg=(CT*Ipeakg);

}


else if (I_minpu > I_high)

{

I_setp=(CT*Ipickp);
I_setg=(CT*Ipickg);

value='u';

}


else

{

I_setp=(CT*Ipickp);

I_setg=(CT*Ipickg);

value='b';

}

}


else

{

I_setp=(CT*Ipickp);

I_setg=(CT*Ipickg);

value='b';

}


C1  =  0.0344;
```

```
C2  =  0.0807;

C3  =  1.9500;

C4  =  0.0577;

C5  = -0.0679;

C6  = -0.7000;

C7  =  0.0199;


TDS = 0.5;


I_multp = (abs(I_max)/I_setp);

I_multg = (abs(I_gnd)/I_setg);



relay_type = 's';



if (relay_type == 's')

{

//  THE TYPE CO-9 RELAY

//  COURTESY OF WESTINGHOUSE
//  7 TERM MODEL


Tp = ((C1) + (C2*(TDS)) + (C3*(TDS/(pow((I_multp-1.0),2))))

+ (C4*(pow(TDS,2)/(I_multp-1)))

+ (C5*(pow(TDS,2)/(pow((I_multp-1.0),2))))

+ (C6*(TDS/(pow((I_multp-1.0),3))))

+ (C7*(pow(TDS,2)/(pow((I_multp-1.0),4)))));


Tg = ((C1) + (C2*(TDS)) + (C3*(TDS/(pow((I_multg-1.0),2))))

+ (C4*(pow(TDS,2)/(I_multg-1)))

+ (C5*(pow(TDS,2)/(pow((I_multg-1.0),2))))

+ (C6*(TDS/(pow((I_multg-1.0),3))))

+ (C7*(pow(TDS,2)/(pow((I_multg-1.0),4)))));

}

return (value);

}


char island   (float Pgr, float Pgi, float S_base, int j)

{

int      m;
```

```
float    Pg[84],Pg_pre,Pg_post,ks,isl;

char     value,t,b,z;

complex  Pg_max(Pgr,Pgi);


ks=0.9;

m=j-N;



Pg[m] = abs(Pg_max);

Pg_post = Pg[m];

Pg_pre  = Pg[0];



isl =(abs(Pg_post - Pg_pre))/S_base;



if (isl>ks)

{

value='t';

}
```

```
else

{

value='b';

}

return (value);

}


char loss(float Zar, float Zai, float Zbr, float Zbi,

float Zcr, float Zci, float Z_base, int j)

{

double   Zapp,X_d;

double   Za,Zb,Zc,imped;

char     value,t,b;

complex  Za_max(Zar,Zai),Zb_max(Zbr,Zbi),Zc_max(Zcr,Zci);


imped  = 0.74;



Za  = abs(Za_max);
```

```
Zb  = abs(Zb_max);

Zc  = abs(Zc_max);

X_d = imped;


 Zapp = (0.5*X_d)*(Z_base);


if (Za < Zapp)

{

value='t';

}


else if (Zb < Zapp)

{

value='t';

}


else if (Zc < Zapp)

{

value='t';

}
```

```
else

{

value='b';

}

return (value);

}




char diff(float Iar, float Iai, float Ibr, float Ibi,float Icr, float Ici,

float I_base, int j)

{

int     m;

float   Ia_post,Ib_post,Ic_post,Ia_pre,Ib_pre,Ic_pre;

float   Ia[84],Ib[84],Ic[84];

float   S_1,S_2,S_3,S;

float   theta;

char    value,t,b;

complex Ia_max(Iar,Iai),Ib_max(Ibr,Ibi),Ic_max(Icr,Ici);
```

```
m=j-N;

S = 0.2;


Ia[m]       = abs(Ia_max);

Ib[m]       = abs(Ib_max);

Ic[m]       = abs(Ic_max);

Ia_post     = Ia[m];

Ib_post     = Ib[m];

Ic_post     = Ic[m];

Ia_pre      = Ia[0];

Ib_pre      = Ib[0];

Ic_pre      = Ic[0];


S_1     = fabs((Ia_post - Ia_pre)/I_base);

S_2     = fabs((Ib_post - Ib_pre)/I_base);

S_3     = fabs((Ic_post - Ic_pre)/I_base);


if (S_1 > S)
```

```
{

value='t';

}


else if (S_2 > S)

{

value='t';

}


else if (S_3 > S)

{

value='t';

}


else

{

value='b';

}

return (value);

}
```

```
char freq(float va1_ang, float va1_del, int j)

{

int     i;

float   freq_nom;

float   high_f,low_f;

float   theta,theta_del,freq_new,freq_del;

char    value,t,b,z;



freq_nom   =  60.0;

high_f        =  66.0;

low_f         =  58.0;


theta      =  va1_ang;

theta_del  =  va1_del;
```

```
freq_del = ((1.0/(2.0*PI))*((theta - theta_del)*(PI/180.0)))

*freq_nom;


freq_new = (freq_nom + freq_del);


if (freq_new > high_f)

{

value='t';

}


else if (freq_new < low_f)

{

value='z';

}



else

{

value='b';

}
```

```
return (value);

}




char voltage(float Var, float Vai, float Vbr, float Vbi,

float Vcr, float Vci, float V_base, int j)

{

float    Vapu,Vbpu,Vcpu;

float    Vinsta,Vinstb,Vinstc;

char     value,t,b;

complex  Va_max(Var,Vai),Vb_max(Vbr,Vbi),Vc_max(Vcr,Vci);

float    Va,Vb,Vc,V_peak,V_high,V_low;


Va = abs(Va_max);

Vb = abs(Vb_max);

Vc = abs(Vc_max);


V_peak = 1.5;

V_high = 1.1;
```

```
V_low  = 0.9;


Vapu = (Va/V_base);

Vbpu = (Vb/V_base);

Vcpu = (Vc/V_base);

Vinsta = ((sqrt(2)*(Va))/V_base);

Vinstb = ((sqrt(2)*(Vb))/V_base);

Vinstc = ((sqrt(2)*(Vc))/V_base);




if (Vinsta >= V_peak)

{

value='z';

}


else if (Vinstb >= V_peak)

{

value='z';

}
```

```
else if (Vinstc >= V_peak)

{

value='z';

}


else if (Vapu >= V_high)

{

value='u';

}


else if (Vbpu >= V_high)

{

value='u';

}


else if (Vcpu >= V_high)

{

value='u';

}
```

```
else if (Vapu <= V_low)

{

value='t';

}


else if (Vbpu <= V_low)

{

value='t';

}


else if (Vcpu <= V_low)

{

value='t';

}


else

{

value='b';

}
```

```
return (value);

}




char negative(float Iar, float Iai, float Ibr, float Ibi,float Icr,

float Ici, float I_base, int j)

{

int     i;

float   Ia2_pu,data[84];

float   T_peak,Tdial,dial_sum,I_peak;

char    value,t,b;

complex a(-0.5,0.866);

complex Ia2;

complex Ia_max(Iar,Iai),Ib_max(Ibr,Ibi),Ic_max(Icr,Ici);

FILE    *s_file;



I_peak = 1.1;

T_peak = 50.0;
```

```
Ia2 = (0.3333)*(Ia_max+(pow(a,2)*(Ib_max))+(a*(Ic_max)));

Ia2_pu = abs(Ia2/I_base);


if ((s_file = fopen("tdial_su.m","r")) == NULL)

printf("Data file tdial_su.m not found\n");


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

{

fscanf(s_file,"%f\n",&data[i]);

}

fclose(s_file);


dial_sum = data[j-N];


if ( Ia2_pu > I_peak)

{

value='t';

}


else if ( dial_sum > T_peak)
```

```
{

value='t';

}


else

{

value='b';

}

return (value);

}



char check(float Var, float Vai, float Vbr, float Vbi, float Vcr, float Vci, int j)

{

int       m;

char     value,t,b;

float    Va_angi,Vb_angi,Vc_angi,Va_angf,Vb_angf,Vc_angf;

float    Va_angle[84],Vb_angle[84],Vc_angle[84];

float    Theta_1,Theta_2,Theta_3,crit_ang;

complex  Va_max(Var,Vai),Vb_max(Vbr,Vbi),Vc_max(Vcr,Vci);
```

```
m = j-N;

crit_ang =  60.0;



if ((real(Va_max) == 0) && (real(Vb_max) == 0) && (real(Vc_max) == 0))

{

Va_angle[m] = (180.0/PI)*(PI/2.0);

Vb_angle[m] = (180.0/PI)*(PI/2.0);

Vc_angle[m] = (180.0/PI)*(PI/2.0);

}


else if ((real(Va_max) == 0) || (real(Vb_max) == 0) || (real(Vc_max) == 0))

{

if ((real(Va_max) == 0) && (real(Vb_max) == 0))

{

 Va_angle[m] = (180.0/PI)*(PI/2.0);

 Vb_angle[m] = (180.0/PI)*(PI/2.0);

 Vc_angle[m] = (180.0/PI)*atan2(imag(Vc_max),real(Vc_max));

}
```

```
else if ((real(Va_max) == 0) && (real(Vc_max) == 0))

{

Va_angle[m] = (180.0/PI)*(PI/2.0);

Vb_angle[m] = (180.0/PI)*atan2(imag(Vb_max),real(Vb_max));

Vc_angle[m] = (180.0/PI)*(PI/2.0);

}


else if ((real(Vb_max) == 0) && (real(Vc_max) == 0))

{

Va_angle[m] = (180.0/PI)*atan2(imag(Va_max),real(Va_max));

Vb_angle[m] = (180.0/PI)*(PI/2.0);

Vc_angle[m] = (180.0/PI)*(PI/2.0);

}


else if (real(Va_max) == 0)

{

Va_angle[m] = (180.0/PI)*(PI/2.0);

Vb_angle[m] = (180.0/PI)*atan2(imag(Vb_max),real(Vb_max));

Vc_angle[m] = (180.0/PI)*atan2(imag(Vc_max),real(Vc_max));
}


else if (real(Vb_max) == 0)

{

Va_angle[m] = (180.0/PI)*atan2(imag(Va_max),real(Va_max));

Vb_angle[m] = (180.0/PI)*(PI/2.0);

Vc_angle[m] = (180.0/PI)*atan2(imag(Vc_max),real(Vc_max));

}


else if (real(Vc_max) == 0)

{

Va_angle[m] = (180.0/PI)*atan2(imag(Va_max),real(Va_max));

Vb_angle[m] = (180.0/PI)*atan2(imag(Vb_max),real(Vb_max));

Vc_angle[m] = (180.0/PI)*(PI/2.0);

}

}


else

{

Va_angle[m] = (180.0/PI)*atan2(imag(Va_max),real(Va_max));
```

```
Vb_angle[m] = (180.0/PI)*atan2(imag(Vb_max),real(Vb_max));

Vc_angle[m] = (180.0/PI)*atan2(imag(Vc_max),real(Vc_max));

}




Va_angi = Va_angle[0];

Vb_angi = Vb_angle[0];

Vc_angi = Vc_angle[0];

Va_angf = Va_angle[m];

Vb_angf = Vb_angle[m];

Vc_angf = Vc_angle[m];




Theta_1 = abs(Va_angf - Va_angi);

Theta_2 = abs(Vb_angf - Vb_angi);

Theta_3 = abs(Vc_angf - Vc_angi);




if    (Theta_1 > crit_ang)

{
```

```
value='t';

}


else if (Theta_2 > crit_ang)

{

value='t';

}


else if (Theta_3 > crit_ang)

{

value='t';

}


else

{

value='b';

}

return (value);

}
```

```
char power(float Par, float Pai, float Pbr, float Pbi, float Pcr, float Pci, int j)

{

complex  S_power;

float    P_t,Q_t,Pa,Pb,Pc,PF,crit_pow;

char     value,t,b;

complex  Pa_max(Par,Pai),Pb_max(Pbr,Pbi),Pc_max(Pcr,Pci);


crit_pow = 0.0;

S_power  = (Pa_max + Pb_max + Pc_max);

P_t = real(S_power);

Q_t = imag(S_power);

PF = (P_t / (sqrt((pow(P_t,2)) + (pow(Q_t,2)))));


if (Par < crit_pow)

{

value='t';

}


else if  (Pbr < crit_pow)
```

```
{

value='t';

}


else if (Pcr < crit_pow)

{

value='t';

}


else

{

value='b';

}

return (value);

}




char cap(float Va_f, float Vb_f, float Vc_f, float V_base, int j)

{
```

```
int     i;

char    value,t,b;

float   Va_pu,Vb_pu,Vc_pu;

float   cap_peak;


cap_peak = 1.7;


Va_pu  = Va_f/V_base;

Vb_pu  = Vb_f/V_base;

Vc_pu  = Vc_f/V_base;


if (Va_pu > cap_peak)

{

value='t';

}


else if (Vb_pu > cap_peak)

{

value='t';

}
```

```
else if (Vc_pu > cap_peak)

{

value='t';

}



else

{

value='b';

}

return (value);

}



char  meter(float Iar, float Iai, float Ibr, float Ibi,

float Icr, float Ici, float Var, float Vai,

float Vbr, float Vbi, float Vcr, float Vci,

float Par, float Pai, float Pbr, float Pbi,

float Pcr, float Pci, int j)
```

```
{

char        value;

float        Va,Vb,Vc,Ia,Ib,Ic,Sa_max,Sb_max,Sc_max;

complex  Va_max(Var,Vai),Vb_max(Vbr,Vbi),Vc_max(Vcr,Vci);

complex  Ia_max(Iar,Iai),Ib_max(Ibr,Ibi),Ic_max(Icr,Ici);

complex  Pa_max(Par,Pai),Pb_max(Pbr,Pbi),Pc_max(Pcr,Pci);

complex  Sa,Sb,Sc,St;

float        Va_angle,Vb_angle,Vc_angle,Ia_angle;

float         Ib_angle,Ic_angle,Sa_angle,Sb_angle;

float         Sc_angle;

float         Pa,Pb,Pc,Qa,Qb,Qc;

float         Pt,Qt;



Va = abs(Va_max);

Vb = abs(Vb_max);

Vc = abs(Vc_max);

Ia = abs(Ia_max);

Ib = abs(Ib_max);

Ic = abs(Ic_max);
```

```
if ((real(Va_max) == 0) && (real(Vb_max) == 0) && (real(Vc_max) == 0))

{

Va_angle = (180.0/PI)*(PI/2.0);

Vb_angle = (180.0/PI)*(PI/2.0);

Vc_angle = (180.0/PI)*(PI/2.0);

}



else if ((real(Va_max) == 0) || (real(Vb_max) == 0) || (real(Vc_max) == 0))

{

if ((real(Va_max) == 0) && (real(Vb_max) == 0))

{

Va_angle = (180.0/PI)*(PI/2.0);

Vb_angle = (180.0/PI)*(PI/2.0);

Vc_angle = (180.0/PI)*atan2(imag(Vc_max),real(Vc_max));

}


else if ((real(Va_max) == 0) && (real(Vc_max) == 0))
```

```
{

Va_angle = (180.0/PI)*(PI/2.0);

Vb_angle = (180.0/PI)*atan2(imag(Vb_max),real(Vb_max));

Vc_angle = (180.0/PI)*(PI/2.0);

}


else if ((real(Vb_max) == 0) && (real(Vc_max) == 0))

{

Va_angle = (180.0/PI)*atan2(imag(Va_max),real(Va_max));

Vb_angle = (180.0/PI)*(PI/2.0);

Vc_angle = (180.0/PI)*(PI/2.0);

}


else if (real(Va_max) == 0)

{

Va_angle = (180.0/PI)*(PI/2.0);

Vb_angle = (180.0/PI)*atan2(imag(Vb_max),real(Vb_max));

Vc_angle = (180.0/PI)*atan2(imag(Vc_max),real(Vc_max));

}
```

```
else if (real(Vb_max) == 0)

{

Va_angle = (180.0/PI)*atan2(imag(Va_max),real(Va_max));

Vb_angle = (180.0/PI)*(PI/2.0);

Vc_angle = (180.0/PI)*atan2(imag(Vc_max),real(Vc_max));

}


else if (real(Vc_max) == 0)

{

Va_angle = (180.0/PI)*atan2(imag(Va_max),real(Va_max));

Vb_angle = (180.0/PI)*atan2(imag(Vb_max),real(Vb_max));

Vc_angle = (180.0/PI)*(PI/2.0);

}

}


else

{

Va_angle = (180.0/PI)*atan2(imag(Va_max),real(Va_max));

Vb_angle = (180.0/PI)*atan2(imag(Vb_max),real(Vb_max));

Vc_angle = (180.0/PI)*atan2(imag(Vc_max),real(Vc_max));
```

```
}


if ((real(Ia_max) == 0) && (real(Ib_max) == 0) && (real(Ic_max) == 0))

{

Ia_angle = (180.0/PI)*(PI/2.0);

Ib_angle = (180.0/PI)*(PI/2.0);

Ic_angle = (180.0/PI)*(PI/2.0);

}


else if ((real(Ia_max) == 0) || (real(Ib_max) == 0) || (real(Ic_max) == 0))

{

if ((real(Ia_max) == 0) && (real(Ib_max) == 0))

{

Ia_angle = (180.0/PI)*(PI/2.0);

Ib_angle = (180.0/PI)*(PI/2.0);

Ic_angle = (180.0/PI)*atan2(imag(Ic_max),real(Ic_max));

}


else if ((real(Ia_max) == 0) && (real(Ic_max) == 0))

{
```

```
Ia_angle = (180.0/PI)*(PI/2.0);

Ib_angle = (180.0/PI)*atan2(imag(Ib_max),real(Ib_max));

Ic_angle = (180.0/PI)*(PI/2.0);

}


else if ((real(Ib_max) == 0) && (real(Ic_max) == 0))

{

Ia_angle = (180.0/PI)*atan2(imag(Ia_max),real(Ia_max));

Ib_angle = (180.0/PI)*(PI/2.0);

Ic_angle = (180.0/PI)*(PI/2.0);

}


else if (real(Ia_max) == 0)

{

Ia_angle = (180.0/PI)*(PI/2.0);

Ib_angle = (180.0/PI)*atan2(imag(Ib_max),real(Ib_max));

Ic_angle = (180.0/PI)*atan2(imag(Ic_max),real(Ic_max));

}


else if (real(Ib_max) == 0)
```

```
{

Ia_angle = (180.0/PI)*atan2(imag(Ia_max),real(Ia_max));

Ib_angle = (180.0/PI)*(PI/2.0);

Ic_angle = (180.0/PI)*atan2(imag(Ic_max),real(Ic_max));

}


else if (real(Ic_max) == 0)

{

Ia_angle = (180.0/PI)*atan2(imag(Ia_max),real(Ia_max));

Ib_angle = (180.0/PI)*atan2(imag(Ib_max),real(Ib_max));

Ic_angle = (180.0/PI)*(PI/2.0);

}

}


else

{

Ia_angle = (180.0/PI)*atan2(imag(Ia_max),real(Ia_max));

Ib_angle = (180.0/PI)*atan2(imag(Ib_max),real(Ib_max));

Ic_angle = (180.0/PI)*atan2(imag(Ic_max),real(Ic_max));

}
```

```
Sa    = Pa_max;

Sb    = Pb_max;

Sc    = Pc_max;


Sa_max = abs(Sa);

Sb_max = abs(Sb);

Sc_max = abs(Sc);




Sa_angle = (180.0/PI)*arg(Sa);

Sb_angle = (180.0/PI)*arg(Sb);

Sc_angle = (180.0/PI)*arg(Sc);


Pa    = real(Sa);

Pb    = real(Sb);

Pc    = real(Sc);


Qa    = imag(Sa);

Qb    = imag(Sb);
```

```
Qc   = imag(Sc);


St   = (Sa + Sb + Sc);

Pt   = (Pa + Pb + Pc);

Qt   = (Qa + Qb + Qc);


value = 'b';

return(value);

}
```

## D.2 AUXILIARY PROGRAM A

The purpose of the auxiliary program A was to calculate the phasors for voltage, current, impedance and power.

```
#include <stdio.h>

#include <math.h>

#include <complex.h>

#include <iostream.h>

#define    PI    3.14159
```

```
#define    M     96

#define    N     12

#define    O     12

#define    Q     96

#define    T     96




void main()

{

int i,k;

float v1y[96],v2y[96],v3y[96];

float i1y[96],i2y[96],i3y[96];

float z1y[96],z2y[96],z3y[96];

float p1y[96],p2y[96],p3y[96],pgy[96];

float Varr = 0,Vaii = 0,Vbrr = 0,Vbii = 0,Vcrr = 0,Vcii = 0;

float Iarr = 0,Iaii = 0,Ibrr = 0,Ibii = 0,Icrr = 0,Icii = 0;

float Zarr = 0,Zaii = 0,Zbrr = 0,Zbii = 0,Zcrr = 0,Zcii = 0;

float Parr = 0,Paii = 0,Pbrr = 0,Pbii = 0,Pcrr = 0,Pcii = 0;

float Pgrr = 0,Pgii = 0;
float Var,Vai,Vbr,Vbi,Vcr,Vci;

float Iar,Iai,Ibr,Ibi,Icr,Ici;

float Zar,Zai,Zbr,Zbi,Zcr,Zci;

float Par,Pai,Pbr,Pbi,Pcr,Pci,Pgr,Pgi;

float Var_new,Var_old,Vai_new,Vai_old,Vbr_new,Vbr_old,Vbi_new,Vbi_old;

float Vcr_new,Vcr_old,Vci_new,Vci_old,Iar_new,Iar_old,Iai_new,Iai_old;

float Ibr_new,Ibr_old,Ibi_new,Ibi_old,Icr_new,Icr_old,Ici_new,Ici_old;

float Zar_new,Zar_old,Zai_new,Zai_old,Zbr_new,Zbr_old,Zbi_new,Zbi_old;

float Zcr_new,Zcr_old,Zci_new,Zci_old,Par_new,Par_old,Pai_new,Pai_old;

float Pbr_new,Pbr_old,Pbi_new,Pbi_old,Pcr_new,Pcr_old,Pci_new,Pci_old;

float Pgr_new,Pgr_old,Pgi_new,Pgi_old;

float Var_del,Vai_del,Vbr_del,Vbi_del,Vcr_del,Vci_del;

float Var_past,Vai_past,Vbr_past,Vbi_past,Vcr_past,Vci_past;

float data[96],data1[96],data2[96],data3[96];

float data4[96],data5[96],data6[96],data7[96],data8[96];

float data9[96],data10[96],data11[96],data12[96],filter[12];

char  choice;

FILE *fp,*fp1,*fp2,*fp3,*fp4,*fp5;

FILE *fp6,*fp7,*fp8,*fp9,*fp10,*fp11,*fp12;

FILE *infile_1,*infile_2,*infile_3,*infile_4,*infile_5,*infile_6,*infile_7;
FILE *infile_8,*infile_9,*infile_10,*infile_11,*infile_12,*infile_13;

FILE *outfile_1,*outfile_2,*outfile_3,*outfile_4,*outfile_5;

FILE *outfile_6,*outfile_7,*outfile_8,*outfile_9,*outfile_10;

FILE *outfile_11,*outfile_12,*outfile_13;

FILE *infile,*x_outfile,*y_outfile,*z_outfile;




if ((fp = fopen("v1.m","r")) == NULL)

printf("Data file v1.m not found\n");


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

{

fscanf(fp,"%e\n",&data[i]);

}

fclose(fp);




if ((fp1 = fopen("v2.m","r")) == NULL)

printf("Data file v2.m not found\n");
```

```
for (i=0;i< M;i++)

{

fscanf(fp1,"%e\n",&data1[i]);

}

fclose(fp1);




if ((fp2 = fopen("v3.m","r")) == NULL)

printf("Data file v3.m not found\n");


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

{

fscanf(fp2,"%e\n",&data2[i]);

}

fclose(fp2);




if ((fp3 = fopen("i1.m","r")) == NULL)

printf("Data file i1.m not found\n");
```

```
for (i=0;i< M;i++)

{

fscanf(fp3,"%e\n",&data3[i]);

}

fclose(fp3);




if ((fp4 = fopen("i2.m","r")) == NULL)

printf("Data file i2.m not found\n");


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

{

fscanf(fp4,"%e\n",&data4[i]);

}

fclose(fp4);




if ((fp5 = fopen("i3.m","r")) == NULL)

printf("Data file i3.m not found\n");
```

```
for (i=0;i< M;i++)

{

fscanf(fp5,"%e\n",&data5[i]);

}

fclose(fp5);


if ((fp6 = fopen("z1.m","r")) == NULL)

printf("Data file z1.m not found\n");


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

{

fscanf(fp6,"%e\n",&data6[i]);

}

fclose(fp6);



if ((fp7 = fopen("z2.m","r")) == NULL)

printf("Data file z2.m not found\n");


for (i=0;i< M;i++)
```

```
{

fscanf(fp7,"%e\n",&data7[i]);

}

fclose(fp7);




if ((fp8 = fopen("z3.m","r")) == NULL)

printf("Data file z3.m not found\n");


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

{

fscanf(fp8,"%e\n",&data8[i]);

}

fclose(fp8);




if ((fp9 = fopen("p1.m","r")) == NULL)

printf("Data file p1.m not found\n");


for (i=0;i< M;i++)
```

```
{

fscanf(fp9,"%e\n",&data9[i]);

}

fclose(fp9);






if ((fp10 = fopen("p2.m","r")) == NULL)

printf("Data file p2.m not found\n");


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

{

fscanf(fp10,"%e\n",&data10[i]);

}

fclose(fp10);


if ((fp11 = fopen("p3.m","r")) == NULL)

printf("Data file p3.m not found\n");


for (i=0;i< M;i++)
```

```
{

fscanf(fp11,"%e\n",&data11[i]);

}

fclose(fp11);




if ((fp12 = fopen("pow_gen.m","r")) == NULL)

printf("Data file pow_gen.m not found\n");


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

{

fscanf(fp12,"%e\n",&data12[i]);

}

fclose(fp12);




if ((infile = fopen("f5.out","r")) == NULL)

printf("Data file f5.out not found\n");


for (i=0;i < O;i++)
```

```
{

fscanf(infile,"%e\n",&filter[i]);

}

fclose(infile);




/* Beginning of Phasor Computation */


for ( k = 0; k < N; k++)

{

if (k==0)

{

v1y[k] = (filter[0]*data[0]);

v2y[k] = (filter[0]*data1[0]);

v3y[k] = (filter[0]*data2[0]);

i1y[k] = (filter[0]*data3[0]);

i2y[k] = (filter[0]*data4[0]);

i3y[k] = (filter[0]*data5[0]);

z1y[k] = (filter[0]*data6[0]);

z2y[k] = (filter[0]*data7[0]);
```

```
z3y[k] = (filter[0]*data8[0]);

p1y[k] = (filter[0]*data9[0]);

p2y[k] = (filter[0]*data10[0]);

p3y[k] = (filter[0]*data11[0]);

pgy[k] = (filter[0]*data12[0]);

}


else if  (k==1)

{

v1y[k] = ((filter[0]*data[1]) +  (filter[1]*data[0]));

v2y[k] = ((filter[0]*data1[1]) +  (filter[1]*data1[0]));

v3y[k] = ((filter[0]*data2[1]) +  (filter[1]*data2[0]));

i1y[k] = ((filter[0]*data3[1]) +  (filter[1]*data3[0]));

i2y[k] = ((filter[0]*data4[1]) +  (filter[1]*data4[0]));

i3y[k] = ((filter[0]*data5[1]) +  (filter[1]*data5[0]));

z1y[k] = ((filter[0]*data6[1]) +  (filter[1]*data6[0]));

z2y[k] = ((filter[0]*data7[1]) +  (filter[1]*data7[0]));

z3y[k] = ((filter[0]*data8[1]) +  (filter[1]*data8[0]));

p1y[k] = ((filter[0]*data9[1]) +  (filter[1]*data9[0]));

p2y[k] = ((filter[0]*data10[1]) +  (filter[1]*data10[0]));
```

```
p3y[k] = ((filter[0]*data11[1]) +  (filter[1]*data11[0]));

pgy[k] = ((filter[0]*data12[1]) +  (filter[1]*data12[0]));

}


else if (k==2)

{

v1y[k] =  ((filter[0]*data[2]) +  (filter[1]*data[1])+ (filter[2]*data[0]));

v2y[k] =  ((filter[0]*data1[2]) +  (filter[1]*data1[1])+ (filter[2]*data1[0]));

v3y[k] =  ((filter[0]*data2[2]) +  (filter[1]*data2[1])+ (filter[2]*data2[0]));

i1y[k] =  ((filter[0]*data3[2]) +  (filter[1]*data3[1])+ (filter[2]*data3[0]));

i2y[k] =  ((filter[0]*data4[2]) +  (filter[1]*data4[1])+ (filter[2]*data4[0]));

i3y[k] =  ((filter[0]*data5[2]) +  (filter[1]*data5[1])+ (filter[2]*data5[0]));

z1y[k] =  ((filter[0]*data6[2]) +  (filter[1]*data6[1])+ (filter[2]*data6[0]));

z2y[k] =  ((filter[0]*data7[2]) +  (filter[1]*data7[1])+ (filter[2]*data7[0]));

z3y[k] =  ((filter[0]*data8[2]) +  (filter[1]*data8[1])+ (filter[2]*data8[0]));

p1y[k] =  ((filter[0]*data9[2]) +  (filter[1]*data9[1])+ (filter[2]*data9[0]));

p2y[k] =  ((filter[0]*data10[2]) +  (filter[1]*data10[1])+ (filter[2]*data10[0]));

p3y[k] =  ((filter[0]*data11[2]) +  (filter[1]*data11[1])+ (filter[2]*data11[0]));

pgy[k] =  ((filter[0]*data12[2]) +  (filter[1]*data12[1])+ (filter[2]*data12[0]));

}
```

```
else if (k==3)

{

v1y[k] =  ((filter[0]*data[3]) +  (filter[1]*data[2])

+ (filter[2]*data[1])  +  (filter[3]*data[0]));


v2y[k] =  ((filter[0]*data1[3]) +  (filter[1]*data1[2])

+ (filter[2]*data1[1])  +  (filter[3]*data1[0]));


v3y[k] =  ((filter[0]*data2[3]) +  (filter[1]*data2[2])

+ (filter[2]*data2[1])  +  (filter[3]*data2[0]));


i1y[k] =  ((filter[0]*data3[3]) +  (filter[1]*data3[2])

+ (filter[2]*data3[1])  +  (filter[3]*data3[0]));


i2y[k] =  ((filter[0]*data4[3]) +  (filter[1]*data4[2])

+ (filter[2]*data4[1])  +  (filter[3]*data4[0]));


i3y[k] =  ((filter[0]*data5[3]) +  (filter[1]*data5[2])

+ (filter[2]*data5[1])  +  (filter[3]*data5[0]));
```

```
z1y[k] =  ((filter[0]*data6[3]) +  (filter[1]*data6[2])

+ (filter[2]*data6[1])  +  (filter[3]*data6[0]));


z2y[k] =  ((filter[0]*data7[3]) +  (filter[1]*data7[2])

+ (filter[2]*data7[1])  +  (filter[3]*data7[0]));


z3y[k] =  ((filter[0]*data8[3]) +  (filter[1]*data8[2])

+ (filter[2]*data8[1])  +  (filter[3]*data8[0]));


p1y[k] =  ((filter[0]*data9[3]) +  (filter[1]*data9[2])

+ (filter[2]*data9[1])  +  (filter[3]*data9[0]));


p2y[k] =  ((filter[0]*data10[3]) +  (filter[1]*data10[2])

+ (filter[2]*data10[1])  +  (filter[3]*data10[0]));


p3y[k] =  ((filter[0]*data11[3]) +  (filter[1]*data11[2])

+ (filter[2]*data11[1])  +  (filter[3]*data11[0]));
```

```
pgy[k] =  ((filter[0]*data12[3]) +  (filter[1]*data12[2])

+ (filter[2]*data12[1])  +  (filter[3]*data12[0]));

}


else if (k==4)

{

v1y[k] =  ((filter[0]*data[4]) +  (filter[1]*data[3])

+ (filter[2]*data[2])  +  (filter[3]*data[1])+ (filter[4]*data[0]));


v2y[k] =  ((filter[0]*data1[4]) +  (filter[1]*data1[3])

+ (filter[2]*data1[2])  +  (filter[3]*data1[1])+ (filter[4]*data1[0]));


v3y[k] =  ((filter[0]*data2[4]) +  (filter[1]*data2[3])

+ (filter[2]*data2[2])  +  (filter[3]*data2[1])+ (filter[4]*data2[0]));


i1y[k] =  ((filter[0]*data3[4]) +  (filter[1]*data3[3])

+ (filter[2]*data3[2])  +  (filter[3]*data3[1])+ (filter[4]*data3[0]));


i2y[k] =  ((filter[0]*data4[4]) +  (filter[1]*data4[3])

+ (filter[2]*data4[2])  +  (filter[3]*data4[1])+ (filter[4]*data4[0]));
```

```
i3y[k] =  ((filter[0]*data5[4]) +  (filter[1]*data5[3])

+ (filter[2]*data5[2])  +  (filter[3]*data5[1])+ (filter[4]*data5[0]));


z1y[k] =  ((filter[0]*data6[4]) +  (filter[1]*data6[3])

+ (filter[2]*data6[2])  +  (filter[3]*data6[1])+ (filter[4]*data6[0]));


z2y[k] =  ((filter[0]*data7[4]) +  (filter[1]*data7[3])

+ (filter[2]*data7[2])  +  (filter[3]*data7[1])+ (filter[4]*data7[0]));


z3y[k] =  ((filter[0]*data8[4]) +  (filter[1]*data8[3])

+ (filter[2]*data8[2])  +  (filter[3]*data8[1])+ (filter[4]*data8[0]));


p1y[k] =  ((filter[0]*data9[4]) +  (filter[1]*data9[3])

+ (filter[2]*data9[2])  +  (filter[3]*data9[1])+ (filter[4]*data9[0]));


p2y[k] =  ((filter[0]*data10[4]) +  (filter[1]*data10[3])

+ (filter[2]*data10[2])  +  (filter[3]*data10[1])+ (filter[4]*data10[0]));


p3y[k] =  ((filter[0]*data11[4]) +  (filter[1]*data11[3])
+ (filter[2]*data11[2])  +  (filter[3]*data11[1])+ (filter[4]*data11[0]));


pgy[k] =  ((filter[0]*data12[4]) +  (filter[1]*data12[3])

+ (filter[2]*data12[2])  +  (filter[3]*data12[1])+ (filter[4]*data12[0]));

}


else if (k==5)

{

v1y[k] =  ((filter[0]*data[5]) +  (filter[1]*data[4])

+ (filter[2]*data[3])  +  (filter[3]*data[2])

+ (filter[4]*data[1])  +  (filter[5]*data[0]));


v2y[k] =  ((filter[0]*data1[5]) +  (filter[1]*data1[4])

+ (filter[2]*data1[3])  +  (filter[3]*data1[2])

+ (filter[4]*data1[1])  +  (filter[5]*data1[0]));


v3y[k] =  ((filter[0]*data2[5]) +  (filter[1]*data2[4])

+ (filter[2]*data2[3])  +  (filter[3]*data2[2])

+ (filter[4]*data2[1])  +  (filter[5]*data2[0]));
```

```
i1y[k] =  ((filter[0]*data3[5]) +  (filter[1]*data3[4])

+ (filter[2]*data3[3])  +  (filter[3]*data3[2])

+ (filter[4]*data3[1])  +  (filter[5]*data3[0]));


i2y[k] =  ((filter[0]*data4[5]) +  (filter[1]*data4[4])

+ (filter[2]*data4[3])  +  (filter[3]*data4[2])

+ (filter[4]*data4[1])  +  (filter[5]*data4[0]));


i3y[k] =  ((filter[0]*data5[5]) +  (filter[1]*data5[4])

+ (filter[2]*data5[3])  +  (filter[3]*data5[2])

+ (filter[4]*data5[1])  +  (filter[5]*data5[0]));


z1y[k] =  ((filter[0]*data6[5]) +  (filter[1]*data6[4])

+ (filter[2]*data6[3])  +  (filter[3]*data6[2])

+ (filter[4]*data6[1])  +  (filter[5]*data6[0]));


z2y[k] =  ((filter[0]*data7[5]) +  (filter[1]*data7[4])

+ (filter[2]*data7[3])  +  (filter[3]*data7[2])

+ (filter[4]*data7[1])  +  (filter[5]*data7[0]));
```

```
z3y[k] =  ((filter[0]*data8[5]) +  (filter[1]*data8[4])

+ (filter[2]*data8[3])  +  (filter[3]*data8[2])

+ (filter[4]*data8[1])  +  (filter[5]*data8[0]));


p1y[k] =  ((filter[0]*data9[5]) +  (filter[1]*data9[4])

+ (filter[2]*data9[3])  +  (filter[3]*data9[2])

+ (filter[4]*data9[1])  +  (filter[5]*data9[0]));


p2y[k] =  ((filter[0]*data10[5]) +  (filter[1]*data10[4])

+ (filter[2]*data10[3])  +  (filter[3]*data10[2])

+ (filter[4]*data10[1])  +  (filter[5]*data10[0]));


p3y[k] =  ((filter[0]*data11[5]) +  (filter[1]*data11[4])

+ (filter[2]*data11[3])  +  (filter[3]*data11[2])

+ (filter[4]*data11[1])  +  (filter[5]*data11[0]));


pgy[k] =  ((filter[0]*data12[5]) +  (filter[1]*data12[4])

+ (filter[2]*data12[3])  +  (filter[3]*data12[2])

+ (filter[4]*data12[1])  +  (filter[5]*data12[0]));

}
```

```
else if (k==6)

{

v1y[k] =  ((filter[0]*data[6])   +  (filter[1]*data[5])

+ (filter[2]*data[4])  +  (filter[3]*data[3])

+ (filter[4]*data[2])  +  (filter[5]*data[1])

+ (filter[6]*data[0]));


v2y[k] =  ((filter[0]*data1[6])   +  (filter[1]*data1[5])

+ (filter[2]*data1[4])  +  (filter[3]*data1[3])

+ (filter[4]*data1[2])  +  (filter[5]*data1[1])

+ (filter[6]*data1[0]));


v3y[k] =  ((filter[0]*data2[6])   +  (filter[1]*data2[5])

+ (filter[2]*data2[4])  +  (filter[3]*data2[3])

+ (filter[4]*data2[2])  +  (filter[5]*data2[1])

+ (filter[6]*data2[0]));


i1y[k] =  ((filter[0]*data3[6])   +  (filter[1]*data3[5])

+ (filter[2]*data3[4])  +  (filter[3]*data3[3])
+ (filter[4]*data3[2])  +  (filter[5]*data3[1])

+ (filter[6]*data3[0]));


i2y[k] =  ((filter[0]*data4[6])   +  (filter[1]*data4[5])

+ (filter[2]*data4[4])  +  (filter[3]*data4[3])

+ (filter[4]*data4[2])  +  (filter[5]*data4[1])

+ (filter[6]*data4[0]));


i3y[k] =  ((filter[0]*data5[6])   +  (filter[1]*data5[5])

+ (filter[2]*data5[4])  +  (filter[3]*data5[3])

+ (filter[4]*data5[2])  +  (filter[5]*data5[1])

+ (filter[6]*data5[0]));


z1y[k] =  ((filter[0]*data6[6])   +  (filter[1]*data6[5])

+ (filter[2]*data6[4])  +  (filter[3]*data6[3])

+ (filter[4]*data6[2])  +  (filter[5]*data6[1])

+ (filter[6]*data6[0]));


z2y[k] =  ((filter[0]*data7[6])   +  (filter[1]*data7[5])

+ (filter[2]*data7[4])  +  (filter[3]*data7[3])
+ (filter[4]*data7[2])  +  (filter[5]*data7[1])

+ (filter[6]*data7[0]));


z3y[k] =  ((filter[0]*data8[6])   +  (filter[1]*data8[5])

+ (filter[2]*data8[4])  +  (filter[3]*data8[3])

+ (filter[4]*data8[2])  +  (filter[5]*data8[1])

+ (filter[6]*data8[0]));


p1y[k] =  ((filter[0]*data9[6])   +  (filter[1]*data9[5])

+ (filter[2]*data9[4])  +  (filter[3]*data9[3])

+ (filter[4]*data9[2])  +  (filter[5]*data9[1])

+ (filter[6]*data9[0]));


p2y[k] =  ((filter[0]*data10[6])   +  (filter[1]*data10[5])

+ (filter[2]*data10[4])  +  (filter[3]*data10[3])

+ (filter[4]*data10[2])  +  (filter[5]*data10[1])

+ (filter[6]*data10[0]));


p3y[k] =  ((filter[0]*data11[6])   +  (filter[1]*data11[5])

+ (filter[2]*data11[4])  +  (filter[3]*data11[3])
+ (filter[4]*data11[2])  +  (filter[5]*data11[1])

+ (filter[6]*data11[0]));


pgy[k] =  ((filter[0]*data12[6])   +  (filter[1]*data12[5])

+ (filter[2]*data12[4])  +  (filter[3]*data12[3])

+ (filter[4]*data12[2])  +  (filter[5]*data12[1])

+ (filter[6]*data12[0]));

}



else if (k==7)

{

v1y[k] =  ((filter[0]*data[7])   +  (filter[1]*data[6])

+ (filter[2]*data[5])  +  (filter[3]*data[4])

+ (filter[4]*data[3])  +  (filter[5]*data[2])

+ (filter[6]*data[1])  +  (filter[7]*data[0]));


v2y[k] =  ((filter[0]*data1[7])   +  (filter[1]*data1[6])

+ (filter[2]*data1[5])  +  (filter[3]*data1[4])

+ (filter[4]*data1[3])  +  (filter[5]*data1[2])
+ (filter[6]*data1[1])  +  (filter[7]*data1[0]));


v3y[k] =  ((filter[0]*data2[7])   +  (filter[1]*data2[6])

+ (filter[2]*data2[5])  +  (filter[3]*data2[4])

+ (filter[4]*data2[3])  +  (filter[5]*data2[2])

+ (filter[6]*data2[1])  +  (filter[7]*data2[0]));


i1y[k] =  ((filter[0]*data3[7])   +  (filter[1]*data3[6])

+ (filter[2]*data3[5])  +  (filter[3]*data3[4])

+ (filter[4]*data3[3])  +  (filter[5]*data3[2])

+ (filter[6]*data3[1])  +  (filter[7]*data3[0]));


i2y[k] =  ((filter[0]*data4[7])   +  (filter[1]*data4[6])

+ (filter[2]*data4[5])  +  (filter[3]*data4[4])

+ (filter[4]*data4[3])  +  (filter[5]*data4[2])

+ (filter[6]*data4[1])  +  (filter[7]*data4[0]));


i3y[k] =  ((filter[0]*data5[7])   +  (filter[1]*data5[6])

+ (filter[2]*data5[5])  +  (filter[3]*data5[4])

+ (filter[4]*data5[3])  +  (filter[5]*data5[2])
+ (filter[6]*data5[1])  +  (filter[7]*data5[0]));


z1y[k] =  ((filter[0]*data6[7])   +  (filter[1]*data6[6])

+ (filter[2]*data6[5])  +  (filter[3]*data6[4])

+ (filter[4]*data6[3])  +  (filter[5]*data6[2])

+ (filter[6]*data6[1])  +  (filter[7]*data6[0]));


z2y[k] =  ((filter[0]*data7[7])   +  (filter[1]*data7[6])

+ (filter[2]*data7[5])  +  (filter[3]*data7[4])

+ (filter[4]*data7[3])  +  (filter[5]*data7[2])

+ (filter[6]*data7[1])  +  (filter[7]*data7[0]));


z3y[k] =  ((filter[0]*data8[7])   +  (filter[1]*data8[6])

+ (filter[2]*data8[5])  +  (filter[3]*data8[4])

+ (filter[4]*data8[3])  +  (filter[5]*data8[2])

+ (filter[6]*data8[1])  +  (filter[7]*data8[0]));


p1y[k] =  ((filter[0]*data9[7])   +  (filter[1]*data9[6])

+ (filter[2]*data9[5])  +  (filter[3]*data9[4])

+ (filter[4]*data9[3])  +  (filter[5]*data9[2])
+ (filter[6]*data9[1])  +  (filter[7]*data9[0]));


p2y[k] =  ((filter[0]*data10[7])   +  (filter[1]*data10[6])

+ (filter[2]*data10[5])  +  (filter[3]*data10[4])

+ (filter[4]*data10[3])  +  (filter[5]*data10[2])

+ (filter[6]*data10[1])  +  (filter[7]*data10[0]));


p3y[k] =  ((filter[0]*data11[7])   +  (filter[1]*data11[6])

+ (filter[2]*data11[5])  +  (filter[3]*data11[4])

+ (filter[4]*data11[3])  +  (filter[5]*data11[2])

+ (filter[6]*data11[1])  +  (filter[7]*data11[0]));


pgy[k] =  ((filter[0]*data12[7])   +  (filter[1]*data12[6])

+ (filter[2]*data12[5])  +  (filter[3]*data12[4])

+ (filter[4]*data12[3])  +  (filter[5]*data12[2])

+ (filter[6]*data12[1])  +  (filter[7]*data12[0]));

}



else if (k==8)
```

```
{

v1y[k] =  ((filter[0]*data[8]) +  (filter[1]*data[7])

+ (filter[2]*data[6])  +  (filter[3]*data[5])

+ (filter[4]*data[4])  +  (filter[5]*data[3])

+ (filter[6]*data[2])  +  (filter[7]*data[1])

+ (filter[8]*data[0]));


v2y[k] =  ((filter[0]*data1[8]) +  (filter[1]*data1[7])

+ (filter[2]*data1[6])  +  (filter[3]*data1[5])

+ (filter[4]*data1[4])  +  (filter[5]*data1[3])

+ (filter[6]*data1[2])  +  (filter[7]*data1[1])

+ (filter[8]*data1[0]));


v3y[k] =  ((filter[0]*data2[8]) +  (filter[1]*data2[7])

+ (filter[2]*data2[6])  +  (filter[3]*data2[5])

+ (filter[4]*data2[4])  +  (filter[5]*data2[3])

+ (filter[6]*data2[2])  +  (filter[7]*data2[1])

+ (filter[8]*data2[0]));


i1y[k] =  ((filter[0]*data3[8]) +  (filter[1]*data3[7])
+ (filter[2]*data3[6])  +  (filter[3]*data3[5])

+ (filter[4]*data3[4])  +  (filter[5]*data3[3])

+ (filter[6]*data3[2])  +  (filter[7]*data3[1])

+ (filter[8]*data3[0]));


i2y[k] =  ((filter[0]*data4[8]) +  (filter[1]*data4[7])

+ (filter[2]*data4[6])  +  (filter[3]*data4[5])

+ (filter[4]*data4[4])  +  (filter[5]*data4[3])

+ (filter[6]*data4[2])  +  (filter[7]*data4[1])

+ (filter[8]*data4[0]));


i3y[k] =  ((filter[0]*data5[8]) +  (filter[1]*data5[7])

+ (filter[2]*data5[6])  +  (filter[3]*data5[5])

+ (filter[4]*data5[4])  +  (filter[5]*data5[3])

+ (filter[6]*data5[2])  +  (filter[7]*data5[1])

+ (filter[8]*data5[0]));



z1y[k] =  ((filter[0]*data6[8]) +  (filter[1]*data6[7])

+ (filter[2]*data6[6])  +  (filter[3]*data6[5])
+ (filter[4]*data6[4])  +  (filter[5]*data6[3])

+ (filter[6]*data6[2])  +  (filter[7]*data6[1])

+ (filter[8]*data6[0]));


z2y[k] =  ((filter[0]*data7[8]) +  (filter[1]*data7[7])

+ (filter[2]*data7[6])  +  (filter[3]*data7[5])

+ (filter[4]*data7[4])  +  (filter[5]*data7[3])

+ (filter[6]*data7[2])  +  (filter[7]*data7[1])

+ (filter[8]*data7[0]));


z3y[k] =  ((filter[0]*data8[8]) +  (filter[1]*data8[7])

+ (filter[2]*data8[6])  +  (filter[3]*data8[5])

+ (filter[4]*data8[4])  +  (filter[5]*data8[3])

+ (filter[6]*data8[2])  +  (filter[7]*data8[1])

+ (filter[8]*data8[0]));


p1y[k] =  ((filter[0]*data9[8]) +  (filter[1]*data9[7])

+ (filter[2]*data9[6])  +  (filter[3]*data9[5])

+ (filter[4]*data9[4])  +  (filter[5]*data9[3])

+ (filter[6]*data9[2])  +  (filter[7]*data9[1])
+ (filter[8]*data9[0]));


p2y[k] =  ((filter[0]*data10[8]) +  (filter[1]*data10[7])

+ (filter[2]*data10[6])  +  (filter[3]*data10[5])

+ (filter[4]*data10[4])  +  (filter[5]*data10[3])

+ (filter[6]*data10[2])  +  (filter[7]*data10[1])

+ (filter[8]*data10[0]));


p3y[k] =  ((filter[0]*data11[8]) +  (filter[1]*data11[7])

+ (filter[2]*data11[6])  +  (filter[3]*data11[5])

+ (filter[4]*data11[4])  +  (filter[5]*data11[3])

+ (filter[6]*data11[2])  +  (filter[7]*data11[1])

+ (filter[8]*data11[0]));


pgy[k] =  ((filter[0]*data12[8]) +  (filter[1]*data12[7])

+ (filter[2]*data12[6])  +  (filter[3]*data12[5])

+ (filter[4]*data12[4])  +  (filter[5]*data12[3])

+ (filter[6]*data12[2])  +  (filter[7]*data12[1])

+ (filter[8]*data12[0]));

}
```

```
else if (k==9)

{

v1y[k] =  ((filter[0]*data[9]) +  (filter[1]*data[8])

+ (filter[2]*data[7])  +  (filter[3]*data[6])

+ (filter[4]*data[5])  +  (filter[5]*data[4])

+ (filter[6]*data[3])  +  (filter[7]*data[2])

+ (filter[8]*data[1])  +  (filter[9]*data[0]));


v2y[k] =  ((filter[0]*data1[9]) +  (filter[1]*data1[8])

+ (filter[2]*data1[7])  +  (filter[3]*data1[6])

+ (filter[4]*data1[5])  +  (filter[5]*data1[4])

+ (filter[6]*data1[3])  +  (filter[7]*data1[2])

+ (filter[8]*data1[1])  +  (filter[9]*data1[0]));


v3y[k] =  ((filter[0]*data2[9]) +  (filter[1]*data2[8])

+ (filter[2]*data2[7])  +  (filter[3]*data2[6])

+ (filter[4]*data2[5])  +  (filter[5]*data2[4])

+ (filter[6]*data2[3])  +  (filter[7]*data2[2])

+ (filter[8]*data2[1])  +  (filter[9]*data2[0]));
```

```
i1y[k] =  ((filter[0]*data3[9]) +  (filter[1]*data3[8])

+ (filter[2]*data3[7])  +  (filter[3]*data3[6])

+ (filter[4]*data3[5])  +  (filter[5]*data3[4])

+ (filter[6]*data3[3])  +  (filter[7]*data3[2])

+ (filter[8]*data3[1])  +  (filter[9]*data3[0]));


i2y[k] =  ((filter[0]*data4[9]) +  (filter[1]*data4[8])

+ (filter[2]*data4[7])  +  (filter[3]*data4[6])

+ (filter[4]*data4[5])  +  (filter[5]*data4[4])

+ (filter[6]*data4[3])  +  (filter[7]*data4[2])

+ (filter[8]*data4[1])  +  (filter[9]*data4[0]));


i3y[k] =  ((filter[0]*data5[9]) +  (filter[1]*data5[8])

+ (filter[2]*data5[7])  +  (filter[3]*data5[6])

+ (filter[4]*data5[5])  +  (filter[5]*data5[4])

+ (filter[6]*data5[3])  +  (filter[7]*data5[2])

+ (filter[8]*data5[1])  +  (filter[9]*data5[0]));


z1y[k] =  ((filter[0]*data6[9]) +  (filter[1]*data6[8])
+ (filter[2]*data6[7])  +  (filter[3]*data6[6])

+ (filter[4]*data6[5])  +  (filter[5]*data6[4])

+ (filter[6]*data6[3])  +  (filter[7]*data6[2])

+ (filter[8]*data6[1])  +  (filter[9]*data6[0]));


z2y[k] =  ((filter[0]*data7[9]) +  (filter[1]*data7[8])

+ (filter[2]*data7[7])  +  (filter[3]*data7[6])

+ (filter[4]*data7[5])  +  (filter[5]*data7[4])

+ (filter[6]*data7[3])  +  (filter[7]*data7[2])

+ (filter[8]*data7[1])  +  (filter[9]*data7[0]));


z3y[k] =  ((filter[0]*data8[9]) +  (filter[1]*data8[8])

+ (filter[2]*data8[7])  +  (filter[3]*data8[6])

+ (filter[4]*data8[5])  +  (filter[5]*data8[4])

+ (filter[6]*data8[3])  +  (filter[7]*data8[2])

+ (filter[8]*data8[1])  +  (filter[9]*data8[0]));


p1y[k] =  ((filter[0]*data9[9]) +  (filter[1]*data9[8])

+ (filter[2]*data9[7])  +  (filter[3]*data9[6])

+ (filter[4]*data9[5])  +  (filter[5]*data9[4])
+ (filter[6]*data9[3])  +  (filter[7]*data9[2])

+ (filter[8]*data9[1])  +  (filter[9]*data9[0]));


p2y[k] =  ((filter[0]*data10[9]) +  (filter[1]*data10[8])

+ (filter[2]*data10[7])  +  (filter[3]*data10[6])

+ (filter[4]*data10[5])  +  (filter[5]*data10[4])

+ (filter[6]*data10[3])  +  (filter[7]*data10[2])

+ (filter[8]*data10[1])  +  (filter[9]*data10[0]));


p3y[k] =  ((filter[0]*data11[9]) +  (filter[1]*data11[8])

+ (filter[2]*data11[7])  +  (filter[3]*data11[6])

+ (filter[4]*data11[5])  +  (filter[5]*data11[4])

+ (filter[6]*data11[3])  +  (filter[7]*data11[2])

+ (filter[8]*data11[1])  +  (filter[9]*data11[0]));


pgy[k] =  ((filter[0]*data12[9]) +  (filter[1]*data12[8])

+ (filter[2]*data12[7])  +  (filter[3]*data12[6])

+ (filter[4]*data12[5])  +  (filter[5]*data12[4])

+ (filter[6]*data12[3])  +  (filter[7]*data12[2])

+ (filter[8]*data12[1])  +  (filter[9]*data12[0]));
```

```
}


else if (k==10)

{

v1y[k] = ((filter[0]*data[10]) +  (filter[1]*data[9])

+ (filter[2]*data[8])  +  (filter[3]*data[7])

+ (filter[4]*data[6])  +  (filter[5]*data[5])

+ (filter[6]*data[4])  +  (filter[7]*data[3])

+ (filter[8]*data[2])  +  (filter[9]*data[1])

+ (filter[10]*data[0]));


v2y[k] = ((filter[0]*data1[10]) +  (filter[1]*data1[9])

+ (filter[2]*data1[8])  +  (filter[3]*data1[7])

+ (filter[4]*data1[6])  +  (filter[5]*data1[5])

+ (filter[6]*data1[4])  +  (filter[7]*data1[3])

+ (filter[8]*data1[2])  +  (filter[9]*data1[1])

+ (filter[10]*data1[0]));


v3y[k] = ((filter[0]*data2[10]) +  (filter[1]*data2[9])

+ (filter[2]*data2[8])  +  (filter[3]*data2[7])
+ (filter[4]*data2[6])  +  (filter[5]*data2[5])

+ (filter[6]*data2[4])  +  (filter[7]*data2[3])

+ (filter[8]*data2[2])  +  (filter[9]*data2[1])

+ (filter[10]*data2[0]));


i1y[k] = ((filter[0]*data3[10]) +  (filter[1]*data3[9])

+ (filter[2]*data3[8])  +  (filter[3]*data3[7])

+ (filter[4]*data3[6])  +  (filter[5]*data3[5])

+ (filter[6]*data3[4])  +  (filter[7]*data3[3])

+ (filter[8]*data3[2])  +  (filter[9]*data3[1])

+ (filter[10]*data3[0]));


i2y[k] = ((filter[0]*data4[10]) +  (filter[1]*data4[9])

+ (filter[2]*data4[8])  +  (filter[3]*data4[7])

+ (filter[4]*data4[6])  +  (filter[5]*data4[5])

+ (filter[6]*data4[4])  +  (filter[7]*data4[3])

+ (filter[8]*data4[2])  +  (filter[9]*data4[1])

+ (filter[10]*data4[0]));


i3y[k] = ((filter[0]*data5[10]) +  (filter[1]*data5[9])
+ (filter[2]*data5[8])  +  (filter[3]*data5[7])

+ (filter[4]*data5[6])  +  (filter[5]*data5[5])

+ (filter[6]*data5[4])  +  (filter[7]*data5[3])

+ (filter[8]*data5[2])  +  (filter[9]*data5[1])

+ (filter[10]*data5[0]));


z1y[k] = ((filter[0]*data6[10]) +  (filter[1]*data6[9])

+ (filter[2]*data6[8])  +  (filter[3]*data6[7])

+ (filter[4]*data6[6])  +  (filter[5]*data6[5])

+ (filter[6]*data6[4])  +  (filter[7]*data6[3])

+ (filter[8]*data6[2])  +  (filter[9]*data6[1])

+ (filter[10]*data6[0]));


z2y[k] = ((filter[0]*data7[10]) +  (filter[1]*data7[9])

+ (filter[2]*data7[8])  +  (filter[3]*data7[7])

+ (filter[4]*data7[6])  +  (filter[5]*data7[5])

+ (filter[6]*data7[4])  +  (filter[7]*data7[3])

+ (filter[8]*data7[2])  +  (filter[9]*data7[1])

+ (filter[10]*data7[0]));
```

```
z3y[k] = ((filter[0]*data8[10]) +  (filter[1]*data8[9])

+ (filter[2]*data8[8])  +  (filter[3]*data8[7])

+ (filter[4]*data8[6])  +  (filter[5]*data8[5])

+ (filter[6]*data8[4])  +  (filter[7]*data8[3])

+ (filter[8]*data8[2])  +  (filter[9]*data8[1])

+ (filter[10]*data8[0]));


p1y[k] = ((filter[0]*data9[10]) +  (filter[1]*data9[9])

+ (filter[2]*data9[8])  +  (filter[3]*data9[7])

+ (filter[4]*data9[6])  +  (filter[5]*data9[5])

+ (filter[6]*data9[4])  +  (filter[7]*data9[3])

+ (filter[8]*data9[2])  +  (filter[9]*data9[1])

+ (filter[10]*data9[0]));


p2y[k] = ((filter[0]*data10[10]) +  (filter[1]*data10[9])

+ (filter[2]*data10[8])  +  (filter[3]*data10[7])

+ (filter[4]*data10[6])  +  (filter[5]*data10[5])

+ (filter[6]*data10[4])  +  (filter[7]*data10[3])

+ (filter[8]*data10[2])  +  (filter[9]*data10[1])

+ (filter[10]*data10[0]));
```

```
p3y[k] = ((filter[0]*data11[10]) +  (filter[1]*data11[9])

+ (filter[2]*data11[8])  +  (filter[3]*data11[7])

+ (filter[4]*data11[6])  +  (filter[5]*data11[5])

+ (filter[6]*data11[4])  +  (filter[7]*data11[3])

+ (filter[8]*data11[2])  +  (filter[9]*data11[1])

+ (filter[10]*data11[0]));


pgy[k] = ((filter[0]*data12[10]) +  (filter[1]*data12[9])

+ (filter[2]*data12[8])  +  (filter[3]*data12[7])

+ (filter[4]*data12[6])  +  (filter[5]*data12[5])

+ (filter[6]*data12[4])  +  (filter[7]*data12[3])

+ (filter[8]*data12[2])  +  (filter[9]*data12[1])

+ (filter[10]*data12[0]));

}


else if (k==11)

{

v1y[k] = ((filter[0]*data[11])   +  (filter[1]*data[10])

+ (filter[2]*data[9])  +  (filter[3]*data[8])
+ (filter[4]*data[7])  +  (filter[5]*data[6])

+ (filter[6]*data[5])  +  (filter[7]*data[4])

+ (filter[8]*data[3])  +  (filter[9]*data[2])

+ (filter[10]*data[1]) +  (filter[11]*data[0]));


v2y[k] = ((filter[0]*data1[11])   +  (filter[1]*data1[10])

+ (filter[2]*data1[9])  +  (filter[3]*data1[8])

+ (filter[4]*data1[7])  +  (filter[5]*data1[6])

+ (filter[6]*data1[5])  +  (filter[7]*data1[4])

+ (filter[8]*data1[3])  +  (filter[9]*data1[2])

+ (filter[10]*data1[1]) +  (filter[11]*data1[0]));


v3y[k] = ((filter[0]*data2[11])   +  (filter[1]*data2[10])

+ (filter[2]*data2[9])  +  (filter[3]*data2[8])

+ (filter[4]*data2[7])  +  (filter[5]*data2[6])

+ (filter[6]*data2[5])  +  (filter[7]*data2[4])

+ (filter[8]*data2[3])  +  (filter[9]*data2[2])

+ (filter[10]*data2[1]) +  (filter[11]*data2[0]));


i1y[k] = ((filter[0]*data3[11])   +  (filter[1]*data3[10])
+ (filter[2]*data3[9])  +  (filter[3]*data3[8])

+ (filter[4]*data3[7])  +  (filter[5]*data3[6])

+ (filter[6]*data3[5])  +  (filter[7]*data3[4])

+ (filter[8]*data3[3])  +  (filter[9]*data3[2])

+ (filter[10]*data3[1]) +  (filter[11]*data3[0]));


i2y[k] = ((filter[0]*data4[11])   +  (filter[1]*data4[10])

+ (filter[2]*data4[9])  +  (filter[3]*data4[8])

+ (filter[4]*data4[7])  +  (filter[5]*data4[6])

+ (filter[6]*data4[5])  +  (filter[7]*data4[4])

+ (filter[8]*data4[3])  +  (filter[9]*data4[2])

+ (filter[10]*data4[1]) +  (filter[11]*data4[0]));


i3y[k] = ((filter[0]*data5[11])   +  (filter[1]*data5[10])

+ (filter[2]*data5[9])  +  (filter[3]*data5[8])

+ (filter[4]*data5[7])  +  (filter[5]*data5[6])

+ (filter[6]*data5[5])  +  (filter[7]*data5[4])

+ (filter[8]*data5[3])  +  (filter[9]*data5[2])

+ (filter[10]*data5[1]) +  (filter[11]*data5[0]));
```

```
z1y[k] = ((filter[0]*data6[11])   +  (filter[1]*data6[10])

+ (filter[2]*data6[9])  +  (filter[3]*data6[8])

+ (filter[4]*data6[7])  +  (filter[5]*data6[6])

+ (filter[6]*data6[5])  +  (filter[7]*data6[4])

+ (filter[8]*data6[3])  +  (filter[9]*data6[2])

+ (filter[10]*data6[1]) +  (filter[11]*data6[0]));


z2y[k] = ((filter[0]*data7[11])   +  (filter[1]*data7[10])

+ (filter[2]*data7[9])  +  (filter[3]*data7[8])

+ (filter[4]*data7[7])  +  (filter[5]*data7[6])

+ (filter[6]*data7[5])  +  (filter[7]*data7[4])

+ (filter[8]*data7[3])  +  (filter[9]*data7[2])

+ (filter[10]*data7[1]) +  (filter[11]*data7[0]));


z3y[k] = ((filter[0]*data8[11])   +  (filter[1]*data8[10])

+ (filter[2]*data8[9])  +  (filter[3]*data8[8])

+ (filter[4]*data8[7])  +  (filter[5]*data8[6])

+ (filter[6]*data8[5])  +  (filter[7]*data8[4])

+ (filter[8]*data8[3])  +  (filter[9]*data8[2])

+ (filter[10]*data8[1]) +  (filter[11]*data8[0]));
```

```
p1y[k] = ((filter[0]*data9[11])   +  (filter[1]*data9[10])

+ (filter[2]*data9[9])  +  (filter[3]*data9[8])

+ (filter[4]*data9[7])  +  (filter[5]*data9[6])

+ (filter[6]*data9[5])  +  (filter[7]*data9[4])

+ (filter[8]*data9[3])  +  (filter[9]*data9[2])

+ (filter[10]*data9[1]) +  (filter[11]*data9[0]));


p2y[k] = ((filter[0]*data10[11])   +  (filter[1]*data10[10])

+ (filter[2]*data10[9])  +  (filter[3]*data10[8])

+ (filter[4]*data10[7])  +  (filter[5]*data10[6])

+ (filter[6]*data10[5])  +  (filter[7]*data10[4])

+ (filter[8]*data10[3])  +  (filter[9]*data10[2])

+ (filter[10]*data10[1]) +  (filter[11]*data10[0]));


p3y[k] = ((filter[0]*data11[11])   +  (filter[1]*data11[10])

+ (filter[2]*data11[9])  +  (filter[3]*data11[8])

+ (filter[4]*data11[7])  +  (filter[5]*data11[6])

+ (filter[6]*data11[5])  +  (filter[7]*data11[4])

+ (filter[8]*data11[3])  +  (filter[9]*data11[2])
+ (filter[10]*data11[1]) +  (filter[11]*data11[0]));


pgy[k] = ((filter[0]*data12[11])   +  (filter[1]*data12[10])

+ (filter[2]*data12[9])  +  (filter[3]*data12[8])

+ (filter[4]*data12[7])  +  (filter[5]*data12[6])

+ (filter[6]*data12[5])  +  (filter[7]*data12[4])

+ (filter[8]*data12[3])  +  (filter[9]*data12[2])

+ (filter[10]*data12[1]) +  (filter[11]*data12[0]));

}




Var =(v1y[k]*(cos(k*(PI/6))));

Varr +=((2.0/12.0)*(Var));

Vai =(v1y[k]*(sin(k*(PI/6))));

Vaii +=((-2.0/12.0)*(Vai));

Vbr =(v2y[k]*(cos(k*(PI/6))));

Vbrr +=((2.0/12.0)*(Vbr));

Vbi =(v2y[k]*(sin(k*(PI/6))));

Vbii +=((-2.0/12.0)*(Vbi));

Vcr =(v3y[k]*(cos(k*(PI/6))));
```

```
Vcrr +=((2.0/12.0)*(Vcr));

Vci =(v3y[k]*(sin(k*(PI/6))));

Vcii +=((-2.0/12.0)*(Vci));

Iar =(i1y[k]*(cos(k*(PI/6))));

Iarr +=((2.0/12.0)*(Iar));

Iai =(i1y[k]*(sin(k*(PI/6))));

Iaii +=((-2.0/12.0)*(Iai));

Ibr =(i2y[k]*(cos(k*(PI/6))));

Ibrr +=((2.0/12.0)*(Ibr));

Ibi =(i2y[k]*(sin(k*(PI/6))));

Ibii +=((-2.0/12.0)*(Ibi));

Icr =(i3y[k]*(cos(k*(PI/6))));

Icrr +=((2.0/12.0)*(Icr));

Ici =(i3y[k]*(sin(k*(PI/6))));

Icii +=((-2.0/12.0)*(Ici));

Zar =(z1y[k]*(cos(k*(PI/6))));

Zarr +=((2.0/12.0)*(Zar));

Zai =(z1y[k]*(sin(k*(PI/6))));

Zaii +=((-2.0/12.0)*(Zai));

Zbr =(z2y[k]*(cos(k*(PI/6))));
```

```
Zbrr +=((2.0/12.0)*(Zbr));

Zbi =(z2y[k]*(sin(k*(PI/6))));

Zbii +=((-2.0/12.0)*(Zbi));

Zcr =(z3y[k]*(cos(k*(PI/6))));

Zcrr +=((2.0/12.0)*(Zcr));

Zci =(z3y[k]*(sin(k*(PI/6))));

Zcii +=((-2.0/12.0)*(Zci));

Par =(p1y[k]*(cos(k*(PI/6))));

Parr +=((2.0/12.0)*(Par));

Pai =(p1y[k]*(sin(k*(PI/6))));

Paii +=((-2.0/12.0)*(Pai));

Pbr =(p2y[k]*(cos(k*(PI/6))));

Pbrr +=((2.0/12.0)*(Pbr));

Pbi =(p2y[k]*(sin(k*(PI/6))));

Pbii +=((-2.0/12.0)*(Pbi));

Pcr =(p3y[k]*(cos(k*(PI/6))));

Pcrr +=((2.0/12.0)*(Pcr));

Pci =(p3y[k]*(sin(k*(PI/6))));

Pcii +=((-2.0/12.0)*(Pci));

Pgr =(pgy[k]*(cos(k*(PI/6))));
```

```
Pgrr +=((2.0/12.0)*(Pgr));

Pgi =(pgy[k]*(sin(k*(PI/6))));

Pgii +=((-2.0/12.0)*(Pgi));


outfile_1 = fopen("v1y.m","a");

fprintf(outfile_1, "%le\n",v1y[k]);

fclose(outfile_1);


outfile_2 = fopen("v2y.m","a");

fprintf(outfile_2, "%le\n",v2y[k]);

fclose(outfile_2);


outfile_3 = fopen("v3y.m","a");

fprintf(outfile_3, "%le\n",v3y[k]);

fclose(outfile_3);


outfile_4 = fopen("i1y.m","a");

fprintf(outfile_4, "%le\n",i1y[k]);

fclose(outfile_4);
```

```
outfile_5 = fopen("i2y.m","a");

fprintf(outfile_5, "%le\n",i2y[k]);

fclose(outfile_5);


outfile_6 = fopen("i3y.m","a");

fprintf(outfile_6, "%le\n",i3y[k]);

fclose(outfile_6);


outfile_7 = fopen("z1y.m","a");

fprintf(outfile_7, "%le\n",z1y[k]);

fclose(outfile_7);


outfile_8 = fopen("z2y.m","a");

fprintf(outfile_8, "%le\n",z2y[k]);

fclose(outfile_8);


outfile_9 = fopen("z3y.m","a");

fprintf(outfile_9, "%le\n",z3y[k]);

fclose(outfile_9);
```

```
outfile_10 = fopen("p1y.m","a");

fprintf(outfile_10, "%le\n",p1y[k]);

fclose(outfile_10);


outfile_11 = fopen("p2y.m","a");

fprintf(outfile_11, "%le\n",p2y[k]);

fclose(outfile_11);


outfile_12 = fopen("p3y.m","a");

fprintf(outfile_12, "%le\n",p3y[k]);

fclose(outfile_12);


outfile_13 = fopen("pgy.m","a");

fprintf(outfile_13, "%le\n",pgy[k]);

fclose(outfile_13);


infile_1 = fopen("va.m","a");

fprintf(infile_1, "%le, %le\n",Varr,Vaii);

fclose(infile_1);
```

```
infile_2 = fopen("vb.m","a");

fprintf(infile_2, "%le, %le\n",Vbrr,Vbii);

fclose(infile_2);


infile_3 = fopen("vc.m","a");

fprintf(infile_3, "%le, %le\n",Vcrr,Vcii);

fclose(infile_3);


infile_4 = fopen("ia.m","a");

fprintf(infile_4, "%le, %le\n",Iarr,Iaii);

fclose(infile_4);


infile_5 = fopen("ib.m","a");

fprintf(infile_5, "%le, %le\n",Ibrr,Ibii);

fclose(infile_5);


infile_6 = fopen("ic.m","a");

fprintf(infile_6, "%le, %le\n",Icrr,Icii);

fclose(infile_6);
```

```
infile_7 = fopen("za.m","a");

fprintf(infile_7, "%le, %le\n",Zarr,Zaii);

fclose(infile_7);


infile_8 = fopen("zb.m","a");

fprintf(infile_8, "%le, %le\n",Zbrr,Zbii);

fclose(infile_8);


infile_9 = fopen("zc.m","a");

fprintf(infile_9, "%le, %le\n",Zcrr,Zcii);

fclose(infile_9);


infile_10 = fopen("pa.m","a");

fprintf(infile_10, "%le, %le\n",Parr,Paii);

fclose(infile_10);


infile_11 = fopen("pb.m","a");

fprintf(infile_11, "%le, %le\n",Pbrr,Pbii);

fclose(infile_11);
```

```
infile_12 = fopen("pc.m","a");

fprintf(infile_12, "%le, %le\n",Pcrr,Pcii);

fclose(infile_12);


infile_13 = fopen("pg.m","a");

fprintf(infile_13, "%le, %le\n",Pgrr,Pgii);

fclose(infile_13);


x_outfile = fopen("va_del.m","a");

fprintf(x_outfile, "%le, %le\n",Varr,Vaii);

fclose(x_outfile);


y_outfile = fopen("vb_del.m","a");

fprintf(y_outfile, "%le, %le\n",Vbrr,Vbii);

fclose(y_outfile);


z_outfile = fopen("vc_del.m","a");

fprintf(z_outfile, "%le, %le\n",Vcrr,Vcii);

fclose(z_outfile);

}
```

```
/*The recursion process will return here*/

Var_old  = Varr;

Vai_old  = Vaii;

Var_past = Varr;

Vai_past = Vaii;

Vbr_old  = Vbrr;

Vbi_old  = Vbii;

Vbr_past = Vbrr;

Vbi_past = Vbii;

Vcr_old  = Vcrr;

Vci_old  = Vcii;

Vcr_past = Vcrr;

Vci_past = Vcii;

Iar_old  = Iarr;

Iai_old  = Iaii;
```

```
Ibr_old  = Ibrr;

Ibi_old  = Ibii;

Icr_old  = Icrr;

Ici_old  = Icii;

Zar_old  = Zarr;

Zai_old  = Zaii;

Zbr_old  = Zbrr;

Zbi_old  = Zbii;

Zcr_old  = Zcrr;

Zci_old  = Zcii;

Par_old  = Parr;

Pai_old  = Paii;

Pbr_old  = Pbrr;

Pbi_old  = Pbii;

Pcr_old  = Pcrr;

Pci_old  = Pcii;

Pgr_old  = Pgrr;

Pgi_old  = Pgii;


for ( k = N; k < Q; k++)
```

```
{

v1y[k] = ((filter[0]*data[k]) + (filter[1]*data[k-1])

+ (filter[2]*data[k-2])  + (filter[3]*data[k-3])

+ (filter[4]*data[k-4])  + (filter[5]*data[k-5])

+ (filter[6]*data[k-6])  + (filter[7]*data[k-7])

+ (filter[8]*data[k-8])  +(filter[9]*data[k-9])

+ (filter[10]*data[k-10])+(filter[11]*data[k-11]));


v2y[k] = ((filter[0]*data1[k]) + (filter[1]*data1[k-1])

+ (filter[2]*data1[k-2])  + (filter[3]*data1[k-3])

+ (filter[4]*data1[k-4])  + (filter[5]*data1[k-5])

+ (filter[6]*data1[k-6])  + (filter[7]*data1[k-7])

+ (filter[8]*data1[k-8])  +(filter[9]*data1[k-9])

+ (filter[10]*data1[k-10])+(filter[11]*data1[k-11]));


v3y[k] = ((filter[0]*data2[k]) + (filter[1]*data2[k-1])

+ (filter[2]*data2[k-2])  + (filter[3]*data2[k-3])

+ (filter[4]*data2[k-4])  + (filter[5]*data2[k-5])

+ (filter[6]*data2[k-6])  + (filter[7]*data2[k-7])

+ (filter[8]*data2[k-8])  +(filter[9]*data2[k-9])
+ (filter[10]*data2[k-10])+(filter[11]*data2[k-11]));


i1y[k] = ((filter[0]*data3[k]) + (filter[1]*data3[k-1])

+ (filter[2]*data3[k-2])  + (filter[3]*data3[k-3])

+ (filter[4]*data3[k-4])  + (filter[5]*data3[k-5])

+ (filter[6]*data3[k-6])  + (filter[7]*data3[k-7])

+ (filter[8]*data3[k-8])  +(filter[9]*data3[k-9])

+ (filter[10]*data3[k-10])+(filter[11]*data3[k-11]));


i2y[k] = ((filter[0]*data4[k]) + (filter[1]*data4[k-1])

+ (filter[2]*data4[k-2])  + (filter[3]*data4[k-3])

+ (filter[4]*data4[k-4])  + (filter[5]*data4[k-5])

+ (filter[6]*data4[k-6])  + (filter[7]*data4[k-7])

+ (filter[8]*data4[k-8])  +(filter[9]*data4[k-9])

+ (filter[10]*data4[k-10])+(filter[11]*data4[k-11]));


i3y[k] = ((filter[0]*data5[k]) + (filter[1]*data5[k-1])

+ (filter[2]*data5[k-2])  + (filter[3]*data5[k-3])

+ (filter[4]*data5[k-4])  + (filter[5]*data5[k-5])

+ (filter[6]*data5[k-6])  + (filter[7]*data5[k-7])
+ (filter[8]*data5[k-8])  +(filter[9]*data5[k-9])

+ (filter[10]*data5[k-10])+(filter[11]*data5[k-11]));


z1y[k] = ((filter[0]*data6[k]) + (filter[1]*data6[k-1])

+ (filter[2]*data6[k-2])  + (filter[3]*data6[k-3])

+ (filter[4]*data6[k-4])  + (filter[5]*data6[k-5])

+ (filter[6]*data6[k-6])  + (filter[7]*data6[k-7])

+ (filter[8]*data6[k-8])  +(filter[9]*data6[k-9])

+ (filter[10]*data6[k-10])+(filter[11]*data6[k-11]));



z2y[k] = ((filter[0]*data7[k]) + (filter[1]*data7[k-1])

+ (filter[2]*data7[k-2])  + (filter[3]*data7[k-3])

+ (filter[4]*data7[k-4])  + (filter[5]*data7[k-5])

+ (filter[6]*data7[k-6])  + (filter[7]*data7[k-7])

+ (filter[8]*data7[k-8])  +(filter[9]*data7[k-9])

+ (filter[10]*data7[k-10])+(filter[11]*data7[k-11]));


z3y[k] = ((filter[0]*data8[k]) + (filter[1]*data8[k-1])

+ (filter[2]*data8[k-2])  + (filter[3]*data8[k-3])
+ (filter[4]*data8[k-4])  + (filter[5]*data8[k-5])

+ (filter[6]*data8[k-6])  + (filter[7]*data8[k-7])

+ (filter[8]*data8[k-8])  +(filter[9]*data8[k-9])

+ (filter[10]*data8[k-10])+(filter[11]*data8[k-11]));


p1y[k] = ((filter[0]*data9[k]) + (filter[1]*data9[k-1])

+ (filter[2]*data9[k-2])  + (filter[3]*data9[k-3])

+ (filter[4]*data9[k-4])  + (filter[5]*data9[k-5])

+ (filter[6]*data9[k-6])  + (filter[7]*data9[k-7])

+ (filter[8]*data9[k-8])  +(filter[9]*data9[k-9])

+ (filter[10]*data9[k-10])+(filter[11]*data9[k-11]));


p2y[k] = ((filter[0]*data10[k]) + (filter[1]*data10[k-1])

+ (filter[2]*data10[k-2])  + (filter[3]*data10[k-3])

+ (filter[4]*data10[k-4])  + (filter[5]*data10[k-5])

+ (filter[6]*data10[k-6])  + (filter[7]*data10[k-7])

+ (filter[8]*data10[k-8])  +(filter[9]*data10[k-9])

+ (filter[10]*data10[k-10])+(filter[11]*data10[k-11]));


p3y[k] = ((filter[0]*data11[k]) + (filter[1]*data11[k-1])
+ (filter[2]*data11[k-2])  + (filter[3]*data11[k-3])

+ (filter[4]*data11[k-4])  + (filter[5]*data11[k-5])

+ (filter[6]*data11[k-6])  + (filter[7]*data11[k-7])

+ (filter[8]*data11[k-8])  +(filter[9]*data11[k-9])

+ (filter[10]*data11[k-10])+(filter[11]*data11[k-11]));


pgy[k] = ((filter[0]*data12[k]) + (filter[1]*data12[k-1])

+ (filter[2]*data12[k-2])  + (filter[3]*data12[k-3])

+ (filter[4]*data12[k-4])  + (filter[5]*data12[k-5])

+ (filter[6]*data12[k-6])  + (filter[7]*data12[k-7])

+ (filter[8]*data12[k-8])  +(filter[9]*data12[k-9])

+ (filter[10]*data12[k-10])+(filter[11]*data12[k-11]));



Var_new = Var_old + ((2.0/12.0)*((v1y[k]-v1y[k-N])*(cos((k)*(PI/6)))));

Vai_new = Vai_old - ((2.0/12.0)*((v1y[k]-v1y[k-N])*(sin((k)*(PI/6)))));

Var_del = Var_past +  ((2.0/12.0)*((v1y[k]-v1y[k-N])*(cos((k-N)*(PI/6)))));

Vai_del = Vai_past - ((2.0/12.0)*((v1y[k]-v1y[k-N])*(sin((k-N)*(PI/6)))));

Vbr_new = Vbr_old +  ((2.0/12.0)*((v2y[k]-v2y[k-N])*(cos((k)*(PI/6)))));

Vbi_new = Vbi_old - ((2.0/12.0)*((v2y[k]-v2y[k-N])*(sin((k)*(PI/6)))));
```

```
Vbr_del = Vbr_past + ((2.0/12.0)*((v2y[k]-v2y[k-N])*(cos((k-N)*(PI/6)))));

Vbi_del = Vbi_past - ((2.0/12.0)*((v2y[k]-v2y[k-N])*(sin((k-N)*(PI/6)))));

Vcr_new = Vcr_old +  ((2.0/12.0)*((v3y[k]-v3y[k-N])*(cos((k)*(PI/6)))));

Vci_new = Vci_old - ((2.0/12.0)*((v3y[k]-v3y[k-N])*(sin((k)*(PI/6)))));

Vcr_del = Vcr_past +  ((2.0/12.0)*((v3y[k]-v3y[k-N])*(cos((k-N)*(PI/6)))));

Vci_del = Vci_past - ((2.0/12.0)*((v3y[k]-v3y[k-N])*(sin((k-N)*(PI/6)))));

Iar_new = Iar_old +  ((2.0/12.0)*((i1y[k]-i1y[k-N])*(cos((k)*(PI/6)))));

Iai_new = Iai_old - ((2.0/12.0)*((i1y[k]-i1y[k-N])*(sin((k)*(PI/6)))));

Ibr_new = Ibr_old +  ((2.0/12.0)*((i2y[k]-i2y[k-N])*(cos((k)*(PI/6)))));

Ibi_new = Ibi_old - ((2.0/12.0)*((i2y[k]-i2y[k-N])*(sin((k)*(PI/6)))));

Icr_new = Icr_old +  ((2.0/12.0)*((i3y[k]-i3y[k-N])*(cos((k)*(PI/6)))));

Ici_new = Ici_old - ((2.0/12.0)*((i3y[k]-i3y[k-N])*(sin((k)*(PI/6)))));

Zar_new = Zar_old +  ((2.0/12.0)*((z1y[k]-z1y[k-N])*(cos((k)*(PI/6)))));

Zai_new = Zai_old - ((2.0/12.0)*((z1y[k]-z1y[k-N])*(sin((k)*(PI/6)))));

Zbr_new = Zbr_old +  ((2.0/12.0)*((z2y[k]-z2y[k-N])*(cos((k)*(PI/6)))));

Zbi_new = Zbi_old - ((2.0/12.0)*((z2y[k]-z2y[k-N])*(sin((k)*(PI/6)))));

Zcr_new = Zcr_old +  ((2.0/12.0)*((z3y[k]-z3y[k-N])*(cos((k)*(PI/6)))));

Zci_new = Zci_old - ((2.0/12.0)*((z3y[k]-z3y[k-N])*(sin((k)*(PI/6)))));

Par_new = Par_old +  ((2.0/12.0)*((p1y[k]-p1y[k-N])*(cos((k)*(PI/6)))));

Pai_new = Pai_old - ((2.0/12.0)*((p1y[k]-p1y[k-N])*(sin((k)*(PI/6)))));
```

```
Pbr_new = Pbr_old +  ((2.0/12.0)*((p2y[k]-p2y[k-N])*(cos((k)*(PI/6)))));

Pbi_new = Pbi_old - ((2.0/12.0)*((p2y[k]-p2y[k-N])*(sin((k)*(PI/6)))));

Pcr_new = Pcr_old +  ((2.0/12.0)*((p3y[k]-p3y[k-N])*(cos((k)*(PI/6)))));

Pci_new = Pci_old - ((2.0/12.0)*((p3y[k]-p3y[k-N])*(sin((k)*(PI/6)))));

Pgr_new = Pgr_old +  ((2.0/12.0)*((pgy[k]-pgy[k-N])*(cos((k)*(PI/6)))));

Pgi_new = Pgi_old - ((2.0/12.0)*((pgy[k]-pgy[k-N])*(sin((k)*(PI/6)))));


 /* The end of Phasor Computation */


outfile_1 = fopen("v1y.m","a");

fprintf(outfile_1, "%le\n",v1y[k]);

fclose(outfile_1);


outfile_2 = fopen("v2y.m","a");

fprintf(outfile_2, "%le\n",v2y[k]);

fclose(outfile_2);


outfile_3 = fopen("v3y.m","a");

fprintf(outfile_3, "%le\n",v3y[k]);

fclose(outfile_3);
```

```
outfile_4 = fopen("i1y.m","a");

fprintf(outfile_4, "%le\n",i1y[k]);

fclose(outfile_4);


outfile_5 = fopen("i2y.m","a");

fprintf(outfile_5, "%le\n",i2y[k]);

fclose(outfile_5);


outfile_6 = fopen("i3y.m","a");

fprintf(outfile_6, "%le\n",i3y[k]);

fclose(outfile_6);


outfile_7 = fopen("z1y.m","a");

fprintf(outfile_7, "%le\n",z1y[k]);

fclose(outfile_7);


outfile_8 = fopen("z2y.m","a");

fprintf(outfile_8, "%le\n",z2y[k]);

fclose(outfile_8);
```

```
outfile_9 = fopen("z3y.m","a");

fprintf(outfile_9, "%le\n",z3y[k]);

fclose(outfile_9);


outfile_10 = fopen("p1y.m","a");

fprintf(outfile_10, "%le\n",p1y[k]);

fclose(outfile_10);


outfile_11 = fopen("p2y.m","a");

fprintf(outfile_11, "%le\n",p2y[k]);

fclose(outfile_11);


outfile_12 = fopen("p3y.m","a");

fprintf(outfile_12, "%le\n",p3y[k]);

fclose(outfile_12);


outfile_13 = fopen("pgy.m","a");

fprintf(outfile_13, "%le\n",pgy[k]);

fclose(outfile_13);
```

```
infile_1 = fopen("va.m","a");

fprintf(infile_1, "%le, %le\n",Var_new,Vai_new);

fclose(infile_1);


infile_2 = fopen("vb.m","a");

fprintf(infile_2, "%le, %le\n",Vbr_new,Vbi_new);

fclose(infile_2);


infile_3 = fopen("vc.m","a");

fprintf(infile_3, "%le, %le\n",Vcr_new,Vci_new);

fclose(infile_3);


infile_4 = fopen("ia.m","a");

fprintf(infile_4, "%le, %le\n",Iar_new,Iai_new);

fclose(infile_4);


infile_5 = fopen("ib.m","a");

fprintf(infile_5, "%le, %le\n",Ibr_new,Ibi_new);

fclose(infile_5);
```

```
infile_6 = fopen("ic.m","a");

fprintf(infile_6, "%le, %le\n",Icr_new,Ici_new);

fclose(infile_6);


infile_7 = fopen("za.m","a");

fprintf(infile_7, "%le, %le\n",Zar_new,Zai_new);

fclose(infile_7);


infile_8 = fopen("zb.m","a");

fprintf(infile_8, "%le, %le\n",Zbr_new,Zbi_new);

fclose(infile_8);


infile_9 = fopen("zc.m","a");

fprintf(infile_9, "%le, %le\n",Zcr_new,Zci_new);

fclose(infile_9);


infile_10 = fopen("pa.m","a");

fprintf(infile_10, "%le, %le\n",Par_new,Pai_new);

fclose(infile_10);
```

```
infile_11 = fopen("pb.m","a");

fprintf(infile_11, "%le, %le\n",Pbr_new,Pbi_new);

fclose(infile_11);


infile_12 = fopen("pc.m","a");

fprintf(infile_12, "%le, %le\n",Pcr_new,Pci_new);

fclose(infile_12);


infile_13 = fopen("pg.m","a");

fprintf(infile_13, "%le, %le\n",Pgr_new,Pgi_new);

fclose(infile_13);




x_outfile = fopen("va_del.m","a");

fprintf(x_outfile, "%le, %le\n",Var_del,Vai_del);

fclose(x_outfile);


y_outfile = fopen("vb_del.m","a");

fprintf(y_outfile, "%le, %le\n",Vbr_del,Vbi_del);
```

```
fclose(y_outfile);


z_outfile = fopen("vc_del.m","a");

fprintf(z_outfile, "%le, %le\n",Vcr_del,Vci_del);

fclose(z_outfile);


Var_old  = Var_new;

Vai_old  = Vai_new;

Var_past = Var_del;

Vai_past = Vai_del;

Vbr_old  = Vbr_new;

Vbi_old  = Vbi_new;

Vbr_past = Vbr_del;

Vbi_past = Vbi_del;

Vcr_old  = Vcr_new;

Vci_old  = Vci_new;

Vcr_past = Vcr_del;

Vci_past = Vci_del;

Iar_old  = Iar_new;

Iai_old  = Iai_new;
```

```
Ibr_old  = Ibr_new;

Ibi_old  = Ibi_new;

Icr_old  = Icr_new;

Ici_old  = Ici_new;

Zar_old  = Zar_new;

Zai_old  = Zai_new;

Zbr_old  = Zbr_new;

Zbi_old  = Zbi_new;

Zcr_old  = Zcr_new;

Zci_old  = Zci_new;

Par_old  = Par_new;

Pai_old  = Pai_new;

Pbr_old  = Pbr_new;

Pbi_old  = Pbi_new;

Pcr_old  = Pcr_new;

Pci_old  = Pci_new;

Pgr_old  = Pgr_new;

Pgi_old  = Pgi_new;

}

}
```

## D.3 AUXILIARY PROGRAM B

The purpose of the auxiliary program B was to calculate the integrated valued of the negative sequence current multiplied by time at each new sample for the negative sequence relay.

```
#include <stdio.h>

#include <math.h>

#include <complex.h>

#include <iostream.h>

#define    PI    3.14159

#define    M     96

#define    N     12




void main()

{

int       i,k;

float     Iar_new,Iai_new,Ibr_new,Ibi_new,Icr_new,Ici_new;

complex   ia_m[96],ib_m[96],ic_m[96];

double    x_ia,y_ia,x_ib,y_ib,x_ic,y_ic;
```

```
char      choice;

FILE      *fp,*fp1,*fp2;




if ((fp = fopen("ia.m","r")) == NULL)

printf("Data file ia.m not found\n");


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

{

fscanf(fp,"%le,%le\n",&x_ia,&y_ia);

ia_m[i]=complex(x_ia,y_ia);

}

fclose(fp);


if ((fp1 = fopen("ib.m","r")) == NULL)

printf("Data file ib.m not found\n");


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

{

fscanf(fp1,"%le,%le\n",&x_ib,&y_ib);
```

```
ib_m[i]=complex(x_ib,y_ib);

}

fclose(fp1);


if ((fp2 = fopen("ic.m","r")) == NULL)

printf("Data file ic.m not found\n");


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

{

fscanf(fp2,"%le,%le\n",&x_ic,&y_ic);

ic_m[i]=complex(x_ic,y_ic);

}

fclose(fp2);


for ( k = N; k < M; k++)

{

Iar_new = real(ia_m[k]);

Iai_new = imag(ia_m[k]);

Ibr_new = real(ib_m[k]);

Ibi_new = imag(ib_m[k]);
```

```
Icr_new = real(ic_m[k]);

Ici_new = imag(ic_m[k]);


char dial_set;

char time_set(float, float, float, float, float, float,int);


dial_set  = time_set(Iar_new,Iai_new,Ibr_new,Ibi_new,

Icr_new,Ici_new,k);

}

}


char time_set(float Iar, float Iai, float Ibr, float Ibi,

float Icr, float Ici, int j)

{

int     i;

float   V_BASE,S_BASE,I_BASE,Ia2_pu;

float   neg_volt,Tdial;

char    value,t,b;

complex a(-0.5,0.866);

complex Ia2;
complex Ia_max(Iar,Iai),Ib_max(Ibr,Ibi),Ic_max(Icr,Ici);

FILE    *s_file1;


V_BASE = 2309.401;

S_BASE = 15E+6;

I_BASE = (S_BASE/((sqrt(3))*(V_BASE)));




Ia2 = (0.3333)*(Ia_max+(pow(a,2)*(Ib_max))+(a*(Ic_max)));

Ia2_pu = abs(Ia2/I_BASE);


Tdial = ((pow(Ia2_pu,2))*(j/96.0));

printf("This is Tdial %f\n",Tdial);

s_file1  = fopen("tdial.m","a");

fprintf(s_file1, "%f\n",Tdial);

fclose(s_file1);


value='b';

return (value);

}
```

## D.4 AUXILIARY PROGRAM C

The purpose of the auxiliary program C was to calculate the accumulated integrated valued of the negative sequence current versus time.

```
#include <stdio.h>

#include <math.h>

#include <complex.h>

#include <iostream.h>

#define    PI    3.14159

#define    B     84

#define    N     12




void main()

{

int       I;

float     Tdial[84],dial_sum = 0;

FILE      *fp,*fp1;
if ((fp = fopen("tdial.m","r")) == NULL)

printf("Data file tdial.m not found\n");



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

{

fscanf(fp,"%e\n",&Tdial[i]);

}

fclose(fp);



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

{

dial_sum += Tdial[i];

fp1  = fopen("tdial_su.m","a");

fprintf(fp1, "%f\n",dial_sum);

fclose(fp1);

}

}
```

## D.5 AUXILIARY PROGRAM D

Auxiliary program D calculated the positive sequence voltage at each new sample for the frequency relay.

```
#include <stdio.h>

#include <math.h>

#include <complex.h>

#include <iostream.h>

#define    PI    3.14159

#define    M     96

#define    N     12




void main()

{

int             i,k;

float         Var_new,Vai_new,Vbr_new,Vbi_new,Vcr_new,Vci_new;

float         Var_del,Vai_del,Vbr_del,Vbi_del,Vcr_del,Vci_del;

complex   va_m[96],vb_m[96],vc_m[96];

complex   va_d[96],vb_d[96],vc_d[96];
```

```
double     x,y,x1,y1,x2,y2;

double     xa_del,ya_del,xb_del,yb_del,xc_del,yc_del;

char         choice;

FILE        *fp,*fp1,*fp2;

FILE        *fp3,*fp4,*fp5;


if ((fp = fopen("va.m","r")) == NULL)

printf("Data file va.m not found\n");


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

{

fscanf(fp,"%le,%le\n",&x,&y);

va_m[i]=complex(x,y);

}

fclose(fp);



 if ((fp1 = fopen("vb.m","r")) == NULL)

printf("Data file vb.m not found\n");
```

```
for (i=0;i< M;i++)

{

fscanf(fp1,"%le,%le\n",&x1,&y1);

vb_m[i]=complex(x1,y1);

}

fclose(fp1);





if ((fp2 = fopen("vc.m","r")) == NULL)

printf("Data file vc.m not found\n");


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

{

fscanf(fp2,"%le,%le\n",&x2,&y2);

vc_m[i]=complex(x2,y2);

}

fclose(fp2);


if ((fp3 = fopen("va_del.m","r")) == NULL)
```

```
printf("Data file va_del.m not found\n");


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

{

fscanf(fp3,"%le,%le\n",&xa_del,&ya_del);

va_d[i]=complex(xa_del,ya_del);

}

fclose(fp3);



if ((fp4 = fopen("vb_del.m","r")) == NULL)

printf("Data file vb_del.m not found\n");



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

{

fscanf(fp4,"%le,%le\n",&xb_del,&yb_del);

vb_d[i]=complex(xb_del,yb_del);

}

fclose(fp4);
```

```
if ((fp5 = fopen("vc_del.m","r")) == NULL)

printf("Data file vc_del.m not found\n");


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

{

fscanf(fp5,"%le,%le\n",&xc_del,&yc_del);

vc_d[i]=complex(xc_del,yc_del);

}

fclose(fp5);


for ( k = 0; k < M; k++)

{

Var_new = real(va_m[k]);

Vai_new = imag(va_m[k]);

Vbr_new = real(vb_m[k]);

Vbi_new = imag(vb_m[k]);

Vcr_new = real(vc_m[k]);

Vci_new = imag(vc_m[k]);

Var_del = real(va_d[k]);
Vai_del = imag(va_d[k]);

Vbr_del = real(vb_d[k]);

Vbi_del = imag(vb_d[k]);

Vcr_del = real(vc_d[k]);

Vci_del = imag(vc_d[k]);


char relay5;

char freq(float, float, float, float, float, float,

float, float, float, float, float, float, int);


relay5  = freq(Var_new,Vai_new,Vbr_new,Vbi_new,

Vcr_new,Vci_new,Var_del,Vai_del,

Vbr_del,Vbi_del,Vcr_del,Vci_del,k);

}

}


char freq(float var_new, float vai_new, float vbr_new,

float vbi_new, float vcr_new, float vci_new,

float var_del, float vai_del, float vbr_del,

float vbi_del, float vcr_del, float vci_del, int j)
```

```
{

int     i;

complex a(-0.5,0.866);

complex Va1,Va1_del;

char    value,t,b,z;

complex Va_max(var_new,vai_new),Vb_max(vbr_new,vbi_new);

complex Vc_max(vcr_new,vci_new),Va_del(var_del,vai_del);

complex Vb_del(vbr_del,vbi_del),Vc_del(vcr_del,vci_del);

 FILE    *u_file;


Va1      =   (0.3333)*((Va_max)+(a*(Vb_max))+

(pow(a,2)*(Vc_max)));


Va1_del  =  (0.3333)*((Va_del)+(a*(Vb_del))+

(pow(a,2)*(Vc_del)));


u_file = fopen("va1.m","a");

fprintf(u_file,"%f %f\n",real(Va1), imag(Va1) );

fclose(u_file);
value = 'b';

return (value);

}
```

## D.6 AUXILIARY PROGRAM E

Auxiliary program E computed the positive sequence voltage angle at each new sample.

```
x=[0.000000 0.000000

-0.224260 -0.198150

2.461992 0.044931

-1.436295 4.087055

-12.065984 9.399324

-2.080023 -98.172441

51.327437 -264.780450

214.199019 -332.372507

379.842856 -255.881695

431.912729 -81.501870

338.013116 74.154234
```

```
159.508326 109.258539

13.697067 0.819597

-3.820159 -179.670648

115.410834 -314.592531

300.960065 -318.881582

433.773109 -190.017157

406.594139 98.199015

207.824189 373.546804

-133.171120 406.628220

-392.969834 175.143766

-393.575954 -173.174767

-136.299402 -407.639693

210.297527 -374.807400

418.805615 -96.271942

352.745015 245.205535

55.500071 425.910042

-278.049205 327.324083

-429.178015 14.244580

-298.984639 -308.085417

27.188036 -428.413699
```

```
335.423603 -267.844754

423.717500 68.228836

234.270728 359.533689

-108.726981 415.091684

-380.350683 198.551896

-402.619652 -148.085259

-160.967429 -397.610770

186.150107 -386.427763

411.179725 -121.954647

366.660012 222.419834

81.784878 420.948272

-256.637982 343.511580

-426.786435 40.923230

-317.163025 -288.452817

0.347578 -428.682415

317.578519 -287.896006

426.595268 41.580909

255.956861 343.765487

-82.428349 420.566798

-366.737488 221.679544
```

```
-410.625394 -122.487675

-185.339244 -386.309513

161.400001 -396.874559

402.290474 -147.291477

379.448177 198.816465

107.886525 414.551254

-234.362677 358.521951

-422.952472 67.500541

-334.271006 -267.725556

-26.496199 -427.418944

298.597823 -306.920172

427.923078 14.752379

276.731547 326.707781

-55.842391 424.469511

-351.772748 243.992235

-417.079266 -96.397585

-208.995949 -373.557899

136.047488 -405.814195

391.871394 -172.058708

390.790848 174.442681
```

```
133.543353 406.556901

-211.206438 372.155112

-417.469712 93.809966

-350.071226 -245.990406

-53.218561 -424.498108

278.480629 -324.733522

427.589060 -12.128838

296.392975 308.389846

-29.064842 426.723844

-335.433768 265.324437

-421.901500 -69.968452

-231.808544 -359.351840

110.206017 -413.155679

379.929042 -196.139820

400.576664 149.418325

158.653286 396.986594

-187.240809 384.297519

-410.363591 119.712558

-364.468473 -223.309324

-79.680354 -419.920835
```

```
257.291893 -341.257201

425.571434 -38.914073

314.876174 288.889864

-2.209388 427.284370

-317.808815 285.587447];


m =96;





for i = 1:1:m


if x(i,1) == 0

P(i) = (180/3.14)* 0;


else

P(i) = (180/3.14)*atan(x(i,2)/x(i,1));

end


infile = fopen('va1_ang.m','a');
```

```
fprintf(infile, '%f\n',P(i));

fclose(infile);


outfile = fopen('va1_del.m','a');

fprintf(outfile, '%f\n',P(i));

fclose(outfile);


end


i=1:1:96;

plot(i,P(i));
```

## D.7 AUXILIARY PROGRAM F

Auxiliary program F calculated the high frequency voltages for the self-excitation relay.

```
#include <stdio.h>

#include <math.h>

#include <iostream.h>

#define    PI    3.14159
```

```
#define    M     96

#define    N     12

#define    O     12

#define    P     36

#define    Q     3

#include "spaincl.h"

#include "spfftc.h"

#include "spfirl.h"

#include "spgain.h"

#include "spplot.h"




void main()

{


float data[96],data1[96],data2[96];

float Va_freq,Vb_freq,Vc_freq;

long neg_i1 = -1;

long pos_i1 =  1;

long pos_i4 =  4;
long i, m;

complex fft[96];

complex fft1[96];

complex fft2[96];

FILE *fp,*fp1,*fp2;

FILE *outfile12,*outfile13,*outfile14;

FILE *infile1,*infile2,*infile3;




if ((fp = fopen("vaf.m","r")) == NULL)

printf("Data file vaf.m not found\n");


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

{

fscanf(fp,"%e\n",&data[i]);

}

fclose(fp);


if ((fp1 = fopen("vbf.m","r")) == NULL)

printf("Data file vbf.m not found\n");
```

```
for (i=0;i< M;i++)

{

fscanf(fp1,"%e\n",&data1[i]);

}

fclose(fp1);




if ((fp2 = fopen("vcf.m","r")) == NULL)

printf("Data file vcf.m not found\n");


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

{

fscanf(fp2,"%e\n",&data2[i]);

}

fclose(fp2);


complex x1[96] ={ { data[0], 0.0 },{data[1], 0.0 },

{data[2], 0.0 },{data[3], 0.0 },

{ data[4], 0.0 },{data[5], 0.0 },
{ data[6], 0.0 },{data[7], 0.0 },

{ data[8], 0.0 },{data[9], 0.0 },

{ data[10], 0.0 },{data[11], 0.0 },

{ data[12], 0.0 },{data[13], 0.0 },

{ data[14], 0.0 },{data[15], 0.0 },

{ data[16], 0.0 },{data[17], 0.0 },

{ data[17], 0.0 },{data[18], 0.0 },

{ data[20], 0.0 },{data[21], 0.0 },

{ data[22], 0.0 },{data[23], 0.0 },

{ data[24], 0.0 },{data[25], 0.0 },

{ data[26], 0.0 },{data[27], 0.0 },

{ data[28], 0.0 },{data[29], 0.0 },

{ data[30], 0.0 },{data[31], 0.0 },

{ data[32], 0.0 },{data[33], 0.0 },

{ data[34], 0.0 },{data[35], 0.0 },

{ data[36], 0.0 },{data[37], 0.0 },

{ data[38], 0.0 },{data[39], 0.0 },

{ data[40], 0.0 },{data[41], 0.0 },

{ data[42], 0.0 },{data[43], 0.0 },

{ data[44], 0.0 },{data[45], 0.0 },
{ data[46], 0.0 },{data[47], 0.0 },

{ data[48], 0.0 },{data[49], 0.0 },

{ data[50], 0.0 },{data[51], 0.0 },

{ data[52], 0.0 },{data[53], 0.0 },

{ data[54], 0.0 },{data[55], 0.0 },

{ data[56], 0.0 },{data[57], 0.0 },

{ data[58], 0.0 },{data[59], 0.0 },

{ data[60], 0.0 },{data[61], 0.0 },

{ data[62], 0.0 },{data[63], 0.0 },

{ data[64], 0.0 },{data[65], 0.0 },

{ data[66], 0.0 },{data[67], 0.0 },

{ data[68], 0.0 },{data[69], 0.0 },

{ data[70], 0.0 },{data[71], 0.0 },

{ data[72], 0.0 },{data[73], 0.0 },

{ data[74], 0.0 },{data[75], 0.0 },

{ data[76], 0.0 },{data[77], 0.0 },

{ data[78], 0.0 },{data[79], 0.0 },

{ data[80], 0.0 },{data[81], 0.0 },

{ data[82], 0.0 },{data[83], 0.0 },

{ data[84], 0.0 },{data[85], 0.0 },
{ data[86], 0.0 },{data[87], 0.0 },

{ data[88], 0.0 },{data[89], 0.0 },

{ data[90], 0.0 },{data[91], 0.0 },

{ data[92], 0.0 },{data[93], 0.0 },

{ data[94], 0.0 },{data[95], 0.0 } };


complex x2[96] ={ { data1[0], 0.0 },{data1[1], 0.0 },

{data1[2], 0.0 },{data1[3], 0.0 },

{ data1[4], 0.0 },{data1[5], 0.0 },

{ data1[6], 0.0 },{data1[7], 0.0 },

{ data1[8], 0.0 },{data1[9], 0.0 },

{ data1[10], 0.0 },{data1[11], 0.0 },

{ data1[12], 0.0 },{data1[13], 0.0 },

{ data1[14], 0.0 },{data1[15], 0.0 },

{ data1[16], 0.0 },{data1[17], 0.0 },

{ data1[17], 0.0 },{data1[18], 0.0 },

{ data1[20], 0.0 },{data1[21], 0.0 },

{ data1[22], 0.0 },{data1[23], 0.0 },

{ data1[24], 0.0 },{data1[25], 0.0 },

{ data1[26], 0.0 },{data1[27], 0.0 },
{ data1[28], 0.0 },{data1[29], 0.0 },

{ data1[30], 0.0 },{data1[31], 0.0 },

{ data1[32], 0.0 },{data1[33], 0.0 },

{ data1[34], 0.0 },{data1[35], 0.0 },

{ data1[36], 0.0 },{data1[37], 0.0 },

{ data1[38], 0.0 },{data1[39], 0.0 },

{ data1[40], 0.0 },{data1[41], 0.0 },

{ data1[42], 0.0 },{data1[43], 0.0 },

{ data1[44], 0.0 },{data1[45], 0.0 },

{ data1[46], 0.0 },{data1[47], 0.0 },

{ data1[48], 0.0 },{data1[49], 0.0 },

{ data1[50], 0.0 },{data1[51], 0.0 },

{ data1[52], 0.0 },{data1[53], 0.0 },

{ data1[54], 0.0 },{data1[55], 0.0 },

{ data1[56], 0.0 },{data1[57], 0.0 },

{ data1[58], 0.0 },{data1[59], 0.0 },

{ data1[60], 0.0 },{data1[61], 0.0 },

{ data1[62], 0.0 },{data1[63], 0.0 },

{ data1[64], 0.0 },{data1[65], 0.0 },

{ data1[66], 0.0 },{data1[67], 0.0 },
{ data1[68], 0.0 },{data1[69], 0.0 },

{ data1[70], 0.0 },{data1[71], 0.0 },

{ data[72], 0.0 },{data[73], 0.0 },

{ data[74], 0.0 },{data[75], 0.0 },

{ data[76], 0.0 },{data[77], 0.0 },

{ data[78], 0.0 },{data[79], 0.0 },

{ data[80], 0.0 },{data[81], 0.0 },

{ data[82], 0.0 },{data[83], 0.0 },

{ data[84], 0.0 },{data[85], 0.0 },

{ data[86], 0.0 },{data[87], 0.0 },

{ data[88], 0.0 },{data[89], 0.0 },

{ data[90], 0.0 },{data[91], 0.0 },

{ data[92], 0.0 },{data[93], 0.0 },

{ data[94], 0.0 },{data[95], 0.0 } };



complex x3[96] ={  { data2[0], 0.0 },{data2[1], 0.0 },

{data2[2], 0.0 },{data2[3], 0.0 },

{ data2[4], 0.0 },{data2[5], 0.0 },

{ data2[6], 0.0 },{data2[7], 0.0 },
{ data2[8], 0.0 },{data2[9], 0.0 },

{ data2[10], 0.0 },{data2[11], 0.0 },

{ data2[12], 0.0 },{data2[13], 0.0 },

{ data2[14], 0.0 },{data2[15], 0.0 },

{ data2[16], 0.0 },{data2[17], 0.0 },

{ data2[17], 0.0 },{data2[18], 0.0 },

{ data2[20], 0.0 },{data2[21], 0.0 },

{ data2[22], 0.0 },{data2[23], 0.0 },

{ data2[24], 0.0 },{data2[25], 0.0 },

{ data2[26], 0.0 },{data2[27], 0.0 },

{ data2[28], 0.0 },{data2[29], 0.0 },

{ data2[30], 0.0 },{data2[31], 0.0 },

{ data2[32], 0.0 },{data2[33], 0.0 },

{ data2[34], 0.0 },{data2[35], 0.0 },

{ data2[36], 0.0 },{data2[37], 0.0 },

{ data2[38], 0.0 },{data2[39], 0.0 },

{ data2[40], 0.0 },{data2[41], 0.0 },

{ data2[42], 0.0 },{data2[43], 0.0 },

{ data2[44], 0.0 },{data2[45], 0.0 },

{ data2[46], 0.0 },{data2[47], 0.0 },
{ data2[48], 0.0 },{data2[49], 0.0 },

{ data2[50], 0.0 },{data2[51], 0.0 },

{ data2[52], 0.0 },{data2[53], 0.0 },

{ data2[54], 0.0 },{data2[55], 0.0 },

{ data2[56], 0.0 },{data2[57], 0.0 },

{ data2[58], 0.0 },{data2[59], 0.0 },

{ data2[60], 0.0 },{data2[61], 0.0 },

{ data2[62], 0.0 },{data2[63], 0.0 },

{ data2[64], 0.0 },{data2[65], 0.0 },

{ data2[66], 0.0 },{data2[67], 0.0 },

{ data2[68], 0.0 },{data2[69], 0.0 },

{ data2[70], 0.0 },{data2[71], 0.0 },

{ data[72], 0.0 },{data[73], 0.0 },

{ data[74], 0.0 },{data[75], 0.0 },

{ data[76], 0.0 },{data[77], 0.0 },

{ data[78], 0.0 },{data[79], 0.0 },

{ data[80], 0.0 },{data[81], 0.0 },

{ data[82], 0.0 },{data[83], 0.0 },

{ data[84], 0.0 },{data[85], 0.0 },

{ data[86], 0.0 },{data[87], 0.0 },
{ data[88], 0.0 },{data[89], 0.0 },

{ data[90], 0.0 },{data[91], 0.0 },

{ data[92], 0.0 },{data[93], 0.0 },

{ data[94], 0.0 },{data[95], 0.0 } };






spfftc(x1, &pos_i4, &neg_i1);


for (m = 0 ; m < M ; ++m)

{

fft[m].r = x1[m].r;

fft[m].i = x1[m].i;


outfile12 = fopen("fft_va.m","a");

fprintf(outfile12, "%e, %e\n",fft[m].r,fft[m].i);

fclose(outfile12);

}


 spfftc(x1, &pos_i4, &pos_i1);
spfftc(x2, &pos_i4, &neg_i1);


for (m = 0 ; m < M ; ++m)

{

fft1[m].r = x2[m].r;

fft1[m].i = x2[m].i;


outfile13 = fopen("fft_vb.m","a");

fprintf(outfile13, "%le, %le\n",fft1[m].r,fft1[m].i);

fclose(outfile13);

}


spfftc(x2, &pos_i4, &pos_i1);


spfftc(x3, &pos_i4, &neg_i1);


for (m = 0 ; m < M ; ++m)

{

fft2[m].r = x3[m].r;
```

```
fft2[m].i = x3[m].i;


outfile14 = fopen("fft_vc.m","a");

fprintf(outfile14, "%le, %le\n",fft2[m].r,fft2[m].i);

fclose(outfile14);

}


spfftc(x3, &pos_i4, &pos_i1);


for (m = 0 ; m < M ; ++m)

{

Va_freq = sqrt((pow((fft[m].r),2)) + (pow((fft[m].i),2)));


infile1 = fopen("Va_mag.m","a");

fprintf(infile1, "%le\n",Va_freq);

fclose(infile1);


Vb_freq = sqrt((pow((fft1[m].r),2)) + (pow((fft1[m].i),2)));


infile2 = fopen("Vb_mag.m","a");
```

```
fprintf(infile2, "%le\n",Vb_freq);

fclose(infile2);


Vc_freq = sqrt((pow((fft2[m].r),2)) + (pow((fft2[m].i),2)));


infile3 = fopen("Vc_mag.m","a");

fprintf(infile3, "%le\n",Vc_freq);

fclose(infile3);

}

}
```

# Bibliography

[1] ANSI/IEEE Standard 1001-1988, "IEEE Guide for Interfacing Dispersed Storage and Generation Facilities with Electric Utility Systems," IEEE, New York, 1989.

[2] Arun G. Phadke and Stanley H. Horowitz, *Power System Relaying*, John Wiley & Sons Inc, New York 1992.

[3] L. A. Kraft and G.T. Heydt, "A Method to Analyze Voltage Resonance in Power Systems", Trans. IEEE Vol.PAS-103, No.5, May 1984, pp. 1033-1037.

[4] Roger C. Dugan and Dwight T. Rizy, "Electric Distribution Protection Problems Associated with the Interconnection of Small Dispersed Generation Devices", Trans. IEEE Vol.PAS-103, No.6, June 1984, pp. 1121-1127.

[5] Herbert M. Pflanz and George N. Lester, "Control of Overvoltages on Energizing Capacitor Banks", Trans. IEEE, Vol.PAS-92, No.3, May/June 1973, pp. 907-915.

[6] C.L. Wagner, W.E. Feero, W.B. Gish, “Relay Performance in DSG Islands”, IEEE Transactions on Power Delivery, Vol.4, No.1, January 1989, pp.122-131.

[7] W.E. Feero and W.B.Gish, “Overvoltages caused by DSG Operation: Synchronous and Induction Generators”, Trans. IEEE, Vol.PWRD-1, 1986, pp.258-264.

[8] J. Rushton and K. Mewes, *Power System Protection 3*, A. Wheaton & Co., Ltd., Exeter, England 1981.

[9] J. W. Hodgkiss, *Power System Protection 2*, A. Wheaton & Co., Ltd., Exeter, England 1981.

[10] H. S. Petch and J. Rushton, *Power System 1*, A. Wheaton & Co., Ltd., Exeter, England, 1981.

[11] IEEE Guide for Transformer Impulse Tests, IEEE No.93, June 1968/ANSI C57.37.

[12] R. S. Bayless, J. D. Delman, D.E. Truax, and W.E. Reid, “Capacitor Switching and Transformer Transients”, IEEE Paper 86 SM 419-6, presented at the IEEE Power Engineering Society Meeting, July 1986.

[13] R. W. Alexander, “Synchronous Closing Control for Shunt Capacitors”, IEEE Transactions on Power Apparatus & Systems, Vol.PAS-104, pages 2619-2626, February 1985.

[14] Samuel D. Stearns and Ruth A. David, *Signal Processing Algorithms in Fortran and C*, PTR Prentice Hall Inc., Englewood Cliffs, New Jersey, 1993.

[15] E. O. Brigham, *The Fast Fourier Transform*, Prentice Hall Inc., Englewood Cliffs, New Jersey, 1974.

[16] R.P. O'Leary and R.H. Harner, "Evaluation of Methods for Controlling the Overvoltages Produced by Energization of a Shunt Capacitor Bank," CIGRE Report No. 13-05 (1988).

[17] Arun G. Phadke and James S. Thorp, *Computer Relaying for Power Systems*, John Wiley & Sons Inc, New York 1988.

[18] M. J. Damborg, R. Ramaswami, S. S. Venkata, and J. M. Postforoosh, "Computer Aided Transmission Protection System Design Part I", IEEE Transactions on Power Apparatus & Systems, Vol.PAS-103, pages 51-57, January 1984.

[19] Jian Chen, *Accurate Frequency Estimation with Phasor Angles*, Virginia Polytechnic Institute and State University, Blacksburg, Virginia 1994.

[20] M.G. Lauby, *Electromagnetic Transients Program (EMTP) Revised Rule Book Version 2.0*, Electric Power Research Institute, California 1989.

# VITA

Dewayne Randolph Brown is originally from Florence, South Carolina. He graduated from South Florence High School in 1984. After high school, he attended the University of South Carolina. He received his Bachelor of Science Degree in Electrical Engineering in 1990. After his undergraduate education, he attended North Carolina A & T State University. He received his Master of Science Degree in Electrical Engineering in 1992. In August of 1992 he attended Virginia Polytechnic Institute and State University to pursue his Doctor of Philosophy in Electrical Engineering. He is a member of IEEE and a recipient of the Office of Naval Research Fellowship.